# Probabilistic uncertainty in an interoperable framework

MATTHEW WILLIAM WILLIAMS

Doctor Of Philosophy


– ASTON UNIVERSITY –

*June 2011*

ASTON UNIVERSITY

# Probabilistic uncertainty in an interoperable framework

MATTHEW WILLIAM WILLIAMS

Doctor Of Philosophy, 2011

**Thesis Summary**

This thesis provides an interoperable language for quantifying uncertainty using probability theory. A general introduction to interoperability and uncertainty is given, with particular emphasis on the geospatial domain. Existing interoperable standards used within the geospatial sciences are reviewed, including Geography Markup Language (GML), Observations and Measurements (O&M) and the Web Processing Service (WPS) specifications. The importance of uncertainty in geospatial data is identified and probability theory is examined as a mechanism for quantifying these uncertainties. The Uncertainty Markup Language (UncertML) is presented as a solution to the lack of an interoperable standard for quantifying uncertainty. UncertML is capable of describing uncertainty using statistics, probability distributions or a series of realisations. The capabilities of UncertML are demonstrated through a series of XML examples.

This thesis then provides a series of example use cases where UncertML is integrated with existing standards in a variety of applications. The Sensor Observation Service - a service for querying and retrieving sensor-observed data - is extended to provide a standardised method for quantifying the inherent uncertainties in sensor observations. The INTAMAP project demonstrates how UncertML can be used to aid uncertainty propagation using a WPS by allowing UncertML as input and output data. The flexibility of UncertML is demonstrated with an extension to the GML geometry schemas to allow positional uncertainty to be quantified. Further applications and developments of UncertML are discussed.

**Keywords:** interoperability, probabilistic uncertainty, UncertML

# Contents

# List of Figures

# List of Tables

# Listings

## List of acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **ARPANET** | Advanced Research Projects Agency Network |
| **ASCII** | American Standard Code for Information Interchange |
| **CDF** | Cumulative Distribution Function |
| **CORBA** | Common Object Request Broker Architecture |
| **CRS** | Coordinate Reference System |
| **CRUD** | Create, Read, Update, Delete |
| **CSV** | Comma Separated Values |
| **DAL** | Data Access Layer |
| **DNS** | Domain Name System |
| **DTD** | Document Type Definition |
| **EPSG** | European Petroleum Survey Group |
| **ERD** | Entity-Relationship Diagram |
| **GEOSS** | Global Earth Observation System of Systems |
| **GIS** | Geographic Information System |
| **GML** | Geography Markup Language |
| **GPS** | Global Positioning System |
| **HTML** | HyperText Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **IDL** | Interface Description Language |
| **INTAMAP** | INTeroperability and Automated MAPping |
| **IP** | Internet Protocol |
| **ISO** | International Organization for Standardization |
| **JSON** | JavaScript Object Notation |
| **LIDAR** | Light Detection And Ranging |
| **MIME** | Multipurpose Internet Mail Extensions |
| **MPEG** | Moving Picture Experts Group |
| **MVC** | Model View Controller |
| **NetCDF** | Network Common Data Form |
| **O&M** | Observations & Measurements |

| | |
|---|---|
| **OGC** | Open Geospatial Consortium |
| **OWS** | Open Geospatial Consortium Web Service |
| **PDF** | Probability Density Function |
| **REST** | Representational State Transfer |
| **SAS** | Sensor Alert Service |
| **SensorML** | Sensor Model Language |
| **SOA** | Service Oriented Architecture |
| **SOAP** | Simple Object Access Protocol |
| **SOS** | Sensor Observation Service |
| **SPS** | Sensor Planning Service |
| **SQL** | Structured Query Language |
| **SRS** | Spatial Reference System |
| **SWE** | Sensor Web Enablement |
| **TCP** | Transmission Control Protocol |
| **TML** | Transducer Markup Language |
| **UDDI** | Universal Description Discovery and Integration |
| **uGML** | Uncertain Geography Markup Language |
| **UML** | Unified Modeling Language |
| **UncertML** | Uncertainty Markup Language |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **URN** | Uniform Resource Name |
| **W3C** | World Wide Web Consortium |
| **WCS** | Web Coverage Service |
| **WFS** | Web Feature Service |
| **WMS** | Web Mapping Service |
| **WNS** | Web Notification Service |
| **WPS** | Web Processing Service |
| **WSDL** | Web Service Description Language |
| **XML** | Extensible Markup Language |
| **XSD** | XML Schema Definition |
| **XSLT** | Extensible Stylesheet Language Transformations |

*To Rosie and Patrick*

# Acknowledgements

Writing this thesis has been the single most challenging task I have faced in my life so far. At times throughout the course of my PhD I have felt depressed, anxious, lost, confused, overwhelmed and exhausted. The result of the past few years of my life is contained within the pages of this thesis, which leaves me with a great sense of accomplishment. However, it would be rather conceited of me not to acknowledge the invaluable input I have received from people both professionally and personally.

First and foremost, I would like to thank my PhD supervisor, Dan Cornford. There have been times during my PhD when I have contemplated giving up and finding an easier life — generally after receiving one of Dan's 'motivational speeches'. However, the passion and enthusiasm that Dan demonstrates towards his work is infectious, and has inspired me to enjoy my work. I hope that the result of this research is something of which Dan can be proud. Not only did Dan help me professionally with reviewing papers, answering my questions and engaging in discussions; he has also been a friend. I have enjoyed working with Dan for the past four years and he has taught me several practical tips about travelling: don't spend more than ten pounds on shampoo, don't check-in any bags and only pack one spare t-shirt. I look forward to working with Dan in the future.

The other half of my supervisory team who I would like to thank is Lucy Bastin. Lucy started as my assistant supervisor in the second year of my PhD and has helped me ever since. Her dedication to my work has been priceless to me over the past few years, reviewing every piece of literature I have written. The thoroughness of her reviewing is unparalleled and no grammatical mistake goes unnoticed (she hasn't read this). The amount of time she spent helping me with this thesis cannot go unacknowledged: thank you. I have also enjoyed working with Lucy over the years and her sense of humour has made me smile on many occasions. The main travelling advice which Lucy provided me with is to always carry an eye mask.

There are so many people I would like to thank for helping with this thesis that it would double in size if I were to dedicate a paragraph to each and every one of them. So in summary I would like to thank everyone from the INTAMAP project. Their biannual meetings were always enjoyable and the discussions I had with various members have had a direct influence on my research. In particular, I would like to thank Edzer, Gregoire, Gerard, Jon, Olivier and John. I would also like to thank Maria Chli for her helpful comments on my first year report, which I have adopted into this thesis. I am grateful to all the people who have shown an interest in my work. UncertML is more popular than I ever imagined; constant stream of interest has helped me to not only to improve my work, but also to enjoy it. I would also like to thank the members of the Open Geospatial Consortium for providing support in producing the UncertML discussion paper, specifically Carl Reed.

The help and support I received professionally has proved vital, but I would not have made it this far if it had not been for the support I received from friends, family and colleagues. Working at Aston has been thoroughly enjoyable; various work colleagues have helped me enjoy my time at various stages of my PhD. In the early days Ben Ingram provided hours of welcome distraction with regular games of 'Pro Evo'; Remi Barillec, Richard Jones, Anthony Jones have also provided me with many enjoyable moments. I would like to thank Andrew 'Fatman' Hall whose daily conversations have kept me from losing the plot on several occasions. I am grateful to all of my

friends as you have no doubt put up with my complaining, I'll stop now, I promise.

I would also like to thank my family. My brothers: Daniel, Joseph, Joshua and Jacob and my parents Jane and Trevor. Growing up in such a large family has given me years of happy memories which have kept me going. My brothers are a constant source of amusement and entertainment which has helped me deeply. My parents have provided me with support, both emotional and financial, throughout my education; I hope that they can now be proud of what I have achieved.

Finally, I would like to thank my wife Rosie, to whom this thesis is dedicated. The people I have mentioned so far have all helped me, but non more so than Rosie, who has been there for me every single day. She has provided me with unquestioned love and support. She comforted me when times were hard, and helped me to enjoy life when times were good. Her faith in me has never faltered, even when my own did. I am under no illusion: without the continuous love that she has provided me I would not be writing these words. Thank you Rosie, I am forever indebted to you.

# 1 Introduction

*CONTENTS*

## 1.1 Interoperability

### 1.1.1 Historical context

The term 'interoperability' can refer to an array of different things. In general terms there are two forms of interoperability, syntactic and semantic. Syntactic interoperability is having a common set of terms for exchange of information, and semantic interoperability is having a shared understanding of what these terms mean. Interoperability is a vital concept in computer science. Without it many of the functions and applications of modern computers would not exist. The best example of interoperability within computing is the Internet. Without the clearly defined standards and protocols that allow communication between machines, the Internet could not function. Historically, many software vendors chose to remove interoperability from their applications by implementing proprietary binary file formats, in an attempt to lock users into their software. For example, the Microsoft Office file formats (e.g. `.doc`) forced users into purchasing their software so that they could open them. However, in 2008 Microsoft freely released the full documentation for their binary file formats, allowing other software vendors to implement them into their applications. With the benefits gained from interoperable, open standards, and the popularity of Extensible Markup Language (XML) Microsoft has since released an XML version of all office file formats (e.g. `.docx`). The generation of open file formats is a good example of syntactic interoperability within computers, which is essential for further levels of interoperability, e.g. semantic interoperability. Semantic interoperability within computer science is an unsolved challenge which constitutes a large research field. Therefore the contents of this thesis focus on syntactic interoperability and most concepts relating to semantic interoperability are considered out of scope.

### 1.1.2 Application in geospatial science

The majority of the work in this thesis is placed in a geospatial context. In a parallel approach to Microsoft, the early Geographic Information System (GIS) applications were based on closed-source, binary file formats. However, partly driven by the Open Geospatial Consortium (OGC), a strong shift towards open file formats is under way. Many of the larger GIS applications are now either compliant with OGC standards, or implement them. The Geography Markup Language (GML) specification is perhaps the key standard within the OGC and it has many uses in describing geometries, raster data and geographic 'features'. A series of Web service specifications for delivering geospatial information, either as raster images (Web Mapping Service (WMS))

or as raw GML data (Web Feature Service (WFS)) have proved popular and are now seeing implementations within mainstream GIS applications, e.g. GRASS [1] and ArcGIS [2]. Building on the success of the early standards, the OGC has recently branched into a new area entitled the 'Sensor Web'. Within the Sensor Web are a whole host of new XML-based standards for describing observations and sensors, as well as services for sharing and processing these observations. The vision of the Sensor Web is to allow any sensor to be plugged into the Web and automatically publish observations for discovery and consumption, a process termed 'plug and measure'.

## 1.2 Uncertainty and probability theory

### 1.2.1 Historical context

The work in this thesis is based around the concepts of probability theory, the brief exploration into the history of probability theory given here may prove useful for some readers. The origins of probability theory come from games of chance and gambling. Two famous French mathematicians, Blaise Pascal and Pierre de Fermat developed the fundamental principles of probability theory after a problem was posed to Pascal by Antoine Gombaud, a French nobleman. The problem was to decide whether or not to bet even money on the occurrence of at least one "double six" during 24 throws of a pair of dice. A seemingly well-established gambling rule led Gombaud to believe that betting on a double six in 24 throws would be profitable, but his own calculations indicated just the opposite. Shortly after Pascal and de Fermat's principles were formulated, Christian Huygens, a Dutch scientist, published the first book on probability in 1657; entitled *De Ratiociniis in Ludo Aleae*. Due to the relation to games of chance, and gambling, probability theory became very popular during the $18^{th}$ century. In 1812 Pierre de Laplace published a book that applied probability theory to many scientific areas outside of games of chance, something that had not been done previously. However, it was not until 1933 that a formal set of axioms of probability, defined by Andrey Kolmogorov, was agreed upon; and it is these axioms that form the basis of most modern probability theory approaches.

Probability theory has evolved since its roots in games of chance, and is now used in a variety of fields including genetics, economics, engineering and geoinformatics. This evolution has seen an extension from primarily discrete events to continuous variables. The large body of work based on probability theory provides numerous tools for inference and other statistical methods that any

[1] http://grass.fbk.eu
[2] http://www.esri.com/software/arcgis/index.html

framework using probability theory can utilise. This is a substantial benefit that other methods of uncertainty quantification do not posses, and is the primary motivating factor for the use of probability theory within this thesis (Papoulis and Unnikrishna Pillai, 1984). A more in depth review of probability theory is given in Section 2.3.

### 1.2.2 Application in geospatial science

Uncertainty exists in many aspects of the geosciences. As *all* geospatial data is gathered by observations of the world there will exist some deviation between an observed value and its corresponding true value. This deviation cannot always be known, if ever, and it must be estimated; the term 'uncertainty' refers to this estimation. Although mankind has been observing the Earth for centuries, the quantification of the inherent uncertainties in the observations is a relatively recent occurrence. Many techniques are used for quantifying such uncertainties including fuzzy set and probability theory. While no one method is 'better' than the other, we feel that the ability of probability theory to quantify complex univariate and multivariate uncertainties via a range of continuous probability distributions is most flexible. Consequently, this thesis adopts probability theory as the mechanism for quantifying uncertainty.

Despite the inherently uncertain nature of geospatial science, no effort has been made to provide an interoperable language for describing these uncertainties using probability theory. While it can be argued that probability theory provides a layer of semantic interoperability, without syntactic interoperability these ideas cannot be shared and processed by other applications.

## 1.3 Scientific contribution

This thesis looks at interoperability within the geosciences, with a specific regard to the quantification of uncertainty using probability theory. The intention here is not to make a judgement on the appropriateness of other methodologies of quantifying uncertainty (e.g., fuzzy set theory, Dempster-Shafer theory). Due to requirements dictated by the INTAMAP project (Chapter 5) and consortium, the work in this thesis focuses on probability theory. However, where it is deemed appropriate, alternative approaches are acknowledged.

The contribution of this thesis to the scientific community, and especially the geoinformatics field, can be summarised in the following points:

- The primary contribution of this work is UncertML: an XML language for describing uncer-

tainty using probability theory. Although many existing interoperable standards within the geospatial domain make references to the importance of uncertainty within their field, none make a conscious effort to model and quantify it. While UncertML was developed for use within the geospatial domain, it remains domain-agnostic and therefore has potential uses within a large number of fields.

- A secondary contribution of this research is the development of an interoperable interface to an automatic interpolation service, INTeroperability and Automated MAPping (INTAMAP). The INTAMAP interface allows users to retrieve interpolation predictions easily, and without prior knowledge of the underlying, complex, geostatistical processes. The system provides a middle ground between interpolation procedures that are often too simplistic (e.g. nearest neighbour interpolation) and those that are too complex for non-geostatisticians (e.g. kriging).

- A further contribution is the development of a GML application schema for quantifying positional uncertainty, Uncertain Geography Markup Language (uGML). GML is a widely adopted language that can be used to describe many elements of geospatial data and processes. One of GML's more successful aspects is the suite of geometry schemas that describe 1, 2 and 3 dimensional geometries. Despite the popularity of the GML geometry schemas, no mechanism for describing positional uncertainty exists. uGML extends the GML geometry types in a way that maintains backwards compatibility with existing GML software, yet allows more advanced software to utilise the positional uncertainty information quantified using probability theory.

- A smaller contribution is the development of an extension to a Sensor Observation Service (SOS) implementation, allowing quantification and retrieval of uncertainty within sensor observations. Utilising the Observations & Measurements (O&M) schema allows UncertML to be added within the `resultQuality` element. These 'uncertain' observations are then published using an extension to the 52° North SOS implementation. The benefits of this extension are demonstrated by the INTAMAP service, which can use the results of a SOS request for interpolation, a primitive form of service chaining.

- A final contribution of this thesis is an investigation and discussion into the term 'interoperability' and specifically the techniques the OGC has used to achieve interoperability within the geospatial domain. The work in this thesis is critical of several aspects of some OGC

standards and suggestions are made where we feel improvements could be made.

### 1.3.1 Thesis aims and objectives

The primary research aims of this thesis are outlined in the three objectives below:

**Objective 1** What is the most flexible, usable and interoperable design for a web-based information model for representing uncertainty in a probabilistic setting?

**Objective 2** Does the designed uncertainty information model integrate with existing standards in the context of observation and positional uncertainties?

**Objective 3** Does the designed uncertainty information model allow interoperable processing of data. Specifically, can UncertML be used in workflows that account for uncertainty?

## 1.4 Outline of thesis

**Chapter 1** is this introduction.

**Chapter 2** introduces the concepts of interoperability and uncertainty. An overview of XML, Web services and the Service Oriented Architecture (SOA) paradigm is given. A discussion follows on the topic of uncertainty and how probability theory can be used to quantify uncertainties. Finally, I discuss how interoperability and uncertainty are relevant in a geospatial context.

**Chapter 3** defines the Uncertainty Markup Language (UncertML) as a solution to the lack of an existing interoperable standard for exchanging uncertainty probabilistically. A conceptual model is outlined, using Unified Modeling Language (UML) models, that allows uncertainty to be described by realisations, summary statistics or probability distributions. An implementation, realised as a set of XML schemas, is then given and is illustrated through the use of numerous XML examples. Finally, a discussion on how UncertML can be integrated into existing standards is given.

**Chapter 4** reviews the SOS specification as a service for querying and retrieving observation data, encoded in the O&M standard. A critical analysis indicates that while uncertainty within observed data is inherent, no standard method for characterising and quantifying these uncertainties exists within the O&M or SOS specifications. An extension to the 52° North implementation of a

SOS is then discussed that allows UncertML to be integrated. The benefit of using UncertML to quantify uncertainties within observed data is motivated via a use case, providing a SOS interface for weather data.

**Chapter 5** introduces the INTAMAP project as an automatic service for interpolating sensor data. INTAMAP is built around a number of OGC standards to provide an interoperable interface. UncertML is a fundamental component of INTAMAP since it is used to quantify the uncertainties on the input data, as well as the inherent uncertainties of the interpolation result. A review on interoperability, and the approach taken by many OGC standards is discussed, with emphasis on the issues faced during the INTAMAP project.

**Chapter 6** introduces an extension to the GML specification that allows UncertML to quantify positional uncertainty. The chapter starts with an overview of positional uncertainty and current techniques for using probability theory to quantify it. A review of the GML standard follows, with a review of the current mechanisms for describing uncertainty with it. A discussion on existing extensions to GML that allow positional uncertainty to be described is given, followed by the implementation of a new extension, uGML.

**Chapter 7** summarises the work presented in the previous chapters and indicates possible future directions of research.

## 1.5 Disclaimer

The work presented in this thesis is original and has not been published anywhere else. Parts of the work, however, have been presented in the following conferences and papers:

- Work on the integration of UncertML with the INTAMAP project was presented at the StatGIS 2009 conference in Milos and published in a special issue of Computers and Geosciences (Williams et al., 2011).
- A review of interoperability in the field of geostatistics using UncertML and INTAMAP was presented in Southampton at the geoENV VII conference. (oral presentation) (Williams et al., 2010).
- A conceptual model for UncertML was presented at the GI Days 2008 conference in Muenster, Germany and was awarded 'best poster'. (poster presentation) (Williams et al., 2008b).

- Preliminary work on the use of UncertML within the Semantic Web was presented at the 7th International Semantic Web Conference in Karlsruhe, Germany. (oral presentation) (Williams et al., 2008c).
- UncertML was accepted as an official OGC discussion paper in 2008 (OGC 08-122r2, 2008).
- A further oral presentation was given at GISRUK in 2008 (Williams et al., 2008a).

# 2 Literature Review

*CONTENTS*

## 2.1 Foreword

This chapter examines the two domains mentioned in the introduction; interoperability (Section 2.2) and uncertainty (Section 2.3). Section 2.2.1 identifies how XML (Section 2.2.1) and Web services (Section 2.2.2) have provided a cornerstone for a new computer design paradigm, SOAs.

The concept of uncertainty is discussed in Section 2.3. Primarily focusing on probability theory, Section 2.3.1 describes how uncertain information may be quantified using the concept of random variables and in some cases two or more (correlated) random variables. Finally, Section 2.3.2 gives a brief overview of some other methods for quantifying uncertainty.

The focus of this work is combining the ideas of uncertainty and interoperability, with particular application to the geospatial domain. Section 2.5 reviews the geospatial context, with Section 2.5.1 looking at how uncertainty is accounted for in GISs, and how it is currently quantified. Existing OGC standards for interoperability are investigated. The following key standards are reviewed: GML (Section 2.5.2), Sensor Web Enablement (SWE) Common (Section 2.5.3) and O&M (Section 2.5.4).

This chapter provides the reader with a basic understanding of the concepts and ideas behind the work described in the subsequent chapters. It also identifies existing work within the fields, reviewing the state of the art and motivating the need for an interoperable standard for quantifying uncertainty, UncertML, described in Chapter 3.

## 2.2 Interoperability

With the creation of Advanced Research Projects Agency Network (ARPANET) in the late 1960s, the idea of distributed systems (concurrent processes running on separate physical machines) was realised. Since the birth of the Internet in the early 1990s, distributed systems have become far more commonplace and have extended beyond their humble beginnings as simple E-mail applications. With this wide-spread adoption comes a pertinent problem. Interoperability, an umbrella term meaning “the ability to exchange and use information”, raises several challenges when applied to computer systems.

This section investigates different emerging technologies that seek to aid interoperability by providing common data formats and communication protocols. Discussion is also provided about how the creation of several standards providing similar functionality, but using different design choices, may actually hinder interoperability in practice.

### 2.2.1 XML

Ironically, XML is not actually a markup language but a set of rules for constructing markup languages (Ray, 2001). XML provides a tag-based syntax for structuring data. Visually, XML is very similar to HyperText Markup Language (HTML). However, where HTML may be considered a presentation language, XML separates presentation from information and can be used to provide meaning to any data source. XML documents are self-describing and are both machine and human-readable. These traits make XML ideal for creating interoperable data formats and interfaces (i.e. Web services).

#### Elements and attributes

Elements make up a large proportion of an XML document. Every XML document has one top-level element, or, root element. XML does not provide any element names but explains how to create elements specific for each domain (Bray et al., 2006b). An element is realised as an open and close tag. The syntax of an open tag is the less-than character (<) immediately followed by the element name (or tag name) and finished with the greater-than character (>). A close tag begins with the character sequence </ followed by the tag name and finished with the greater-than character. A close tag must have the same name as the open tag to terminate the element.

```
<Aircraft code="G-NASW">
   <Manufacturer>Airbus</Manufacturer>
   <Model>A380</Model>
   <Engine>
      <Name>Trent 900</Name>
      <Thrust>311 Kn</Thrust>
   </Engine>
</Aircraft>
```

Listing 2.1: A simple example demonstrating the key components of XML: elements, attributes and data.

Elements can have children which may be elements themselves or processing instructions, comments or character data. When an element does not contain any children it is referred to as empty. Empty elements can be written in the shorthand form `<element />`. Elements may also contain any number of attributes. An attribute resides within the element open tag and takes the form of a `key=value` pair (e.g. `id="15123"`). Attributes are traditionally used to provide elements with a unique label so that they can be easily located within a larger document.

Listing 2.1 shows a simple XML fragment describing a particular aircraft. The `<Aircraft>`

element is the root element containing three children. An attribute, `code`, exists to distinguish this aircraft from others within the same document. The `<Manufacturer>` and `<Model>` elements have character data as their children whereas the `<Engine>` element contains a further 2 child elements. It is this hierarchical, tree structure of data that allows XML to describe practically any concept, regardless of its complexity.

### Namespaces

With XML providing the framework for anyone to construct data models, a problem quickly becomes apparent. Because XML designers are free to choose their own tag names it is entirely plausible that two, or more, designers will choose the same tag names for some of their elements (Skonnard and Gudgin, 2001). While not necessarily a problem when the two elements possess the same meaning, problems arise when two distinct concepts share a common name. Take, for example, the concept of a tree. To a horticulturist a tree represents a perennial woody plant, however, a computer scientist may refer to a tree as a 'commonly used data structure with a set of linked nodes'. Solving this problem relies on XML namespaces (Bray et al., 2006a).

A namespace in XML allows distinction between elements that share the same name, but are from different vocabularies. Usage of namespaces relies on two parts. Firstly, the namespace must be declared in an element. The syntax of a namespace declaration is `xmlns:prefix='URI'`, which maps the namespace Uniform Resource Identifier (URI) to a typically shorter prefix. When no prefix is specified the resulting namespace is referred to as the default namespace. Namespaces have a scope — that is, a set of elements to which they apply. The scope of a namespace declaration includes the element on which it is declared and any descendants. Secondly, elements that belong to a particular namespace are referenced using the prefix, `<prefix:Element>text</prefix:Element>`. If an element omits the prefix and a default namespace declaration exists, and is in scope, then it is assumed that the unqualified element belongs to the default namespace. If, however, no default namespace exists then the element does not belong to any namespace. Though such an element is valid XML, it would prove difficult integrating it into other XML vocabularies.

```
<hrt:Tree xmlns:hrt="urn:example:horticulture">
  <hrt:Name scientific="Corylus avellana">Hazel</hrt:Name>
</hrt:Tree>
```

Listing 2.2: XML example describing a tree. The addition of a namespace suggests that it is a plant and not a data structure.

```
<cs:Tree xmlns:cs="urn:example:computer_science">
   <cs:Node>
      <cs:SomeElement />
   </cs:Node>
</cs:Tree>
```

Listing 2.3: XML example describing a tree. The namespace suggests it is a data structure and not a plant.

Listing 2.2 provides a simple XML fragment describing a 'tree'. The inclusion of a namespace allows us to infer that it is a 'perennial woody plant' and not a commonly used data structure (Listing 2.3). Using namespaces also allows different vocabularies to be used within the same document. Imagine, for example, a computer scientist is constructing a tree (data structure) of trees (plants). Listing 2.4 shows how this may be accomplished by assigning the element 'Tree' to two different namespaces. The ability to combine different vocabularies allows new XML languages to be constructed from individual elements taken from several sources. This separation of concerns (Dijkstra, 1982) allows individual data modellers to define small, domain specific, vocabularies, thus promoting reusability — a fundamental requirement of interoperability (Erl, 2005).

```
<cs:Tree xmlns:cs="urn:example:computer_science">
   <cs:Node>
      <hrt:Tree xmlns:hrt="urn:example:horticulture">
         <hrt:Name scientific="Corylus avellana">Hazel</hrt:Name>
      </hrt:Tree>
   </cs:Node>
</cs:Tree>
```

Listing 2.4: XML example combining two types of 'tree' into a single data structure. The use of namespaces provides a clear distinction between the two `tree` elements.

Namespace declarations use URIs to distinguish themselves from one another. While these URIs may be Uniform Resource Names (URNs), demonstrated in Listings 2.2– 2.3, they are typically realised as Uniform Resource Locators (URLs). A detailed discussion about the benefits of both URLs and URNs can be found on page 33.

#### XML schema

The effectiveness of an XML language as an interoperable data format relies on the availability of tools to validate a given instance document, i.e. to ensure the instance document conforms to the logical structure defined in the data model. Without such validation tools the automatic processing

| Type | Description |
|---|---|
| decimal | An arbitrary-precision decimal number. |
| integer | An arbitrary-precision integer. |
| float | A single-precision floating point number. |
| double | A double-precision floating point number. |
| date | A Gregorian calendar date. |
| dateTime | An instant in time. |
| string | A general string type. |

Table 2.1: A list of commonly used XML schema simple data types.

of data, which underpins the ideology of interoperability, would be impossible. Within the XML framework there exists various tools for validating the structure, and content, of an XML instance document — the most popular of which is XML schema. XML schema was developed as a natural extension to Document Type Definitions (DTDs) and addresses some of their shortcomings (Bex et al., 2004). XML schema is a complex language; consequently, only the concepts that are used throughout this thesis are outlined. A more complete description of the XML schema language can be found in Vlist (2002) and Fallside and Walmsley (2004).

XML schema is itself an XML vocabulary providing a set of built-in elements that allow users to construct rules and constraints to apply to a class of XML documents. All elements defined by XML schema reside in the XML schema namespace and are thus prefixed with `xs`. In its simplest form, an XML schema defines a set of allowed XML elements. For instance, in the example in Listing 2.1, six elements exist: `Aircraft`, `Manufacturer`, `Model`, `Engine`, `Name` and `Thrust`. An accompanying schema would need to define all six of these elements, and their corresponding data types. For simple data types (e.g. string, integer etc.) XML schema provides a series of built-in types that may be used (Table 2.1). Listing 2.5 demonstrates how the Manufacturer element in Listing 2.1 could be defined in XML schema language.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Manufacturer" type="xs:string" />
</xs:schema>
```

Listing 2.5: An XSD example.

While these simple data types are adequate for elements such as `Manufacturer` and `Model`, where the content is a string value, certain elements (e.g. `Aircraft`) have complex content, such as other XML elements. Complex types in XML schema are defined by the xs:complexType. Depending upon the desired restrictions, a choice of elements can be placed inside a xs:complexType

| Element name | Description |
|---|---|
| all | A model group that allows elements in any order. |
| any | An element wildcard. |
| choice | A model group that allows one of the elements contained within it. |
| element | An element declaration or reference. |
| group | A reference to a named model group |
| sequence | A model group that allows elements in a fixed order |

Table 2.2: A list of allowed XML schema elements within an xs:complexType element.

element. Typically, this is a xs:sequence element that specifies a series, and order, of XML elements that can reside within this type. Table 2.2 provides the full list of options for a xs:complexType. Attributes are also considered complex and therefore must be defined within a complexType element.

One of the key benefits of XML schema over DTD is that it allows for modularisation (Bex et al., 2004). Identifying commonly used elements within a domain and placing them in the root of the schema allows these components to be reused throughout. Elements can then reference these common components via the `ref` attribute. An example of this modularisation can be seen in Listing 2.6 which gives a complete XML schema for the example in Listing 2.1. Notice how the `Aircraft` and `Engine` elements are defined at the root level and that the `Aircraft` element makes a reference to the `Engine` as one of its child nodes. Due to the `Engine` element being defined at root level, and not inside the `Aircraft` element, it has a global scope. If, for example, a user wished to create a schema allowing aircraft engineers to monitor engine maintenance records, importing the aircraft schema would allow reference to the pre-existing `Engine` element, precluding the need to duplicate elements. The XML schema language also provides the mechanism for deriving from existing types. For example, if the aircraft engineers wished to add more detailed information to the basic `Engine` model then inheriting by extension allows this. Another method of inheriting in XML schema is by restriction. Inheriting by restriction allows a child element to use only a subset of the values that are provided by the parent element. However, inheritance by restriction is less commonly used than extension and does not form any part of the UncertML schemas.

There are many choices to be made when designing an XML schema. One high-level choice is whether to employ a strong-typed or weak-typed methodology. In a strongly-typed schema all of the elements and attributes are known *a priori* and can therefore be defined within the schema. Defining all elements and attributes relies on a domain expert to identify which concepts require modelling and which do not. Listing 2.7 provides an example of a strong-typed XML fragment

```
<xs:schema>

  <xs:element name="Aircraft">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Manufacturer" type="xs:string"/>
        <xs:element name="Model" type="xs:string"/>
        <xs:element ref="Engine" maxOccurs="8"/>
      </xs:sequence>
      <xs:attribute name="code" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <!-- The Engine element may exist outside of an aircraft -->
  <xs:element name="Engine">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Thrust" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Listing 2.6: An XML schema for defining aircraft (Listing 2.1). A separate Engine element is defined providing some modularisation.

describing a Golden Eagle. Contrast this with the example in Listing 2.8, which describes the same Golden Eagle, but with a weak-typed design.

```
<GoldenEagle tag="GE1312">
   <description>...</description>
  <wingspan units="m">2.4</wingspan>
  <height units="m">0.8</height>
</GoldenEagle>
```

Listing 2.7: A Golden Eagle defined using a strong-typed design. All elements and attributes have been defined by a domain expert *a priori*.

Each design methodology has its relative strengths and weaknesses. A strong-typed design provides easier validation of XML fragments as all possible elements and attributes have been defined in a schema. This means that if two computer systems implement the code necessary to parse and understand all elements and attributes in this schema, they can then communicate with each other. A disadvantage of strong-typed designs is the difficulty of extension. Due to all known elements and attributes being well defined in a schema, the addition of a new element, or attribute, is not trivial. Extending the schema to provide additional elements or attributes breaks any in-

```
<Bird type="GoldenEagle">
   <property name="tag" type="string">GE1312</property>
   <property name="description" type="string">...</property>
   <property name="wingspan" type="double">2.4</property>
   <property name="height" type="double">0.8</property>
</Bird>
```

Listing 2.8: A Golden Eagle defined using a weak-typed design. The elements and attributes are not known *a priori* so generic terms are used.

teroperability that existing software systems had, unless they update their code to accommodate the changes. Consider the computer systems in the scenario above. If a change occurs to the schema which they rely on to communicate, then they must either both chose to ignore the update, or modify their systems to accommodate the change. In the event that only one system converts to the new schema they can no longer communicate safely in the knowledge that they speak the same language. A weak-typed design, on the other hand, only has generic elements and attributes predefined. Using URIs to provide semantics (possibly through an online dictionary) allows the XML vocabulary to be extended easily. This methodology is particularly suited to domains where there are an undetermined number of concepts that need modelling, e.g. types of probability distributions (Section 2.3.1). A major disadvantage, however, is that weak-typed XML fragments can only be validated to a simple degree. Take the Golden Eagle example; the `property` elements can contain any information with no way to validate it against a schema. The weak-typed schema simply states that a `Bird` element may contain any number of `property` elements, not what these properties should contain, or indeed how many properties a particular bird should have. This poses the question: "how can two computer systems communicate with each other efficiently when their common language is not well-defined?". This question results in a trade-off, namely -: provide a strong-typed schema to allow interoperability and risk future extensions breaking this, or provide a generic, weak-typed, schema that limits the effectiveness of interoperability? In this work it is argued that perhaps some middle ground can be found that provides a set of common elements and attributes (strong-typed), but also provides generic elements and attributes to allow extension (weak-typed).

The existing literature on strong and weak-typed designs does not provide a comparative analysis of the merits of each philosophy. However, discussion can be found on the use of strong and weak-typed designs in programming languages (Madsen et al., 1990) and database design (Yang and Parker, 2007). Providing a comparison between strong and weak-typed methodologies in an XML schema context is a minor contribution of this thesis.

### URIs — URNs vs URLs

The success of the Web could be attributed to the role played by resource identification. In a Web context this is handled by the URI scheme. While Web users will be accustomed to the URL scheme as the main means of resource identification, several other schemes are in existence. Jacobs and Walsh (2004) state that users of URIs should not try to infer properties of the referenced resource through the URI — URIs should be opaque. One of the common misconceptions of the Web is that URLs should describe the resource they identify. While this is typically the case, Mendelsohn and Williams (2007) explain that a URL such as: `http://example.org/weather/Birmingham/today.html` need not necessarily refer to today's weather in Birmingham, nor that it points to an HTML file. The role of the URI agent is to provide a unique identifier for resources under their governance, the use of metadata in these URIs is optional, although considered good practice, where the URIs are intended for direct use by people (Mendelsohn and Williams, 2007).

Identifiers on the Web have two roles:

- to name or denote a resource,
- to provide a resolvable locator for a Web resource.

The strength of the URL scheme is that it combines both of these roles and uses well established protocols for resolutions (e.g., Domain Name System (DNS), Hypertext Transfer Protocol (HTTP) etc.). However, certain weaknesses of URLs have led to other schemes being developed. Perhaps the main disadvantage of the URL is the issue of persistence. Due to the fact that the resolution of URLs is handled partly by DNS servers, the resource identifier is left prone to changes in server configuration, e.g. a resource may move within the server and therefore nullify any existing identifiers of that resource. The URN syntax was developed to overcome the issues of URLs by providing persistent, location-independent, resource identifiers (Moats, 1997). The syntax of a URN takes the form of `urn:<namespace identifier>:<namespace specific string>`. An example URN could be written as `urn:airbus:a380` where 'airbus' is the namespace identifier and 'a380' is the namespace specific string. In stark contrast to URLs, the location-independence of URNs make them particularly difficult to resolve. While a universal URN resolver does exist [1], it merely points to a series of documents for each scheme, i.e. it is manual. However, it is not essential to resolve resources. Reasoning can be achieved by simply knowing that resource A is

[1] `http://www.iana.org/assignments/urn-namespaces/`

the same as resource B. If both these resources share the same identifier then the resolution of that identifier may not be required.

With each scheme having disadvantages, the choice then focusses on the relative importance placed by the URI agent on persistence and resolvability. The OGC use the URN scheme extensively through their specifications for defining phenomena, units of measure and other such concepts (OGC 07-036, 2007; OGC 07-022r1, 2007; OGC 07-000, 2007). However, UncertML uses the URL scheme for all identifiers due to the ease of resolving URLs and the benefits that this brings.

### 2.2.2 Web Services

The term 'Web service', in its simplest form, refers to an architecture that allows applications to talk to each other. It was coined by Microsoft in 2000 to describe a series of standards for allowing machines to communicate via a network (typically the Internet). There are now over 50 Web service standards in circulation today (Josuttis, 2007), referred to as the WS-* stack, performing a variety of tasks ranging from security to service management through business processes. However, it is widely agreed that 5 core standards exist (Josuttis, 2007; Erl, 2004, 2005): XML, HTTP, Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description Discovery and Integration (UDDI).

**Extensible Markup Language (XML)** is used to describe models, formats and data types. Most other Web service standards are also based on XML, XML schema or XML namespaces.

**Hypertext Transfer Protocol (HTTP)** is a low-level protocol used by the Internet. While it is not a stipulation that Web services communicate via HTTP, it is the most frequently used mechanism. Details of the HTTP protocol are outside the scope of this thesis, but more information can be found in Fielding et al. (1999).

**Web Service Description Language (WSDL)** is used to define service interfaces. Typically split into three distinct sections, a WSDL file first explicitly describes the interface, i.e. the operations available and any input and output parameters. The second section is the binding, i.e. the protocol and format for which the operations are provided. The final section is the physical location (endpoint) of the Web service, i.e. a URL or address where the service is available.

**Simple Object Access Protocol (SOAP)** defines the Web services protocol. Existing at a higher

level than HTTP, SOAP is the specific format for exchanging Web service data. In a typically SOAP/WSDL Web service all messages are wrapped in a SOAP envelope that provides additional information to the contained data, e.g. security information.

**Universal Description Discovery and Integration (UDDI)** is a standard for managing Web services, i.e. registering and discovering services. However, early indications suggest that UDDI has not been adopted as widely as the other 4 standards (Josuttis, 2007).

The combination of these standards aims to provide interoperability between systems. Describing data payloads and protocols using XML and SOAP allows communication between platform and language-neutral vendors. However, with the recent uptake of Web services and the generation of numerous other service standards, implementing a 'Web service' becomes a daunting task. In addition to the numerous service standards, account must be taken of the different versions of these standards. It is not safe to assume that version 1.1 of a particular standard can interoperate with version 1.0 (and vice versa). Even when specific versions of specific standards have been decided upon, the breadth of some standards is overwhelming and interoperability remains a problem. The culmination of these issues was the creation of a Web services Interoperability Organisation (WS-I). A core task faced by the WS-I is the creation of profiles. A profile is a defined set of specific-version standards accompanied by guidelines and conventions for using these standards in order to ensure interoperability. In 2006 a 'basic profile 1.1' was specified indicating versions of the 5 core specifications that should be used in combination with 150 additional conformance requirements (Ballinger et al., 2006). The fact that in accounting for only 5 of the 50 specifications, 150 additional conformance requirements were amassed clearly demonstrates the huge task faced by the WS-I, and interoperability as a whole.

#### RESTful Web services

The phrase Web service is often used to refer to Web services constructed using SOAP and the WS-* standards, as discussed above. However, the concept of a Web service is more abstract and should be free from any implementation assumptions. The popularity of SOAP services has led to this confusion but it should be noted that other frameworks for constructing Web services exist. One such architecture is called Representational State Transfer (REST).

Introduced by Fielding (2000), REST style architectures involve both clients and servers who interchange messages. These requests and responses are based upon representations of resources. A resource is defined as "any information that can be named", for example, a document or image.

| Verb | Description |
|---|---|
| PUT | Creates a new resource |
| GET | Gets the current state of a resource |
| POST | Transfers a new state onto a resource |
| DELETE | Deletes a resource |

Table 2.3: A list of HTTP verbs and their perceived meaning in a 'RESTful' architecture.

Each resource in a RESTful system (a system conforming to the REST constraints) must have a unique identifier, or URI, and be accessed through a uniform interface. Traditionally, RESTful services use the HTTP protocol as it provides a complete vocabulary of verbs (or methods) for manipulation of resources (Table 2.3). The perceived simplicity of REST, due to the use of existing standards (HTTP, XML, Multipurpose Internet Mail Extensions (MIME), URI etc.), is arguably its greatest strength (Pautasso et al., 2008). HTTP client and server software is widely available in all major programming languages meaning RESTful services can be constructed and consumed with minimal overheads. Scalability and discovery are also key components of a REST architecture, which is in contrast to WS-* based services. However, the simplicity of REST is also a weakness, since the limited set of verbs available to HTTP makes it difficult for extended functionality to be exposed via a REST interface.

There is an extensive literature on comparisons of REST and SOAP-based services (Pautasso et al., 2008; Vinoski, 2007; Muehlen et al., 2005; Richardson and Ruby, 2007). While opinions vary wildly depending on the individual's background, a general consensus can be drawn that both REST and SOAP services aim to solve different problems and that the choice of interface adopted should be made according to the functional requirements of a software system. Advocates of RESTful services would argue that, despite the limited number of verbs, RESTful services are capable of solving any problems. Likewise, SOAP-based advocates would argue that SOAP/WSDL services are not overly complicated and provide much more functionality than RESTful services. However, it can be agreed that for simple services that provide access and limited manipulation (e.g., edit, delete etc.) of resources a RESTful solution is preferred. When more complex processing is required, or where a service extends beyond the manipulation of resources a SOAP/WSDL architecture is preferable.

#### OGC Web Services

The OGC is an international consortium of companies that provide standards and interfaces for geospatial information on the Web. One standard is the OGC Web Services Common Specifi-

cation (OGC 06-121r3, 2007) which outlines the common operations that "geo-enabled" Web services should provide. The OGC decided that existing Web service architectures (SOAP and REST) lacked the required functionality, instead preferring to outline a new, geospatial-oriented, framework. The hub of an Open Geospatial Consortium Web Service (OWS), is the capabilities document. Within a capabilities document are comprehensive details of a particular service instance, including any operations, inputs and outputs. Comparisons can be made between the capabilities document and a WSDL file from traditional Web services as they both describe the contract between service and user. In fact, efforts have been made to fuse existing OGC services (WFS, WMS, Web Coverage Service (WCS) etc.) with WS-* standards (WSDL, SOAP, UDDI etc.) (OGC 03-028, 2003; OGC 03-014, 2003; OGC 04-049r1, 2005; OGC 04-050r1, 2005) with OGC 04-060r1 (2004) drafting a second version of the OWS Common specification to incorporate the WS-* stack. The official stance of the OGC towards WSDL is explained in the OWS specification (OGC 06-121r3, 2007):

> ... operations can return service metadata using different data structures and/or formats, such as WSDL ... When such operation(s) have been sufficiently specified and shown more useful, the OGC may decide to require those operation(s) instead of, or in addition to, the current GetCapabilities operation.

The design ethos of OWSs is that customising services to individual use cases detracts from interoperability, a view shared by Josuttis (2007). Instead, a set of generic services have been created to cater for a broad range of situations. Each OWS specifies a number of methods, or verbs, that adhere to the foundations laid out in the OWS specification (OGC 06-121r3, 2007). The combination of capabilities document and a succinct set of verbs culminates in a form of hybrid SOAP/REST service that inherits the benefits of both architectures. A more detailed discussion of some OWSs, including WFS and WMS can be found in Section 2.5.

#### Service-Oriented Architecture

SOA is an abstract programming paradigm. Frequently, Web services are used as a mechanism for implementing a SOA, so frequently in fact, that it has become a common misconception that Web services are a requirement of a SOA (Erl, 2004, 2005; Josuttis, 2007). In its simplest form a SOA is a methodology for decomposing logic required to solve a large problem into several smaller pieces, or services. Erl (2005) draws comparison from a cosmopolitan city, with each business providing a distinct service to a larger community. This separation of concerns allows a larger, distributed, environment to exist.

| Principle | Description |
|---|---|
| *Loose coupling* | Services maintain a relationship that minimises dependencies and only requires that they retain an awareness of each other. |
| *Service contract* | Services adhere to a communications agreement, as defined collectively by one or more service descriptions and related documents. |
| *Autonomy* | Services have control over the logic they encapsulate |
| *Abstraction* | Beyond what is described in the service contract, services hide logic from the outside world. |
| *Reusability* | Logic is divided into services with the intention of promoting reuse. |
| *Composability* | Collections of services can be coordinated and assembled to form composite services. |
| *Statelessness* | Services minimise retaining information specific to an activity. |
| *Discoverability* | Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms. |

Table 2.4: Key principles of a Service-Oriented Architecture.

While distributing logic into distinct logical units is nothing new, there are several notable differences to an SOA. Tight coupling of units imposes dependencies that may inhibit their potential by limiting any future extension. SOA emphasises a loose coupling idea whereby each unit should be self-governed, allowing them to evolve and grow independently from the rest of the system. While encouraging independence between the units of logic there still exists a series of fundamental conventions they must adhere to. Referring back to the cosmopolitan city example, each business must share a common currency for exchange of goods and services and it is necessary that all employees speak the same language as the consumers. Within an SOA, units of logic, or services, are required to conform to a set of principles that allow them to remain autonomous while maintaining a sufficient amount of commonality.

For services to interact within a SOA they must be aware of one another, and this is achieved through the use of a service description. A service description, in its most basic form, describes the name and location of a particular service, as well as its data exchange requirements. The similarity of this service description and a WSDL, or capabilities, document is just one reason why Web services have become synonymous with a SOA. Table 2.4 outlines the key principles of service-orientation according to Erl (2005).

SOAs are one method of implementing a distributed system, however, other architectures exist. Common Object Request Broker Architecture (CORBA) is a standard that allows software

components written in multiple languages to work together, regardless of system hardware. This definition is similar to that of a Web service, but the implementation of both methods is very different. CORBA uses an Interface Description Language (IDL) that describes the interface to shared objects. Mappings are then provided that convert the IDL into a native programming language (e.g., Java, C++ or ADA). The IDL in CORBA can be compared to XML, in that they both describe the interface to an object, or service, respectively. A benefit of CORBA, over an XML-based approach, is that the data is transported in binary format, which results in drastically smaller file sizes. However, a problem faced by CORBA implementations is that the connections between distributed components are made over Transmission Control Protocol (TCP)/Internet Protocol (IP), but unlike Web services, these connections are not on the commonly-used port 80. The result of this is that communication can often be impossible when a restrictive firewall, which only allows communication via port 80, is in place. The use of existing ports and protocols by Web services has seen a faster adoption by users than CORBA.

While there are benefits of using CORBA (and other distributed architectures) in some scenarios, it is generally considered that an XML and Web service approach provides a more loosely coupled and interoperable solution (Chung et al., 2003; Gokhale et al., 2002).

## 2.3 Uncertainty

Uncertainty affects many aspects of our lives each and every day. Take, for example, the weather — will it rain today? This is a frequently asked question and answering with absolute certainty is not generally possible. Uncertainty, however, extends far beyond the future weather conditions. What will the lottery numbers be this coming weekend? Will the bus arrive on time? What is the chance I will get hit by a car today? In science, uncertainty is principally concerned with the measurement of quantities and the accuracy of these measurements. Uncertainty may also arise as a by-product of computational models — which often propagate the measurement uncertainties from inputs. When making decisions based on uncertain information, or concepts, it is important to quantify the uncertainties. For instance, if a person believes that the probability that it will rain today (perhaps using a weather forecasting system) is 0.1 then they may decide not to take an umbrella. Using the available information, they made the decision that the 10% risk of it raining was not large enough to warrant the inconvenience of carrying an umbrella. It is the quantification of uncertainties that allows such reasoning. Uncertainty can often be distinguished as being either aleatory or epistemic. The former arises because of the natural, unpredictable

variation of the world while the latter is caused by a fundamental lack of knowledge about the world. The concept of aleatory and epistemic uncertainties are summarised concisely by the US Environmental Protection Agency (2008) below:

> "Uncertainties in the scientific sense are a component of all aspects of the modeling process. Uncertainties that affect model quality include aleatory uncertainty (due to the inherent stochastic nature of the world), uncertainty in the underlying science and algorithms of a model (model structure uncertainty), data uncertainty, and uncertainty regarding the appropriate application of a model (application niche uncertainty)."

This section outlines the principles behind probability theory, a mechanism for quantifying uncertainty, as well as other complimentary theories.

### 2.3.1 A probabilistic approach to quantifying uncertainty

Probability theory is a branch of mathematics that quantifies uncertainties through the use of probabilities (Papoulis and Unnikrishna Pillai, 1984). A simple scenario, often referred to, is the toss of a coin. Assuming it is a fair coin one can assign a probability of $\frac{1}{2}$, or 50%, to each possible outcome (head or tails). Extending this idea allows us to apply the same principle to the roll of a die. Assuming a fair die, the probability of a six appearing is $\frac{1}{6}$. While these are trivial examples the same principles can be applied to more complex events allowing us to make decisions based on calculated risk.

#### Objective and subjective probabilities

There are two ways to think of probabilities: objective and subjective. The two examples given previously are both examples of an objective probability. An objective probability is where it is known that, repeated enough times, a particular event will occur a given proportion of the time. For example, rolling a die enough times will demonstrate that the number six appears $\frac{1}{6}$ of the time. Of course, with probability nothing is certain. It is plausible that a die could roll 100 times and not have a single six appear. However, roll the die an infinite number of times and eventually the occurrences of the number six will converge to $\frac{1}{6}$. The concept of calculating probabilities by repeating experiments, or measurements, is called a 'frequentist' approach to probability theory and is how probability theory is commonly taught. However, there are many situations where taking repeat measurements is not possible. In such circumstances a subjective, or Bayesian, approach is better suited.

Subjective (Bayesian) probability differs in that it is a belief about an event. For instance, one might have a belief that there is a 20% chance it will rain tomorrow. Subjective probabilities are

liable to change as our knowledge of the event increases. For instance, if tomorrow morning there are dark rain clouds in the sky the prior belief of 20% chance of rain may increase significantly. This 'updating' of prior beliefs into a 'posterior', conditional, probability is achieved using Bayes' theorem, hence 'Bayesian'. A key benefit of Bayesian over frequentist theory is that it can be adopted for all scenarios. For example, if a Bayesian statistician was asked to provide a belief that the roll of a die would produce a 6, they could base their belief on the repeated throw of a die (an objective approach). However, if a frequentist statistician was asked the probability that it will rain tomorrow then they must adopt a subjective approach, i.e. the event is not repeatable. Bayesian probability theory provides a robust, flexible framework that allows uncertainty to be quantified using a variety of methods. A Bayesian approach to probabilities is adopted throughout this thesis.

#### Probability distributions

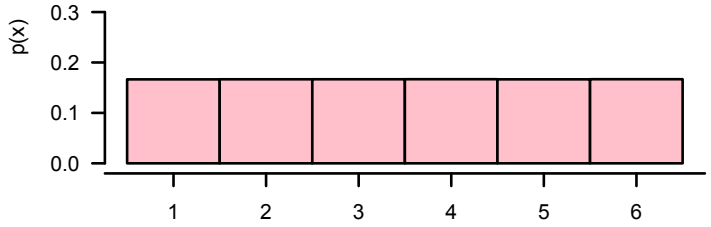



Figure 2.1: A uniform probability distribution for the roll of a six-sided die.

When rolling a die there are six discrete possible outcomes. The set of these outcomes is referred to as the sample space (denoted by $\Omega$). For any given outcome there is a probability that it will occur. These probabilities are identified by a probability distribution. In the example of rolling a die the outcomes have an equal probability of occurring and are therefore uniformly distributed, Figure 2.1. However, probability distributions can take any form. Imagine two dice are rolled and the sum of the numbers shown is recorded. The sample set (all possible outcomes of the experiment) can be defined as:

$$\Omega = \{2,3,4,5,6,7,8,9,10,11,12\} \tag{2.1}$$

However, P(2) = $\frac{1}{36}$ whereas the P(7) = $\frac{6}{36}$, i.e., not all outcomes are equally likely. Figure 2.2 shows the full probability distribution of this experiment.

In all the examples so far the sample set has had a finite number of possible outcomes. The flip of a coin has two outcomes, the roll of a die has six and the sum of two dice has eleven

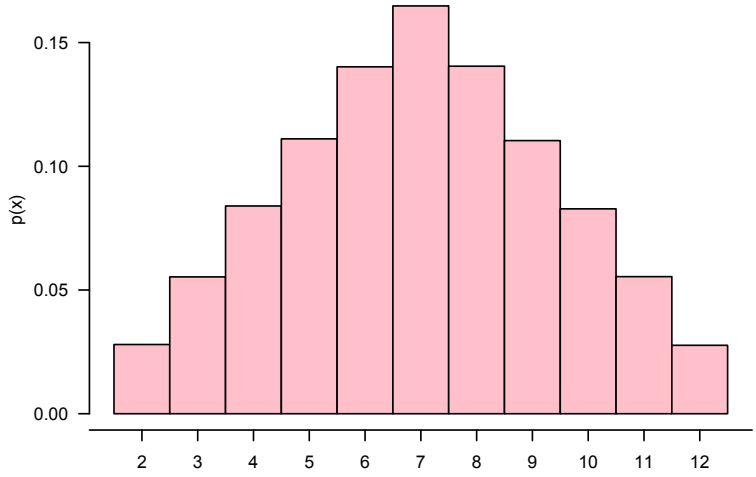



Figure 2.2: Probability distribution for the sum of the scores shown by two dice.

possibilities. The probability distributions in such examples are referred to as discrete probability distributions. However, what happens when our sample space has an infinite set of outcomes, for example, possible values for the current air temperature? In these instances a continuous probability distribution needs to be considered.

A continuous probability distribution can be defined by a probability density function, or PDF. The PDF is a curve that describes the density of probability at each point in the sample space. However, it should be noted that the probability of any single value occurring from a continuous sample space is 0. When dealing with continuous probability distributions the area under the PDF curve must be considered, that is, the probability that the value lies between two bounds.

#### Random variables

The previous section looked at probability distributions, or functions, that map events from a sample space to real numbers. When a quantity follows a particular probability distribution it can be referred to as a random variable. Examples of random variables could be the temperature at a given point, the number of cars on a section of road at a given time or even the number of hairs on a person's head. Both continuous and discrete examples are described as random variables. For instance, if a die is rolled, the random variable $X$ could be the number shown, or the random variable $Y$ could be the number of sixes shown. Examples like this, where multiple functions exist on the same sample space, are commonplace and are one of the benefits of working with random variables.

A random variable is typically described by a probability distribution, as discussed in the previous section. As with probability distributions, random variables come in two main types,

discrete and continuous. Discussion so far has mainly focussed on discrete scenarios, but often the more interesting cases are continuous. Imagine a random variable, $D$, that is the distance (in miles) travelled by a jet engine before a significant malfunction occurs. A random variable is described by a probability density function, of which there exist an infinite number. However, in practice it is easier to work with a smaller class of well-understood probability distributions, e.g. the Gaussian, or normal, distribution. For instance, the random variable $D$ is not normally distributed (it is positive and the Gaussian distribution allows for negative values), but because $D$ is large, a Gaussian assumption is acceptable in this instance. A Gaussian distribution can be characterised by its probability density function and the parameters location ($\mu$) and scale ($\sigma^2$). The distribution for random variable $D$ can be seen in Figure 2.3).

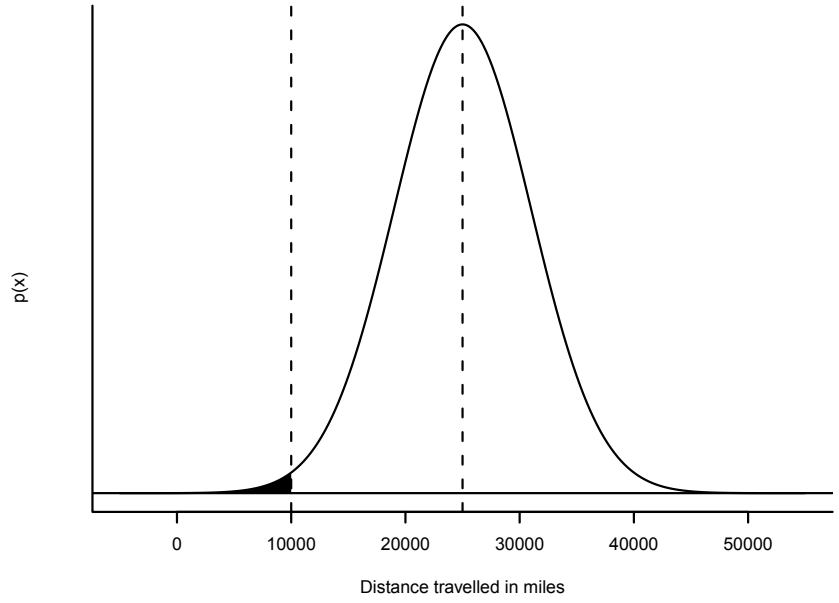



Figure 2.3: Gaussian distribution of the random variable $D$, describing the distance travelled by a jet engine before significant malfunction.

Using the distribution in Figure 2.3 one can ask such questions as "what is the probability a malfunction will occur before 10,000 miles?" or "what is the expected distance for an engine to travel without malfunction?". Probabilities, such as in the first query, can be generated by calculating the area under the curve, which equates to a probability. The expected value of a continuous random variable is calculated differently and leads us on to the topic of moments.

While the probability distribution fully quantifies a random variable, often it is sufficient to describe it through a series of statistics, called moments. The first moment of interest is the expected value. The expected value is typically the 'average' value or mean of a random variable. Unfortunately, the first moment alone is generally not sufficient to describe a random variable.

This is because the probability of any single value of a continuous random variable is 0. For the expected value to be of use in decision making, consideration of the second order moment, or variance is required. The variance is a measure of spread that indicates how far from the expected value the values of a random variable can be. For the random variable, $D$, given above, calculating these two moments is trivial. This is because the location parameter of a normal distribution is equal to the expected value and the scale parameter is equivalent to the variance. However, with other distributions the parameters are not the same as the moments, and in such instances a moment generating function is required.

In some instances the distribution of a random variable may not be known and subsequently it cannot be described using a probability distribution or its moments. To counter this, a series of possible outcomes, or realisations, of a random variable can be provided. Given sufficient realisations it is possible to infer the underlying probability distribution and consequent uncertainty. Such realisations are often generated from a Monte Carlo procedure.

Quantiles may also be used to quantify a random variable. A quantile is a point on the Cumulative Distribution Function (CDF). Typically, a series of quantiles are taken at regular intervals along the CDF. Providing a collection of quantiles instead of a probability distribution provides an approximation of the random variable that may provide a sufficient quantification. This non-parametric approximation is necessary when the parameters of a variable are not known.

There are a large range of probability distributions including the beta, gamma, log-normal and Poisson distributions. The choice of distribution used is often dependant on the variable one wishes to model. For instance, the Poisson distribution is often used for modelling spatial densities (Ripley, 1977). However, due to the relative simplicity of the Gaussian distribution it is commonly used across a range of domains. This thesis does not attempt to list all possible probability distributions as the implementation of a weak-typed design (Section 2.2.1) allows the encoding of any distribution.

#### Joint random variables

All probability distributions considered up to this point have been of single random variables. However, often the behaviour of two or more random variables together need to be considered. As previously stated a random variable, $X$, is a real-valued function on $\Omega$ (the sample space). Often there will be two or more random variables of interest defined on the same sample space and their joint behaviour may be of interest. Consider an example where a weather station records

| Time | Air temperature, $T$ | Air pressure, $P$ |
|---|---|---|
| 00:00 | 6.6 | 994.8 |
| 06:00 | 3.6 | 993.8 |
| 12:00 | 10.0 | 995.8 |
| 18:00 | 9.1 | 994.8 |

Table 2.5: Values taken from two variables measuring air temperature and air pressure at four sample times.

the air temperature, air pressure and wind speed. Each of these random variables may depend on the other and knowing the probabilities of the joint outcomes may be of interest. If the air temperature is random variable, $T$, and the air pressure is random variable $P$ then the distribution of $(T,P)$ is called the joint distribution. The distributions of $T$ and $P$ are referred to as the marginal distributions. While it is possible to obtain the marginal distributions from the joint distribution it should be noted that the inverse is not true. A basic rule of probability theory (independence) states that two events $A$ and $B$ are independent if

$$P(A,B) = P(A)P(B). \tag{2.2}$$

The same applies for joint distributions. If $T$ and $P$ have a joint distribution function $f(t,p)$ then $T$ and $P$ are independent if and only if for all $t$ and $p$

$$p(t,p) = p_t(t)p_p(p). \tag{2.3}$$

However, when two random variables are not independent there needs to be some mechanism for quantifying the degree of correlation between them. Let us expand on the weather example. Imagine that the air temperature, $T$, and air pressure, $P$, are measured at four moments in time throughout the day (00:00, 06:00, 12:00 and 18:00), the corresponding results are recorded in Table 2.5.

To quantify the degree to which the above variables, $T$ and $P$, vary together (covary) the covariance coefficient must be calculated, this is given as

$$cov[T,P] = \frac{\sum(T - E[T])(P - E[P])}{n}. \tag{2.4}$$

Where $E[]$ denotes the expectation of a specified variable and $n$ is the number of samples (4 in this instance). Applying (2.4) to the data in Table 2.5 results in $cov[T,P] = 1.6$. A positive covariance, as in this instance, means that the two variables move together, i.e., as one variable

increases, so does the other. A negative covariance means that as one variable increases the other decreases.

Typically a jointly Gaussian distributed random variable is described by the expected value, or mean, of each of the random variables and a variance-covariance matrix. The variance-covariance matrix, $\Sigma$, (just covariance matrix from herein) has variances in its diagonal and covariance in its off-diagonal elements.

$$\Sigma = \begin{bmatrix} cov[T,T] & cov[T,P] \\ cov[P,T] & cov[P,P] \end{bmatrix}$$

The joint random variable can therefore be quantified by the following information

$$E[T] = [7.325]$$

$$E[P] = [994.8]$$

$$\Sigma = \begin{bmatrix} 6.1 & 1.6 \\ 1.6 & 0.5 \end{bmatrix}$$

While the example given only involves two random variables, scaling these rules to larger collections is natural. In general terms the covariance matrix has the size $n^2$ where $n$ is the number of variables.

The multivariate Gaussian distribution provides a powerful mechanism for quantifying uncertainties for multiple, correlated, variables. However, there are other methods of quantifying multivariate uncertainty. Other multivariate probability distributions include the multivariate Student distribution and the Wishart distribution. Probability distributions provide a complete quantification of joint random variables. However, as with univariate scenarios, sometimes the underlying distribution is not known. In this case it is possible to provide a summary of jointly distributed random variables by providing their expected values and a covariance matrix. It is also possible to represent joint random variables via a series of multivariate realisations. All examples of joint random variables in this thesis use the multivariate Gaussian distribution.

### 2.3.2 Other methods of quantifying uncertainty

In the previous section probability theory was proposed as the natural mechanism to quantify uncertainty. The idea of a random variable as a function mapping events in the sample space to a real number was introduced along with a discussion of how these random variables can be described

by their probability distributions. In this section a brief overview of some other mechanisms for quantifying uncertainty is given.

#### Fuzzy set theory

In traditional set theory, the membership of an element in a set is binary, i.e. it is either a member or it is not. Fuzzy set theory extends on this idea and says that elements of a set have a degree of membership. The degree to which an element belongs to a set is calculated through the use of a membership function which maps membership to the real unit interval $[0,1]$, with 0 representing absolute falseness and 1 absolute truth. Consider the following statement: "Madrid is hot". In traditional set theory the element (Madrid) is either a member of the set (hot) or not. However, the concept of 'hot' is subjective and the membership of Madrid might be considered fuzzy. Given the average temperature in Madrid is 24 degrees Celsius during August, a truth value of 0.6 could be assigned to the statement . This can be represented symbolically as $m_{HOT}(Spain) = 0.6$, where $m$ is the membership function operating on the set of hot cities which returns a value between 0 and 1. Although both fuzzy set theory and probability theory operate over the same numeric range and have similarities with 0 representing false and 1 representing truth, distinctions can be made. In the above statement, the frequentist probabilistic approach would say "there is a 60% chance Madrid is hot", while the fuzzy approach says "Madrid's degree of membership within the set of 'hot' cities is 0.6". The semantic difference is important as the probabilistic approach states that Madrid is hot or not, with a 60% certainty. However, the fuzzy approach might state that Madrid is 'quite hot'. The probabilistic approach mentioned above is a frequentist view. A Bayesian statistician, on the other hand, would simply state a belief about the description of the temperature in Madrid.

A corollary to fuzzy set theory was fuzzy logic, a mechanism for reasoning with fuzzy sets. Fuzzy logic is built upon the use of `IF-THEN` rules that apply to fuzzy sets, which take the typical form `IF` *`variable`* `IS` *`property`* `THEN` *`action`*. A simple example for an air-conditioning unit is given below.

```
IF temperature IS hot THEN full power;
IF temperature IS warm THEN low power;
IF temperature IS cold THEN off;
```

Fuzzy logic has been adopted in a wide range of applications including air conditioning units,

on-board vehicle controls, remote sensing and even the Massive [2] engine used to simulate large-scale battles (used in the Lord of the Rings films). Fuzzy logic provides an alternative representation of uncertainty to probability theory. However, for the concepts discussed within this thesis, probability theory provides sufficient representation and is therefore the chosen methodology for the work in Chapters 3, 4, 5 and 6.

#### Dempster-Shafer theory

Probability theory can effectively model uncertainty (Jaynes, 2003). However, probability theory cannot describe ignorance. Consider the following example:

If no information exists about a coin, in probability theory it is assumed that there is a 0.5 probability of heads and 0.5 probability of tails. However, in another scenario, the coin is known to be fair, it is a fact that it is 0.5 probability of heads and 0.5 probability of tails. In the two different scenarios, the same conclusion was reached. Representing total ignorance in probability theory is problematic. In Dempster-Shafer theory, for the ignorance scenario, the belief of heads and the belief of tails is 0. For the fair coin scenario, the belief of heads is 0.5, the belief of tails is also 0.5.

An alternative parametrization can say that the probability of heads is $p$ and tails is $1-p$. In the case of ignorance, we say that $p$ is uniformly distributed on the $[0,1]$ interval. Upon a fair coin assumption, we say that $p = 0.5$ with probability 1 and has other values with probability zero. Another kind of ignorance might be that $p$ has some beta distribution on the interval $[0,1]$. Put in context, a gambler is offered the choice of two bets: to bet that the toss of a coin will result in heads, or to bet on the outcome of a motor race between a Formula 1 champion and a World Rally Championship driver. Assuming the gambler is ignorant about motor sports, they would be more inclined to take the bet on the coin, where the probabilities are known. Dempster-Schafer theory allows consideration of the confidence of specific outcomes.

Dempster-Shafer theory is a generalisation of Bayesian probability theory, and some argue that total ignorance does not exist, i.e. we always have some belief.

#### Imprecise probability and Bayes Linear theory

A key principle in Bayesian probability theory is that the probabilities are subjective. In a typical Bayesian framework it is assumed that these probabilities are precise. However, it can be argued

[2] http://www.massivesoftware.com/

that it is impossible to specify the subjective probabilities of an individual precisely, especially when there is a lack of information, or data. Imprecise probability theory does not use a single probability to represent uncertainty, but upper and lower probabilities, or expectations. (Walley, 1996) illustrates this with an example. Marbles are drawn blindly from a bag, the event $R$ is the event that a red marble is drawn. (Walley, 1996) argues that because nothing is known about the constitution of the bag (i.e. there could be no red marbles) assigning a non-informative prior over the event $R$, as in a classical objective Bayesian approach, is problematic. The author states that "the problem is not that Bayesians have yet to discover the truly non-informative priors, but rather that no precise probability distribution can adequately represent ignorance." (Walley, 1991). Imprecise probability theory eliminates the need for a single probability associated with $R$ and replaces it with an upper and lower probability. In the case of the bag of marbles the author argues that as nothing is known about the make-up of the bag they should not bet on or against the event $R$. Thus, before any marbles are drawn, the lower probability is $\underline{P}(R) = 0$ and the upper probability is $\overline{P}(R) = 1$. In layman's terms, the lower probability is the degree to which one is confident the drawn marble will definitely be red. As nothing is known about the marbles in the bag a value of 0 is chosen. The upper probability represents the degree to which one is concerned the next marble might be red. As all the marbles might be red a value of 1 has been chosen. The values of 0 and 1 for lower and upper probabilities is a special case that represents no constraint on the event $R$. However, as more marbles are drawn from the bag the lower and upper probabilities will change. In the case where the lower and upper probabilities are equal for $R$ then this is said to be a precise probability, i.e. a fair gamble. More information on imprecise probability theory can be found in Walley (1991, 1996).

Bayes linear statistics adopts a similar view to imprecise probabilities; subjective probabilities are difficult to specify at the necessary level of detail. In a traditional Bayesian approach, every possible outcome of an event must be enumerated. Bayes linear statistics uses subjective expectation as a primitive, probability is then defined as the expectation of an indicator variable. Instead of specifying a subjective probability for every outcome of an event, an analyst specifies the subjective expectation for just a few quantities that they are interested in or feel knowledgeable about. An adjusted expectation is then calculated using a generalisation of Bayes' theorem.

Imprecise probability theory, Bayes linear statistics and Dempster-Schafer theory all provide alternative approaches to representing uncertainty. However, traditional Bayesian statistics (i.e. precise probability theory) is a complete, principled and sufficient framework for the work in this

thesis. Therefore, all future examples of uncertainty quantification assume a Bayesian position, unless stated otherwise.

## 2.4 Positional uncertainty

The process of locating an object on the surface of the Earth has intrinsic inaccuracies. These inaccuracies may be relatively large, when using primitive techniques (e.g. measuring from a known reference point), or small, when using a high-precision Global Positioning System (GPS) device. Regardless of the process used to locate an object and the relative error, the resulting position is still uncertain. An interesting experiment to prove this theory would be to measure the longitude of the Prime Meridian in Greenwich using a consumer-grade GPS device; depending on the precision of the device the longitude is unlikely to show exactly 0°E, and will vary across time.

In order to describe the positional uncertainty of geospatial objects Heuvelink et al. (2007) state that the objects should be classified by their primitive parts and by the types of movement they support under uncertainty. Heuvelink et al. (2007) continue by suggesting the three following, first-order, classifications:

- point objects — objects that are single points;
- rigid objects — multiple point objects whose relative positions cannot change under uncertainty (i.e. the internal angles of a polygon would remain fixed); and
- deformable objects — multiple point objects whose relative positions can vary under uncertainty, i.e., internal angles may change.

The same classification is adopted here, and the framework developed should provide the flexibility to quantify the positional uncertainty of all object types.

#### Spatial autocorrelation

Many GIS operations involve objects with multiple points. If each point contains some degree of uncertainty, the spatial autocorrelation of that uncertainty must be considered. Longley et al. (2005, p. 143) provide an example that demonstrates the effect of spatial autocorrelation in this context. A common GIS query might be to determine the distance between two points. Imagine that these points were measured using a GPS device and that each measurement has a mean

distance error of 50m. Longley et al. (2005) argue that if these two points were measured at significantly different time intervals, with different satellites above the horizon, then the errors of the two points may be considered independent. Independence between the errors of the two points means that one point's measured position might be 50m North of its true location, while the other was 50m South, giving a possible error in distance of 100m. This example assumes that the two points are on the same longitude, illustrated in Figure 2.4. However, Longley et al. (2005) continue to say that if both points are measured at similar times, using the same combination of satellites, then the errors would be more likely to be similar (e.g. 50m North and 40m North). With similar errors, the error in distance becomes much smaller (10m in this instance). The differences between these two examples can be quantified by the degree of spatial autocorrelation in the factors influencing errors in measurement. There may also exist a correlation between the errors in the dimensions of a coordinate (i.e, $x$ and $y$), however, this is typically not as common (Longley et al., 2005, p. 142). Locations obtained via a GPS receiver are relatively accurate, i.e. they have small positional errors. The effects of spatial autocorrelation on positions with small errors could be perceived as insignificant. However, in many situations, it is either too expensive or simply not feasible to obtain locations via GPS receivers. Commonly, a process known as geocoding, where a location is approximated from address information within a GIS system, is used. These locations contain much larger errors and the effects of spatial autocorrelation, where it exists, are greater (Zimmerman et al., 2010).

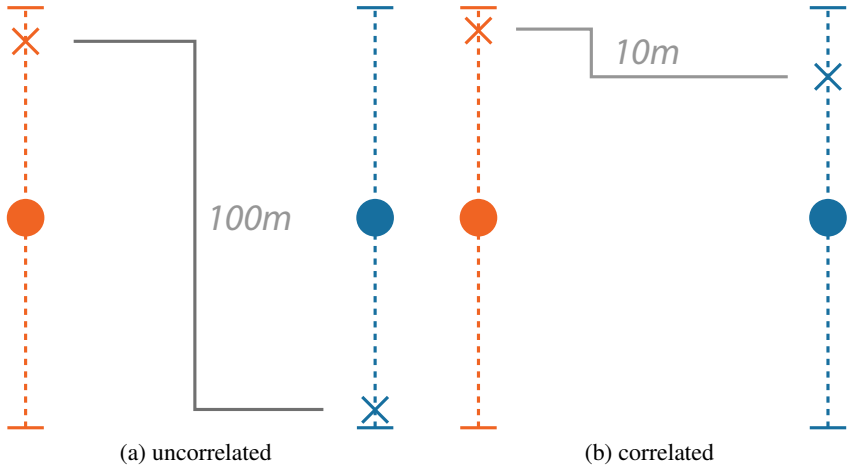



Figure 2.4: Effects of spatial auto correlation on distance of errors. The circles represent the true location of two points. The crosses are the measured locations in two scenarios.

Since spatial autocorrelation is a common phenomenon in geospatial sciences, it is imperative that the developed framework can support not only the correlation of errors between points

of an object, but also the correlation of errors between the individual coordinates at a location. A methodology that provides the necessary flexibility to quantify positional uncertainty and its spatial autocorrelation is probability theory (Section 2.3.1).

### 2.4.1 A probabilistic approach to positional uncertainty

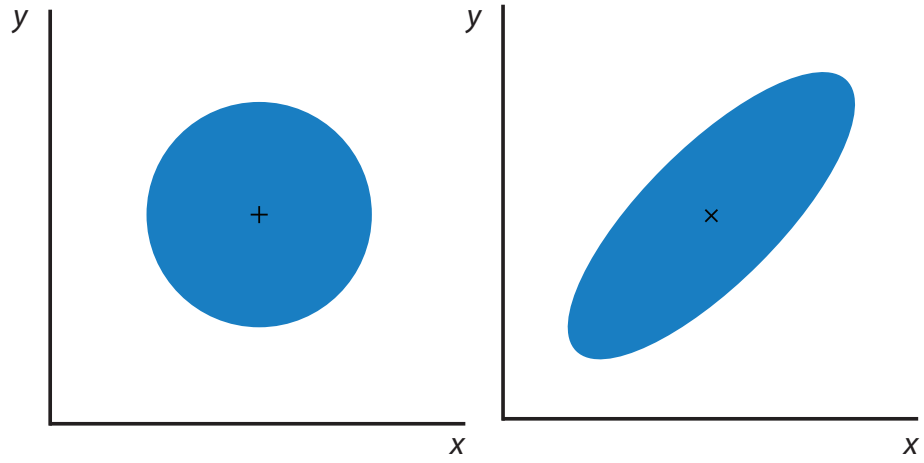
When faced with uncertainty in spatial data, Hunter and Goodchild (1995) outlined three approaches. The first approach is labelled the 'do nothing' approach, whereby the user simply ignores the errors. Clearly this approach is unsatisfactory in some cases. Another approach is to use an epsilon band (from the Greek letter (ε) for error) (Perkal, 1966). The epsilon band approach defines a 'buffer' around a point, or line segment, which represents an area of uncertainty. For example, a measured point in space with an epsilon band of 50m, means that the point, in reality, may fall anywhere within this 50m buffer. However, this approach does not provide an explicit confidence level (typically it is assumed that an epsilon band has a 100% confidence) and, consequently, lacks the required information for meaningful processing, or propagation (Shi, 1998; Heuvelink, 1998). The final approach in Hunter and Goodchild (1995), and the one discussed here, is the use of probability theory. In simple terms, probability theory provides the ability to add a confidence (or probability) to the error. For example, the 50m epsilon band could be improved by saying "the point in reality falls within 50m of the measured point, with 95% confidence". While providing more information, an error and confidence level does not provide a complete quantification of an uncertain location. For instance, it does not tell us how the uncertainty is distributed; is the true location equally likely to be anywhere within the 50m bounds? An uncertain variable can be more completely described by its Probability Density Function (PDF), discussed in Section 2.3.1.

The following sections discuss how probability theory can be used to quantify positional uncertainty for the three classifications outlined by Heuvelink et al. (2007).

#### Points

Point objects that exist in 3-dimensional Cartesian space may contain 3 coordinates: $x$, $y$ and $z$ (though time, $t$, may also be considered as an extra dimension). In the presence of uncertainty, the 'true' value of each coordinate (e.g. $x$) is unknown and therefore should be represented by a random variable, $X$ (Section 2.3.1). When enough information is known about $X$ it can be quantified by its marginal PDF. However, in some situations, a complete description (PDF) of

$X$ is not possible. A less complete summary description of $X$ may be given by providing a mean (expected value) and a standard deviation. The mean value provides a measure of central tendency, while the standard deviation is a measure of spread (the average deviation of $X$ from the mean).




(a) uncorrelated (b) correlated

Figure 2.5: Visualisation of point uncertainty.

A complete description of an uncertain point object that exists in space and time would therefore contain four marginal PDFs for $x$, $y$, $z$ and $t$. However, this assumes that each coordinate is independent of the others. When the uncertainties are statistically dependent a multivariate PDF is required. The parameters of a multivariate PDF, as discussed in Section 2.3.1, are a vector of expected values (of each coordinate) and a covariance matrix. The covariance matrix contains the variance (square of the standard deviation) for each coordinate, and the covariance between each pair of coordinates. Figure 2.5 demonstrates the effects of statistical dependency between coordinates (2 in this case). Figure 2.5a (left) assumes that the uncertainty in the $x$ and $y$ coordinates are independent, whereas Figure 2.5b (right) has some correlation between the uncertainty in $x$ and $y$. The distorted shape in Figure 2.5b suggests that as the value of $x$ is overestimated, it is likely that the value of $y$ will also be overestimated. If two coordinates have a perfect correlation (i.e. 1.0) the plot would look like a straight line. However, in practice, variables rarely have a correlation of 1.0.

#### Deformable objects

A deformable object, defined by Heuvelink et al. (2007), is an object with multiple points whose relative distances and angles can vary under uncertainty. This is in contrast to a rigid object whose internal angles and distances cannot vary, i.e. the object's topology is always preserved.

The same techniques discussed in the previous section may be used here to quantify the uncertainties of a deformable object. However, when using summary statistics (i.e. mean and standard deviation), some assumptions must be made. Firstly, it has to be assumed that the individual coordinates of each point are independent. Secondly, it has to be assumed that no statistical dependence between the points of the deformable object exists.

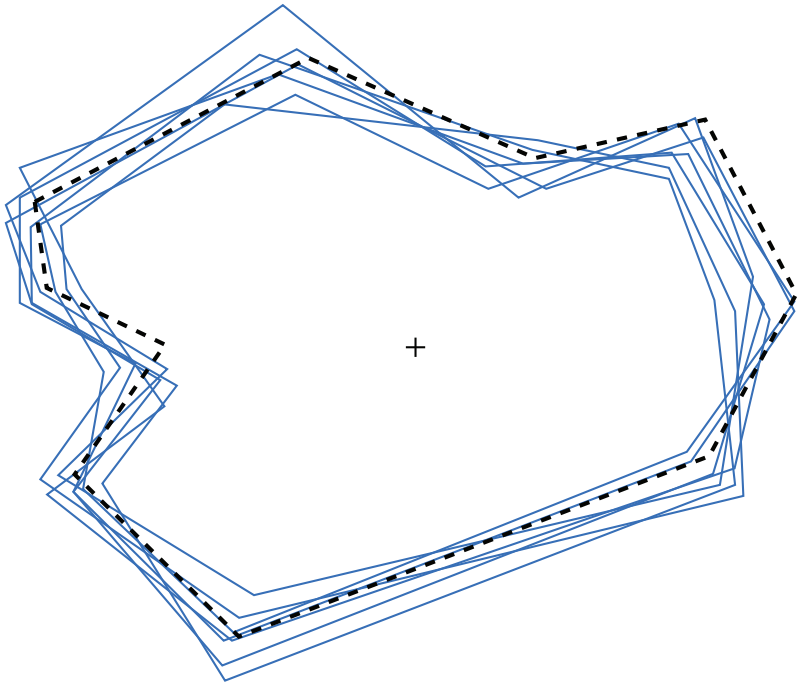


Figure 2.6: Illustration of positional uncertainty within a deformable object. The black, dashed, outline represents the measured object and each blue outline represents a possible realisation of that object. The realisations were generated according to the positional uncertainties in each of the objects points.

These limitations dictate that only on rare occasions are summary statistics satisfactory for completely quantifying the uncertainties of a deformable object. Typically, a joint probability distribution is used to describe deformable objects. This joint PDF must encode the dependencies between each coordinate of a given point, but also the dependence between the primitive points of an object. Due to the complexity of such a joint PDF, a joint normal distribution is often assumed. The parameters of a joint normal PDF would be a vector of expected values, for all dimensions at each point within the object, and a covariance matrix. Assuming that each point has 4 dimensions ($x$, $y$, $z$ and $t$) the covariance matrix would have the size $4n$ x $4n$, where $n$ is the number of points within the object. The variances of each marginal PDF form the diagonal of the covariance matrix and the covariances between each pair of coordinates and points are placed elsewhere. Figure 2.6 illustrates a deformable object and a number of possible realisations due to uncertainty.

### Rigid objects

A rigid object can be thought of as a deformable object where the correlation between any deflection of the primitive points is 1. This prohibits the changing of internal angles and distances between points. However, rigid objects may still undergo a rotation and translation under uncertainty. Figure 2.7 shows a rigid object (black dashed line) and a number of realisations. Each realisation has undergone a rotation about the centroid and a translation.

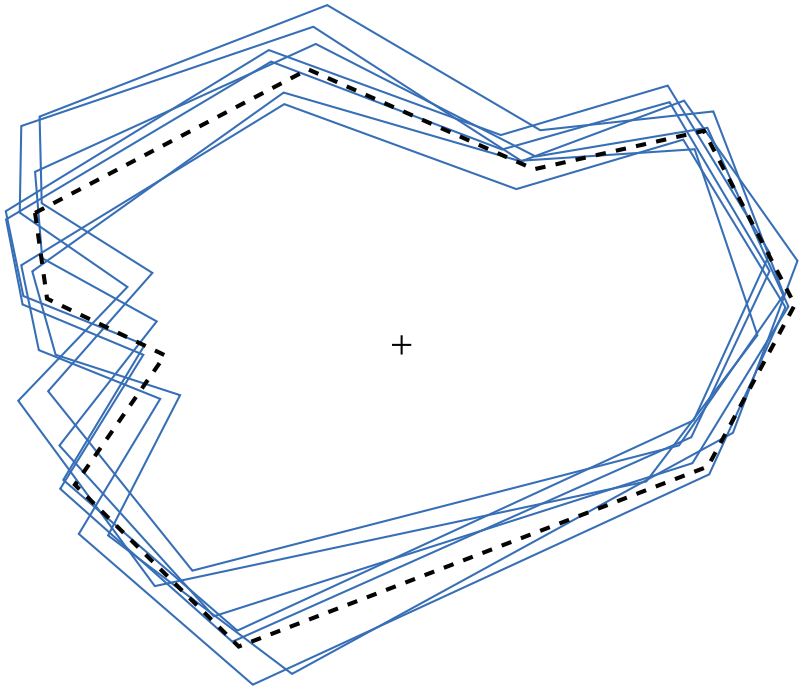


Figure 2.7: Illustration of positional uncertainty within a rigid object. The black, dashed, outline represents the measured object and each blue outline represents a possible realisation of that object. Each realisation has undergone a rotation and a translation about the centroid (marked with a black cross).

The rigid topology of the object means that its positional uncertainty can be characterised with a single point. For example, when one point moves, the position of every other point is immediately known. Therefore, quantifying the uncertainty for a rigid object is far simpler than for a deformable object. In fact, only two pieces of information are required: the joint PDF of one of the primitive points (or another reference point, e.g., a centroid) and a series of rotation angles. The joint PDF is required for the positional coordinates of the reference point ($x$, $y$, $z$ and $t$). There must also exist a rotation angle for every combination of coordinates (e.g. $xy$), for 4 dimensions this gives the rotation angles $\theta_{xy}$, $\theta_{xz}$, $\theta_{xt}$, $\theta_{yz}$, $\theta_{yt}$ and $\theta_{zt}$. The order in which the rotations are made must be fixed, as it affects the movement of the object (Heuvelink et al., 2007). Although far more complicated, it is possible and plausible to quantify the positional uncertainty

of a rigid object using a joint PDF — as described for a deformable object. However, this may be unnecessarily complicated when the same information can be represented with a joint PDF of a single point, in conjunction with a number of rotations.

#### Data or metadata?

When discussing uncertainty there are two schools of thought. The first perceives uncertainty as metadata, i.e., the temperature is observed to be $10°C$, but there is some associated uncertainty about that observation. The uncertainty could be characterised by a Gaussian distribution with a zero mean and a variance of $2°C^2$, for example. This method of representing uncertainty is suited to situations whereby a variable is directly observed, i.e., the distribution represents the discrepancy between the measured value and reality, and this discrepancy may include many uncertainties.

However, a variable that is not directly observed i.e., one that is derived is inherently uncertain. The value of a derived variable is a distribution — the exact value is not known. Taking the previous example, a temperature derived from an observation could be represented by a Gaussian distribution with a mean of $10°C$ and a variance of $2°C^2$. Both approaches can also be used for measurements of location (e.g. using a GPS device), however, whether or not the location observed is a direct or derived observation, in this case is not so clear. One could argue that all observations are in fact derived at some level, so that all variables are inherently uncertain and should be characterised by a probability distribution.

While both approaches result in the same information, the semantic difference between whether uncertainty is data or metadata affects how one presents the information.

### 2.4.2 Other approaches to positional uncertainty

The discussion thus far has focussed on a probabilistic view of positional uncertainty. However, as mentioned in Section 2.3.2, other approaches to quantifying uncertainty exist. The other methodologies discussed in Section 2.3.2 are also applicable to positional uncertainty. For instance, fuzzy set theory has been widely used in various applications and is supported by a substantial and varying literature (e.g., Fritz and See (2005); Lucieer and Kraak (2004); Fisher (1999) and Liu et al. (2009)). Despite the volume of literature, confusion often arises between the concepts of probability theory and fuzzy set theory (De Bruin, 2000; Fisher, 1994). The confusion stems from the fact that both theories use the scale of 0.0–1.0 to represent, respectively, probabilities or degrees

of membership. Despite this similarity both methodologies are in fact different enough to prove useful in complementary areas of geospatial science. Work has also been done using rough set theory (Duckham et al., 2001) but is not considered in this thesis.


Figure 2.8: Conceptual model of uncertainty in spatial data from Fisher (1999).

Fisher (1999) outlines a conceptual model of uncertainty in spatial data (Figure 2.8), adapted from Klir and Yuan (1995). The crux of the conceptual model is whether a spatial object is 'well defined' or 'poorly defined'. In cases where an object is well defined, probability theory is deemed applicable. However, if the object is poorly defined and, more specifically, could be described as vague, then fuzzy set theory may be used. An example of a poorly defined object could be a locality, e.g. the city of Birmingham (Liu et al., 2009). The Sorites Paradox, discussed in Section 2.3.2, can also apply to the citizenship of people in the Birmingham area. For example, if a person living in the town centre of Birmingham was asked whether they lived in Birmingham they would undoubtedly say yes. If a person living 1 mile outside of the city centre was asked they would also probably say yes. However, at some stage a person located at, say, 5, 6 or 7 miles outside the centre may not consider themselves to be a citizen of Birmingham. It is the inherent vagueness of the boundary of Birmingham, and all other localities, that causes this phenomenon. Fuzzy set theory allows citizens to partially belong to the set of 'Birmingham citizens', via a membership function. Scenarios such as this are typically where fuzzy set theory has become popular. The use

of fuzzy set theory for quantifying vagueness does have benefits (e.g., simplicity), however, we would argue that probability theory can also be applied to the concept of vagueness. In this context it is important to raise the difference between objective (frequentist) and subjective (Bayesian) probability theory (Section 2.3.1). Using the Birmingham example, one could say that they have a belief that a particular person is a citizen of Birmingham with a probability of 0.75. That is not to say, however, that given a sample of 100 citizens, 75 will claim to live in Birmingham and 25 will deny it. The citizenship of a given person is not a repeatable event and cannot, therefore, be accurately represented by a frequentist interpretation of probability.

## 2.5 Environmental and spatial context

The previous two sections looked at the concepts of interoperability and uncertainty. Key technologies in XML and Web services were examined that are striving towards interoperable, distributed systems through the adoption of SOA. The method of quantifying uncertainty through the use of probability theory and the concept of random variables was also examined. This section looks at how uncertainty representation and interoperability have been implemented in the geospatial domain. Current efforts for fusing the two fields are identified and a discussion about how further work in this area would be beneficial is given.

### 2.5.1 Uncertainty in the geospatial domain

Uncertainty in geographic information arises due to the fact that almost all representations of the world are incomplete; they are approximations. The importance of a GIS framework capable of handling uncertainty in data inputs, in decision rules and in the geometries and attributes modelled has been highlighted by authors from Burrough (1992) to Heuvelink et al. (2007). Uncertainties in data within a GIS framework arise from various sources including measurement error, observation operator error, processing/modelling errors, or corruption. These uncertainties can be further categorised into two distinct types: attribute uncertainties and positional uncertainties.

#### Attribute uncertainty

To understand attribute uncertainty, a brief explanation of how data is represented in a typical GIS must be given. In a vector-based system, spatial objects (or features), such as rivers, roads, cities or houses comprise one or more attributes, such as nitrate concentration, surface material, population size or number of rooms, respectively. Vector features also have boundaries that contain positional

information, which may also be uncertain (see the following section). Attribute values may be defined at one or many locations for which the feature is defined (e.g. nitrate concentration as it varies along a river) or as integral properties of the feature (e.g. average nitrate concentration along the river). In raster-based GIS, the attribute is recorded at regularly-spaced locations. Features must be extracted, if necessary, by post-processing of these arrays of pixels. These attributes may be uncertain — typically for the reasons outlined above. Characterising the uncertainty of a feature attribute depends on the value scale on which the attribute is measured. As discussed in Section 2.3.1 there are two main classes of scale:

- Attributes measured on a continuous scale.
- Attributes measured on a discrete scale. These can be further split into ordinal and categorical.

If the attribute is discrete (ordinal), a discrete probability distribution can be used to quantify the uncertainty. When the attribute is continuous a PDF should be used, as discussed in Section 2.3.1. For attributes that are distributed in space, a PDF is required for each location, as well as any correlations between them (Heuvelink and Brown, 2008). The 'true' probability distribution of an uncertain attribute could be described by one of an infinite number of PDFs. In practice, however, only a small number of well-understood probability distributions are considered, as an approximation of the true probability distribution. One such distribution is the Gaussian distribution, a benefit of which is that it is easily generalised to a multi-variate Gaussian distribution, which may be used to quantify jointly-distributed random variables.

#### Positional uncertainty

Characterisation of the positional uncertainty of a spatial feature relies on classifying its primitive parts (coordinates) and the forms of movement they support under uncertainty (Heuvelink and Brown, 2008). These classifications are listed below:

- objects that are single points;
- objects that comprise multiple points but whose geometry cannot change under uncertainty (rigid);
- objects that comprise multiple points and whose geometry can vary under uncertainty (deformable).

Firstly, consider the positional uncertainty of a single point feature. The effects of positional uncertainty on the feature always results in a translation of the feature's position. This translation occurs in as many directions as the feature has coordinate dimensions. Positional uncertainty of a rigid spatial feature involves a rotation about an origin as well as a simple translation. When considering a deformable spatial feature, the positional uncertainty of its individual points results in arbitrarily complex changes to the features position or shape. These complexities occur when the individual points are either partially or completely independent. Once the spatial feature has been classified, probability distributions can be assigned. Further information about positional uncertainty and how it may be quantified can be found in Chapter 6.

### 2.5.2 Geography Markup Language

In order to address the issue of interoperability within the domain of geospatial information, the OGC is developing and maintaining a series of geospatial information models and services. GML is one such standard that allows the expression of geographical features (OGC 07-036, 2007). Within GML a distinction is made between a feature and a geometry object; this is not necessarily true of all GIS systems. GML defines a feature as an application object that represents a physical entity, e.g. a river, road or house. A feature may contain geometric aspects, but this is not required. On the other hand, a geometry object defines a location or region and is thus different from a feature. GML conforms to the General Feature Model laid out in ISO/TC 211 19109 (2003) which states that a feature is composed of a series of properties, which themselves may be realised as features. A feature may have one, or more, geometric properties (e.g. a river may have a bounding box as well as a line representing its centre-line).

The current version of GML (3.X) contains a large set of primitive elements which may be used to construct a domain-specific application schema. Included within these elements are the following:

**Feature** This is the general feature model discussed above. In the majority of application schemas, the abstract feature type will form the base.

**Geometry** GML provides a suite of elements for describing geometries. These range from simple point geometries through to polygons and rectified grids for encoding raster layers. The geometry schema is one of the more popular aspects of GML and has been adopted in several other standards, as discussed later.

**Coordinate Reference System** When describing spatial data the coordinate system of that data is often required. GML provides a mechanism for describing reference systems.

**Coverage** Coverages are functions that map a domain (e.g. a collection of points in space or time) to a range, or collection of values.

**Observations** GML provides a schema for describing observations. However, O&M (another OGC specification) also provides this functionality. The duplication of standard specifications hinders interoperability. Therefore, it is anticipated that the GML observation schema will be deprecated in the future.

While this is not an exhaustive list of the GML primitive elements (the full suite includes concepts such as topologies, and units of measure and time), it should provide sufficient awareness of the broad spectrum that GML aims to cover. In fact, the sheer breadth of the GML specifications may be too complex to implement completely, or even sufficiently, as many software applications remain conformant to version 2.0 of GML only (which is far simpler). The OGC has recognised this fact and acted by releasing a series of profiles which are a series of rules restricting the GML specification to a simpler dialect (OGC 06-049r1, 2006; OGC 05-099r2, 2005; OGC 05-096r1, 2005; OGC 05-094r1, 2005; OGC 05-095r1, 2005). Concentrating on one or two key aspects as required these profiles allow large sections of the specification to be ignored.

#### GML geometry schema

The GML geometry schema provides a collection of 0, 1, 2 and 3 dimensional geometry primitives. These include, but are not limited to:

**Point** Represents a simple point in space.

**LineString** Represents a special curve consisting of a single segment with linear interpolation, defined by two or more coordinate tuples.

**Polygon** A geospatial polygon defined by an interior and exterior ring.

**MultiPoint** A collection of points.

**MultiPolygon** A collection of polygons.

Each GML geometry inherits from a super-type, `AbstractGeometryType`. Consequently, each geometry inherits an attribute group for associating it with a Spatial Reference System (SRS).

When dealing with collections of geometry types (MultiPoint, MultiPolygon etc) all constituent members are assumed to have the same SRS as the containing type, unless stated otherwise. Adding to these base attributes, each geometry type is typically constructed with a set of coordinate tuples, organised into various feature properties (Listing 2.9). The expressive capabilities and inherent simplicity of the GML geometry types has seen the schema adopted by several other standards including O&M, Sensor Model Language (SensorML), Filter Encoding Specification and the Web Processing Service (WPS) specification.

```
<gml:Point gml:id="1312" srsName="urn:ogc:def:crs:EPSG:6.6:4326">
  <gml:pos>52.01 0.56</gml:pos>
</gml:Point>
```

Listing 2.9: A GML fragment showing how a spatial point may be encoded.

The importance of quantifying positional uncertainties, discussed in Section 2.5.1, is outlined by Burrough (1992). It is surprising therefore that no means of describing positional uncertainty exists within GML. This ommission from GML motivates the work found in Chapter 6.

### 2.5.3 Sensor Web Enablement

The Sensor Web Enablement (SWE) is an initiative run by the OGC that is looking at providing a framework for exploiting Web-connected sensors (Botts et al., 2008; Percivall and Reed, 2006). The framework, comprising of a series of open standards and services, aims to convert existing sensor networks into a "sensor web". Botts et al. (2008) compare the technologies of the Sensor Web to those of the standard Web, or Internet, claiming to have a similar impact to HTML and HTTP. The OGC has outlined a list of key functionalities a sensor web should have, some of which are listed below.

- Discovery of sensor systems.
- Determination of sensors' functionality and quality.
- Retrieval of real-time or time-series observations.
- Tasking of sensors, where the sensor is capable.
- Subscription to and publishing of alerts.

Meeting these criteria lays the foundation for the OGCs vision of "plug and play" Web-based sensor networks. Utilising a plug and play foundation would allow new Sensor Web-enabled

sensors to be "plugged in" and automatically start publishing their observations, as well as be discovered by interested users. At the heart of SWE are two XML encoding specifications: O&M and SensorML. O&M is capable of describing and exchanging observations, typically produced by some sensor, which may be described using SensorML (OGC 07-000, 2007). More information on the O&M specification can be found in Section 2.5.4. The importance of location information for a Web-connected sensor is highlighted numerous times by Botts et al. (2008) where they state: "sensor location is usually a critical parameter for sensors on the Web" and "for both fixed and mobile sensors, sensor location is often a *vital* sensor parameter". Despite these statements there appears to be no definitive solution for encoding sensor location information within the Sensor Web. However, as the O&M specification states, the location of an observation may not be trivially available. Take, for example, remote sensing applications where the location of the sensor is different to that of the actual feature being observed. For this reason there is no explicit property of an observation for encoding a geospatial location, rather a recommendation is made that it should appear within the `featureOfInterest` property. The specification states that the `featureOfInterest` property can be any feature that inherits the GML `AbstractFeatureType`, which adds further confusion because, as previously stated, a GML feature may contain one or more geometric properties, or even none. If a feature contains multiple geometric properties there is no mechanism within O&M to explicitly specify which geometry should be taken as the observation location. With such emphasis put on the importance of encoding the location of sensors, perhaps an explicitly defined location property would have been beneficial.

A series of XML Web services have been outlined within SWE to enable discovery, retrieval and notification of sensor data as well as tasking capable sensors to provide tailored observations. The SOS provides an interface for retrieving and managing sensors and their observations (OGC 06-009r6, 2007). The SOS specification uses O&M for the encoding of observation data and either SensorML or Transducer Markup Language (TML) (OGC 06-010r6, 2006), another SWE standard not covered here, for describing sensor models. A detailed review of the SOS specification can be found in Chapter 4. Tasking of sensors is achieved through the use of the Sensor Planning Service (SPS) which defines a set of operations for the collection of information from capable devices (sensors) of varying specifications (OGC 07-014r3, 2007). Subscription to sensor alerts is handled through a combination of standards: the Sensor Alert Service (SAS) (OGC 06-028r3, 2006) and the Web Notification Service (WNS) (OGC 06-095, 2006). The SAS provides publishing and subscription to sensor notifications and uses the WNS for asynchronous delivery of said

notifications.

The Sensor Web is a collection of emerging technologies that has seen early adoption from a host of different domains. Terhorst et al. (2006) present the evolution of a satellite-based wild fire detection system into a Sensor Web application. Achieving this relies on an extension to the current SWE technologies to provide an open, service oriented, multi-agent platform called the Sensor Web Agent Platform (SWAP) (Moodley and Simonis, 2006). The need for such a framework was partly driven by the distinct lack of ontological structure within the SWE framework. van Zyl et al. (2008) also acknowledge this limitation, claiming it to be a big challenge, but outline efforts by the Global Earth Observation System of Systems (GEOSS) project to compile a collection of commonly used terms and expressions across the various communities. OntoSensor, developed by Russomanno et al. (2005), is another project addressing the ontological requirements of the Sensor Web.

A further limitation that is not discussed in the literature is the lack of standardisation pertaining to quality information. One of the key functionalities highlighted by Botts et al. (2008) is "determination of sensors' functionality and quality", yet information regarding sensor observation quality is opaque. The SensorML standard specifies that "the role of the SensorML is to provide characteristics required for processing, georegistering, and assessing the quality of measurements from sensor systems". The quantification of quality is delegated to the SWE Common schema.

#### Sensor Web Enablement Common schema

The SWE Common model defines basic types and data encodings for elements that are common amongst all SWE encodings and services. While the SWE Common elements belong to the SWE Common namespace, they do not form a standalone specification as yet. Formal information about SWE Common can be found in OGC 07-000 (2007) and OGC 07-022r1 (2007) where the encoding currently resides. SWE Common is responsible for defining a set of data types and related components that fall into the following categories (OGC 07-000, 2007):

1. primitive data types, complementing those implemented in GML (Section 2.5.2)
2. general purpose aggregate data types, including records, arrays, vectors and matrices
3. aggregate data types with specialised semantics, including position, curve, and time-aggregates

4. standard encodings to add semantics, quality indication and constraints to both primitive and aggregate types

5. specialised components to support semantic definitions, as required above

6. a notation for the description of XML and non-XML array encodings.

#### Simple data types

Fundamentally, SWE Common is a conceptual model for representing primitive data types and aggregations of these data types. As shown in Figure 2.9 these simple data types can be limited to `Quantity`, `Count`, `Boolean`, `Category` and `Time`.


Figure 2.9: Simple data types in SWE Common from the SensorML specification (OGC 07-000, 2007).

Each simple type inherits a set of standard properties from the `AbstractDataComponent` class, including simple `name` and `description` properties. A more informative property, `definition`, exists to provide an optional link to semantics. The intended use of the `definition` property is to identify the phenomenon association or other context of the value (OGC 07-000,

2007). For example, the `definition` may indicate that the value (of a simple SWE Common data type) represents an atmospheric pressure. The value of the `definition` property should be realised as a URI and should point to a community-accepted dictionary or registry. An important concept of the SWE Common simple data types is that the value property is optional. Consequently, when a simple data type is used without value it serves as a descriptor for some data structure. The key data types and their intended use are detailed below.

**Boolean** A simple true/false value of a specified property.

**Category** Textual data that is a member of a particular dictionary.

**Quantity** A continuous value represented by a floating point number.

**Count** A discrete value represented by an integer.

Each of these data types may have a `Quality` component, taken from the quality union (Figure 2.9). This restricted and rather simplified model allows data quality to be encoded as either a `Category`, `Quantity`, `QuantityRange` or `Text` type. The SWE Common specification states that a quality measure may be expressed as "values of precision, accuracy, tolerance and confidence" and that the type of quality measure should be stated "explicitly in the definition attribute".

```
<Quantity definition="urn:ogc:def:property:OGC:scanAngle">
    <uom xlink:href="urn:ogc:unit:degree"/>
    <quality>
        <QuantityRange definition="urn:ogc:def:property:OGC:tolerance2std">
            <value>-0.02 0.02</value>
        </QuantityRange>
    </quality>
    <value>25.3</value>
</Quantity>
```

Listing 2.10: A SWE Common Quantity type with an associated quality property, given as a tolerance to 2 standard deviations, from the SensorML specification (OGC 07-000, 2007).

Listing 2.10 is an example taken from the SWE Common specification showing how a `quality` property can be assigned to a `Quantity` type. From this example it can be said, with 95% certainty, that the actual value will be between 25.28 and 25.32, assuming a Gaussian distribution. As discussed in Section 2.3, this is an adequate method of quantifying one's uncertainty; however, there are limitations to SWE Common's capabilities. For example, it is better to explicitly specify that the random variable is Gaussian distributed, rather than forcing users to make an assumption. This is more relevant when the random variable is non-Gaussian distributed. Another method of

describing uncertainty, which was discussed in Section 2.3, was through the use of realisations. With the current SWE Common standard this is simply not possible. These shortcomings leave the SWE lacking the tools for fully quantifying the complex uncertainties found in all sensor observations and one could even argue that it falls short of fulfilling the functionality requirements outlined by Botts et al. (2008).

#### Aggregate data types


Figure 2.10: Aggregate data types in SWE Common, from the SensorML specification (OGC 07-000, 2007).

Aggregate types within SWE Common inherit from the same `AbstractDataComponent` as the simple types and thus posses the same properties (`name`, `description`, `definition` and `fixed`). However, the `definition` property now provides semantics for the aggregated data, not the individual components. Figure 2.10 outlines the key aggregate data types, which follow conventions set in ISO/IEC 11404 (1996), two of which (`DataRecord` and `DataArray`) are discussed below.

**DataRecord** The `DataRecord` is a `RecordType` that defines a logical collection of simple data values. Each constituent part of a `DataRecord` exists within a `field` property which may contain any SWE data type. It should be noted that the `field` property may also contain aggregate data types so `DataRecord`s can include other `DataRecord`s or `DataArray`s. An example of a `DataRecord` can be seen in Listing 2.11.

**DataArray** The `DataArray` is an `ArrayType` that defines a collection of values of a given data type, defined by the `elementType` property. As with `DataRecord`s, the `elementType` property may contain aggregate data types allowing arrays of arrays and arrays of records. A separate `elementCount` property exists containing the number of elements within the array. The SWE Common array types utilise a flexible encoding mechanism that allows data to

be encoded in XML, or in more efficient formats such as a text block, base64 or binary. Section 2.5.3 discusses the details of this encoding specification in greater detail. Listings 2.12 – 2.13 give examples of the SWE Common `DataArray` with a variety of encodings..

```
<DataRecord definition="urn:ogc:def:property:OGC:WeatherConditions">
  <field name="Temperature">
    <Quantity definition="urn:ogc:def:property:OGC:AirTemperature">
      <uom code="Cel" />
      <value>21.4</value>
    </Quantity>
  </field>
  <field name="Pressure">
    <Quantity definition="urn:ogc:def:property:OGC:AtmosphericPressure">
      <uom code="hPa" />
      <value>1014.0</value>
    </Quantity>
  </field>
</DataRecord>
```

Listing 2.11: A SWE Common DataRecord describing the weather with temperature and pressure measurements.

#### SWE Common encoding

Arrays in SWE Common may be encoded using a variety of methods. Providing binary encodings as alternatives to XML and American Standard Code for Information Interchange (ASCII) allows large datasets to be transmitted efficiently. An overview of the key encoding types is given below.

**TextBlock** The `TextBlock` encoding is an ASCII string of values separated by a set of specific characters. The order of the values is specified in the `dataComponents` section (the `elementType` property of a `DataArray`). There are three separator tokens in a `TextBlock`. The `tokenSeparator` is used to separate the individual components of a data structure, i.e. if the `DataArray`'s `elementType` property is not a scalar value but a `DataRecord`. The `blockSeparator` separates each data structure, be it a single scalar value or multiple values within a `DataRecord`. The `decimalSeparator` specifies the character that separates the integer and fractional part of a decimal number. Listing 2.12 gives an example of a `TextBlock` encoding.

**BinaryBlock** The `BinaryBlock` encoding is similar to the `TextBlock`, such that it defines a stream of values whose order is determined by the order of values in the `dataComponents` section. However, the payload of the data is encoded in binary. The `byteEncoding` specifies

```
<DataArray>
   <elementCount>
      <Count>
         <value>4</value>
      </Count>
   </elementCount>
   <elementType name="WeatherConditions">
      <DataRecord>
         <field name="Temperature">
            <Quantity definition="urn:ogc:def:property:OGC:AirTemperature">
               <uom code="Cel"/>
            </Quantity>
         </field>
         <field name="Pressure">
            <Quantity definition="urn:ogc:def:property:OGC:AtmosphericPressure">
               <uom code="hPa"/>
            </Quantity>
         </field>
      </DataRecord>
   </elementType>
   <encoding>
      <TextBlock decimalSeparator="." blockSeparator="," tokenSeparator=" "/>
   </encoding>
   <values>
      21.4 1014.0,
      19.8 1005.0,
      20.1 1008.0,
      21.3 1009.0
   </values>
</DataArray>
```

Listing 2.12: A SWE Common `DataArray` describing the weather with temperature and pressure measurements. A `TextBlock` encoding is used.

the particular byte encoding method, either raw, base64 or base16 (hex). The `byteOrder` attribute specifies whether multiple byte data types are bigEndian or littleEndian. Further information about the encoding of the values can be found in a list of either `Component` or `Block` properties. Component properties allow each data type to be attributed a `dataType` (i.e., byte, integer, float etc.), a number of significant bits, a number of padding bits before and after the value and the encryption method used, if any. The `Block` property is similar to the `Component` property in that it allows specification of padding bytes and encryption. However, it relates to a block of data rather than a scalar value so replaces the `dataType` and `significantBits` properties with `byteLength` and `compression` (e.g. JPEG compression) properties. For both `Component` and `Block` classes the `ref` attributes refer back to the data components defined earlier in the structure. An example of the `BinaryBlock` can be seen in Listing 2.13.

**StandardFormat** The `StandardFormat` encoding allows data to be encoded in a MIME format, e.g. JPEG or GIF. The `mimeType` property defines the format that is being referred to. The MIME type should apply to the whole data structure.

### 2.5.4 Observations & Measurements

Produced as part of the OGC SWE activity, O&M is a conceptual model, realised as an XML application schema, for describing observations. The O&M specification (OGC 07-022r1, 2007) defines an observation as "an act associated with a discrete time instant or period through which a number, term or other symbol is assigned to a phenomenon". The distinction between an observation and a measurement is made according to Fowler (1996) who states that an observation may have any result (i.e. a category, coverage etc.) whereas a measurement may only have a numeric quantity. The conceptual model, built around the General Feature Model defined in ISO/TC 211 19101 (2001) and ISO/TC 211 19109 (2003), is shown in Figure 2.11.

An observation is modelled as a Feature (ISO/TC 211 19101, 2001) with several properties, discussed below.

#### featureOfInterest

The `featureOfInterest` property is the feature of which the observation was made. Conceptually this is realised as any Feature type, according to ISO/TC 211 19101 (2001). O&M is a generic model for describing observations and as such does not provide an application schema for features

```
<DataArray>
   <elementCount>
      <Count>
         <value>4</value>
      </Count>
   </elementCount>
   <elementType name="WeatherConditions">
      <DataRecord>
         <field name="Temperature">
            <Quantity definition="urn:ogc:def:property:OGC:AirTemperature">
               <uom code="Cel"/>
            </Quantity>
         </field>
         <field name="Pressure">
            <Quantity definition="urn:ogc:def:property:OGC:AtmosphericPressure">
               <uom code="hPa"/>
            </Quantity>
         </field>
      </DataRecord>
   </elementType>
   <encoding>
      <BinaryBlock byteEncoding="base64" byteOrder="bigEndian">
         <member>
            <Component ref="WeatherConditions/Temperature" dataType="
urn:ogc:def:dataType:float"/>
         </member>
         <member>
            <Component ref="WeatherConditions/Pressure" dataType="
urn:ogc:def:dataType:float"/>
         </member>
      </BinaryBlock>
   </encoding>
   <values>
      MjEuNCAxMDE0LjAgMTkuOCAxMDA1LjAgMjAuMSAxMDA4LjAgMjEuMyAxMDA5LjA
   </values>
</DataArray>
```

Listing 2.13: A SWE Common `DataArray` describing the weather with temperature and pressure measurements. A `BinaryBlock` encoding is used.


Figure 2.11: Conceptual model of O&M, from the O&M specification (OGC 07-022r1, 2007).

of interest. Typically, such features are defined in either a domain-specific application schema or a cross-domain schema such as the sampling features specification (OGC 07-002r3, 2007). O&M stipulates that the feature types defined in the domain-specific application schema must conform to ISO/TC 211 19109 (2003). As such, the feature type must contain the observed property (see below) as part of its definition.

Frequently, observations are associated with geospatial locations. OGC 07-022r1 (2007) explains that while it is an important property, the location is not always trivially available. Consequently there is no pre-defined location property of an observation within O&M. The location of interest for an observation is often associated with the feature of interest and as such the encoding of geospatial locations falls within the governance of the domain-specific feature type.

#### observedProperty

The `observedProperty` describes some phenomenon for which the `result` of the observation is an estimate. There is an inherent association between the observed property and feature of interest such that the observed property must exist as a property of the feature of interest. Moreover the units of measure of the observation result must map to those of the observed property.

#### procedure

The `procedure` of an observation is the process used to obtain the result. A common example is an instrument or sensor, however, it could also be a human observer or a computer simulation. The encoding of the `procedure` is out of the scope of OGC 07-022r1 (2007), however, SensorML (OGC 07-000, 2007), another outcome of the SWE activity, is capable of describing sensor processes and may be used within the `procedure` property.

#### result

The `result` property of an observation is the value generated by the procedure and is represented as an XML `anyType` element with a single constraint: it must be suitable for the observed property. There are numerous types of observation result, two of which are included below.

**scalar** When the property of the feature of interest is single-valued the result of an observation will be a scalar value.

**coverage** When the property of the feature of interest depends on some parameter (e.g. space or time) then the result of the observation is a function, or, coverage (ISO/TC 211 19123, 2004).

#### resultQuality

The `resultQuality` property allows "event-specific" quality information to be associated with an observation. Within O&M, result quality is considered metadata and as such is modelled on the `DQ_Element` type from ISO/FDIS 19115 (2003), which will be discussed in Section 3.4.1. However, the XML schema allows any XML type to be used within this `resultQuality` property. This generality underlines the fact that there is currently no standardised tool for quantifying uncertainties quantitatively and in an interoperable fashion. While it is possible to use the SWE Common data model to express uncertainties, this method lacks the capability to express random variables through their probability distributions, as discussed in Section 2.5.3.

## 2.6 Conclusions

This chapter reviewed the ideas and technologies behind two distinct domains: interoperability and uncertainty representation. In terms of interoperability the Extensible Markup Language (XML)

was examined, identifying how a generic language allows for a multitude of domain-specific vocabularies to be constructed. Using the XML Schema language it was shown how a set of rules, governing the structure, can aid in creating these vocabularies, consequently allowing multiple computer systems to interoperate. The differences between a weak and strong-typed schema design were discussed, and the strengths and weaknesses of both approaches were detailed. It was concluded that while a strong-typed design was preferential for interoperability it was not always possible and depended upon the size of the domain to be modelled. An amalgamation of the two design patterns was suggested as the best solution.

The discussion progressed onto Web services and the way in which a set of core technologies laid the foundations for interoperable distributed systems, examining three distinct approaches to Web services including: SOAP and WSDL; RESTful; and OGC Web services. SOAP and WSDL services are suited to systems where complex processing of data needs to occur, whereas a RESTful approach is simpler when only retrieval, updating and deleting of resources is required. Many tools exist for constructing and consuming both SOAP and RESTful-based services, meaning that adoption of these methodologies has been popular. The OWS specification, on the other hand, has not seen such mainstream adoption. This lack of popularity may be partly attributed to the lack of tools that support the capabilities document standard, thus making it hard to automatically create marshalling classes. With the maturity of the SOAP and WSDL standards always improving, and with the suite of other WS-* specifications on offer, perhaps now would be an ideal time for the OGC to consider migrating their services to a SOAP based system. The possibility of a RESTful architecture for a typical OGC service was excluded as they primarily deal with complex geospatial data processing, which may be ill-suited to a RESTful approach.

The discussion on uncertainty outlined the idea of random variables and how they can be described using a series of statistics, realisations or by a probability distribution. When dealing with random variables it is important to be able to use any of the aforementioned methods to describe uncertainty, as in many instances a full probability distribution is not known. However, when sufficient information is available it is beneficial to be able to explicitly quantify the probability density function of a particular random variable, as this provides a complete picture of its uncertainty.

Section 2.5 looked at interoperability in the field of geosciences. The OGC has made great strides by developing a host of data and service specifications. However, the size of some of the specifications (e.g. GML) and the generic nature of others (e.g. O&M) mean that the OGC is facing the same problems that the SOAP and WSDL communities faced previously. These

problems, discussed in Section 2.2.2 mean that interoperability is suffering due to the overly-complex specifications. This issue led to the foundation of the Web Services Interoperability Organization (WS-I), which produced a series of profiles that further restrict the use of the WS-* specifications. Specifically, they state which versions of which standards should be used as well as a series of other rules that should be adhered to in order to conform to a specific profile. With the current popularity of interoperability within the geospatial domain, the OGC should create a subsidiary organisation and mimic the successes achieved by the WS-I. This could be accomplished by creating documents detailing specific versions of OGC services and standards that should be used, and with further restrictions put on these standards. Existing work includes the publication of several GML profiles, however, this is likely to be only the beginning of a substantial reorganisation.

Considering the prevalence of uncertainty in geospatial data, and the benefits of providing interoperable standards, one might imagine that a standard mechanism for quantifying uncertainty, interoperably, would exist. The efforts made by SWE Common and O&M to provide for quality measures were discussed, however, they lacked the expressiveness required to fully and flexibly quantify a random variable through its probability density function. The importance of such quantification, coupled with the limited power of existing standards, motivated the main work of this thesis, the Uncertainty Markup Language (UncertML).

# 3 UncertML

CONTENTS

## 3.1 Foreword

A conclusion from Chapter 2 was that, while interoperability and uncertainty both play an important role within geospatial informatics, the two have never been combined, i.e. no interoperable language for describing uncertainty exists. The work in this chapter seeks to solve this problem through the development of an interoperable model for quantifying probabilistic uncertainty, UncertML.

Section 3.2 provides detail of the underlying conceptual model employed within UncertML. A series of UML diagrams detail how, within UncertML, uncertainty can be described via a series of realisations (Section 3.2.2), summary statistics (Section 3.2.3) or probability distributions (Section 3.2.4). Section 3.3 provides examples of an implementation of the conceptual model via XML; detailing how UncertML can describe uncertainty via realisations, statistics and distributions in Sections 3.3.1– 3.3.3 respectively. Section 3.3.4 provides a series of examples of how UncertML can be used to describe the uncertainty types within the International Organization for Standardization (ISO) 19138 (ISO/TS 19138, 2006) standard. The role of existing ISO standards and how UncertML fits within them is discussed in Section 3.3.4. Finally, the chapter concludes in Section 3.5.

### 3.1.1 UML notation

The diagrams in Section 3.2 use the UML notation to illustrate the UncertML data model. However, as UML is not specifically designed to conceptualise XML schemas some unconventional notations have been adopted. Below are a set of rules that have been used throughout:

- The `Leaf` stereotype been used to represent individual XML schema files. The name of the `Leaf` is the file name for that schema.
- The `DataType` stereotype represents an XML element and corresponding complex type.
- All types and properties are represented as public. There is no concept of visibility in XML schemas.
- All XML attributes and properties are represented by UML attributes, no explicit differentiation is made. The supporting text and examples will state the differences where necessary.
- A multiplicity of `[0..1]` or `[0..*]` represents an optional attribute or property.

- If no multiplicity is specified it can be assumed that the attribute or property is mandatory and may exist only once, i.e. a multiplicity of `[1..1]`.
- Inheritance (or XML extension) is denoted by a solid line with a hollow triangle.
- Certain properties are denoted using the UML association notation (an arrow). Where this notation is used it can be assumed that the source element has a property with the name above the association, which is of the target type.

## 3.2 Conceptual model

The Uncertainty Markup Language (UncertML) is an XML encoding for the transport and storage of information about uncertain quantities, with emphasis on quantitative representations based on probability theory. This chapter describes the XML schema syntax and conventions that allow an interoperable description of uncertain data (i.e., random quantities) in a variety of ways including:

- probability distributions including both uni- and multi-variate distributions and mixture models;
- statistics, including means, (co-)variances, standard deviations and quantiles;
- realisations or sampled data.

These three categories, illustrated in Figure 3.1, cover the full range of representations which one might commonly use for random quantities. In the diagram, a package is considered as a distinct XML schema file. A dependency between packages illustrates that an XML schema import has been used. The intricate relationships between the members of the different packages is not demonstrated in Figure 3.1, but is explained in the subsequent sections. However, a common assumption can be made that most elements extend the `AbstractUncertainty` type in the base type package.

The most precise description of a random quantity is in terms of its probability distribution, which is appropriate where the distributional form of that random quantity is known. Where a strong parametric form for the distribution is not appropriate, a more flexible semi-parametric mixture model may be used. A weaker, but often more realistic, option which is still useful and is widely employed, is to represent a random quantity in terms of its statistics. Within UncertML this is achieved with a range of `statistics` types, ranging from moments (e.g. mean and variance) to

histogram and quantile based representations. Finally UncertML allows for fully non-parametric representations in terms of samples / realisations from the given distribution, such as might arise from a Bayesian Markov chain Monte Carlo analysis.

The ISO/IEC guide to the expression of uncertainty in measurement (GUM) (ISO/IEC Guide 98:1995, 1995) outlines the importance of quantifying uncertainty in observations by stating that it is "obligatory that some quantitative indication of the quality of the result be given so that those who use it can assess its reliability". The guide goes on to state that it is necessary to have a readily-implemented and generally-accepted procedure for characterizing the quality of a result of a measurement; however, it does not outline a mechanism for describing this information via an exchangeable medium. The GUM guide aspires to provide a worldwide consensus on the evaluation and expression of uncertainty in measurement, not dissimilar to the International System of Units.

Section 2.5 discussed the recent developments for sensor observation modelling within the OGC (e.g. the O&M standard). These developments have opened opportunities for interoperable, sensor-derived datasets to be exchanged over the Internet. As this Sensor Web community grows, an increasing volume of data becomes available which requires processing; much of this data will be used for decision support. However, rational decision-making using incomplete knowledge (i.e. sensor measurements) is only possible if the inherent uncertainties in those measurements, and the uncertainties which are introduced or increased by subsequent processing, are quantified. To be truly valuable in the context of discoverable Web services and datasets (for example, within automatic online risk management chains), this uncertainty must be represented in an interoperable manner. Chapter 2 concluded that no formal method of quantifying complex uncertainties (e.g. probabilistic representations) within the Sensor Web framework exists.

This section describes a conceptual model for uncertainty capable of describing all aspects of probability theory discussed in Section 2.3.1. The contents of this section form the basis for UncertML, with Section 3.3 providing an XML implementation of the conceptual model, detailed here. The conceptual model is realised as a series of UML diagrams, split into three packages: realisations (Section 3.2.2), statistics (Section 3.2.3) and probability distributions (Section 3.2.4).

All types are designed to allow encoding of both uni- and multi-variate uncertainty, and can thus be used for specification of marginal and joint distributions. Both continuous and discrete random quantities are catered for.

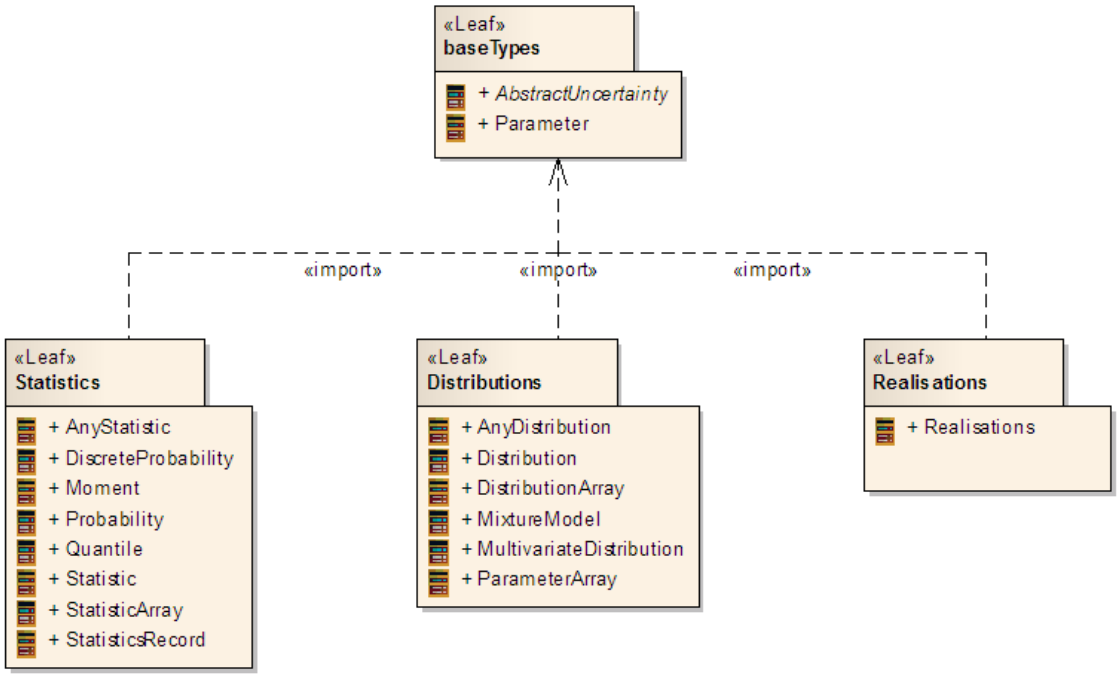



Figure 3.1: An overview of the UncertML package dependencies.

There is a clear separation of concerns in the design of UncertML, in that it is not designed to address issues covered in other schemas. For example, there is no notion of units of measure in UncertML — the intention is that UncertML is used with other schemas, and essentially replaces primitive value types in scalar and vector form. In subsequent chapters of this thesis, GML, SWE Common and O&M are used to illustrate the application of UncertML, but its design means that it could as easily be used to describe uncertainty in datasets from other fields of research; for example, genetics, economics or linguistics.

UncertML is, at present, restricted to probabilistic representations of uncertainty in random quantities, and does not address concepts such as fuzzy sets, random processes or belief functions.

### 3.2.1 Base types

Certain types are common to all three UncertML packages. These are referred to as base types. There are two base types within UncertML — the `AbstractUncertainty` type and the `Parameter` type.

#### AbstractUncertainty

The `AbstractUncertainty` type provides a root for the conceptual model of uncertainty in UncertML — all common uncertainty types extend this base type and inherit any common properties.

These common types include: `Statistic` and all it's child elements (`Quantile`, `Moment`, `DiscreteProbability` and `Probability`), `Distribution` and `MultivariateDistribution`. The decision to provide a root type, common to all uncertainty types, was influenced by the abstract types present throughout the GML schema.

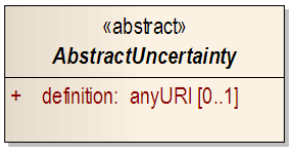



Figure 3.2: AbstractUncertainty type is the root type of the UncertML model of uncertainty. All other uncertainty types inherit from this.

Figure 3.2 illustrates the `AbstractUncertainty` type with a single property, `definition`. The `definition` property provides a link, via the use of URIs, from any uncertainty type (for example, `Distribution`) to a definition describing the uncertainty type of interest (for example, 'Gaussian').

Section 2.2.1 outlined two common design methodologies for XML schemas: strong and weak typed. It concluded that, while both methodologies have strengths and weaknesses, an amalgamation of the two would provide the best solution. UncertML follows a 'weak-typed' design pattern which promotes the use of generically named elements with links to dictionary definitions to provide semantics. However, rather than having a pure weak-typed solution (e.g., with a single `Statistic` element for all statistics), UncertML provides some hard-typed elements (e.g., `Quantile`). This approach is far more flexible than a traditional strong-typed design. However, crucial to the use of a weak-typed design is the implementation of a dictionary describing the concepts referenced from within the schema. A dictionary is currently under development for UncertML which describes the most common statistics, distributions and other related concepts, using mathematical formulae where appropriate.

The design and implementation of this dictionary is fundamental to the successful adoption and use of UncertML. Based loosely around the structure of the data quality measures outlined in ISO/TS 19138 (2006), the definitions are encoded according to the GML dictionary schema specification. Included within the definition for each statistic, distribution and all other concepts are the following pieces of information:

- URI
- Name(s).

| URI | http://dictionary.uncertml.org/statistcs/mean |
|---|---|
| Name(s) | Mean, arithmetic mean, average, expectation |
| Definition | The arithmetic mean (typically just the mean) is what is commonly called the average. It is defined as $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$ where $x_i$ represents with $i$'th observation of the quantity $x$ in the sample set of size $n$. It is related to the expected value of a random variable, $\mu = E[X]$ in that the population mean, $\mu$, which is the average of all quantities in the population and is typically not known, is replaced by its estimator, the sample mean $\bar{x}$. Note that !UncertML however does not deal with issues of sample size, rather the mean is taken to refer to the population mean. |

Table 3.1: An example excerpt from the UncertML dictionary defining the concept of the arithmetic mean.

- Definition and textual description.

The root of the dictionary is located at `http://dictionary.uncertml.org` and the URIs are influenced by the RESTful design pattern whereby they accurately describe the resource that they refer to. All statistics are located at '/statistics' with the name of the statistic following — e.g. the definition of the statistic 'mean' would be located at the following URL: `http://dictionary.uncertml.org/statistics/mean`. If the specified statistic has any parameters, these can be found at: `http://dictionary.uncertml.org/statistics/<statisticname>/parameters`. Individual parameter information can be located at the following: `http://dictionary.uncertml.org/statistics/<statisticname>/parameters/<parametername>`. The same structure is applied to all parametric distributions, with the word 'distributions' substituted for 'statistics'.

An Extensible Stylesheet Language Transformations (XSLT) stylesheet exists to allow a human-readable view of the dictionary and allows easy navigation through the various statistics and distributions.

An example of a dictionary entry is given in Table 3.1. With a strict mathematical definition of each term, and a unique identifier (URI), two UncertML users are able to correspond securely in the knowledge that they are discussing the same statistical term.

### Parameter

The second base type in UncertML is the `Parameter`, this type is common to all parametric distributions and certain statistics.

Displayed in Figure 3.3, the `Parameter` type contains two properties: `definition` and `value`. The `definition` of a `Parameter` is identical to that of the `AbstractUncertaintyType` and references a dictionary definition of a particular statistic (or distribution) parameter. The `value` of a

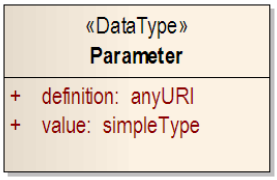



Figure 3.3: The UncertML Parameter type is common to distributions and certain statistics.

`Parameter` contains any simple XML type, typically an integer or double value.

### 3.2.2 Realisations

In some situations, a user may not be able to simply represent the uncertainties of the data they are working with. In such a scenario, a sample from the random variable might be provided, allowing uncertainty to be described implicitly. However, when using this approach, a sufficiently large sample is required to accurately deduce the uncertainties inherent in the data, which means that efficient encapsulation of large volumes of data was an important issue for UncertML. The following sections discuss the `Realisations` type available within UncertML, for describing a sample of data through a series of realisations.

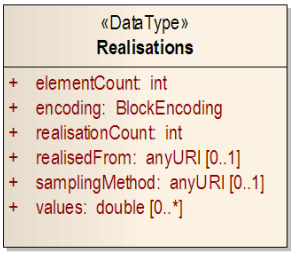



Figure 3.4: Structure of a `Realisations` type in UncertML.

Inheriting from the `AbstractUncertainty` type, discussed in Section 3.2.1, provides a `definition` property. In this instance the URI, if desired (note that the `definition` property is optional), resolves to a definition of the concept of a realisation. Two optional properties are included to provide more information about any particular set of realisations. The `realisedFrom` property is a URI that references a definition of the underlying distribution from which the realisations are generated. The second property, `samplingMethod`, is a URI referencing the definition of a particular method that was used to sample the realisations. The `realisationCount` contains the number of realisations in each sample; this information is useful when describing multiple random variables at multiple domain points. 'Domain point' here refers to a unique sampling location in a simulation series, which will often, but not always, be distinguished by its location in space and/or time.

As with all other array types in UncertML, the `Realisations` type is based around the SWE Common `DataArray` type. However, as the `Realisations` type can only describe a series of realisations, the `elementType` property of the `DataArray` is not used. The `elementCount` property is used to indicate the total number of values contained within the array. In cases where a dataset describes a single variable at a single domain point this value will be the same as the `realisationsCount` property. When describing multiple variables and/or or multiple domain points, the size of the `elementCount` will be the product of the number of variables, number of domain points and the number of individual realisations. More information about how to decode the information within the array may be found in Section 3.3.

The last two properties are directly inherited from the SWE Common encoding schema, which provides an efficient and flexible solution to encoding data arrays. Loosely speaking, the format of the data (binary, ASCII, XML etc.) is described in the `encoding` property and the `values` property contains the data which relates to the `elementType`; i.e. the actual values realised through sampling.

Aggregate types within UncertML, i.e. arrays or records, do not extend the `AbstractUncertainty` type, as they are merely perceived as 'containers' for uncertainty types with each individual constituent containing its own definition.

### 3.2.3 Statistics

This section discusses the range of options available in UncertML for describing summary statistics. Such statistics are used to provide a summary of a random variable, ranging from measures of centrality (mean, mode, median, etc.) through measures of dispersion (range, standard deviation, variance etc.) to higher order moments, such as skewness and kurtosis. While certain statistics (e.g. mean, mode) do not provide any information about uncertainty in isolation, they are often used in conjunction with other statistics (e.g. variance, standard deviation) to provide a concise summary, and there is considerable value in the explicit information that a given value represents the mean, rather than some other statistic such as the mode, or even a single realisation.

Figure 3.5 displays the entire collection of types available within UncertML for describing statistics. The `Statistic` type is the simplest method of describing a statistic. Inheriting from the `AbstractUncertainty` type provides a `definition` property which in this instance references a definition of the particular statistic, e.g. mean, variance, mode etc. Every `Statistic` type has a `value` property containing the actual value of the statistic, which is encoded as any simple

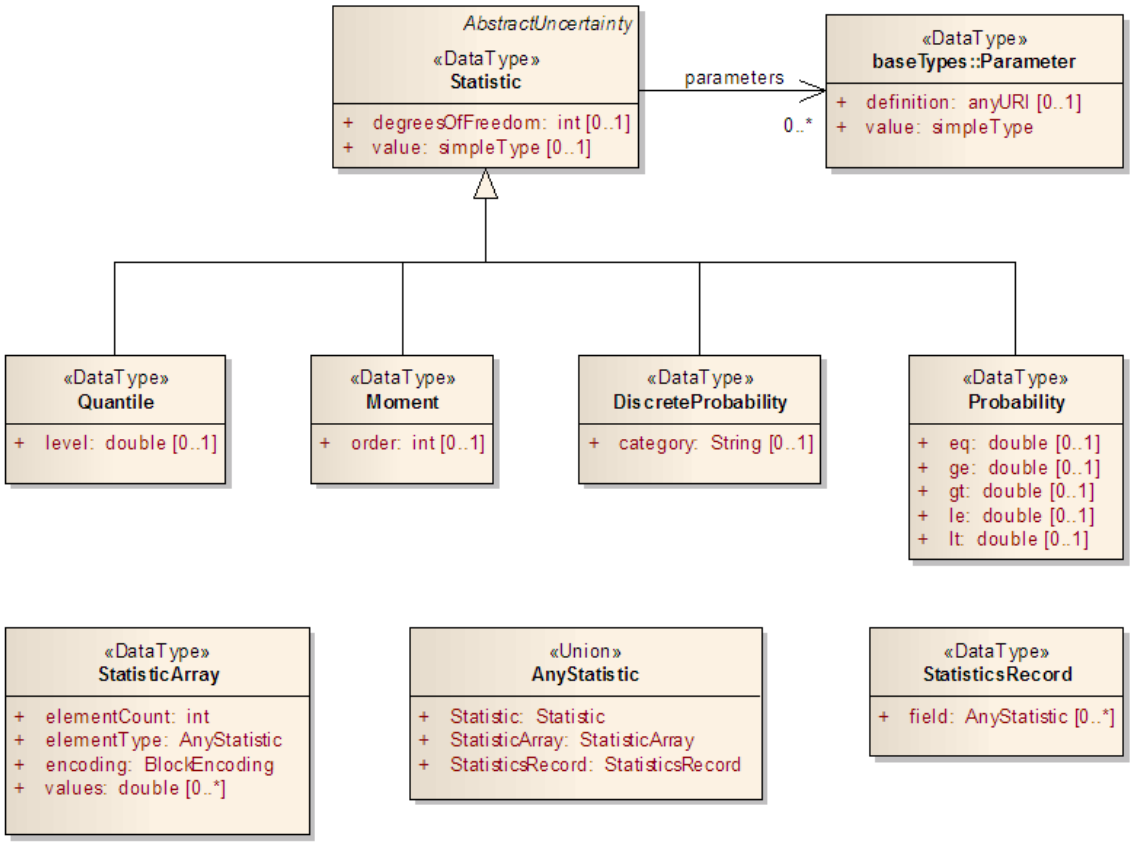



Figure 3.5: UncertML provides a suite of data types for describing various statistics including quantiles, probabilities and general statistics such as mean and variance.

XML type but would typically be an integer or double. When using statistics to describe a dataset it is often valuable to know the effective degrees of freedom, in order to assess the power and appropriate uses of the statistics. The base `Statistic` type in UncertML provides an optional attribute, `degreesOfFreedom`, which may be used for this purpose and is inherited by all subtypes (`Quantile`, `Moment`, `DiscreteProbability` and `Probability`). This generic and concise concept of a statistic allows most statistics to be encoded, but for certain statistics more information is required. In such cases the `Statistic` type contains one or more `Parameter` types (Section 3.2.1), each with a `definition` and `value`.

While the `Statistic` type is capable of describing all common statistics, certain statistics are used so frequently as to warrant their own concrete type. One example is a quantile. When working with quantiles, a user needs to know which quantile is being referred to. UncertML provides a specific `Quantile` type which inherits from the `Statistic` type and provides an additional property, `level`. Within UncertML, particular quantiles are specified by their level. For example, the 0.3 quantile (the value of the random variable below which a proportion of 0.3 of the probability mass, which always sums to 1.0, lies) has a `level` property of value 0.3.

Another frequently-used statistic is the probability value, used to frame questions such as: "What is the probability that the water level at a specific location will exceed 25m?", "What is the probability that the temperature will fall below 24° Celsius?", "What is the probability that this tree is an oak tree?". For such instances UncertML provides two further types, `DiscreteProbability` and `Probability`, which both inherit from the `Statistic` type. A further level of inheritance for probability types was decided against as the intermediate probability type would offer no extra properties. Furthermore, the use of XML schema choice elements negates the semantic linking between probability types that would have been provided by a second level of inheritance. `DiscreteProbability` is used to describe the probability that a variable falls within a certain enumerated class, as in the third example above. The addition of a `category` property provides the ability to specify any data type as a class but will typically be realised as a string; in the example above the category would be "Oak". A more explicit design might have allowed XML `anyType` as a category. However, this would place a requirement on UncertML users to know *a priori* what elements they are being sent within the `category` in order for meaningful processing. It was decided that to improve interoperability the `category` property should be limited to a simple textual representation.

`Probability` may be used to describe the probability that a variable exceeds (or does not exceed) a certain threshold. Such thresholds are defined through the use of the `gt` (greater than), `lt` (less than), `eq` (equal to), `ge` (greater than or equal to) and `le` (less than or equal to) properties, which may be used either individually or in combination. The `value` property of both the `DiscreteProbability` and `Probability` types always contains a value restricted to fall between 0.0 and 1.0, in contrast to the other statistic types, which contain a value in the relevant scale of the random quantity. Unfortunately, XML schema does not provide any mechanism for restricting which attributes can be used in combination, this can result in illogical encodings. For example, it is syntactically legal for a user to encode a `Probability` with both `gt` and `ge` attributes. Therefore, it is the responsibility of the user to identify which of the probability elements suits their particular use case and which attributes they should use.

The final statistic available within UncertML is the `Moment` type. A random quantity is often described by a collection of its (centered) moments. A `Moment` inherits from the `Statistic` type and adds an `order` property which contains the order of the moment as an integer; for example '3' for the $3^{rd}$ order centered moment often referred to as the skewness.

From herein, reference to 'statistics' means the generic `Statistic` type as well as all children

(`Quantile`, `DiscreteProbability`, `Probability` and `Moment`).

#### StatisticsRecord

Grouping of statistics provides a mechanism by which a random quantity can be summarised in terms of its centrality and dispersion or other attributes, UncertML provides the `StatisticsRecord` type for such use cases. As with all record types within UncertML, the `StatisticsRecord` is closely modelled on the SWE Common `DataRecord` type. However, the SWE Common models also include extra attributes and properties that are not required by UncertML so consequently the UncertML aggregate types are only loosely modelled on them and not a direct inheritance (OGC 07-000, 2007).

A `DataRecord` type in SWE Common, and therefore a `StatisticsRecord` type in UncertML, consists of a number of field properties. Each field of a `StatisticsRecord` may be any type that belongs to the `AnyStatistic` union: `Statistic`, `Quantile`, `Moment`, `DiscreteProbability`, `Probability`, `StatisticArray` or `StatisticsRecord`. A combination of general statistics in a `StatisticsRecord` provides a clearly-structured set of summary statistics for a given variable. The ability to construct complex structures such as records of arrays and records of records allows users to construct complex representations of uncertainty with relative ease.

#### StatisticArray

Arrays of statistics are useful when describing a variable at several domain points, or several variables at a given domain point. The `StatisticArray` type in UncertML, closely modelled on the `DataArray` of SWE Common, provides such a mechanism. The `elementType` property of a `StatisticArray` may be any type from within the `AnyStatistic` union, allowing multiple summaries to be encoded flexibly as arrays of single statistics, or an array of `StatisticsRecords` types. More complex structures, such as two dimensional arrays, are also possible. Exploiting the efficiency of the SWE Common encoding schema provides an efficient structure for encoding complex structures.

### 3.2.4 Distributions

When the uncertainties of a data set are better understood, it is desirable to describe them through the use of probability distributions. The types contained within this section of UncertML are specifically designed to allow a concise parametric or semi-parametric encapsulation of all proba-

bility distributions.

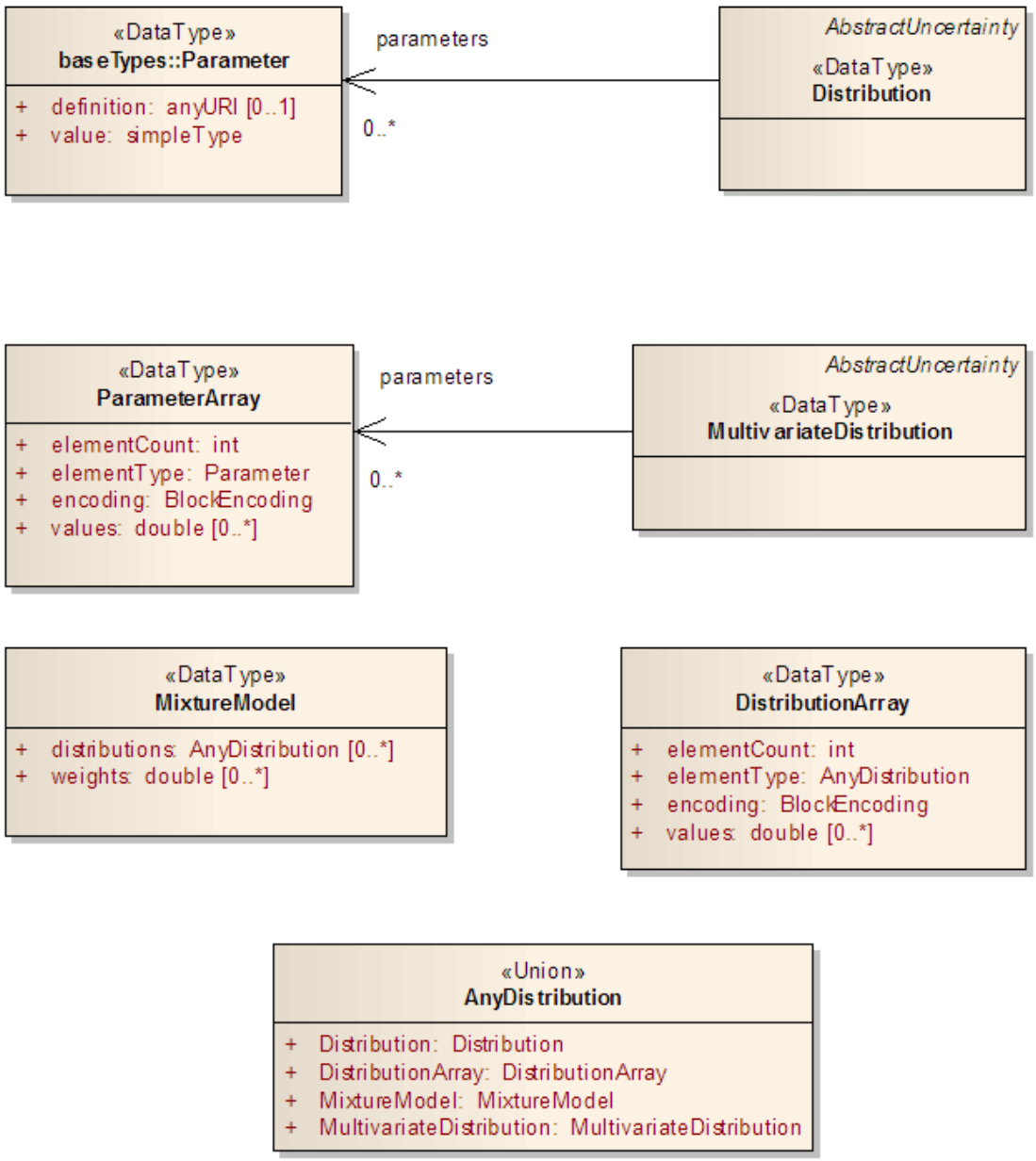



Figure 3.6: Class diagram of the UncertML distributions package.

### Distribution

For the simplest case (describing the probability distribution of a single variable), UncertML provides the `Distribution` type. Like all uncertainty types in UncertML, the `Distribution` type inherits from the `AbstractUncertainty`, inheriting the `definition` property. In the case of distributions, this definition may contain both a textual description, and a complex mathematical description of the distribution (for example cumulative distribution function and probability density function). It is important to note that the `Distribution` type is not a mechanism for

completely describing a probability distribution in terms of its functions, parameters and how they relate to each other; it should be thought of as a mechanism for describing an instance of a distribution which is defined elsewhere. As previously stated, a dictionary containing definitions of common distributions is currently under development.

Complementing the `definition` property is a `parameters` property which contains a number of `Parameter` types. Each `Parameter` of a distribution is not considered to be an uncertainty type in itself, but it contains a `definition` property which can be used to specify this particular parameter. Each `Parameter` also has a `value` property holding the actual value of that parameter.

#### DistributionArray

When describing the marginal distributions of several variables at a given domain point, or a univariate, marginal distribution over several domain points, the need for an array emerges. The `DistributionArray` type provided by UncertML is similar to the `StatisticArray`. However, in this instance the `elementType` property is realised as any type from the `AnyDistribution` union. Typically, this will be a simple `Distribution` type, however, the mechanism for describing arrays of arrays, and arrays of `MultivariateDistribution`s, is also available. The rest of the properties remain the same as in the `StatisticArray` (Section 3.2.3), with one subtle difference. Distributions often have numerous parameters that help describe them (e.g. a Gaussian distribution has both a location (centrality) and a scale (variance) parameter). In this instance the `Distribution` contained within the `elementType` property acts as a form of 'record'. Therefore, when interpreting distributions which have been encoded within the `values` property, care should be taken to clearly understand which set of values refers to which parameter. The examples of `DistributionArray`, in Section 3.3 demonstrate how one may encode such a scenario using a logical ordering of values to clarify their meaning. The benefit of providing a distinct array type for distributions is that it drastically reduces file sizes. For example, a user may wish to quantify the probability distribution of a variable over a large spatial field (e.g., 1000 by 1000 cells), perhaps as a result of an interpolation process (Chapter 5). In such a case this would result in 1,000,000 distinct probability distributions. Without the aggregating mechanism of the `DistributionArray` `values` property, a distinct `Distribution` element would have to exist for each distribution. The result of which is a file size that is disproportionately large for the amount of data encoded, as the majority of the file is consumed with opening and closing `Distribution` tags and URIs, rather than the actual data. With the aggregation method described above, only a single opening and

closing tag and URI is required, dramatically reducing file sizes — a major benefit when working in a distributed context.

#### MixtureModel

A `MixtureModel` is a specialised form of record. Typically, when describing a variable using a mixture of distributions, a specific weight is assigned to each distribution specifying the relative influence of that distribution in the mixture. This constraint means that a simple '`DistributionRecord`' is not sufficient, so a dedicated `MixtureModel` is included.

The `distributions` property is equivalent to the `fields` property of a standard record type which may contain a type from the `AnyDistribution` union. The addition of a `weights` property allows a weight (floating point value) to be assigned to each distribution within the `distributions` property. It should be noted that when constructing `MixtureModel`s of `DistributionArray`s or `MultivariateDistribution`s care should be taken to ensure that there is a value for each individual distribution within the `distributions` property. In addition, though the sum of all values within the `weights` property should equal 1.0, this is not enforced within UncertML and it is the responsibility of the user to ensure such constraints are met.

Examples of `MixtureModel`s encoded using UncertML can be found in Section 3.3.

#### MultivariateDistribution

The final and most complex type in UncertML is the `MultivariateDistribution` type. Typical use cases for a multivariate (joint) distribution are when two variables are correlated, or a single variable is spatially correlated. As each of these scenarios requires the inclusion of a covariance matrix, the `DistributionArray` is not sufficient to describe the structure of the uncertainty. Also, because the `MultivariateDistribution` is fundamentally different to the `Distribution` element, the use of inheritance is not warranted. The major difference is the inclusion of a covariance matrix in most multivariate use cases.

A `MultivariateDistribution` is similar to the `Distribution` type, containing both a `definition` and `parameters` property. However, a significant difference is that the `parameters` property of a `MultivariateDistribution` now contains a number of `ParameterArray`s rather than `Parameter` types, due to the fact that multivariate distributions, by definition, always deal with arrays of parameters.

The `ParameterArray` type is similar to all other array types within UncertML, consisting of

an `elementType`, `elementCount`, `encoding` and `values` properties. The `elementType` property contains a `Parameter` type which provides a `definition` property. The `values` property then contains all values for that given parameter. A collection of such arrays allows the description of complex joint distributions in an efficient manner. Examples of how to use the `Multivariate-Distribution` type are detailed in Section 3.3.

## 3.3 XML encoding and examples

All models discussed in the previous section are encoded as a set of XML schemas. When using UncertML, instance documents conforming to the rules set out in these schemas shall be created. All types within UncertML are concrete types that may be used without the development of an additional application schema. All elements within the UncertML schema belong to the `http://www.uncertml.org` namespace. However, all namespaces within the examples in this section have been omitted for brevity. Any element using the prefix `un` belongs to the UncertML namespace.

The UncertML schemas also utilise two external schemas:

- Geography Markup Language, version 3.1.1, `http://www.opengis.net/gml`
- Sensor Web Enablement Common, version 1.0, `http://www.opengis.net/swe/1.0`

Elements prefixed with either `swe` or `gml` belong to the SWE Common or GML schemas respectively.

The rules used to encode the UncertML models into XML schemas are similar to those outlined in the GML specification (OGC 07-036, 2007). UncertML uses the design principle that there are objects that have properties which may themselves be realised as objects. According to these rules objects are realised as XML elements and are named using `UpperCamelCase` and properties are named using `lowerCamelCase`. XML types are named in `UpperCamelCase` and end with the word 'Type'.

Most properties and elements in the conceptual model are encoded as elements in XML. However, several properties are realised as XML attributes where deemed necessary.

### 3.3.1 Realisations

`Realisations` exists for situations where no *a priori* knowledge of the underlying probability distribution is available, or where complex processing of data, such as Monte Carlo analysis, is

required. Typically, a data sample made up of realisations will contain hundreds or thousands of values and an efficient encoding is required.

```
<un:Realisations definition="http://dictionary.uncertml.org/realisation"
samplingMethod="http://dictionary.uncertml.org/realisations/sampling_methods/direct
" realisedFrom="http://dictionary.uncertml.org/distributions/gaussian">
  <un:realisationsCount>100</un:realisationsCount>
  <un:elementCount>100</un:elementCount>
  <swe:encoding>
    <swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=","/>
  </swe:encoding>
  <swe:values>
    [100 space separated values]
  </swe:values>
</un:Realisations>
```

Listing 3.1: A set of realisations encoded using the UncertML `Realisations` type.

Listing 3.1 shows how a set of 100 sample realisation values can be encoded. The `realisationsCount` property states that 100 realisations were generated, and in combination with the `elementCount` (also 100) it is possible to deduce that these realisations refer to a single variable, at one domain point. However, information of this type is not intended to be extracted from the UncertML, as it would typically be encoded in a higher level schema, representing the domain points and phenomena sampled

The three URIs present in this example provide information about this particular realisation. The `definition` property resolves to a description about the concept of a realisation, `realisedFrom` resolves to a definition of a Gaussian distribution (from which this sample is realised) and the `samplingMethod` resolves to a description of a direct sampling technique. Resolving these URIs gives the user a complete picture of the true meaning of these realisations, when combined with the `values` property. For this example, and all subsequent examples, the `TextBlock` encoding is used as this best illustrates the examples.

The `TextBlock` contains three properties. The `decimalSeparator` defines a single character that represents a decimal place, `blockSeparator` represents the character that separates each new element in the `values` block and `tokenSeparator` separates individual items within an element. In the above example there is only a single value within each element — the actual realisation itself — therefore the need for a token separator is removed. Later sections demonstrate how the `tokenSeparator` can be used to encode complex elements, e.g. distributions.

### 3.3.2 Statistics

Statistics can be used to provide a concise summary of a particular variable. The examples in this section demonstrate how UncertML can be used to encode simple statistics, quantiles and probabilities. Examples of grouping several statistics to form a summary of a variable and a set of statistics at several domain points are also given. Finally, an example of how to encode complex structures, such as histograms, is provided.

#### Statistics, Quantiles, Probabilities and Moments

Due to the weak-typed design of UncertML all simple statistics appear identical in structure. What separates a 'mean' from a 'median' is the URI (and definition upon resolving) of the `definition` property. This is demonstrated in Listing 3.2 and Listing 3.3, where almost identical `Statistic` elements point to different definition URIs. Assuming the existence of a dictionary containing definitions of the most common statistics, only the URIs is needed in order for an application to 'understand' how to process the data.

```
<un:Statistic definition="http://dictionary.uncertml.org/statistics/mode">
  <un:value>34.67</un:value>
</un:Statistic>
```

Listing 3.2: The `Statistic` type encoding a basic statistic, 'mode'. A URI is used to provide the necessary semantics.

```
<un:Statistic definition="http://dictionary.uncertml.org/statistics/
standard_deviation" degreesOfFreedom="251">
  <un:value>12.08</un:value>
</un:Statistic>
```

Listing 3.3: A standard deviation encoded in UncertML using the `Statistic` type.

A whole array of statistics can be encoded in a similar fashion to the examples in Listing 3.2 and Listing 3.3, including mean, variance and median. However, certain statistics require slightly more detail. One such example is a quantile (Listing 3.4).

When encoding quantiles, it is crucial to quantify the particular quantile of interest. This information is encoded in the `level` property. In Listing 3.4 the 0.95 level quantile has a value of 34.34.

Similar to the quantile example is the `Moment` (Listing 3.5). When working with moments a user must know the moment order for meaningful processing. In Listing 3.5 the $2^{nd}$ order centred

```
<un:Quantile definition="http://dictionary.uncertml.org/statistics/quantile" level=
"0.95">
  <un:value>34.34</un:value>
</un:Quantile>
```

Listing 3.4: A `Quantile` encoded using UncertML. The `Quantile` type inherits from the `Statistic` type, adding a `level` attribute.

moment, or variance, is described.

```
<un:Moment definition="http://dictionary.uncertml.org/statistics/centred_moment"
order="2">
   <un:value>34.5</un:value>
</un:Moment>
```

Listing 3.5: A `Moment` encoded using UncertML. A `Moment` extends the base `Statistic` type, adding an `order` attribute.

Section 3.2.3 listed probabilities, both discrete and continuous, as another type of statistic that required more information. A discrete probability requires an association to a particular class. An example of how to encode a discrete probability in UncertML is seen in Listing 3.6.

```
<un:DiscreteProbability definition="http://dictionary.uncertml.org/
discrete_probability">
  <un:value>0.25</un:value>
  <un:category definition="http://www.mydomain.com/trees/list">Oak</un:category>
</un:DiscreteProbability>
```

Listing 3.6: A `DiscreteProbability` encoded in UncertML. A `DiscreteProbability` is used for quantifying variables which can be catagorised into discrete values.

The value of the `category` property can be any type, but the typical use case is to use a string (as demonstrated in Listing 3.6). It should be noted that the `value` property of all probabilities represents a probability in the range of 0.0–1.0, in contrast to other statistic types.

The final example is a continuous probability (Listing 3.7). Continuous probabilities see the addition of five properties: `gt` (greater than), `lt` (less than), `eq` (equal to), `ge` (greater than or equal to) and `le` (less than or equal to). Using these properties individually, or in combination, allows a user to specify thresholds of importance for a particular variable. Examples include: "What is the probability that a variable exceeds a given value?" Or, "what is the probability that a variable falls between an upper and lower bound?". Utilising the upper and lower bounds of several `Probability` types in combination with the grouping of the `StatisticsRecord` type allows the encoding of a histogram, seen in Listing 3.11.

```
<un:Probability definition="http://dictionary.uncertml.org/probability">
  <un:value>0.25</un:value>
  <un:gt>35.6</un:gt>
</un:Probability>
```

Listing 3.7: A continuous probability encoded in UncertML. The `Probability` type adds five properties to the `Statistic` type, specifying the thresholds that a variable may or may not exceed.

### StatisticArray

Large numbers of statistics can be efficiently encoded using the `StatisticArray` type. This aggregate type utilises the SWE Common `EncodedValuesGroup` to provide a host of encoding options. In its simplest form, a `StatisticArray` provides a collection of instances of a single statistic, as in Listing 3.8, which encodes an array of five 'mean' values.

```
<un:StatisticsArray>
  <un:elementType>
    <un:Statistic definition="http://dictionary.uncertml.org/statistics/mean"/>
  </un:elementType>
  <un:elementCount>5</un:elementCount>
  <swe:encoding>
    <swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=","/>
  </swe:encoding>
  <swe:values>46.76 25.75 57.432 12.42 53.64</swe:values>
</un:StatisticsArray>
```

Listing 3.8: A `StatisticArray` encoded in UncertML. This `StatisticArray` contains an array of five 'mean' values.

Encoding a series of summary statistics (i.e. mean and standard deviation) can be achieved through a combination of the `StatisticArray` and `StatisticsRecord` types. An example of such a scenario can be seen in Listing 3.9, which describes an array of five 'mean' and 'standard deviation' values of some variable(s).

When using a record inside an array, the `tokenSeparator` separates each individual item within the record. Dissecting the `values` block in Listing 3.9, a user can deduce that the values are in the order: `Mean`, `Standard Deviation`.

### StatisticsRecord

The previous section included an example of a `StatisticsRecord` inside a `StatisticArray`, but sometimes only a single group of statistics may be required. An example of a single `StatisticsRecord` can be seen in Listing 3.10, demonstrating how the mean, variance and the probability

```
<un:StatisticsArray>
  <un:elementType>
    <un:StatisticsRecord>
      <un:field>
        <un:Statistic definition="http://dictionary.uncertml.org/statistics/mean" />
      </un:field>
      <un:field>
        <un:Statistic definition="http://dictionary.uncertml.org/statistics/
standard_deviation" />
      </un:field>
    </un:StatisticsRecord>
  </un:elementType>
  <un:elementCount>5</un:elementCount>
  <swe:encoding>
    <swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=","/>
  </swe:encoding>
  <swe:values>
    34.64,67.86 45.65,78.41 56.7,45.75 29.86,56.74 45.65,76.43
  </swe:values>
</un:StatisticsArray>
```

Listing 3.9: A `StatisticArray` used in conjunction with the `StatisticsRecord` type to provide an array of summary statistic groups.

that a variable exceeds a certain threshold can be meaningfully grouped into a single structure.

Grouping statistics in this fashion allows complex structures to be formed. One such example is given in Listing 3.11, which illustrates how a histogram can be constructed by grouping several `Probability` types into a `StatisticsRecord`.

### 3.3.3 Distributions

The final subset of UncertML is concerned with the encoding of probability distributions. A range of options are available, including single distributions, arrays of distributions, joint distributions and mixture models. Examples of common distributions, e.g. Gaussian, arrays of distributions, mixture models and multivariate distributions are given.

#### Distribution

A `Distribution` type in UncertML is a record type. However, rather than having an unbounded number of `fields` it has `parameters`. The decision to extract all mathematical functions from the encoding of a distribution has enabled complex concepts, such as a Gaussian distribution to be easily encoded in a simple structure, shown in Listing 3.12.

Generating a weak-typed framework allows any distribution to be encoded in a single generic `Distribution` type. Provided that any processing applications understand which distribution is

```
<un:StatisticsRecord>
  <un:field>
    <un:Statistic definition="http://dictionary.uncertml.org/statistics/mean">
      <un:value>34.5</un:value>
    </un:Statistic>
  </un:field>
  <un:field>
    <un:Statistic definition="http://dictionary.uncertml.org/statistics/variance">
      <un:value>2.34</un:value>
    </un:Statistic>
  </un:field>
  <un:field>
    <un:Probability>
      <un:value>0.12</un:value>
      <un:gt>45.6</un:gt>
    </un:Probability>
  </un:field>
</un:StatisticsRecord>
```

Listing 3.10: `StatisticsRecord` encoded in UncertML to allow statistics to be grouped into a logical structure.

```
<un:StatisticsRecord>
  <un:field>
    <un:Probability definition="http://dictionary.uncertml.org/statistics/
probability">
      <un:value>0.24</un:value>
    <un:lt>30</un:lt>
    <un:ge>10</un:ge>
    </un:Probability>
  </un:field>
  <un:field>
    <un:Probability definition="http://dictionary.uncertml.org/statistics/
probability">
      <un:value>0.57</un:value>
    <un:lt>50</un:lt>
    <un:ge>30</un:ge>
    </un:Probability>
  </un:field>
  <un:field>
    <un:Probability definition="http://dictionary.uncertml.org/statistics/
probability">
      <un:value>0.19</un:value>
    <un:lt>80</un:lt>
    <un:ge>50</un:ge>
    </un:Probability>
  </un:field>
</un:StatisticsRecord>
```

Listing 3.11: `StatisticsRecord` in UncertML encoding a histogram. In this example, only a few bins are encoded, for brevity.

```
<un:Distribution definition="http://dictionary.uncertml.org/distributions/gaussian"
>
  <un:parameters>
    <un:Parameter definition="http://dictionary.uncertml.org/distributions/gaussian/
mean">
      <un:value>34.564</un:value>
    </un:Parameter>
    <un:Parameter definition="http://dictionary.uncertml.org/distributions/gaussian/
variance">
      <un:value>7.45</un:value>
    </un:Parameter>
  </un:parameters>
</un:Distribution>
```

Listing 3.12: A Gaussian distribution encoded in UncertML using the generic `Distribution` type.

being described (by resolving the URIs) then there is no need to include complex mathematical functions in the encoding. This is demonstrated in the example in Listing 3.13, which encodes an exponential distribution.

```
<un:Distribution definition="http://dictionary.uncertml.org/distributions/
exponential">
  <un:parameters>
    <un:Parameter definition="http://dictionary.uncertml.org/distributions/
exponential/rate">
      <un:value>34.564</un:value>
    </un:Parameter>
  </un:parameters>
</un:Distribution>
```

Listing 3.13: An exponential distribution encoded in UncertML using the generic `Distribution` type.

### DistributionArray

As with statistics, a situation can arise where a user wishes to encode multiple marginal distributions in an efficient structure. The `DistributionArray` type included in UncertML is similar in structure to a `StatisticArray` containing a `StatisticsRecord`.

Listing 3.14, illustrates that each parameter of the distribution (mean and variance) is separated by the `tokenSeparator`. Each individual distribution is then separated by the `blockSeparator` giving the following pattern: `mean,variance mean,variance` etc...

```
<un:DistributionArray>
  <un:elementType>
    <un:Distribution definition="http://dictionary.uncertml.org/distributions/
gaussian">
      <un:parameters>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
gaussian/mean"/>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
gaussian/variance"/>
      </un:parameters>
    </un:Distribution>
  </un:elementType>
  <un:elementCount>5</un:elementCount>
  <swe:encoding>
    <swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=","/>
  </swe:encoding>
  <swe:values>
    35.2,56.75
    31.2,65.31
    28.2,54.23
    35.6,45.21
    41.5,85.24
  </swe:values>
</un:DistributionArray>
```

Listing 3.14: A DistributionArray containing several Gaussian distributions. The values block contains 5 'mean,variance' tuples, one for each Gaussian distribution.

### MixtureModel

Mixture models are distributions that are convex combinations of other, weighted, distributions. Semantically, a `MixtureModel` is not dissimilar to any other record type in UncertML. There are a number of constituent fields (distributions) that make up a larger structure. However, the addition of a `weights` property allows a relative weight, or influence, to be assigned to each constituent part. Certain guidelines should be followed when using `MixtureModel`s to ensure interoperability. These are as follows:

- There should be a corresponding weight for each constituent distribution within the `distributions` property.
- Each weight should be a value between 0.0–1.0 inclusive (but this is not enforced).
- The sum of all weights should equal 1.0 (though again, this is not enforced).

An example of a `MixtureModel` is given in Listing 3.15, consisting of multiple Gaussian distributions and adhering to the above rules.

```
<un:MixtureModel>
  <un:weights>0.24 0.76</un:weights>
  <un:distributions>
    <un:Distribution definition="http://dictionary.uncertml.org/distributions/
gaussian">
      <un:parameters>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
gaussian/mean">
          <un:value>34.564</un:value>
        </un:Parameter>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
gaussian/variance">
          <un:value>67.45</un:value>
        </un:Parameter>
      </un:parameters>
    </un:Distribution>
    <un:Distribution definition="http://dictionary.uncertml.org/distributions/
gaussian">
      <un:parameters>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
gaussian/mean">
          <un:value>21.564</un:value>
        </un:Parameter>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
gaussian/variance">
          <un:value>34.45</un:value>
        </un:Parameter>
      </un:parameters>
    </un:Distribution>
  </un:distributions>
</un:MixtureModel>
```

Listing 3.15: A `MixtureModel` including a number of weights that corresponds to the number of constituent distributions.

Combining distributions and weights in this fashion allows the construction of complex mixture models. The ability to include `DistributionArrays` into the `distributions` property allows large mixture models to be efficiently created. Care should be taken when working with large mixture models to ensure adherence to the above rules.

#### MultivariateDistribution

The final type in the distributions subset of UncertML is the `MultivariateDistribution`. This type is required when there is correlation between variables, or a variable varies over a spatial-temporal domain. Much like a `Distribution` type, a `MultivariateDistribution` consists of a `definition` and `parameters` properties. However, instead of each parameter having a single value, it now consists of multiple values, depending upon either the number of variables or the extent of the domain. The notion of a multivariate distribution is complex, but the example in Listing 3.16 demonstrates the relative ease with which a multivariate Gaussian distribution may be encoded.

The first `ParameterArray` contains a vector of mean values whose dimensions depend upon the number of variables, and the size of the spatial or temporal domain. The second `ParameterArray` contains the covariance matrix, which contains the appropriate number of covariance values. Utilising the SWE Common `EncodedValuesGroup` ensures that the most efficient encoding of covariances is achieved in applications with large numbers of domain points, since these matrices can become exceedingly large under normal use.

### 3.3.4 ISO 19138 data quality measures

ISO/TS 19138 (2006) outlines a list of commonly used data quality reporting measures for the data quality sub-elements identified in ISO/TC 211 19113 (2002). This section provides several examples of how UncertML can be used to encode these data quality measures in XML.

Listing 3.17 demonstrates how the statistic 'mean value of positional uncertainties' may be encoded using UncertML. The example is identical to previous statistic examples with only the definition URL differentiating it. With the addition of a `Parameter` property, Listing 3.18 demonstrates how the 'mean value of positional uncertainties' can be extended to exclude outliers. In the example, the $e_{max}$ parameter is used to specify the definition of what an outlier is in this particular context, and its threshold nature and value type is therefore also specified within the dictionary.

The inclusion of data quality elements from ISO/TS 19138 (2006), such as 'data quality value

```
<un:MultivariateDistribution definition="http://dictionary.uncertml.org/
distributions/multivariate_gaussian">
  <un:parameters>
    <un:ParameterArray>
      <un:elementType>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
multivariate_gaussian/mean"/>
      </un:elementType>
      <un:elementCount>5</un:elementCount>
      <swe:encoding>
        <swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=","/>
      </swe:encoding>
      <swe:values>
        45.42 53.12 12.53 64.21 55.22
      </swe:values>
    </un:ParameterArray>
    <un:ParameterArray>
      <un:elementType>
        <un:Parameter definition="http://dictionary.uncertml.org/distributions/
multivariate_gaussian/covariance"/>
      </un:elementType>
      <un:elementCount>25</un:elementCount>
      <swe:encoding>
        <swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=","/>
      </swe:encoding>
      <swe:values>
        2.71828 0 0 0 0
        0 2.71828 0 0 0
        0 0 2.71828 0 0
        0 0 0 2.71828 0
        0 0 0 0 2.71828
      </swe:values>
    </un:ParameterArray>
  </un:parameters>
</un:MultivariateDistribution>
```

Listing 3.16: A multivariate Gaussian distribution encoded in UncertML using the `MultivariateDistribution` type.

```
<un:Statistic definition="http://dictionary.uncertml.org/statistics/
Mean_value_of_positional_uncertainties" degreesOfFreedom="228">
  <un:value xsi:type="xs:double">24.21</un:value>
</un:Statistic>
```

Listing 3.17: Mean value of positional uncertainties, (ISO/TS 19138, 2006) (Table D.29), encoded in UncertML.

```
<un:Statistic definition="http://dictionary.uncertml.org/statistics/
Mean_value_of_positional_uncertainties_excluding_outliers">
  <un:parameters>
    <un:Parameter definition="http://dictionary.uncertml.org/statistics/
Mean_value_of_positional_uncertainties_excluding_outliers/parameters/e_max">
      <un:value xsi:type="xs:double">25.12</un:value>
    </un:Parameter>
  </un:parameters>
  <un:value xsi:type="xs:double">23.45</un:value>
</un:Statistic>
```

Listing 3.18: Mean value of positional uncertainties excluding outliers, (ISO/TS 19138, 2006) (Table D.30), encoded in UncertML.

type' and 'data quality value structure', within the dictionary allows more complex statistics, such as a covariance matrix (Listing 3.19), to be encoded.

```
<un:Statistic definition="http://dictionary.uncertml.org/statistics/
Covariance_matrix">
  <un:value xsi:type="xs:base64Binary">
Mi43MTgyOCAwIDAgMCAwIDAgMi43MTgyOCAwIDAgMCAwIDAgMi43MTgyOCAwIDAgMC
AwIDAgMi43MTgyOCAwIDAgMCAwIDAgMi43MTgyOA==
  </un:value>
</un:Statistic>
```

Listing 3.19: Covariance matrix encoded in UncertML, (ISO/TS 19138, 2006) (Table D.33).

Listing 3.19 demonstrates the ability of a statistic value to contain any simple XML type; in this example the covariance matrix is encoded in base64 binary.

Listing 3.20 is an extract from the statistics dictionary that describes the concept and structure of a covariance matrix. The `dataQualityValueType` property states that it is of type 'Measures', and the `dataQualityValueStructure` defines it as 'matrix'. With this knowledge it is possible to deduce what the statistic value property contains.

Another example of how the `dataQualityValueType` and `dataQualityValueStructure` properties within the `definitions` are used can be seen in Listing 3.21. The uncertainty ellipse, as defined in ISO/TS 19138 (2006), does not contain a single value, rather a list (a, b, φ) of values. Including this information within the dictionary definition allows the base `Statistic` type to encode a multitude of different statistics, including the uncertainty ellipse, while maintaining the same structure.

```
<un:StatisticDefinition gml:id="Covariance_matrix">
  <gml:description>
  The covariance matrix generalises the concept of variance from one to n
dimensions, i.e. from scalar-valued random quantities to vector-valued random
quantities (tuples or scalar random quantities).
  </gml:description>
  <gml:name>Covariance matrix</gml:name>
  <gml:name>variance-covariance matrix</gml:name>
  <un:definition>
  A symmetrical square matrix with variances or point coordinates on the main
diagonal and covariances between these coordinates as off-diagonal elements
  </un:definition>
  <un:dataQualityValueType>Measures</un:dataQualityValueType>
  <un:dataQualityValueStructure>matrix</un:dataQualityValueStructure>
</un:StatisticDefinition>
```

Listing 3.20: UncertML dictionary extract describing a covariance matrix.

```
<un:Statistic definition="http://dictionary.uncertml.org/statistics/
Uncertainty_ellipse">
  <un:value>21.14 12.42 125.12</un:value>
</un:Statistic>
```

Listing 3.21: Uncertainty ellipse encoded in UncertML, (ISO/TS 19138, 2006) (Table D.52).

## 3.4 Relation to ISO standards

UncertML is not intended to be a solely geospatial standard, since the fundamental principle of UncertML (interoperable representation of probabilistic uncertainty) applies to a wider set of application domains. A conscious effort has been made to separate concerns and to ensure that UncertML is complete with respect to its remit: the probabilistic representation of uncertainty. Probabilistic uncertainty representation is pertinent to many applications, and provides significant benefits in terms of a well defined theoretical basis for propagating and working with uncertainty. However, while UncertML has wide applicability the largest potential application of (and motivation for) developing UncertML lies within the Sensor Web.

This section summarises the relationship of UncertML to existing ISO standards, while later examples address the specific relevance of UncertML to these standards in practice. For example, Section 3.3.4 shows the encoding of specific data quality measures from ISO/TS 19138 (2006).

### 3.4.1 ISO 19115: Metadata

There are two ways of thinking when it comes to uncertainty and metadata — either that uncertainty is in fact data about data, or that the data itself is uncertain and thus the uncertainty is the

| Property | Nature |
|---|---|
| valueType[0..1] | RecordType |
| valueUnit | UnitOfMeasure |
| errorStatistic | CharacterString |
| Value[1..*] | Record |

Table 3.2: Properties of a ISO 19115 DQ_QuantitativeResult.

actual data and not metadata. This is a concept that is discussed later in Section 2.4.1. Both cases are applicable in different situations. For instance, consider an observation, the result (or observed value), of this observation is considered to be the data. Any uncertainty in this observed value (e.g., measurement bias) can be considered metadata. However, consider a process, such as interpolation, the output of that process is itself uncertain — in this instance, the data is inherently uncertain.

With that in mind, the concepts that are discussed in ISO/FDIS 19115 (2003) suggest that UncertML is actually a realisation of the `DQ_QuantitativeResult` element, although UncertML does not to include units of measure. Comparing UncertML to a `DQ_QuantitativeResult` in this manner suggests that uncertainty is metadata, and this is certainly a domain where UncertML might be used. We have deliberately extended the definition in the `DQ_QuantitativeResult` section to allow very explicit but flexible representation of the uncertainty in the various types supported by UncertML. Concepts such as units of measure are not included in UncertML, but are intended to be represented by existing schemas, which will fully describe the phenomena to which the uncertainty pertains while using UncertML to efficiently describe that uncertainty. Table 3.2 lists the properties of a `DQ_QuantitativeResult` from ISO/FDIS 19115 (2003).

The information in Table 3.2 can be encoded using UncertML by delegating the `valueUnit` to a 'wrapper' which could also encode measurement-specific information such as location and sensor characteristics. The uncertainty would be represented by an UncertML type in which the nature of the statistic is given by a URI, rather than the `errorStatistic` string used here.

An alternative option would be to supply an UncertML type as the actual value of the `DQ_QuantitativeResult`, allowing the representation of qualified estimates for metrics which are more usually supplied as a single value for a dataset. For example, traditionally a `DQ_QuantitativeResult` would be used to represent the fact that the value of 'TopologicalConsistency' (ISO/TS 19138, 2006) for a polygon dataset is 75%. UncertML allows a more informative representation of this estimate, which is likely to be uncertain and to have been derived from a set of samples. This set of sample consistencies is also highly unlikely to have a symmetrical Gaussian

distribution, given the 0–100 bounded nature of the measurement scale. UncertML can represent the variability of the samples without over-simplifying their distribution; for example, (a) by encoding a histogram/set of quantiles recording the full range of consistency values recorded; or (b) by recording an exceedance probability which tells us that, based on our evidence, there is a 99% chance that the 'TopologicalConsistency' exceeds 75%; or (c) by recording the raw sample values, representing the variation across the dataset. In those cases where the value of a data quality metric genuinely is certain, UncertML also offers benefits: encoding a value as a Dirac (delta) distribution conveys the fact that there is a strong and certain belief in the value.

### 3.4.2 ISO 19114: Quality evaluation procedures

UncertML fits with this standard in the same way as with ISO/FDIS 19115 (2003) — via the `DQ_QualityResult` (realised as the subclass `DQ_QuantitativeResult`). ISO/TC 211 19114 (2003) draws a strong distinction between the different types of data quality measures (e.g., thematic, positional or temporal). This is a topic that is not addressed in UncertML; all these uncertainties can be described using the same basic UncertML types, within elements which deal with the context of the uncertainty in another schema. Thus the semantic description of, for example, how a 'vertical measurement error' differs from a 'percentage of incorrectly-classified pixels' would be the responsibility of elements specifically designed to describe these phenomena, while the statistical representation of uncertainty would be performed by UncertML. There is a fundamental difference between the conceptual approach of UncertML and ISO 19114, since while the ISO standard looks at the different ways to describe the quality of a data set, UncertML looks at how the results of these measures can be encoded concisely.

### 3.4.3 ISO 19138: Data quality measures

ISO/TS 19138 (2006) defines an extensive list of data quality measures that may be applied to datasets, but looks at describing these measures, rather than offering standard-specific means of actually representing those measures. It can be argued that UncertML should be capable of encoding all of the listed data quality measures; however, it is not clear that this falls within the scope of UncertML. Such a function might belong in a broader "Data Quality Markup Language" — which would make extensive use of the types in UncertML.

The concept of random variables were discussed in Section 2.3.1. ISO/TS 19138 (2006) relies on three assumptions when quantifying random variables; these are:

1. *Uncertainties are homogeneous for all observed values*

2. *The observed values are not correlated*

3. *The observed values are normally distributed*

UncertML addresses these issues as follows:

1) UncertML provides the flexibility to encode individual and global measures of uncertainty for a dataset, rather than using a single statistic. This is quite essential for many applications; for example in a heterogeneous sensor network it is quite likely that different instruments, and deployments of those instruments, may generate very different observational errors / uncertainties. It is also true that the result of a processing method (for example an interpolation operation) would produce different uncertainties at each location within the interpolation domain.

2) This assumption seems contradictory, since the standard later discusses the possibility of describing 2 or 3 random variables using a covariance matrix. As discussed in Section 2.3.1 and again in Chapter 6, correlated random variables are an important, and frequent occurrence in geospatial data. Consequently, the assumption made within ISO/TS 19138 (2006) is too restrictive for the requirements of UncertML.

3) Many of the uncertainties in this document's examples are assumed to be normally distributed — however, the flexibility of UncertML means that a user is not limited to using a normal distribution and may in fact describe a variable using any probability distribution. As users better understand and model the uncertainty in their data, this will become increasingly important. For example the results of many processing applications are likely to result in non-Gaussian distributions over their outputs, even when the inputs have an approximately Gaussian distribution. Also for many sensor types, for example most used in remote sensing, Gaussian errors are the exception, not the norm.

The work in this section has demonstrated that UncertML is flexible and capable not only to represent uncertainty using probability theory, but also to integrate neatly with the existing ISO standards. Conforming to existing standards, such as the ones discussed above, allows users to adhere to a common framework, thus ensuring that interoperability is maximised.

## 3.5 Conclusions

The lack of any existing XML standards for representing probabilistic uncertainty, discussed in Chapter 2, motivated the work in this chapter. The solution, UncertML, is an interoperable model

for representing uncertainty probabilistically. Section 3.2 introduced a conceptual model that underpins the whole of UncertML. Illustrated via a series of UML diagrams, a detailed description was given into how uncertainty can be represented within UncertML as realisations (Section 3.2.2), statistics (Section 3.2.3) and distributions (Section 3.2.4). The design methodology for UncertML, explained in the conceptual model, was a weak-typed design. The relative advantages and disadvantages of weak and strong-typed designs were discussed in the previous chapter. Section 2.2.1 concluded that a weak-typed design, with specifically chosen additional hard-typed elements, was the best approach. Consequently, the conceptual model in Section 3.2 made use of hard-typed elements where applicable, e.g. `Quantile`. The design outlined in this chapter attempts to answer the question raised in objective 1 (Section 1.3.1). The flexibility of the design is provided by the use of a weak-typed schema, allowing any uncertainty concept to be encapsulated. Including hard-typed elements for commonly used concepts (e.g. quantile) improves the usability of the design by affording additional statistic-specific attributes. For example, the use of quantile requires a level attribute. Judging the interoperability of an information model such as UncertML is challenging. A test of interoperability is perhaps the ability of a given data model to integrate with a diverse range of existing standards. This was demonstrated in Section 3.3.4 and Section 3.4 where examples, with supporting discussion, were given indicating how UncertML sits with existing ISO standards. The number elements within UncertML, including realisations, statistics and distributions and the combination of these elements, through the use of the aggregate types demonstrated the wide applicability of UncertML. The subsequent chapters in this thesis investigate how this flexibility can be utilised in real-world scenarios and seek to answer the remaining research questions outlined in Section 1.3.1.

The implementation of UncertML, based on the conceptual model, was achieved using the XML schema language. Section 3.3 provided an in-depth analysis of the structure of UncertML using numerous XML examples. Examples were given for all the basic types in UncertML including realisations, statistics and distributions (Sections 3.3.1– 3.3.3 respectively). Complex examples were provided to demonstrate the flexibility of UncertML, including aggregate types (e.g. arrays) and histograms. The XML examples illustrated how different statistics or distributions can be encoded using the same types, due to the weak-typed model. Emphasis was put on the importance of identifiers to provide the necessary semantic information. UncertML utilises the URL scheme for providing identifiers, adopting a REST-influenced approach for generating them. The development of a supporting ‘dictionary’, containing a governed list of identifiers commenced, however,

was not fully implemented.

The GUM guide, discussed in the introduction to this chapter, aspires to provide a worldwide consensus on the evaluation and expression of uncertainty in measurement, not dissimilar to the International System of Units. With the development of the UncertML standard and an accompanying dictionary a small step has been taken toward achieving this goal.

The work described in this chapter was accepted by the OGC as a discussion paper (OGC 08-122r2, 2008). The interest sparked by this discussion paper, alongside practical implementations and demands for usability with large datasets, led to the decision to redesign UncertML. At the time of publication (September 2010) an initial draft of UncertML version 2 has been prepared, some details of which are given in the future work section of Chapter 7. The main problem with the first version of UncertML is the adoption of a weak-typed solution. It is acknowledged in Chapter 2 that both weak and strong-typed designs have advantages and disadvantages. The major benefit of adopting a weak-typed solution is that it is flexible enough to allow the encoding of any statistic and distribution. However, with hindsight it is clear that only a small subset of the possible statistics and distributions are actually used in practice, and that the benefits of a hard-typed solution begin to outweigh those of a weak-typed design. While the design described in this chapter is both flexible and usable, it is thought that the migration to a hard-typed solution in version 2 will drastically increase the usability with only a slight reduction in flexibility.

# 4 UncertML and SOS

CONTENTS

## 4.1 Foreword

Chapter 3 presented a conceptual model and XML encoding for representing uncertainty using realisations, statistics and probability distributions. A key design choice was to encapsulate the representation of uncertainty, while keeping it separate from other concepts including phenomena and units of measure. Removing the responsibility of encoding such concepts from UncertML allows it to be integrated into a number of different domain standards. This chapter shows how UncertML may be integrated into the O&M and SOS standards.

Section 4.2 provides a critical analysis of the SOS standard. A discussion of the SOS requirements provides an insight into possible problems faced when implementing SOS-based software. The compromises made by software implementers to produce a usable product are discussed in Section 4.3. Focusing on the 52° North implementation, Section 4.4 demonstrates how an SOS may be extended to allow UncertML into the workflow. Section 4.4 concludes with a use case — providing access to Weather Underground data through a SOS interface.

Finally, Section 4.5 concludes the chapter with an overview of the contributions made and discussion about future work to be carried out.

## 4.2 Sensor Observation Service

Originally developed in 2005, the SOS is primarily used as an Application Programming Interface (API) for "managing deployed sensors and retrieving sensor data, specifically observation data" (OGC 06-009r6, 2007). The main aim, and challenge, of the SOS specification is to define a standard method for accessing sensor data that is consistent for all sensor systems. Considering the differences between remote (e.g., Light Detection And Ranging (LIDAR)), fixed (e.g., weather station) and mobile (e.g., radioactivity monitoring van) sensors, and their respective communities, it is clear that a flexible and robust solution is required. Two distinct data types are provided for by the SOS specification: sensors and observations. By utilising the OGC specifications, O&M and SensorML, all varieties of sensors and sensor data may be described. However, the generic conceptual models provided by SensorML and, more specifically, O&M, pose several significant problems when working at an implementation level, as discussed later in this chapter.

A comparison is made within SOS specification OGC 06-009r6 (2007) to the WFS specification (OGC 04-094, 2005). The WFS specification provides a service-based interface to GML features. These features are generic and may encompass any real-world entity. Features within

a WFS are accessed via the `GetFeature` request and may be queried using the OGC Filter Encoding specification (OGC 04-095, 2005). More detailed information on the WFS specification may be found in its implementation specification OGC 04-094 (2005). The WFS specification states that due to the generic feature definition, interoperability requires that organisations agree on domain-specific GML application schemas. Consequently, a WFS client for rich processing, in a particular domain, must have *a priori* knowledge of the application schemas used in that domain. Comparisons are made within this OGC document to the SOS approach, where a common model for all sensors and their observations is defined. The authors continue to state that this model is 'horizontal', since it applies to all domains, with specific details delegated to the second layer (feature of interest, observed properties and sensor descriptions). The argument is that because of this basic, universal observation description, a generic client can process any observation. However, that statement is misleading, and one may argue that it is in fact *incorrect*. While it is true that a generic client may parse an observation in enough detail to understand that it has `featureOfInterest`, `observedProperty` and `result` properties, it cannot reliably parse the contents of these properties. The problem stems from the fact that the observation model described in the O&M specification (OGC 07-022r1, 2007) is at a higher level of abstraction than is necessary for implementation. One example of this abstraction is the `result` property. O&M states that an observation may contain any data as a result and, due to the broad spectrum of possible types of observation, one would be inclined to agree. However, when dealing with software that implements this observation model it is simply not sufficient to suggest that the result may consist of anything; constraints must be imposed. Another, perhaps more significant, example is that of geospatial location. As discussed in Section 2.5.4, the O&M model does not provide an explicit property for encoding the spatial domain of an observation — instead, this responsibility is delegated to the `featureOfInterest` property. Conceptually, the idea that an observation is an event observing some feature of interest which may, or may not, have spatial properties is logical. However, as before, when implementing software this proves insufficient. Due to the `featureOfInterest` being any GML feature, typically defined by domain experts, the problems faced by the WFS specification, (i.e., a requirement for *a priori* domain knowledge) also apply to the SOS specification. In fact, the topic of abstract specifications and implementation problems within SOS and O&M is so pertinent that discussions are still ongoing throughout the OGC mailing lists, with compelling arguments for both sides.

#### Observation offerings

The notion of an 'observation offering' is introduced in the SOS specification as a collection of related sensor system observations. Offerings may be constrained temporally, spatially, by phenomena or by sensor type. The necessity of observation offerings is not made clear within the SOS specification, which only provides the following scenario.

> "... suppose an SOS instance advertises two sensors — one that reports wind speed and the other that reports air temperature. If these sensors are attached to the same weather station, they should probably be included in the same offering. That is because the GetObservation request for weather data for a given area that includes the weather station to which the sensors are attached and for time periods that the weather station is reporting will almost always have data for both sensors. If the client asks for wind speed only, air temperature only, or both, the time and location of the results should be the same.
>
> On the other hand, if the two sensors are located on weather stations that are far apart in space or which report during non-overlapping time periods, then they should probably be factored into two distinct offerings. If they were put into the same offering, then the combinatorial space of that single offering would be relatively sparse. In other words, it would frequently be the case that a `GetObservation` request asking for temperature might return a result where one for wind speed might not. The client might have to make quite a few `GetObservation` requests on a "sparse" offering before finding data, which is clearly undesirable."

The uncertain terms such as 'probably', 'almost', 'should' and 'might' within those paragraphs serve only to confuse what is already a flawed argument. Logically, a request for air temperature or wind speed observations constrained both temporally and spatially should return any observations that match these criteria, regardless of which offering they belong to. The addition of an observation offering therefore adds another layer of complexity when constructing a query. Including a particular observation offering in a spatio-temporal query runs the risk of excluding observations which belong to another offering, but which otherwise match the search criteria. This problem is further amplified when the limitation that only one offering may be specified in any given request, requiring the user to make several requests (a side-effect that observation offerings sought to remove) is considered.

### 4.2.1 Core operations profile

A SOS may conform to one of four profiles, each providing a different set of operations. The operations of the core profile, discussed in this section, form the basis of any SOS instance. The enhanced and transactional profiles add further operations (discussed later) while the entire profile implements all operations of the previous three profiles. While this thesis discusses the operations


Figure 4.1: A sequence diagram showing a typical work flow for a consumer of sensor observation data. Taken from OGC 06-009r6 (2007).

of all SOS profiles, the work contained within this chapter focuses on the core set of operations and subsequently no reference is made to either enhanced or transactional profiles in Sections 4.3– 4.4.

The SOS specification (OGC 06-009r6, 2007) describes two distinct perspectives that may be taken of a SOS; the perspective of a consumer of sensor observations and the perspective of a provider of sensor observations. Throughout this chapter only the consumer perspective is considered, and a typical work flow is given in Figure 4.1. The three operations implemented by the core profile are `GetCapabilities`, `DescribeSensor` and `GetObservation`. In the figure, SOS 1 and SOS 2 represent different SOS implementations, the workflow demonstrates how the same requests can be sent to SOS implementations from different providers.

#### GetCapabilities

Common to all OGC Web services is the `GetCapabilities` request. As discussed in Chapter 2 this method is defined by the OWS Common specification (OGC 06-121r3, 2007). The responsibility of this operation is to allow users to access metadata about a particular service instance. A `GetCapabilities` request is constructed as described in Subclauses 7.2 and 7.3 of the OWS Common specification OGC 06-121r3 (2007). The `Capabilities` document generated by an SOS instance includes all sections defined within the OWS specification plus an additional two sections: `Contents` and `FilterCapabilities`. The `Contents` section of a `Capabilities` response includes all relevant information about observation offerings for a given SOS instance. As mentioned previously, the SOS specification utilises the OGC Filter language as a mechanism for ad-hoc querying. However, any given SOS instance may not support all facets of the Filter language and such limitations must be detailed in the `Capabilities` document. The `FilterCapabilities` section does precisely that, informs clients about what query parameters are supported by the service.

#### DescribeSensor

The SOS specification was influenced by the WFS specification (OGC 04-094, 2005) with respect to how a user might interact with the service. This pattern can be seen in Figure 4.1 where a user first queries the capabilities of a service, through the `GetCapabilities` request, then proceeds to retrieve more information, through a `describe` operation, before finally making a query for data using a `get` operation. The `DescribeSensor` process conforms to this interaction pattern

by allowing users to retrieve detailed sensor metadata. A list of sensor models associated with a given SOS instance can be listed in the response to a `GetCapabilities` request. A user can then request more information about a particular sensor model via the `DescribeSensor` operation. A response to the `DescribeSensor` process would typically be encoded in either SensorML (OGC 07-000, 2007) or TML (OGC 06-010r6, 2006) and may include lists and definitions of observables supported by the sensor. A detailed SensorML example is outside the scope of this thesis, however, more detail can be found in the SensorML specification (OGC 07-000, 2007). While detailed information about the associated sensor models my be relevant for particular applications, it is not a prerequisite for obtaining observation data and consequently a `DescribeSensor` request may be omitted from a user's work flow.

#### GetObservation


Aston University
Illustration removed for copyright restrictions

Table 4.1: The parameters of the GetObservation request, taken from OGC 06-009r6 (2007).

The GetObservation request provides the core functionality of a SOS, designed to retrieve observation data formatted as an O&M document. A GetObservation request contains one or more constraints on the observations being retrieved from a SOS. Constraints can be specified on the time, location and observed properties of an observation as well as the encoding and format of the response. A full list of parameters for a GetObservation request can be seen in Table 4.1.

Specific parameters listed in Table 4.1 highlight the potential issues, mentioned earlier in Section 4.2, regarding the ability of generic clients to consume SOSs. For instance, the responseFormat parameter allows observation results to be encoded in a variety of standard formats. Forgetting, for a moment, the inherent complications of parsing an O&M document, to suggest that a generic client should be capable of parsing an arbitrary number of possible formats is nonsensical. The supported formats are described in the Capabilities document so, arguably, a generic client may select a format which it understands. However, these formats are decided upon by a particular service provider and therefore may consist only of formats not supported by a particular client.

Perhaps such difficulties are encountered due to the ambitious requirements set out in the SOS specification. Specifically, the requirement that a SOS should provide access to observations that is consistent for all sensor systems (remote, in-situ, fixed and mobile), may prove too broad a scope to allow an efficient and reliable interface to be implemented. These problems stem from the high-level conceptual model of observations, outlined in the O&M specification OGC 07-022r1 (2007), around which the SOS specification is built. A possible solution might be to maintain O&M as a conceptual model, free of implementation specifics, yet provide a series of profiles that restrict the model for various implementation use cases. An attempt is already in place to refine the O&M specification via a series of 'specialised observations', including a Measurement, CategoryObservation, CountObservation and GeometryObservation type. However, these restrictions only apply to the result property of an observation. Developing this idea further would allow tighter restrictions to be applied to other observation properties, in particular the featureOfInterest. While the featureOfInterest is a domain-specific property, and consequently difficult to standardise, it is the property that should contain the geometric location of an observation — important for many use cases. Providing a common set of features of interest would enable re-

liable spatial querying. However, a more satisfactory solution would be to provide a separate sampling domain property, leaving the feature of interest intact. Section 4.3 examines how current implementations of SOS software handle spatial querying.

#### Additional operations

Other operations outside of the core profile are detailed below.

**RegisterSensor** The `RegisterSensor` operation is a mandatory operation for the transactional profile. A `RegisterSensor` request includes a SensorML or TML document and an O&M instance that is a template for the observations that will be published for this sensor (using the `InsertObservation` operation). However, the specification is not clear about why such a template is required.

**InsertObservation** The `InsertObservation` operation is an optional operation for the transactional profile. An `InsertObservation` request must include the sensor ID generated from a successful `RegisterSensor` request and an O&M instance that follows the template provided when the sensor was registered.

**GetObservationById** The `GetObservationById` operation is an optional operation for the enhanced profile. A `GetObservationById` request contains the ID of the requested observation and optional parameters that control the output format of the returned observation. The optional parameters are the `resultModel` and `responseMode` parameters seen in Table 4.1.

**GetResult** The `GetResult` operation is an optional operation for the enhanced profile. The motivation for the `GetResult` operation is to allow a user to request sensor data from the same sensor repeatedly. The benefit is that it uses much less bandwidth than a full `GetObservation` request. However, in practice the `GetResult` operation is cumbersome as it relies on a successful `GetObservation` request beforehand.

**GetFeatureOfInterest** The `GetFeatureOfInterest` operation is an optional operation for the enhanced profile. A `GetFeatureOfInterest` request allows the user to fetch a particular feature of interest, advertised within the capabilities document, encoded as a GML feature.

**GetFeatureOfInterestTime** The `GetFeatureOfInterestTime` operation is an optional operation for the enhanced profile. Given a particular feature ID, this request returns one or more

GML time primitives that define the time periods for which observations exist for this feature of interest. However, the granularity (i.e., every year, month, day etc.) of observations is not provided.

**DescribeFeatureType** The `DescribeFeatureType` operation is an optional operation for the enhanced profile. A `DescribeFeatureType` request takes a particular feature ID (advertised in the capabilities document) and returns the XML schema.

**DescribeObservationType** The `DescribeObservationType` operation is an optional operation for the enhanced profile which returns the XML schema for a specified phenomenon.

**DescribeResultModel** The `DescribeResultModel` is an optional operation for the enhanced profile. This operation returns the XML schema for the result element of an observation.

The transactional profile offers two useful operations for the data provider perspective — `RegisterSensor` and `InsertObservation`. However, due to the high level abstraction in the O&M schema, similar difficulties, in terms of domain specificity, are faced within the `InsertObservation` operation to those posed by the `GetObservation` operation.

The enhanced profile adds a collection of varied operations that allow specific pieces of information to be retrieved from the service. There are certain similarities between the SOS `GetFeatureOfInterest` operation and the WFS `GetFeature` operation. With such a robust interface for querying features already defined in the WFS specification, it would seem logical to delegate the retrieval of features from within a SOS to a separate WFS instance. The same argument could be applied to the `DescribeFeatureType` operation.

### 4.2.2 Uncertainty within an SOS

All sensor observation data is uncertain, be it from sensor measurement error, observation operator error or processing errors. The degree of uncertainty varies per observation: some are essentially certain, whereas others may be significantly erroneous. Decision making based on uncertain data requires these uncertainties to be quantified. The current SOS specification (OGC 06-009r6, 2007) does not mention uncertainty, or data quality, explicitly. However, as it is based upon the O&M specification, one can assume that the methods discussed in Section 2.5.4 apply. The omission of quality information from the specification suggests that data filtering based on uncertainty is currently not supported, a possible side-effect of the absence of a defined standard for quantifying uncertainty.

Since the SOS aims to provide free access to sensor observed data it is a logical extension to integrate UncertML within the service. Doing so would allow users to assess the quality of the observation data before using it in their work flows. Extending the idea of uncertainty representation further into the OGC Filter language could allow observation data to be filtered based upon specific quality assurance policies. As the SWE specifications mature, an increase in available data will emerge; with no means of governing data suppliers, quantifying the quality of data becomes essential.

## 4.3 Existing SOS implementations

At the time of writing, few implementations of SOSs exist. As the SOS standard is still in its infancy, the lack of available software is not unexpected. However, it may be partly attributed to the possible implementation issues discussed in the previous section. Currently, a lack of software is not critical; however, the wide-spread adoption of standards can often go hand-in-hand with functional software implementations.

This section investigates existing solutions to implementing the SOS interface and the methods they use to bypass the issues raised in the previous section. A particular focus is put on the 52° North implementation.

### 4.3.1 52° North SOS

The 52° North SOS is a free piece of software written in Java that conforms to the OGC SOS specification version 1.0. Data storage within the 52° North SOS is delegated to the PostgreSQL database management system (DBMS)[1], with PostGIS extensions [2], while XMLBeans[3] is utilised to parse and encode the XML data. The two primary goals of the 52° North SOS are to:

1. Allow data providers to easily install and publish domain-specific sensor observation data.
2. Allow data consumers to query and retrieve sensor observation data, according to the SOS specification.

These two goals should be applicable for all domains, a constraint that is not easily fulfilled. The following sections discuss the operations, filtering capabilities and underlying data structure employed by 52° North and highlight any assumptions or compromises that have been made.

[1] http://www.postgresql.org/
[2] http://postgis.refractions.net/
[3] http://xmlbeans.apache.org/

### Supported operations

As discussed in Section 4.2 there are a number of different SOS profiles outlined in the specification (OGC 06-009r6, 2007). Each defined profile has a set of operations that should be implemented to conform to that particular profile. Within the 52° North SOS, the following operations have been implemented.

- **Core profile**
  - `GetCapabilities`
  - `DescribeSensor`
  - `GetObservation`
- **Transactional profile**
  - `RegisterSensor`
  - `InsertObservation`
- **Enhanced profile**
  - `GetResult`
  - `GetObservationById`
  - `GetFeatureOfInterest`
  - `GetFeatureOfInterestTime`

Comparing this list to Table 4.1, it is clear that the current version of the 52° North SOS implements a transactional profile. With only a few enhanced operations missing, a relatively small amount of work is required to progress the 52° North SOS to conform to the enhanced profile.

### Filter capabilities

The OGC Filter specification covers a broad spectrum of possible filtering options. It is unreasonable, therefore, to expect any given piece of software to support all filter capabilities; the 52° North SOS is no different. Below is a list of the currently supported filter capabilities of the 52° North SOS. It is split into three distinct sections: spatial, temporal and result. These relate to the properties of an observation (according to the O&M conceptual model). Spatial filters are applied to the feature of interest, temporal filters are applied to the `samplingTime` property and result filters are applied to the `result` property.

- **Spatial filters**
  - `Contains`
  - `Intersects`

  - Overlaps
  - BBox
- **Temporal filters**
  - T_Equals
  - T_After
  - T_Before
  - T_During
- **Result filters**
  - PropertyIsEqualTo
  - PropertyIsNotEqualTo
  - PropertyIsLessThan
  - PropertyIsGreaterThan
  - PropertyIsLessThanOrEqualTo
  - PropertyIsGreaterThanOrEqualTo
  - PropertyIsLike
  - PropertyIsNull
  - PropertyIsBetween

52° North have provided substantial filtering capabilities within their SOS implementation, but certain compromises had to be made. For instance, the O&M specification states that spatial properties of an observation should reside in the `featureOfInterest` element, which may contain any GML feature. Also, the GML specification states that a feature may have several spatial properties. The combination of these two requirements ultimately converts what should be the trivial task of filtering observations spatially, into a complex and unreliable operation. For example, if the spatial filter `Contains` was applied in a SOS `GetObservation` request on an observation whose feature of interest contained multiple spatial properties, it is uncertain which of the spatial properties should be filtered, if any, leading to possible non-deterministic behaviour. In order to alleviate such issues, the 52° North project have limited the feature types which may reside in the feature of interest. Specifically, in the current implementation, an observation may either have a `SamplingPoint` or a `SamplingSurface` as a feature of interest. These feature types are defined in Sampling Features specification OGC 07-002r3 (2007) and each may act as a proximate feature of interest, referencing the ultimate feature of interest through the `sampledFeature` property. This solution provides a simple method of filtering observations spatially, as a `SamplingPoint` may only contain a GML `Point` geometry whereas a `SamplingSurface` may only contain a surface geometry. However, an obvious compromise has been made by limiting the types of geometry which an observation within the 52° North SOS may contain. Similar problems exist with the

filtering of the result property. The O&M specification states that the result of an observation is the XML `anyType`. This requirement is even more problematic than those referring to the feature of interest, as it does not restrict the result to any specific schema. 52° North utilise the SWE Common `DataRecord` for all observation results within the SOS, providing a method of filtering the `result` property. Both the compromises discussed here underline the difficulties faced by a generic client of a SOS.

Perhaps a more interesting omission is present within the temporal filtering. A typical use case for an observation data consumer may be to gather the latest observations from each sensor within a particular offering. With the available temporal filters this is currently not possible, a filter like `T_Last` or `T_Latest` is required, and this cannot be built up from available component operators. A possible workaround might be to request all observations after a specific time. However, this is not satisfactory, since one sensor may record observations every minute whereas another may only record every hour. Filtering after a specific time in this scenario would either lack observations from the second sensor or contain duplicate observations from the first sensor. This duplication of observations at the same location can raise exceptions in certain processes, specifically interpolation methods, as discussed in Chapter 5. It is not clear whether this omission is the responsibility of the filter encoding specification, the SOS specification, or even the client applications. However, as requesting the latest observations is a fairly common requirement, one could argue it should be handled by the SOS. 52° North have developed a 'hack' that allows a request to retrieve the latest observations, but unfortunately it is not reliable and is consequently not documented.

#### Data model

The previous sections discussed the shortcomings of the SOS specification and the compromises made by 52° North in order to produce a usable system. This section dissects the underlying data model used within the 52° North SOS implementation and how the compromises made are reflected within this model. Discussion then evolves onto how uncertainty is currently handled underlining the strengths and weaknesses of the current approach.

Figure 4.2 displays a subset of the key tables within the 52° North data model. In the diagram, a database table is represented by a rectangle and the columns each represent a piece of data. The data types are listed on the right of the column names. A key symbol represents a primary key and an arrow symbol is a foreign key. An 'N' symbol signifies that the column is optional, or nullable. Relationships are represented via broken lines; a crow's foot represents a 'many' relationship

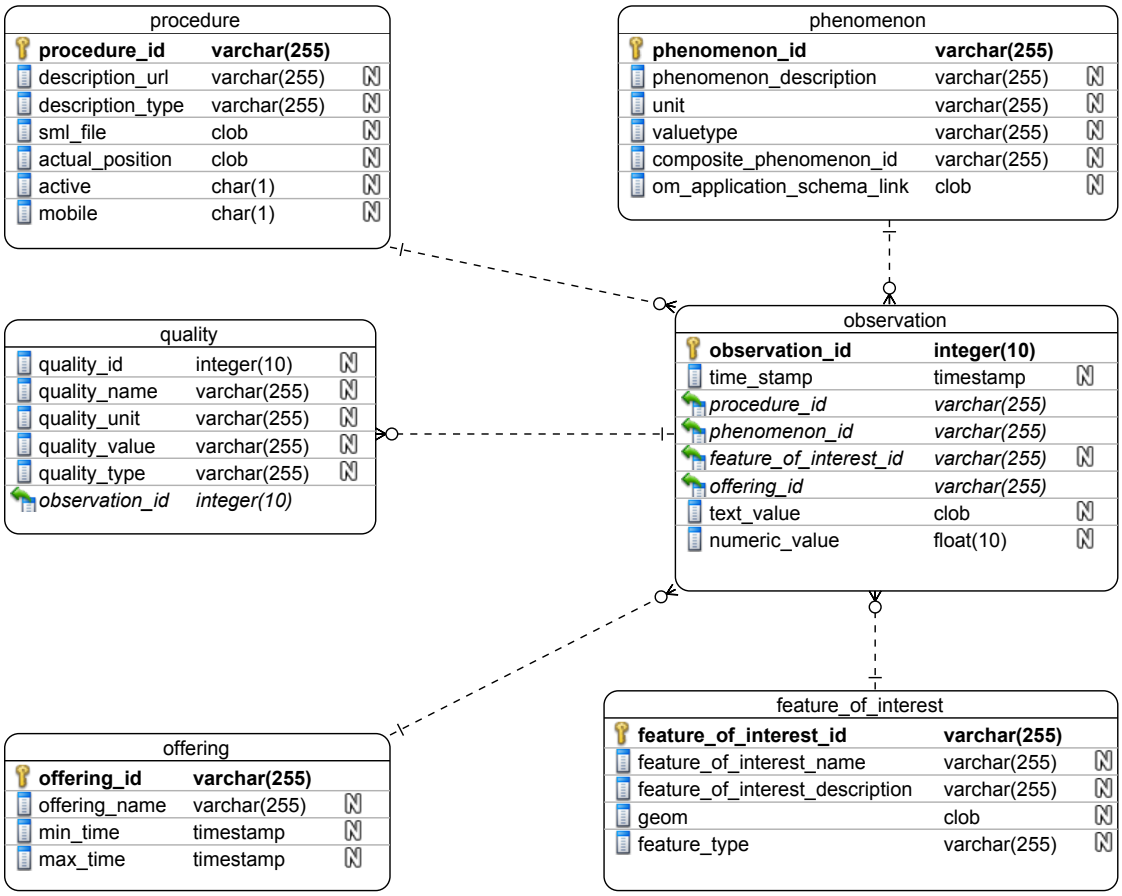



Figure 4.2: ERD of the 52 North SOS.

and a single horizontal bar, perpendicular to the broken line, represents a 'one' relationship. For example, the relationship between the observation and phenomenon table in Figure 4.2 is a 'many-to-one' relationship.

The central table is the observation table which stores a time stamp, value (either textual or numeric) and a series of foreign keys linking to other tables. The design of this table closely imitates the conceptual model outlined in the O&M specification. One of the compromises mentioned before is strikingly obvious upon examining the observation table; the result. Within the 52° North SOS a result may be stored as either a textual or a numeric value. This is a vast limitation on the O&M model which states that an observation result may be anything; however, it is necessary in order to provide a feasible implementation. Although, due to the `text_value` column being of data type `TEXT` one could argue that any XML could conceivably be stored within the database. Whether an observation can have both textual and numeric results is not made clear within the 52° North documentation.

The `observation` table contains four foreign key relationships. Firstly, there is an `offering_id` which links to an `offering` table. Observation offerings are a way of logically grouping a collection of sensor observations and are discussed in Section 4.2. An offering in the data model consists of a name and bounding times for the observations contained within the offering. A second relationship exists between the observation and `phenomenon` tables. Referring to the O&M specification, the `phenomenon` table can be thought of as the `observedProperty` of an `Observation`. The `phenomenon_id` should be the URN of the phenomenon as specified by the OGC. A typical value might be `urn:x-ogc:def:phenomenon:OGC:Temperature`. The `value_type` column specifies what value type an observation of this phenomenon will contain. The possible value types are `integerType`, `doubleType`, `floatType` and `textType`. The `uom` column contains the units of measurement for a particular phenomenon, encoded as a URN. The third relationship exists between the `observation` and `procedure` tables. A `procedure`, as specified by the O&M specification, is the sensor which produced the observation. While no restriction is made within the O&M specification, the value of a procedure is typically a SensorML document. 52° North specify that the only mandatory column within the `procedure` table is the `procedure_id`, which should contain a URN pertaining to the procedure as specified by OGC. However, the `description_url` column allows a reference to be made to a SensorML document located on the SOS server. The fourth, and most interesting, relationship is between the `observation` and the `feature_of_interest` tables. Within the `feature_of_interest` table several design choices

have been made. Firstly, a PostGIS geometry column (`geom`) has been included, which contains a point. This inclusion is to facilitate the application of spatial queries to the data. Secondly, there is a `feature_type` column which is used to store the type of the feature of interest. The documentation on what should go in this column is missing, however, a quick look at the underlying code reveals that it should either be `om:SamplingPoint` or `om:SamplingSurface` otherwise the feature of interest, for that observation, is ignored. Furthermore, it is not clear how a sampling surface is handled using the point geometry specified in PostGIS. While this feature of interest model is greatly simplified from the O&M 'any feature' requirement it enables spatial querying, which is a fundamental requirement of such a service. It should also be stated that a table exists to encode a 'domain feature' which links to a particular feature of interest; this allows further, domain-specific, information to be included within the model. It is envisaged that the domain feature would include a URL referencing a WFS instance to provide the feature information.

Examining the foreign key relationships within the diagram reveals a poor design choice made throughout the 52° North data model. Each of the four foreign keys discussed is of Structured Query Language (SQL) type `VARCHAR`. There are numerous problems with using `VARCHAR` data types as primary and foreign keys. One issue is memory use. Consider that for every observation in the table it must store a character string for the reference to the procedure. Now, consider that this string must be an OGC specified URN which typically takes the following form: `urn:ogc:object:feature:Sensor:IFGI:ifgi-sensor-1`. This string contains 48 characters, each consuming a single byte. If a more conventional database design was implemented, where the primary and foreign keys were integer values (each consuming 32 bits, or 4 bytes) a reduction of 44 bytes, per record, is achieved. In a database containing a modest 5,000 observations, this reduction translates to a total of 220,000 bytes. Taking the other three relationships into account, assuming similar sized URNs, brings the total reduction to 880,000 bytes. There are also numerous many-to-many relationships between these tables, each of which must contain two `VARCHAR` primary keys. While memory use is not as critical as perhaps it once was, there are further disadvantages of using `VARCHAR` primary keys that compound the memory issue. Indexing on integer values is relatively quick compared to `VARCHAR`, which results in a performance hit when performing an SQL `JOIN` between tables. Also, if for some unpredicted reason the OGC change the URN for a sensor then not only does the entry in the procedure table have to change, every observation that references that sensor must also be updated. For a relatively simple change, possible performance improvements could be seen.

```
<om:Measurement gml:id="o_23">
   <om:samplingTime>
      <gml:TimeInstant xsi:type="gml:TimeInstantType">
         <gml:timePosition>2008-04-01 17:44:00</gml:timePosition>
      </gml:TimeInstant>
   </om:samplingTime>
   <om:procedure xlink:href="urn:ogc:object:feature:Sensor:IFGI:ifgi-sensor-1"/>
   <om:resultQuality>
      <swe:Quantity>
         <gml:name>completeness</gml:name>
         <swe:uom code="percent"/>
         <swe:value>10.0</swe:value>
      </swe:Quantity>
      <swe:Category>
         <gml:name>accuracy</gml:name>
         <swe:codeSpace xlink:type="mm"/>
         <swe:value>1</swe:value>
      </swe:Category>
   </om:resultQuality>
   <om:observedProperty xlink:href="urn:ogc:def:phenomenon:OGC:1.0.30:waterlevel"/>
   <om:featureOfInterest/>
   <om:result uom="cm">50.0</om:result>
</om:Measurement>
```

Listing 4.1: An O&M `Measurement` retrieved from a 52° North SOS implementation. The SWE Common quality model is used for quantifying the uncertainty.

Despite uncertainty not being explicitly mentioned in the SOS specification, 52° North have identified the importance of quantifying uncertainty on sensor observed data by including a `quality` table. However, the facility to output uncertainty is switched off by default and thus mandates a change in settings to enable it. The structure of the `quality` table (Figure 4.2) includes a `quality_name`, `quality_unit`, `quality_value`, `quality_type` as well as a reference to the `observation` table via an `observation_id` column. The `quality_unit` and `quality_value` columns are self-explanatory, however, the purpose of the `quality_type` is not so clear. Examining the sample data supplied with the 52° North SOS indicates that the `quality_type` column should be populated with `quantity`, `count` or `category`. While it is not explicitly stated in the documentation, it is evident that 52° North are using the SWE Common model for quantifying uncertainty, as discussed in Chapter 2. The problems discussed in Chapter 2 with the SWE Common uncertainty model are evident in the sample data supplied by 52° North. Listing 4.1 is an O&M sample output from the 52° North SOS with example data. In this example the observation `resultQuality` property contains two SWE common quantities. The meaning of these quantities is not immediately clear, however, with the terms ‘completeness’ and ‘accuracy’ not providing the necessary semantics for numerical processing. More explicitly quantitative names could be assigned within

the database, for example 'range' or 'average'. However, many such terms are ambiguous and without a standard method for quantifying uncertainties, such as described in Chapter 3, reliable processing of uncertain data cannot be achieved. Extending the code base of the 52° North SOS implementation to facilitate the use of UncertML for quantifying the observation uncertainty will not only provide a clearer understanding of the quality of the supplied data, but also open up the possibility of filtering data based on stringent quality demands.

### 4.3.2 Other SOS implementations

While the 52° North SOS is arguably the most widely used SOS software, there are a couple of alternatives.

The deegree project [4] is an extensive collection of software tools for implementing OGC standards. The latest stable release of deegree boasts robust implementations of the WMS, WFS, WCS and Catalogue Service Web-Profile. The package also includes support for GML 3.1, including complex features. Unfortunately, at the time of writing, support for the OGC SOS is only included in the third iteration of the deegree project, deegree3, which is in the development stages.

An alternative to the deegree project is MapServer [5]. Originally developed as an OGC WMS, MapServer has been extended to include a number of other standards including WFS, WCS and SOS. Unfortunately, the roots of MapServer as a WMS are evident in the implementation of the SOS specification. For example, each SOS server needs a unique `MapFile`, a MapServer specific file format. A `MapFile` refers to concepts such as layers, a term borrowed from the WMS specification that is not directly relevant to a SOS instance. References are also made to image formats and paths, which may cause confusion within a SOS infrastructure, where the primary output format is XML.

While the choice of SOS implementations is varied, for the purpose of this thesis the 52° North implementation was chosen for a number of reasons. Firstly, the current latest release is version 3 of the software, providing a robust framework. More importantly, the lead developers of the 52° North SOS are also active members of the SOS Standards Working Group, allowing the 52° North implementation to develop in tandem with the evolving SOS specification.

[4] `http://deegree.org`
[5] `http://mapserver.org`

## 4.4 Integrating UncertML with 52° North

Section 4.2 discussed how the SOS specification functions as an API for accessing sensor observed data formatted as an O&M document. Due to the flexible model outlined defined by O&M, any XML type may be used within the `resultQuality` property (Chapter 2). This flexibility allows UncertML to be integrated within O&M, and thus within a SOS, without the requirement to alter the underlying schema. Leaving the O&M core schema unaltered maintains backwards compatibility with existing SOS clients.

### 4.4.1 Extended data model

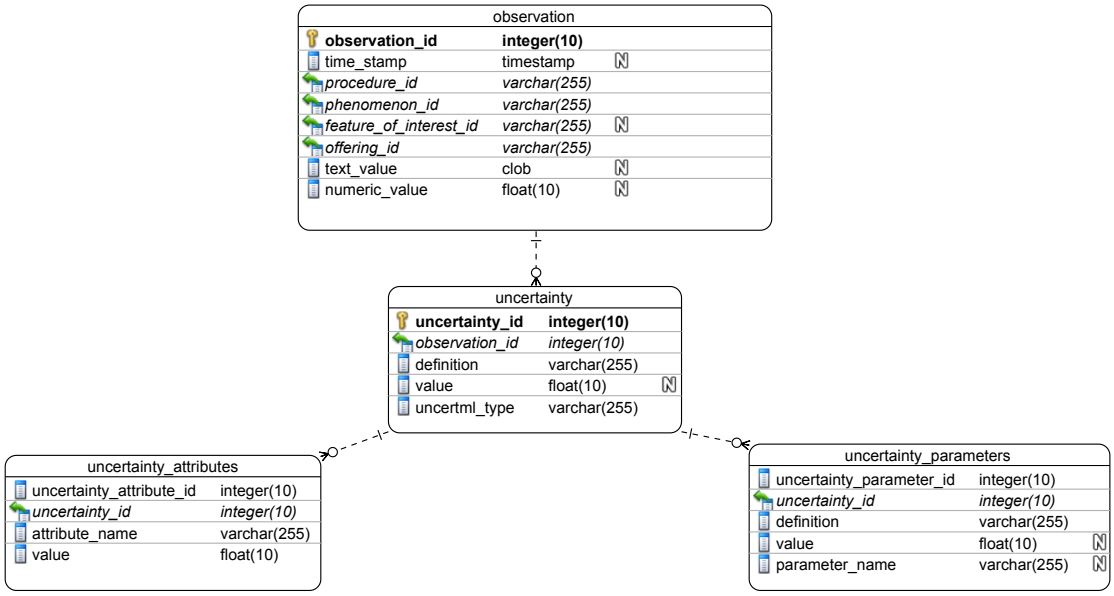



Figure 4.3: Uncertainty within SOS

As discussed in Chapter 3, UncertML has been developed with a weak-typed design philosophy. The extensibility provided by a weak-typed design has to be balanced against some consequent problems faced in implementing software. The first challenge faced when integrating UncertML with 52° North was creating a flexible data model that would allow the large array of different types present in UncertML to be stored efficiently. Due to the structural similarity between many UncertML types a simple data model was formulated. Figure 4.3 displays the complete UncertML model linking to the `observation` table, which can be seen in more detail in Figure 4.2. The relationship between the `observation` and `uncertainty` tables is a many-to-one relationship, i.e. one observation entry can have many uncertainty entries. Each uncertainty entry may only have one observation referenced by the `observation_id` field. In

response to the issue raised in the previous sections about VARCHAR keys, all primary and foreign keys in the UncertML extension are integer values. Individually, the uncertainty table is able to store 'simple' statistics. A simple statistic is defined as a Statistic, within UncertML, that does not contain any additional attributes or parameters, for example a mean or variance. The definition column stores the URL relating to the UncertML definition of that statistic (e.g., http://dictionary.uncertml.org/statistics/mean). The value column stores the value of a particular statistic as a floating point number, while the uncertml_type column informs the UncertML API about the UncertML type which should be used for encoding. For the mean example a value of Statistic should be stored in the uncertml_type column. It should be noted that a better design for the UncertML data model would be to delegate the definition and uncertml_type VARCHAR values to subsidiary tables and reference them through surrogate (integer) keys. However, due to a design choice within the 52° North framework, which is discussed later, a decision was made to implement the data model in a similar fashion to the existing 52° North model.

Particular statistics require more information than is available within the base uncertainty table. For instance, a Quantile within UncertML contains a level attribute. Statistics which require attributes, such as a quantile, need to populate the uncertainty_attributes table. An uncertainty attribute consists of a foreign key linking to the uncertainty table, an attribute name and a value. The attribute name translates to the 'key' section of the key=value pair. The relationship between the uncertainty and uncertainty_attributes table is a many-to-one relationship, allowing any entry in the uncertainty table to contain multiple attributes, if such a scenario should occur. Care should be taken when storing an UncertML Quantile in the data model so that the value of the uncertml_type column is Quantile, otherwise the UncertML API will encode it as a base Statistic.

Storage of distributions, or statistics that require additional parameters, requires use of the uncertainty_parameters table. As with the uncertainty_attributes table, there exists a many-to-one relationship between the uncertainty and uncertainty_parameters table, allowing a distribution or statistic to contain multiple parameters. The structure of the uncertainty_parameters table is also similar to the uncertainty_attributes table, but a simple name is no longer adequate to characterise the parameter so a definition URL must be supplied. When encoding a Distribution in UncertML (Listing 3.12) the base Distribution element has no value property. To reflect this in the data model the value column of the uncertainty table is optional. When storing distributions in the database the value column of the uncertainty

table should be left NULL, instead delegating the values to the individual parameters. For example, a Gaussian distribution has no value *per se*; however, its parameters, location and scale, do. In certain instances one may wish to store a statistic that has a value as well as parameters. In this instance the value column of the uncertainty table may be populated.

While the data model shown in Figure 4.3 is simple in structure, it is flexible enough to allow a wide range of differing statistics and distributions to be stored. This flexible solution, combined with design choices made by 52° North, raises several implementation challenges, which are discussed in the following section.

### 4.4.2 Implementation challenges

In some popular computer design architectures a tiered hierarchy is adopted to provide a separation of concerns. Part of this hierarchy is a layer that separates database access from logic code. In an n-tier architecture this layer is often called the Data Access Layer (DAL) and in a Model View Controller (MVC) architecture the Model layer may sometimes be considered as a data and data access layer. The purpose of this data layer is to provide a simplified interface to the tables contained within an underlying database. The (often complex) SQL commands used to manipulate the database are hidden, allowing logical code to interface with the database without having to contain any SQL. This allows efficient updating of code, should the database structure change.

While the 52° North SOS advocates the use of a 3-tier architecture, the design and implementation of the data layer is overly-complicated and somewhat cumbersome. In a typical DAL a class should exist for each of the database tables and relationships are accessed via method calls. For example, in a SOS, one would expect to find an Observation class, which contains references to FeatureOfInterest and Phenomenon. In such a case a call to observation.getFeatureOfInterest(); would return the feature of interest for the specified observation. Relationships are also expected to work in reverse, so that in the previous example, featureOfInterest.getObservations() would return all observations that refer to that feature of interest. A modular design of this sort allows complex queries to be run on the database with minimal code. Unfortunately, the design of the 52° North SOS does not provide such facilities. There are a number of data objects, but each one refers to a single operation within the SOS. Even in instances where code reuse would be applicable, it is not used. For example, there is a GetObservationDAO class which provides access for the GetObservation operation. There

is also a GetFeatureOfInterestDAO class, which facilitates the GetFeatureOfInterest operation. However, within the GetObservationDAO class, there is code to load in the feature of interest which could, and should, have been delegated to the GetFeatureOfInterestDAO class. Due to this design choice, rather than executing a series of small, manageable SQL statements to retrieve the various constituent parts of an observation, a large, unwieldy, SQL statement is constructed that retrieves all information at once. This results in a series of SQL JOIN conditions which are difficult to maintain and slow to execute.

```
// HashMap mapping observation IDs to processed observations
HashMap<int, Observation> observations;

// Loop through all records returned from SQL statement
while(results.next()) {

   // Get the id of the current observation
   int observation_id = results.getString("observation_id");

   if(observations.containsKey(observation_id)) {
      // This observation already exists in the list,
      // must be a duplicate row

      // Calculate whether the duplication is due
      // to multiple uncertainties or multiple
      // uncertainty parameters
   } else {
      // New observation

      // Insert this observation into the list
      observations.insert(new Observation(observation_id));
   }
}
```

Listing 4.2: A fragment of code taken from the 52° North SOS implementation. The if statement is required due to a lack of well structured code.

A further consequence of this non-modular design is that, for observations containing multiple uncertainty types, or uncertainty types with multiple parameters, duplicate rows of data are returned from the SQL statement. In practice this causes two problems. Firstly, the duplication of VARCHAR data increases the memory throughput unnecessarily. Secondly, it poses a logical challenge in ensuring that the data is processed correctly. Listing 4.2 demonstrates a piece of code that is required to successfully process the result of the GetObservation SQL request. The bulk of the processing occurs inside a while loop, which loops until all rows returned from the SQL statement are processed. In a typical piece of software, this would be one row per observation. However, due to the structure of the SQL statement within 52° North, there are sometimes multiple rows for

a single observation. In such instances an if statement is required to determine whether the current row is a new observation, or a duplicated row. In situations when a duplicate row is found a series of checks (not shown in Listing 4.2) must be carried out to determine what new information has to be processed. For example, if an observation's uncertainty is quantified by a Gaussian distribution, the first row would contain the mean parameter while the second row would contain the variance. Any extra row for a given observation requires a check to determine what the previous uncertainty type was and whether this row represents a new parameter of the previous type, or a new uncertainty type altogether. This is a source of confusion that could have easily been avoided, had a better DAL been implemented.

Despite these obstacles, a working prototype of an UncertML-enhanced 52° North SOS was developed, capable of quantifying sensor-observed data uncertainties via the approaches outlined in this chapter, using UncertML. Providing this level of detailed uncertainty in sensor data allows more reliable processing to be achieved, as demonstrated by the use case below.

### 4.4.3 Use case — Weather Underground

```
<current_observation>
  <location>
    <full>Monkspath, Solihull, West Midlands</full>
    <neighborhood>Monkspath</neighborhood>
    <city>Solihull</city>
    <state>West Midlands</state>
    <zip></zip>
    <latitude>52.390202</latitude>
    <longitude>-1.794913</longitude>
    <elevation>440 ft</elevation>
  </location>
  <station_id>I90579489</station_id>
  <station_type>OS WMR928NX</station_type>
  <observation_time>Sat, 19 December 2009 13:41:10 GMT</observation_time>
  <temp_c>0.6</temp_c>
</current_observation>
```

Listing 4.3: An observation from a sensor within the Weather Underground. The encoding does not conform to a recognised XML Schema standard. This example only displays a small part of the information provided by Weather Underground.

Weather Underground is an online community of weather enthusiasts providing up-to-the-minute information about current weather conditions around the globe. Under the surface lies a vast repository of freely available weather data recorded by thousands of individual weather stations. For this experiment a subset of data gathered from the Weather Underground repositories

was used. Due to the large number of observations, obtained from various types of sensors, the data contained within Weather Underground is perfectly suited for a SOS interface. Unfortunately, the current interface to the data uses a custom XML format that does not adhere to a recognised standard (Listing 4.3). Utilising a 'screen scraping' algorithm, a subset of Weather Underground data was converted into the 52° North SOS data model. Listing 4.4 shows how a Weather Underground observation of temperature can be encoded using O&M, and returned from a SOS instance. Conformance to a recognised standard, such as O&M, allows third-party users to quickly integrate Weather Underground data into their workflow.

```
<om:Measurement gml:id="I90579489_12412">
  <om:samplingTime>
    <gml:TimeInstant xsi:type="gml:TimeInstantType">
      <gml:timePosition>2009-12-19T13:41:10</gml:timePosition>
    </gml:TimeInstant>
  </om:samplingTime>
  <om:procedure xlink:href="urn:ogc:object:feature:Sensor:WU:I90579489"/>
  <om:observedProperty xlink:href="urn:ogc:def:phenomenon:OGC:temperature"/>
  <om:featureOfInterest>
    <sa:SamplingPoint gml:id="monkspath">
      <sa:position>
        <gml:Point>
          <gml:pos>-1.794913 52.390202</gml:pos>
        </gml:Point>
      </sa:position>
    </sa:SamplingPoint>
  </om:featureOfInterest>
  <om:result uom="degC">0.6</om:result>
</om:Measurement>
```

Listing 4.4: The observation from Listing 4.3 encoded using the O&M standard.

The data obtained from Weather Underground is submitted by a range of users, who will apply differing levels of quality control to their data, and site their sensors in a wide variety of locations. Combined with the varying quality of sensing equipment, these influences lead to results which may contain drastically different levels of uncertainty. The data supplied by Weather Underground does not record the information pertaining to such uncertainties — however, the uncertainty may be estimated *a posteriori* with reference to other data sources. In Williams et al. (2011), we used a novel technique to correct the bias of Weather Underground data, using the INTAMAP interpolation service, and thus to provide an estimate of uncertainty. With the estimated uncertainties inserted into the UncertML-enhanced SOS, a request can be made which will retrieve the fully quantified Weather Underground data. Listing 4.5 shows an example output from the SOS, where the uncertainty is quantified as a Gaussian distribution with zero bias (due to bias correction) and

a variance.

```
<om:Measurement gml:id="I90579489_12412">
   <om:samplingTime>
      <gml:TimeInstant xsi:type="gml:TimeInstantType">
         <gml:timePosition>2009-12-19T13:41:10</gml:timePosition>
      </gml:TimeInstant>
   </om:samplingTime>
   <om:procedure xlink:href="urn:ogc:object:feature:Sensor:WU:I90579489"/>
   <om:resultQuality>
      <un:Distribution definition="http://dictionary.uncertml.
org/distributions/gaussian">
         <un:parameters>
            <un:Parameter definition="http://dictionary.uncertml.
org/distributions/gaussian/parameters/mean">
               <un:value>0.0</un:value>
            </un:Parameter>
            <un:Parameter definition="http://dictionary.uncertml.
org/distributions/gaussian/parameters/variance">
               <un:value>0.07595</un:value>
            </un:Parameter>
         </un:parameters>
      </un:Distribution>
   </om:resultQuality>
   <om:observedProperty xlink:href="urn:ogc:def:phenomenon:OGC:temperature"/>
   <om:featureOfInterest>
      <sa:SamplingPoint gml:id="monkspath">
         <sa:position>
            <gml:Point>
               <gml:pos>-1.794913 52.390202</gml:pos>
            </gml:Point>
         </sa:position>
      </sa:SamplingPoint>
   </om:featureOfInterest>
   <om:result uom="degC">0.6</om:result>
</om:Measurement>
```

Listing 4.5: Estimated uncertainty in Weather Underground data can be encoded using UncertML. This example was retrieved from an enhanced version of the 52° North SOS, capable of returning UncerML types.

As sensors become cheaper and people are increasingly connected to the Web it seems likely that user-contributed data will proliferate, and that the collection and use of this data could become a significant part of our environmental monitoring networks. Quality control and uncertainty assessment will therefore be crucial to the effective use of user-contributed data. With its current closed interface, access to and subsequent processing of Weather Underground data is difficult. Providing an open, XML-based, API, like the SOS, opens up this wealth of information for consumption by standards-compliant software.

## 4.5 Conclusions

This chapter extended the ideas discussed in Chapter 3 by integrating the conceptual model of UncertML into existing OGC standards. A thorough review of the SOS revealed that the abstract conceptual model of O&M, on which it is built, caused potential problems with implementation. The broad scope of the SOS, seeking to provide a common API for any sensor observed data, is unrealistic and forces software providers to make compromises. Limiting the scope of the SOS and providing a series of O&M profiles was suggested to alleviate the majority of implementation problems.

An investigation into the current SOS software, focused mainly on the 52° North implementation, examined the methods used to handle the problems caused by the SOS specification. Though it makes several intelligent compromises, the 52° North software was found to be flawed by choices made in the underlying data model and data access layers, discussed in Section 4.3. Despite these flaws, Section 4.4 showed that only a small extension to the data model was needed in order to provide the 52° North SOS with UncertML capabilities. Use of an UncertML API provided the ability to convert seamlessly between native Java objects and the XML form of UncertML. A use case involving Weather Underground data motivated the need for UncertML-enhanced Sensor Observation Services to allow more sophisticated processing and use of data.

With the work in this chapter providing a solid framework, future extensions should become trivial. For instance, extending the OGC Filter specification will allow SOS data to be filtered at source based on data quality requirements, rather than requiring post-hoc data quality checks on the client machine. For instance, data could be filtered based on a probability of exceedance, i.e. only the most critical observations are returned. Alternatively, one may wish to omit data that does not contain any uncertainty quantification.

The work in this chapter addresses research objectives 1 and 2 outlined in Section 1.3.1. Primarily it demonstrates how, by integrating with O&M, UncertML can be used to quantify probabilistic uncertainty in the context of observational errors. Implicitly, the integration with O&M and the SOS strengthens the evidence that the UncertML data model is flexible, usable and interoperable.

# 5 INTAMAP

CONTENTS

## 5.1 Foreword

Chapter 3 outlined a conceptual model and XML schema for representing uncertainty. This was followed in Chapter 4 with a look at how UncertML could be integrated into existing OGC standards, specifically the SOS. INTAMAP is a European Union funded project providing an interoperable service for automatic interpolation. Due to the uncertainties of both inputs and outputs of the interpolation process, a language capable of quantifying these in a web-based framework was required. INTAMAP, can be considered to be the motivation for the development of UncertML. This chapter discusses the INTAMAP project, introducing the OGC WPS specification on which INTAMAP is based, in Section 5.2.

Section 5.3 then provides a history and overview of the INTAMAP project. The early prototype of INTAMAP, based on a WFS framework, is discussed along with reasons for the change to a WPS. One of the key advantages of the WPS approach over the WFS was the ability to provide several input parameters to customise the interpolation request. A list of available input parameters is given at the end of Section 5.3, explaining how a client can tailor their request.

Designing software that is interoperable is not trivial. Implementing a standards-based service does not guarantee interoperability. Section 5.4 provides a detailed discussion about the challenges faced while implementing INTAMAP, why they occurred and possible solutions to the problems. Section 5.4 also provides a critical analysis of UncertML and examines how the hard-typed design of UncertML version 2 will be a beneficial change prior to the adoption of UncertML as a recognised standard.

Finally, Section 5.5 concludes this chapter with a recap of the contributions made.

The INTAMAP project combines the work of several researchers across Europe. The development of the novel interpolation methods used within INTAMAP was undertaken by other partners and consequently is not part of this thesis. However, the investigation into OGC standards, and the development of the WPS interface including the integration of UncertML does form part of this thesis.

The work in this chapter is a collaboration between the partners of the INTAMAP consortium. The contribution of this thesis includes:

- The development of a Web service interface for interpolation.
- The integration of UncertML into the WPS standard.
- The development of an UncertML API.

- The discussion of interoperability and the costs required to achieve it.

The author did not contribute to the development of the novel interpolation techniques used behind the WPS interface.

## 5.2 Web Processing Service

The WPS specification became an OGC standard in 2007. The primary function of the WPS is to provide a standardised interface to allow "the publishing of geospatial processes". A process may include any algorithm, calculation or model that operates on spatially referenced data (OGC 05-007r7, 2007). A secondary function of a WPS is to provide machine-readable binding information in conjunction with human-readable metadata, allowing service discovery and consumption. Unlike many other OGC services, the behaviour of a WPS is not defined by its operations. Because of this a strong emphasis is placed on developing 'application profiles' of WPSs. An application profile is a specific implementation of a WPS that provides a geospatial process. All application profiles consist of a mandatory OGC URN that uniquely identifies the process and a response to a `DescribeProcess` request for that process. An application profile may optionally include a human-readable description of the process (recommended) and a WSDL description.

The flexibility of the WPS specification allows any type of GIS functionality to be provided. Combined with the ability to specify WPS inputs remotely provides a convincing framework for creating a wholly Web-based GIS. In fact, the WPS specification provides methods for handling both vector and raster data allowing processes to range from simple 'point in polygon' algorithms to a global climate model.

The specification allows a service provider to expose a geospatial process, of arbitrary complexity, without exposing clients to the underlying mechanics. Adhering to this standardised interface allows 'naive' clients to integrate complex geospatial processes into their workflow, that were not previously possible. INTAMAP, discussed in Section 5.3, provides a use case for such a scenario by providing complex interpolation algorithms via the WPS interface. However, Section 5.4 discusses the challenges faced by potential INTAMAP users and reviews the practicality of the WPS specification.

### 5.2.1 WPS operations

Unlike the SOS specification, discussed in Chapter 4, the WPS specification has a small set of operations. Specifically, there are three operations, the implementation of which is mandatory for all servers. The operations are:

- `GetCapabilities`
- `DescribeProcess`
- `Execute`

This small number of operations makes the WPS standard appear simple. However, the complexities of the geospatial processes which a WPS provides are hidden behind the `Execute` process. The three processes are described in more detail in the following sections.

#### `GetCapabilities`

Common to all OWSs, the `GetCapabilities` operation allows a potential client to query metadata, or capabilities, of a specific service implementation. For a WPS, the `GetCapabilities` operation returns the names and general (textual) description of all processes offered. For a more detailed description of a particular processes, including the required inputs and supported outputs, a request to the `DescribeProcess` operation is required.

#### `DescribeProcess`

A familiar interaction pattern is noted within several different OGC service standards. The WFS, WMS and SOS all outline the pattern a client should adopt when interfacing with the service. Initially, a client should request the capabilities of a service, via the `GetCapabilities` operation. Once the capabilities have been parsed a client may wish to retrieve more information about a particular part of the service. Such a query is satisfied by a '`Describe`' operation. A WFS provides a '`DescribeFeature`' operation, a SOS exposes a '`DescribeSensor`' operation and a WPS has the '`DescribeProcess`' operation. Following the retrieval of this information the client typically calls the main operation of the service, these are '`GetFeature`', '`GetObservation`' and '`Execute`', respectively.

The `DescribeProcess` operation of a WPS provides a client with a full description of a specified process. The response of the `DescribeProcess` operation includes the input and output parameters, including format, of the process. For example, a capabilities response may indicate that a

'point in polygon' process is available. The result of a `DescribeProcess` request for that process would indicate that it requires two inputs, a point and a polygon, encoded in GML, and returns a boolean value as an output. The WPS specification states that the result of a `DescribeProcess` request "...can be used to automatically build a user interface to capture the parameter values...", however, this is not always possible, as discussed later.

Encoded as either an HTTP POST request with an XML body, or, an HTTP GET request with key-value pairs, the `DescribeProcess` request has few parameters. The mandatory `service` and `request` parameters must be set to '`WPS`' and '`DescribeProcess`', respectively. The `version` parameter can be obtained from the `GetCapabilities` response. The key parameter is `Identifier`, which should contain a unique name of the process of interest. As mentioned previously, a list of available processes, and their names, is contained within the response to a `GetCapabilities` request.

The response to a `DescribeProcess` operation is a `ProcessDescription` document. The `ProcessDescription` document contains the brief metadata (e.g., the process title), present within the capabilities document, but also descriptions of all input and output parameters. Each process may have any number of input and output parameters. The description of each parameter specifies the supported formats and encodings. However, the WPS specification identifies three distinct data types:

**ComplexData** typically used for XML, but often for raster images too.

**LiteralData** is used for simple data types. Each literal parameter may specify a data type (string, integer etc), allowed values, default value and possible units of measure (where applicable).

**BoundingBoxData** specified in a supported coordinate reference system.

The distinction between literal and complex parameters is logical and can be witnessed in other areas of computer science — primitive types and classes in object-oriented programming languages, for instance. However, the decision to include a specific data type for bounding boxes is curious because a bounding box is a complex data type. Perhaps it was perceived to be a commonly used data type, and thus an attempt was made to improve interoperability between geospatial processes by providing explicit types. However, it may have proved useful to extend this idea and provide other common types, for example, points, lines and polygons. Ultimately, the `BoundingBoxData` parameter can be considered unnecessary because a bounding box is complex data. All input parameters must also specify the minimum and maximum occurrences.

While literal data types are self-explanatory (string, integer, boolean etc), complex data types are capable of containing any MIME types, including XML and raster image formats. Each complex input defines which MIME type is expected, and a supported encoding and schema URL may optionally be added if the complex type is an XML fragment. Listing 5.1 is an example of a `DescribeProcess` response, describing a complex input. The example MIME type is `text/XML`, suggesting that an XML document is required as this input. A schema URL is provided, in this instance a reference to the O&M 1.0.0 schema. Employing a generic interface to complex types provides support for any MIME type and XML schema to be used as an input, or output, of a WPS process. This flexibility enables WPS interfaces to be simply wrapped around existing legacy systems. Conversely, however, the flexibility also casts doubt over the previous claim about automatically building an interface based on the result of a `DescribeProcess` request. The most obvious problem arises from the reference to a schema, not a schema element. Taking the O&M schema as a relatively simple example, there are two root elements: `Observation` and `ObservationCollection`. While it is plausible that the input in Listing 5.1 accepts either of those elements, particular scenarios might require that only a single `Observation` is supported. In the WPS specification the distinction between which elements are supported currently resides in the textual descriptions of each input — i.e., not machine-readable. This problem is further compounded when more complex schemas are considered. For example, if a service requires a polygon as an input then a reference to the GML geometry schema might be used. Unfortunately, there are many other elements within the geometry schema that are not suitable. Distinguishing between those elements that are accepted and those that are not, automatically, is not possible. A further difficulty with the automation of generating WPS interfaces is the volume of possible MIME types and XML schemas that may be utilised. Creating an interface that can handle any possible complex data type is impossible. With a certain level of abstraction, an interface may be generated to the level of validating a given input against the specified schema. However, when considering the previous problem of not being able to specify particular elements within a schema, this level of abstraction would prove insufficient for most scenarios.

In addition to the input and output parameters, each process description may support two features. First, a process may support storage of output parameters. This feature proves useful when dealing with large XML payloads or when service chaining. If storage is supported, a client can request that the data be stored on the server, in which case a URL pointing to the output data is provided. The default behaviour of a WPS is to disable storage, in which case all outputs, if

```
<Input minOccurs="0" maxOccurs="1">
   <ComplexData>
      <Default>
         <Format>
            <MimeType>text/XML</MimeType>
            <Schema>http://schemas.opengis.net/om/1.0.0/om.xsd</Schema>
         </Format>
      </Default>
      <Supported>
         <Format>
            <MimeType>text/XML</MimeType>
            <Schema>http://schemas.opengis.net/om/1.0.0/om.xsd</Schema>
         </Format>
      </Supported>
   </ComplexData>
</Input>
```

Listing 5.1: An example ComplexData type input for a WPS.

requested, are returned within the response to an execute operation. The second feature that a process may support is asynchronous processing. When a process requires a substantial amount of time to complete a client may wish to execute the process asynchronously. When a process is executed asynchronously, a response is returned immediately containing a URL to a status document for that process, which should describe the current progress. A client is able to poll this status document and determine the amount of time still remaining. Once the process has completed successfully, the URL points to the execute response that would have been returned via a synchronous request.

This section has discussed the key elements of the `DescribeProcess` request and response. A complete description of these parameters is found in OGC 05-007r7 (2007).

### Execute

The `Execute` operation allows clients to run the processes specified in the `GetCapabilities` response. A valid `Execute` request must specify, via an identifier, the process which should be executed, and any mandatory inputs specified in the `DescribeProcess` response. Reference is also made to the outputs that a client requires. Recognising that geospatial processes often require large data inputs (e.g., raster files or XML fragments) the WPS specification provides a mechanism for reducing client-server bandwidth. Within an execute request the data inputs may be specified via a URL, rather than embedded within the request. This feature proves useful when processing large data files that reside on a central server. Remote access to data is also provided for the outputs of an executed process, discussed previously. The use of remote data stores in a WPS is ideal for

creating service chains with large data payloads.

In a typical service chain, data will be sent from the client to a server. The server processes these inputs and returns some data as outputs. The client then sends these outputs to another service, or process. This pattern can be repeated numerous times. When the data flow is considered, the client must send and receive several, potentially large, data payloads. If, however, the chain is constructed from WPSs, and remote caching is specified, then the flow of data between client and server(s) is hugely reduced. Only the initial input request, with its associated input data, and the ultimate outputs are handled by the client.

The WPS specification does not allow for automatic generation of client interfaces, as mentioned. However, by enforcing geospatial processes to conform to the same framework, a substantial saving on the amount of programming required by a client can be gained. The volume of software tools supporting the more traditional WSDL/SOAP Web services is an attractive alternative for service providers. However, the inclusion of 'out-of-the-box' features such as remote caching of data and asynchronous processing make the WPS perfect for geospatial processing.

### 5.2.2 Uncertainty within a WPS

Unlike the SOS discussed in Chapter 4, the WPS is not built around any particular Web standards. The generic framework defined in the WPS specification allows any MIME type, including XML languages, to be included as both inputs and outputs. The broad range of possibilities presented via the WPS specification means that there are no limitations on how uncertainty can be included. Integrating UncertML into a WPS can be achieved by specifying a `ComplexData` input or output and referring to the UncertML schema. Inclusion of UncertML within another XML schemas, e.g. O&M, is also a possibility. The INTAMAP project, discussed below, utilises UncertML in both of these use cases. An O&M input may contain an UncertML fragment within the `resultQuality` property. Also, one output of the INTAMAP process is a standalone UncertML fragment.

## 5.3 INTAMAP

INTAMAP is a Web service for the automatic interpolation of measured point data. Both the interface and statistical modelling and computation algorithms are built around open standards.

Knowledge of the current state of an environmental system is often critical to decision making, for example in contexts such as disaster response, public health protection or routine environmental monitoring and management. This knowledge of the state of a system is obtained from obser-

vations. However, the process of observing the system of interest can be costly, both financially and temporally. When making decisions at locations where observations do not exist, predictions must be made. This process of prediction is often carried out through interpolation between the existing observations. Spatial interpolation can be used for a wide variety of situations including meteorological, air quality or environmental radioactivity variables. Unfortunately, no single simple solution exists. A recent experiment examined the methods used by several experts to interpolate the same data set (Dubois and Galmarini, 2005). The results showed that the methods varied drastically, with no single solution proving greatly superior in all aspects. The lack of a generally accepted interpolation method has culminated in domain-specific experts developing tightly-coupled, specialised, tools. This means that certain domains, where interpolation would prove useful, do not utilise the most advanced methods. This can be attributed to the large overheads of gaining the necessary expertise, so that the methods used are sometimes too simplistic. INTAMAP provides a solution by automating the process of interpolation and allowing access to the automated process through an interoperable interface. Several novel geostatistical methods are housed on backend servers and accessed through an OGC WPS, discussed in detail in Section 5.3.1.

### 5.3.1 The INTAMAP interface

Accessibility is a key ethos of the INTAMAP project. Providing a simple interface on top of complex interpolation algorithms maximises the number and range of potential clients. Hiding this complexity is achieved by automating, wherever possible, the decisions needed during interpolation. However, to ensure that the INTAMAP service remains useful to more experienced clients, optional full control over the interpolation process was also a requirement. Interoperability is fundamental to a cross-domain service and, consequently, the latest open standards were adopted. Interpolation, and geostatistics in the broader sense, are geospatial processes and, as mentioned in Chapter 2, the OGC are the current pioneers in Web-based standards for geospatial data. Implementing OGC standards ensures that the INTAMAP service can be efficiently integrated into existing OGC compliant software.

The INTAMAP interface underwent several revisions during the project life cycle. The following sections provide an overview of the various iterations of the interface, highlighting the advantages and disadvantages.

**WFS interface**

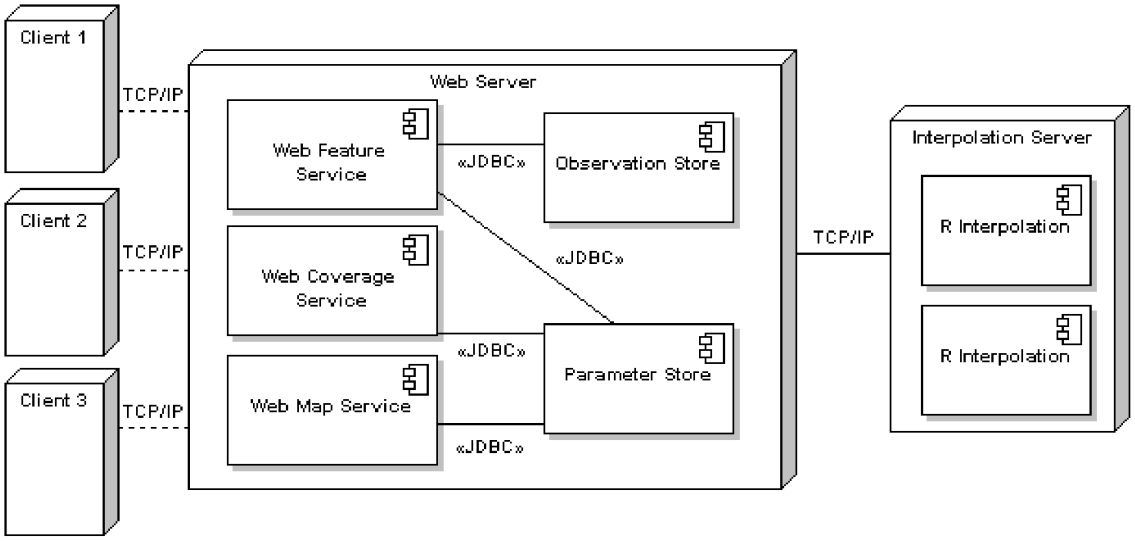



Figure 5.1: Architectural overview of the initial INTAMAP prototype. A WFS interface provided access to the underlying interpolation algorithms.

Prior to the release of the WPS as an OGC standard INTAMAP was designed around a WFS interface (Figure 5.1). The WFS specification defines interfaces for describing data manipulation operations of geographic features, including:

- `Get` or query features based on spatial or non-spatial constraints (using the OGC Filter encoding).
- `Create` features.
- `Delete` features.
- `Update` features.

These operations, from a transactional WFS, are synonymous with the classic Create, Read, Update, Delete (CRUD) methods of persistent storage. However, the operations of a WFS are limited to features, usually encoded in GML. Since WFS operations were tailored towards a feature-store interface, implementing an interpolation process with it posed several problems. The fundamental issue was that no method provides the necessary parameters to input the observations while at the same time returning an interpolation result. For example, the `GetFeature` operation allows a feature, possibly a map generated as a result of interpolation, to be returned — but does not provide the ability to include input parameters. Conversely, the `InsertFeature` operation allows observations to be sent to the server, but does not provide the mechanism to return a generated map. The solution was to utilise both methods (Williams et al., 2007).

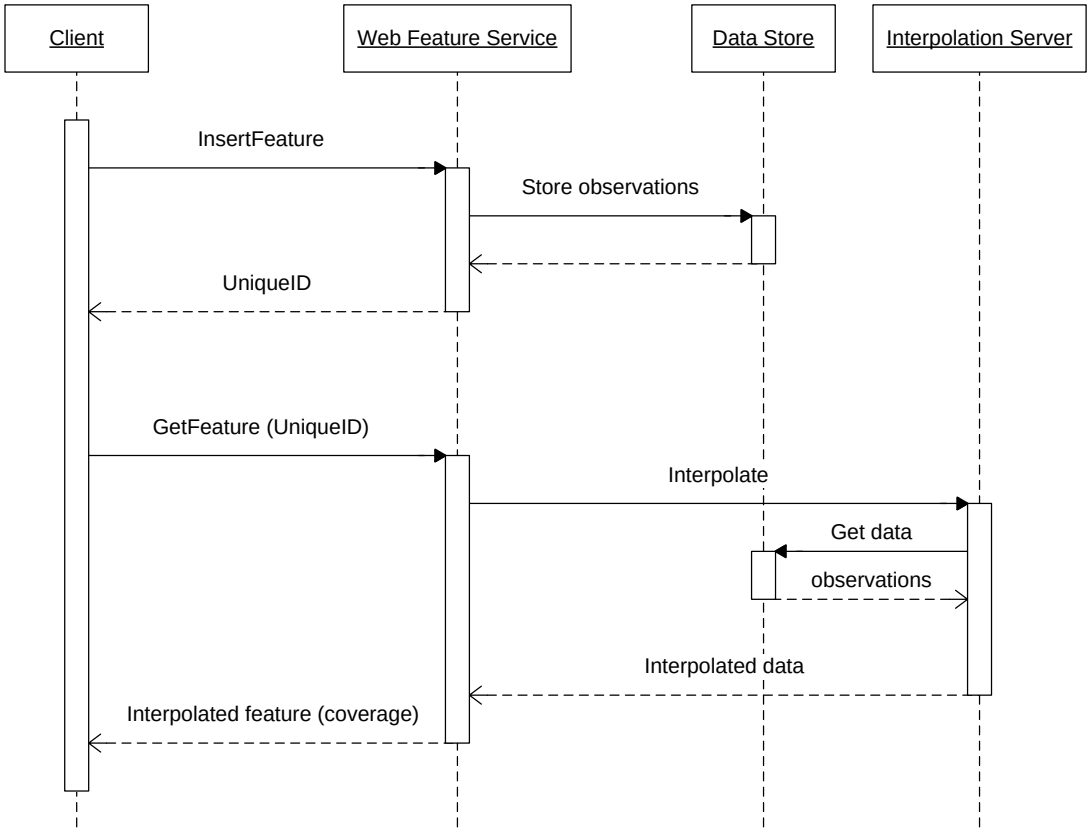



Figure 5.2: Sequence diagram showing how an interpolation request was made to INTAMAP via a WFS interface.

To initiate an interpolation request via INTAMAP a client was first required to supply the observation data via the `InsertFeature` operation. Initially, the observations were encoded using the GML observation schema, but this was later superseded by the O&M model. INTAMAP inserted these observations into a PostGIS database and returned a unique identifier for that particular data set to the client. An interpolation request could then be made via the `GetFeature` operation. Requests were made for a `RectifiedGrid` feature (included in GML) and, using the Filter encoding, constraints were placed on this grid. Typically, these constraints were the bounding box of the grid and the coarseness, or resolution, desired. At this stage INTAMAP initiated the interpolation process, retrieving the observations from the database. A sequence diagram of the complete interpolation process can be seen in Figure 5.2.

Despite producing a working system, the WFS based server had many problems. The use of the `InsertFeature` operation necessitated the storage of observations in a database. This created several issues. Firstly, there was no recommended storage time for inserted observations. Once a client had called the `InsertFeature` operation, they were not obliged to immediately proceed with the `GetFeature` request. This resulted in a database that could rapidly increase in size. Another issue with persistent storage of observations was data protection. The observations submitted to INTAMAP might not have been publicly available, and users might not wish to have them stored in a database. Perhaps a more fundamental problem was one of semantics. INTAMAP was using the WFS interface in a manner for which it was not intended. Using the `InsertFeature` and `GetFeature` operations to form a composite operation caused issues. For example, a typical WFS can describe all the features available to query via the `GetCapabilities` and `DescribeFeature` operations. However, with INTAMAP the `GetCapabilities` response would only describe the observations that had been inserted, not the interpolated results. The reason for this is that the service does not store the interpolation results, and thus in any audit of resources they don't exist. Only once a client makes a `GetFeature` request is an interpolation executed, with the result being returned directly to the client. Furthermore, interpolation can be a lengthy process whereas a typical `GetFeature` request should return instantaneously. All the problems with the WFS implementation produced a 'square peg in a round hole' implementation; a neater solution was required.

### Web Processing Service interface

2007 saw the public release of version 1.0.0 of the WPS as an official OGC standard. The interface provided by the WPS, discussed in Section 5.2, is far better suited to geospatial processes such as interpolation. The ability to supply input parameters with an `Execute` request allows the removal of the two-stage process of the WFS (Figure 5.3). The removal of the `InsertFeature` stage also allows INTAMAP to function without a persistent data store, thus alleviating some issues present in the WFS prototype. A further benefit of the WPS system is that there is no reliance on the GML standard. While the WFS specification is tailored towards the GML feature model, the WPS specification remains generic. Removing the ties to GML allows INTAMAP to use specifications such as O&M and UncertML without any extensions to the service standard.

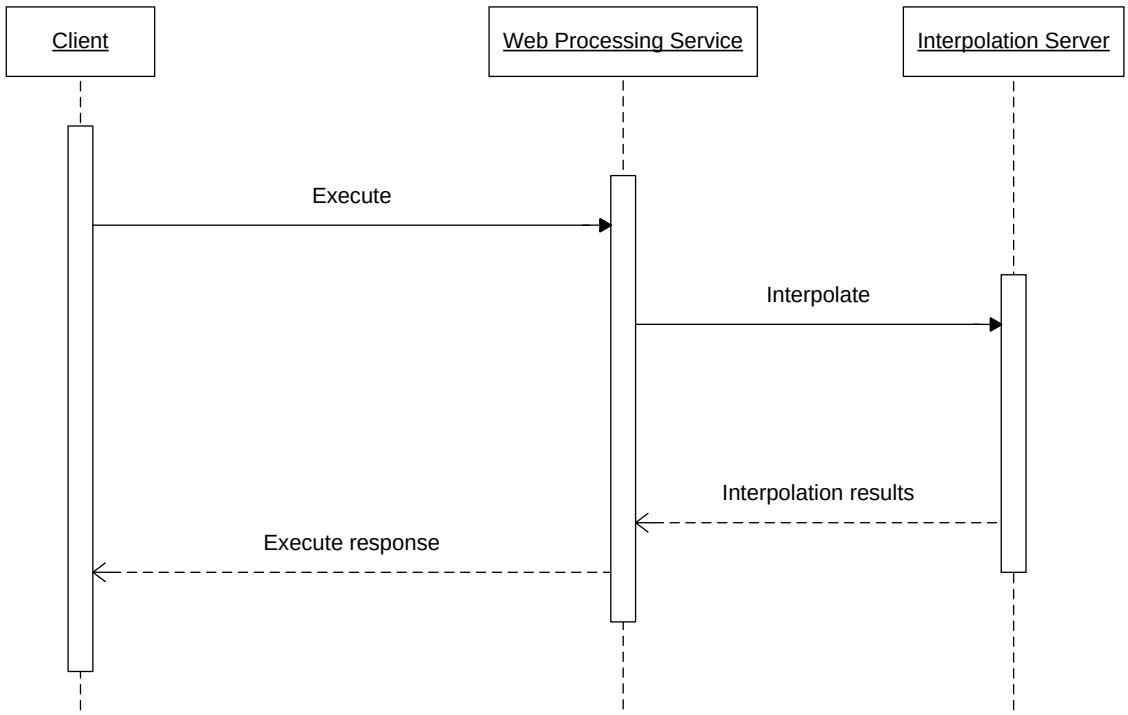



Figure 5.3: Sequence diagram showing how an interpolation request was made to INTAMAP via a WPS interface.

Advanced features of the WPS are also relevant to INTAMAP. When interpolating large datasets, the O&M document required can become large. The remote caching ability of the WPS specification provides a method of reducing bandwidth between client and server. This proves particularly useful when mobile clients interact with INTAMAP, as the large data sets can be stored on a central server. The ability to call the `Execute` operation asynchronously also provides a benefit for INTAMAP. Some of the geostatistical methods implemented within INTAMAP require a substantial amount of processing time, especially on larger datasets. Executing these

time consuming processes asynchronously enables the client to resume operation immediately.

The advantages of using a WPS over a WFS for INTAMAP are numerous. A decision to migrate to a WPS interface was made.

### Inputs

A requirement of INTAMAP was that it should be completely automatic. This resulted in only a single mandatory input parameter — the observations which should be used to make the predictions. However, it had to remain flexible enough to allow experienced users to control the interpolation process efficiently if required. The INTAMAP WPS contains many input parameters, allowing a substantial amount of customisation. This section outlines the input parameters, detailing the effect they have upon the interpolation result. Given the use of generic schemas (e.g., O&M) concessions were made to facilitate a workable system. Details are given where certain restrictions have been made to an underlying schema.

**ObservationCollection** This is the only mandatory input. Its contents should be an O&M `ObservationCollection` type, containing all observations for use within the interpolation. It should be stated here that the minimum number of observations required by INTAMAP is 20; any fewer and a sensible result cannot be obtained. As discussed in Chapter 2, the O&M observation model contains several properties. A few of these are not used by INTAMAP and are ignored upon parsing. However, those properties that are required may also contain restrictions on their use. Below is a list of properties that INTAMAP uses, how they are used and the restrictions which are enforced.

**featureOfInterest** INTAMAP uses this as the location of the observation. O&M specifies that any GML feature can reside here. Unfortunately, reliable parsing of any GML feature, to extract the location, is not currently possible and consequently only a single feature is allowed within INTAMAP. A `SamplingPoint` feature, defined in OGC 07-002r3 (2007), contains a GML `Point` geometry, adequate for use within INTAMAP. All other features within the `featureOfInterest` are ignored by INTAMAP.

**resultQuality** INTAMAP is able to parse UncertML documents. If the error of the observation is known it can be encoded as UncertML and embedded within the `resultQuality` property. Typically, observation errors are quantified as normally distributed.

**result** According to the O&M specification this can be any XML type, although INTAMAP only accepts a single real value. The `Measurement` model is a restricted `Observation` limiting the result property to a single real value within the schema. `Measurement` encodings are therefore preferred within INTAMAP.

**Domain** In this context, the term domain refers to the prediction locations, i.e. the point(s) at which INTAMAP should predict. GML geometry types are utilised for this parameter. The types currently accepted by the INTAMAP service are listed below:

- `Point` — predict at a single location.
- `Polygon` — predict the average and other summaries within this polygon.
- `MultiPoint` — predict at multiple point locations
- `MultiPolygon` — predict within multiple polygons.
- `RectifiedGrid` — predict at every location within a regular grid.

Specification of the prediction locations is critical to most applications. However, within INTAMAP, this parameter is optional. When the `Domain` parameter is omitted, a regular grid that encloses all provided observations with a resolution of 100 x 100 cells is created. This is an example of how the automated nature of INTAMAP can be overridden by experienced users, if desired.

**PredictionType** The result of an interpolation is typically an estimated value and a measure of uncertainty, or, prediction variance. However, it is possible to request a number of alternative statistics by supplying a relevant UncertML fragment. Below is a list of supported statistics:

- `Mean`
- `Variance`
- `Probability` (of exceeding a specified value)
- `Quantile`

If no `PredictionType` is supplied a default assumption is made and the mean and variance are returned. Any combination of the above statistics may be requested, in which case a value is returned for each one, at each prediction location.

**MethodName** As mentioned previously, there are a number of interpolation methods available within INTAMAP. If no method is specifically requested, one is chosen based on some heuristics applied to the supplied dataset. However, if a client believes that a particular method is more suitable it can be selected via this parameter. Unlike the previous parameters, the method name is a `LiteralData` type, accepting a string which may be one of the following values:

- `psgp`
- `copula`
- `automap`
- `idw`
- `automatic`

**SOSURL** An advantage of using a WPS over WFS is that the O&M model can be used. A direct consequence of this is that input can be taken from a SOS instance, rather than being embedded within the request. This parameter is a string which is a URL reference to a SOS instance. If this URL is a valid SOS GET request it may be used alone, however, if a POST request is required, this URL is used in conjunction with the `SOSRequest` parameter, below. The returned observations will be parsed and used within the interpolation. SOS observations can be combined with other observations, described in the `ObservationCollection` parameter.

It should be noted that only observations that conform to the restrictions outlined above are accepted. This is helped by requesting that the SOS returns O&M measurements, rather than observations. Care should also be taken to ensure that no two observations occur at the same location, as this causes exceptions during interpolation. If an UncertML-enhanced SOS is being used, as outlined in Chapter 4, INTAMAP will use any observation errors present.

**SOSRequest** This parameter can only be used in conjunction with the `SOSURL` parameter above. This parameter contains an XML fragment that should validate against the SOS `GetObservations` schema. The contents of this parameter are sent to the URL specified in the `SOSURL` parameter. The result from the SOS is parsed and the observations are merged with any observations specified in the `ObservationCollection` parameter.

**maximumTime** This parameter is a literal integer value specifying the maximum time allowed for processing, in seconds. INTAMAP uses this information to select a suitable method. If no method can complete within the allowed time an exception is raised.

#### Outputs

There are a number of outputs generated by the WPS, customised by the supplied inputs. Below is a description of all available outputs.

**PredictedValues** The actual result of the interpolation. The contents of this parameter is an UncertML `StatisticArray` element. Within the values section is a result for each statistic specified in the `PredictionType` input, at every location specified in the domain input.

**Domain** Although in many use cases the client is aware of the prediction locations, in certain instances the prediction locations are generated automatically by INTAMAP. For instance, when no `Domain` input is supplied a regular grid is constructed, or, if a polygon domain is specified, INTAMAP generates regular points within that polygon. In these cases it is imperative that a client can retrieve the locations for which predictions were made, and thus relate to the values contained within the `PredictedValues` output.

**PredictionReport** Often a client using interpolated data in decision making wishes to know exactly what processing was applied to their data. The `PredictionReport` output supplies a user with a structured string, listing all processing steps performed. This enables clients to assess whether the results can be used reliably in decision making.

### 5.3.2 Example INTAMAP request

Figure 5.3 gives an overview of the INTAMAP architecture and how a request is processed by the two-tier architecture. However, it does not provide a detailed overview of the different components that are executed in order to process a request. This section discusses an example request, demonstrating how the INTAMAP service decomposes a request and then encodes the response.

A more detailed sequence diagram can be seen in Figure 5.4. This diagram contains five constituent objects: the client, WPS interface, O&M and UncertML APIs and an interpolation server. The shaded objects (the WPS interface and APIs) represent work that was carried out as part of this thesis, the other two objects (client and interpolation server) were developed by third-parties. The client object represents any INTAMAP client, whether this is a web application, a desktop application or browser plugin. All INTAMAP clients initiate a request via the same method — sending a WPS `Execute` request (Listing 5.2) document to the INTAMAP URL. The request document contains all the information required to perform an interpolation request, encoded in XML,

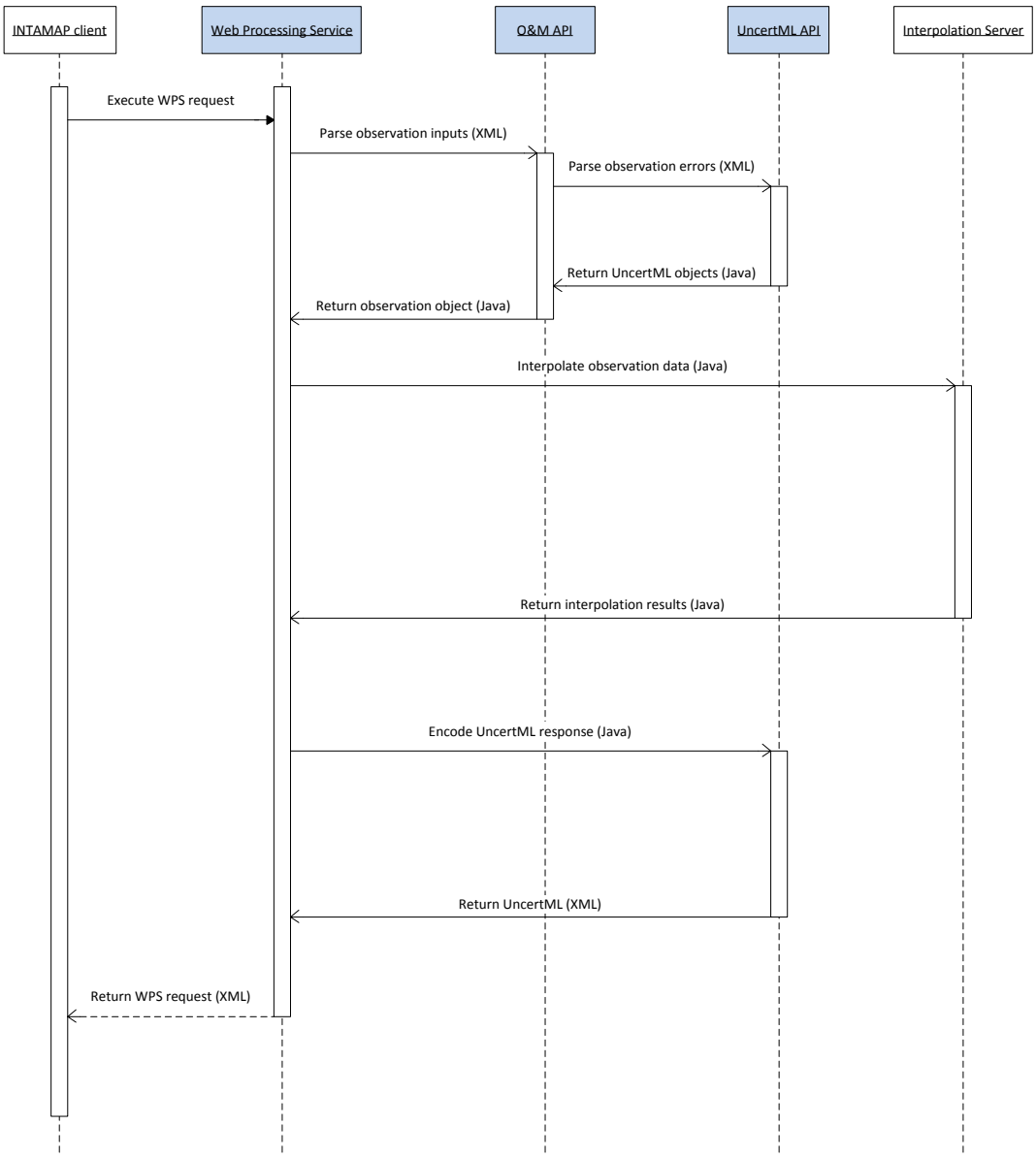



Figure 5.4: A sequence diagram showing the interaction between the WPS, APIs and interpolation server. Shaded boxes are contributions to this thesis.

```
<Execute service="WPS" version="1.0.0">
  <Identifier>org.intamap.wps.Interpolate</Identifier>
  <DataInputs>
    <Input>
      <Identifier>ObservationCollection</Identifier>
      <Data>
        <ComplexData>
        <!-- O&M observations -->
      </ComplexData>
      </Data>
    </Input>
    <!-- Other inputs go here -->
  </DataInputs>
  <!-- Response options go here -->
</Execute>
```

Listing 5.2: A truncated example of an INTAMAP request. Several options have been removed for brevity.

these inputs can be seen in Section 5.3.1. Once the WPS receives the request document, it uses the developed O&M API to extract the observation information and convert it into native objects. Typically, the information of interest is the location of the observation ($X$ and $y$) and the observed result ($z$). However, if the observation has an attached error, encoded in UncertML, the O&M API delegates the parsing to the UncertML API (the fourth object in Figure 5.4). This interaction can be seen in the Java code in Listing 5.3. The INTAMAP service also contains code for parsing the other inputs to the service, including GML, however, these are minor contributions.

```
private Uncertainty parseResultQuality(Element root) throws IOException {
    // Get the child node of the resultQuality element.
    // A check has previously been done to ensure one exists.
    Element uncertml = root.getChild("resultQuality", Namespaces.OM).getChild();
    UncertMLParser parser = new UncertMLParser();
    return parser.parse(uncertml);
}
```

Listing 5.3: A Java code snippet from the O&M API that utilises the UncertML API to parse an uncertainty element.

Once all inputs have been parsed into native Java objects, the INTAMAP WPS communicates via TCP/IP with the interpolation server to request an interpolation. The interpolation is handled by the statistical language, R. The interpolation algorithms were not developed as part of this thesis, however, more information can be found in Williams et al. (2011). The result of an interpolation is a series of arrays containing the predicted values at each location. These values are passed back to the INTAMAP WPS which in turn passes them to the UncertML API. Due to

a requirement of the INTAMAP project, all interpolation requests return a mean and variance at each prediction location. Therefore, the UncertML API always converts the provided values to a `StatisticArray` of `StatisticsRecord` containing both mean and variance values. These values, now encoded in XML are sent back to the INTAMAP WPS and wrapped with the remaining outputs that were requested by the client and returned. The whole interpolation process typically takes twenty seconds to execute.


Figure 5.5: A screenshot of a web application that simplifies the INTAMAP WPS interpolation process. The image displayed is an interpolation of $NO_2$ data over the United Kingdom.

Once the client retrieves the result from the INTAMAP server the results can be processed in a number of ways. A typical use case is for the client to display the mean values overlaid with the observation data, usually with some background mapping. An example of this, generated as part of the INTAMAP project can be seen in Figure 5.5, which uses Google Earth as the background mapping. The benefit that UncertML brings to this client, and the numerous other clients discussed in Section 5.4.1 is that the meaning of the returned data is explicitly described. For instance, the image in Figure 5.5 shows the mean level of $NO_2$ over the United Kingdom. Without UncertML the meaning of this data is lost and it becomes nothing more than a static image. The quality of the inter-operation of UncertML with INTAMAP can be inferred from the success of the service and by the number of supporting clients that have been developed (Section 5.4.1).

## 5.4 Interoperability review

Interoperability was defined in Chapter 2 as "the ability to exchange and use information". Putting this broad definition into the context of computer systems can refer to the ability of two machines to communicate while remaining hardware, platform and language independent. The recent popularity of XML and Web services has proven that interoperability is a growing trend, yet it is difficult to measure how 'interoperable' a service is. This section provides a review of the INTAMAP service, investigating how interoperability has been achieved, and at what cost.

### 5.4.1 Open standards, good or bad?

A key requirement of INTAMAP was to produce an easily accessible interface to the underlying, complex interpolation algorithms. In order to achieve this requirement, a series of recognised open standards were adopted. The core of INTAMAP revolves around the O&M standard, but also uses GML, SWE Common and SensorML. A further XML standard, UncertML, is also used extensively. The service itself is constructed using the WPS specification, discussed above. The combination of all these XML based standards certainly allows multiple hardware, platform and language independent devices to communicate with INTAMAP. By that measurement, one could assess that INTAMAP is indeed interoperable. However, the use of so many open standards also generates difficulties when creating supporting software.

Another important measurement of how interoperable INTAMAP is might be how easily it can be integrated into existing software. It is here that the double-edged sword of utilising the OGC open standards becomes apparent. Currently very few tools exist to aid software designers with OGC standards. For instance, no API exists for parsing and creating valid O&M documents. The reasons for this are likely to be because of the problems discussed in Chapter 2 about the abstract nature of O&M. It should be stressed that O&M is a conceptual model and thus relies on the creation and adoption of application profiles. However, with the lack of any official O&M profiles, the absence of an API is not surprising. This lack of a definitive API for O&M leads software designers to create their own, based on their interpretations of the O&M specification. Paradoxically, the differences between software designers' perceptions of O&M can result in APIs that cannot parse valid O&M generated by an other. For example, two designers may require an O&M profile whose result is an array of floating point values. One designer may choose to encode this as a series of comma separated values, while the other adopts the SWE Common standard (Section 2.5.3). Both of these profiles conform to the O&M model and both encode the same data,

yet they are syntactically different and thus may not be 'interoperable'. One could argue that, given the openness of O&M, a sufficient API can never exist. This same problem exists for other OGC standards, but is less prominent. Ultimately this means that the overheads of a software designer integrating INTAMAP into their existing software are potentially costly. In fact, for INTAMAP to be consumed a developer must create APIs for parsing O&M, GML and WPS requests/responses, even before they start to investigate the various input parameters of INTAMAP. Had INTAMAP created a set of heavily restricted profiles of the standards instead, many of the implementation strains could have been relieved. However, when using profiles so specific to a particular project, it raises doubt about the benefits of adhering to the overruling conceptual standards. For instance, it is not guaranteed that an existing piece of O&M software would be able to process the INTAMAP profile without alterations. In such an example the benefits of conforming to the O&M model are not clear.

The effort required by INTAMAP clients was noticed during the development stage. Several approaches to alleviate the problem were attempted.

#### SimpleInterpolate

Devised particularly with mobile INTAMAP clients in mind, `SimpleInterpolate` was developed as a middleware client to INTAMAP. Based around the WPS specification, it provided an `interpolate` process. However, rather than relying on the use of O&M, GML and UncertML as inputs and outputs, `SimpleInterpolate` uses Comma Separated Values (CSV) and bitmap images. The bandwidth required by the classic INTAMAP server for a typical interpolate request (10,000 prediction locations) is large. The motivation, then, was to create a service that was still 'interoperable' but used more concise inputs. The simplicity provided by the `SimpleInterpolate` interface became immediately popular and it was adopted by a number of users. However, there are several disadvantages to the `SimpleInterpolate` service. The limitations placed on it by the CSV inputs mean that a client cannot specify many types of observation error, or sensor models. Indeed, the CSV inputs are limited to a `x,y,z,s` format where `x` and `y` are coordinates, `z` is the observed value and `s` is a standard deviation. The expressive capabilities provided by the OGC standards have been lost. The same applies to the output. While many clients simply wish to display the result as an image, usually overlaying a digital map, many wish to analyse the results for decision making. SimpleInterpolate provides output as raster formats (PNG, GIF, JPEG etc) but also the more expressive GeoTIFF format. It does not, however, provide a fully

quantified representation of the uncertainty generated by INTAMAP. This can only be achieved using UncertML.

#### WCS/WMS interaction

While software supporting OGC standards such as O&M and WPS is scarce, there are other standards which have, arguably, seen more support. Two such OGC services are the WMS and the WCS. The WMS serves features as images, while the WCS serves coverages, defined by their domain and range or results. A result of an `interpolate` request can be considered as an image (as described above) but also as a coverage. This led to the creation of an extension to the classic INTAMAP service. The idea is similar to that of `SimpleInterpolate`, but it does not lose the expressive capabilities of the input parameters. A request is made to INTAMAP in exactly the same way as the original INTAMAP WPS. The extension stores the interpolation result within a WCS or WMS server, generates a URL containing a valid HTTP GET request for the newly created 'feature' and returns that with the INTAMAP response. This allows clients to use existing WMS/WCS compliant software to parse and consume the results generated by INTAMAP.

Unfortunately, several problems exist with the INTAMAP extension that hindered its adoption. Primarily the extension was written with a specific WMS/WCS server implementation that was only available for Windows servers. Also, clients can't solely rely on the WMS/WCS software to interface with INTAMAP. An INTAMAP request still needs to be constructed, which relies on an O&M and WPS API as a minimum but would benefit from a GML API also. This means that clients willing to put the effort into generating an INTAMAP request, may as well create the software for parsing a response. This would allow them to use the full capabilities of INTAMAP. The disadvantages with both this approach, and the `SimpleInterpolate` approach underpinned the need for a smarter solution.

#### INTAMAP API

With potential users discouraged by the overhead of integrating with INTAMAP, an accompanying API was produced. Developed in Java, the INTAMAP API implemented basic parsers and generators for O&M, WPS, and GML. It also interfaced with the UncertML API. Within the API a client can create an instance of a Java class representing an interpolation request. All the input parameters can then be modified using native Java method calls. Eventually, an `interpolate` method can be called which constructs the XML from the supplied parameters and sends it to the

INTAMAP server. Once the API receives a response, it parses the XML to construct a Java class representation. Effectively, this allows a client to interface with INTAMAP through native Java code, without even having to understand the underlying XML messages. The benefits of this are huge, since any Java certified developer could integrate INTAMAP into their Java application with ease.

There are particular disadvantages to the API. For instance, it is a Java implementation, and thus might not be compatible with existing software written in another language. Also, a limited model of O&M, GML and other supporting languages is used — enough to satisfy an INTAMAP request/response. This may cause friction if a client has another model of O&M, in which case the INTAMAP API might not operate correctly.

The creation and use of a common API does raise one question though. If every client deems it too much effort to create their own software and would rather use the provided API, is it necessary to use so many open standards? For example, if everyone uses the Java API then the underlying code could easily communicate via binary, which would be much more efficient than converting everything to and from XML. This idea can be extended by suggesting that it is not the open standards that make a service interoperable, but the software that ultimately supports it. However, there is one, major, benefit of implementing an interface using open standards. The possibility exists for other software to be developed by unrelated clients, should the need arise.

The concept of interoperability is certainly useful. Achieving it, however, is extremely difficult. This section identified that in one respect, INTAMAP could be conceived as 'interoperable' — it provides an open interface, independent from any such machine restrictions. However, it transpires that implementing software to communicate with INTAMAP is not trivial. On one level, the idea of interoperability is for machines to communicate with one another. Ironically, if the standards adopted to achieve this are too complex, creating software that communicates could be deemed too difficult. The issues faced during the INTAMAP project underlined the difficulty of creating interoperable software. In fact, a separate 'INTAMAP' system was created devoid of any XML standards or services and exists simply as a piece of software, written in the statistical language 'R'. One way of simplifying the process may be to create restricted profiles of the abstract standards. Creating profiles of standards, restricted to a manageable level, would enable software that was 'standards-compliant' to be created more efficiently, reliably and quickly.

### 5.4.2 UncertML and weak-typed schema

Chapter 2 introduced the concepts of strong and weak-typed schema design. Chapter 3 introduced UncertML, an XML schema to quantify uncertainty. The approach taken with UncertML is a weak-typed schema, providing flexibility. The problem with weak-typed designs is that writing compliant, usable, software is impossible. The flexibility provided by the weak-typed design means that there are no clearly defined boundaries or scope. Within UncertML, for instance, a probability distribution is encoded via a definition, providing the necessary semantics, and a series of parameters (each with their own definitions (Chapter 3)). Theoretically, this allows any distribution to be encoded with UncertML. It is entirely possible that a user describes a distribution that only they are aware of. Providing software capable of parsing any distribution, at a high level, is trivial. The current UncertML API is capable of doing just that. However, these distributions contain little or no meaning. Parsing the XML into programming language objects does not provide the necessary information to meaningfully process these distributions. For example, the UncertML parser can parse a distribution with the definition `urn:uncertml:distribution:gaussian`, but to use it appropriately requires another level of detail.

The solution to the problem is simple. UncertML defines all possible statistics and distributions as hard-typed elements. Providing a definitive list of supported elements gives software a clear scope. Once boundaries have been established, hard-typed classes can be created within the API. Rather than a generic `Distribution` class, capable of representing any distribution but with little processing ability, a series of distribution classes would exist. For example, a `GaussianDistribution` class would be able to process its parameters to good effect, such as generating realisations or exceedance probabilities. This is not simple with a generic framework as it relies on comparisons between URIs, which may differ between users. Another indirect benefit is that by establishing an explicit scope, software implementing UncertML can claim to be 'UncertML compliant'. While this may seem trivial, it allows two UncertML compliant pieces of software to communicate effectively. With a generic framework this is not possible. For instance, another piece of software could send a perfectly valid UncertML fragment that another piece of software has not encountered before and consequently is unable to process.

Hard-typed schemas are plausible for domains where a small, well-defined, set of features exist. Unfortunately, with statistics and probability distributions this is not so easily achieved. There are a huge set of probability distributions that could be defined in UncertML and a hard-typed schema of that size suffers from the same problems as a weak-typed design. Fortunately,

there are a relatively small number of distributions and statistics that are more commonly used than others; for instance, the statistics 'mean', 'variance' and 'standard deviation', or a Gaussian distribution. Although limiting UncertML to this small subset of features is too restricting, a compromise may be found through the use of profiles. Following the OGC's example of the GML simple feature profile (OGC 06-049r1, 2006), a similar design could be utilised with UncertML, maintaining the existing, weak-typed, statistics described in Chapter 3 but complementing them with a collection of hard-typed, commonly used statistics and distributions. Software compliant with this UncertML 'simple feature profile' could then communicate reliably, yet the flexibility of UncertML remains.

The problems faced by software designers due to extensible, weak-typed, XML schemas are great. This is emphasised by the challenges faced in INTAMAP and the UncertML API. To tackle these problems and provide the community with more effective software, thus increasing the uptake of open standards, an effort has to be made. Focus should be placed on creating smaller, discrete, independent profiles of the large, flexible, standards. These profiles should fall under the governance of the standards organisation committee, and not the software providers. The creation of manageable profiles will allow software designers to focus on creating the APIs that are needed to see wide-scale adoption of OGC standards.

## 5.5 Conclusions

The work carried out in Chapter 4 demonstrated the ease with which UncertML can be integrated into existing standards-based software. The work in this chapter continues the trend by introducing the INTAMAP project. Developed as an "interoperable service for automated mapping", INTAMAP was the primary motivation for the development of UncertML.

After an initial prototype based on a WFS, INTAMAP was built using the OGC WPS standard. Section 5.2 provided an overview of this standard, highlighting some particular advantages over other service based architectures. Particular attention was paid to the remote caching of data and asynchronous processing, both of which are perfectly suited to the interpolation algorithms found within INTAMAP. Section 5.3 gave a history of the INTAMAP project. While the early, WFS-based, prototypes did produce working systems, there were many problems, in particular, the reliance on a two stage process. During 2007, the OGC released version 1.0.0 of the WPS and the potential advantages that could be gained by its use in INTAMAP were recognised. Migration to the new WPS based system provided users with more functionality. A clear benefit of the WPS

over WFS was the ability to pass numerous input parameters with the execute request, allowing a user to customise an interpolation request. The end of Section 5.3 provides a list of parameters accepted by INTAMAP and how they can be used. One parameter of importance was the ability to provide a SOS request. This ability is enhanced by the fact that INTAMAP can parse UncertML. If a user has implemented an UncertML-enhanced SOS, detailed in Chapter 4, they can propagate those errors through INTAMAP directly.

The chapter finished with a discussion of interoperability, and what it means to be 'interoperable'. It argued that providing an open standards-based service is often not enough to ensure interoperability. During the development of INTAMAP, several users found it difficult to create the necessary software to process the number of different standards used. This led to several methods to try and alleviate the strain. One method was to create a simpler interface to the interpolations using comma separated values and bitmap images. However, the flexibility lost for the sake of simplicity was a great disadvantage. A Java API was actually most efficient as it provided users with the full capabilities of INTAMAP but with a much simpler integration process.

An overview of weak-typed schema designs and the problems that they cause to users was given in Section 5.4.2. Making reference to UncertML, a strong case for hard-typed designs was forged. However, acknowledging that hard-typed designs are not always possible, a recommendation to standards organisation committees was made. The development of simple, atomic profiles of larger, extensible standards facilitates growth in available software. A suite of software APIs to accompany the OGC standards would enable domain users to adopt them with greater ease and thus increase cross-domain interoperability.

The research in this chapter found several notable disadvantages to weak-typed XML schemas that hinder their usability. However, the success of the INTAMAP project is a clear demonstration of how UncertML can be used within a geoprocessing workflow. The ability of INTAMAP to accommodate observation errors on the data inputs enhances the justification that the work in this thesis has met objective 2 in Section 1.3.1. Furthermore, the propagation of uncertainty through the workflow to the data outputs clearly indicates that objective 3 has been satisfied. The success of the INTAMAP project and the integration with UncertML can be quantified by analysing the client applications and tools. Section 5.4.1 analysed three separate client tools that interfaced with INTAMAP. Each of these tools utilises the UncertML API and thus can be considered an indication of the success of the integration with INTAMAP. Furthermore, the diversity of the client tools is a demonstration of the interoperability provided by INTAMAP and UncertML. However,

the effort required to achieve this level of interoperability taints this success.

# 6 uGML

CONTENTS

## 6.1 Foreword

UncertML, proposed in Chapter 3, outlines an interoperable language for probabilistically quantifying uncertainty. Chapters 4 and 5 demonstrate how UncertML may be integrated into a number of different frameworks. However, as discussed in Section 2.5, there are two areas where uncertainty is present in geospatial information. Both Chapters 4 and 5 provide examples of how UncertML may be used to encode attribute uncertainty. This chapter outlines a framework whereby UncertML can be utilised to encode positional uncertainty.

Section 6.2 critically analyses the GML schema, discussed previously in Section 2.5.2. In particular, Section 6.2 will discuss existing methods of representing quantified positional uncertainty within GML, and explore ways in which GML can be extended to allow a more thorough representation of uncertainty.

Section 6.3 documents an XML schema, uGML, that allows positional uncertainty to be encoded using a similar syntax to that used for existing GML geometries. The flexibility of uGML to encode uncertainty in a variety of geometries is demonstrated with a series of examples towards the end of the section. Section 6.3.3 further demonstrates the benefits of uGML with a Web-based software application that allow users to visualise the positional uncertainty of lines and polygons using the Google Maps API [1].

Finally, Section 6.4 concludes the chapter by outlining the contributions of uGML and discussing possible further work.

## 6.2 GML

Initially titled xGML, and later Simple Features XML (SFXML), GML was first presented to the OGC in 1999. The first proposal of GML was based around the Resource Description Framework (RDF) [2], however, due to strong opposition within the OGC a complementary DTD version was developed. GML version 1.0 was adopted by the OGC as a recommendation paper in 2000. During the same period work was being undertaken by the World Wide Web Consortium (W3C) to produce another XML schema language, titled XML Schema language. With many benefits over DTD (discussed in Section 2.2.1) the OGC pursued the development of an XML Schema-based GML — leading to the adopted standard, GML version 2, in May 2001.

GML 2 offered a simple view of geographic features described through the object model. Fun-

[1] `http://maps.google.co.uk`
[2] `http://www.w3.org/RDF/`

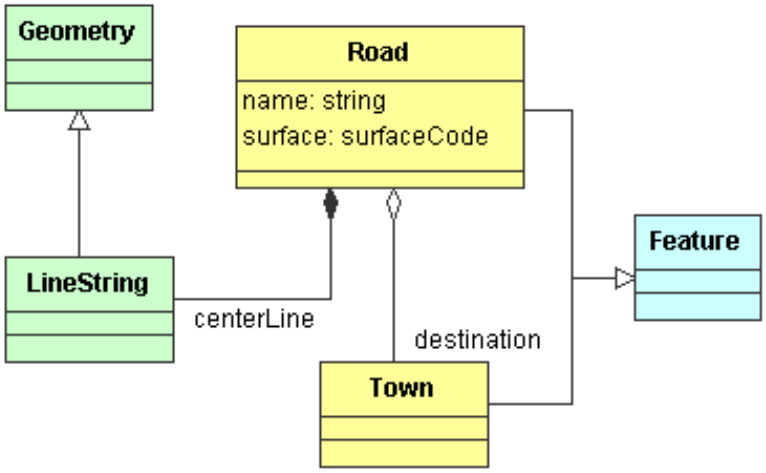



Figure 6.1: GML 2 example of a feature, 'Road', illustrated in UML.

damentally, the object model states that a geographic feature is a named list of properties, some of which may be geometric or other features. The basic premise that geographic features are made up of properties, which may themselves be features, underpins one of the fundamental design principles of GML, the feature-property model. The names of specific features and properties are not specified by GML, but left to the developer of a GML application schema. Figure 6.1, taken from OGC 02-069 (2002), illustrates a simple application schema where a `Road` feature has 4 properties: `name`, `surface`, `centerLine` and `destination`. Both the `name` and `surface` properties are primitive types, but the `destination` and `centerLine` properties are more complex. The `destination` property is another feature, `Town`, and the `centerLine` property is a geometry. While the feature-property design allows for arbitrarily complex, nested features, the properties themselves are always primitive types, geometries or complex features (which are themselves composed of only primitive, geometric or complex feature properties). This simplicity led to a large user base for GML 2. Unfortunately, partly motivated by developments in a rival standard, G-XML, it was established that the simple feature/geometry model was not expressive enough for many purposes. Work was undertaken to include key ideas from G-XML (e.g., observations, temporal and topological elements) in the next version of GML. There was also a need to bring GML in line with the abstract specifications published by the ISO TC/211 committee; a committee primarily involved with the standardisation of geographic information models. This work culminated in GML 3.

GML 3 represented a significant step forwards for GML in terms of scope but also complexity. The number of XML schema files increased from 2 (feature and geometry schemas) to over 10. The basic feature-property model still exists within GML 3, however, it has been modified to conform to the General Feature Model outlined in ISO/TC 211 19109 (2003). The increased

complexity has caused concern (OGC 06-049r1, 2006; OGC 03-003r10, 2003; Tamayo et al., 2010) over the usability of GML 3, with many software implementations preferring to conform to GML 2. Due to the large number of elements interoperability can only be achieved through the development of profiles, e.g., the Simple Features profile (OGC 06-049r1, 2006). Despite the drastically increased complexity of GML 3, the ability to select and use only a subset of the XML schemas allows users to ignore the irrelevant portions for their projects. One such subset is the geometry schemas.

### 6.2.1 GML geometry schemas

During the transition from GML 2 to GML 3 the geometry suite of schemas underwent a substantial change to bring the components of the GML geometry schemas in line with the ISO/TC 211 19107 (2003) spatial schema and form a partial implementation. These changes produced a large increase in the geometry types available within GML 3. This section provides a brief overview of the GML 3 geometry schemas, and later discusses a selection of geometries in more detail.

At the head of the geometry schema is the `AbstractGeometry` element, which is a child of the `AbstractGML` element. All geometries within GML 3 will inherit, either directly or indirectly, from the `AbstractGeometry` element. Consequently, a geometry inherits (via `AbstractGML`) an identifying attribute (`gml:id`) and metadata elements `gml:name`, `gml:identifier` and `gml:description`. Additionally, any element inheriting from `AbstractGeometry` may be associated with a spatial reference system, via the `SRSReferenceGroup`. The `SRSReferenceGroup` provides the facility to identify which specific Coordinate Reference System (CRS) is being used by a geometry. In theory, the attribute `gml:srsName` points to an instance of `AbstractCoordinateReferenceSystem`, however, in practice a URI referring (but not resolving) to a well-known CRS (e.g., European Petroleum Survey Group (EPSG) codes[3]) is used. The `srsName` attribute is optional and if omitted from a geometry it is implicitly assumed that the CRS is specified as part of a larger context, e.g., a geometric aggregate. OGC 07-036 (2007) states that "any geometry that inherits the semantics of `AbstractGeometryType` may be viewed as a set of direct positions". Within the GML 3 geometry schemas there are two elements for encoding direct positions: `DirectPositionType` (used for geometries with a single direct position) and `DirectPositionListType` (used for geometries containing multiple direct positions). Instances of the `DirectPo-`

[3] `http://www.epsg.org/`

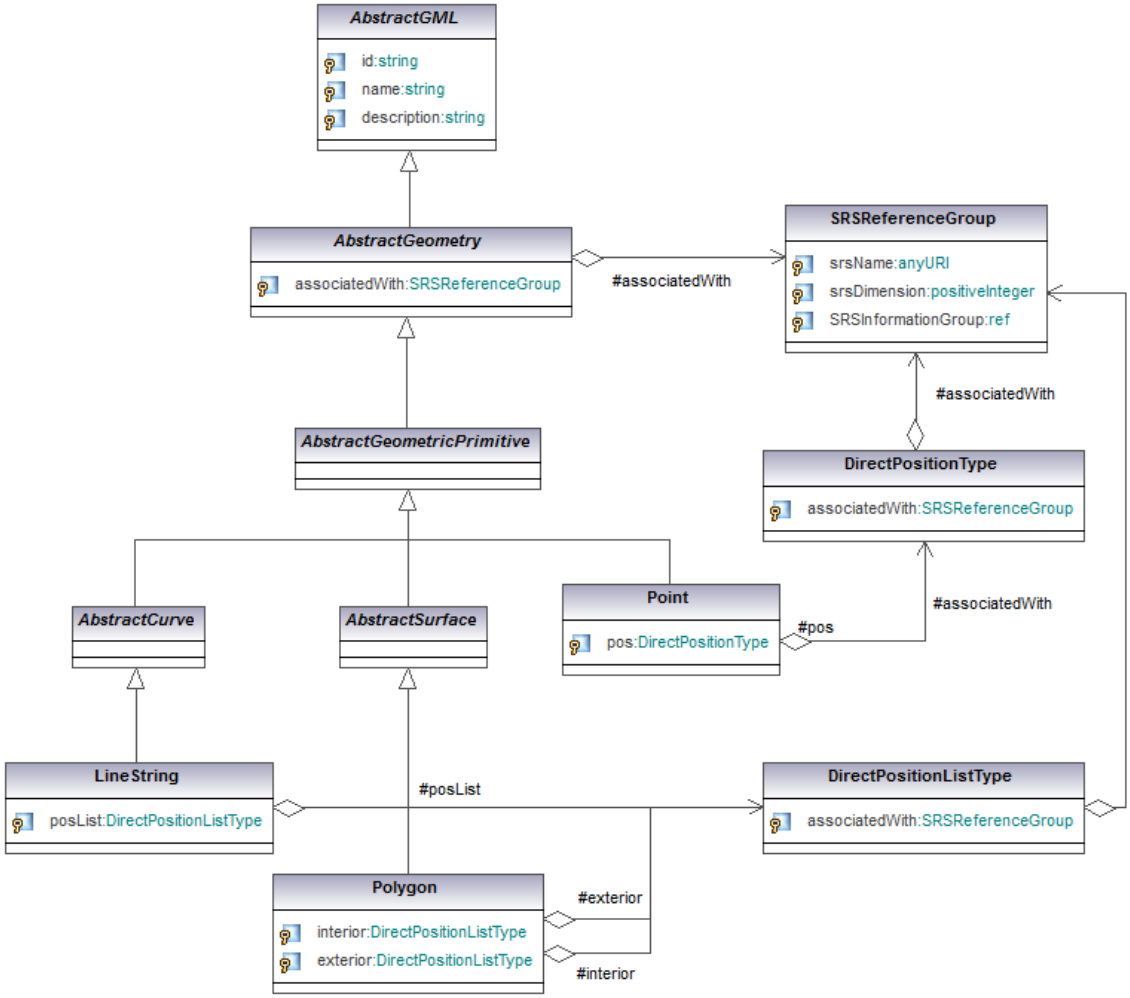



Figure 6.2: A concise overview of the GML 3 geometry schemas illustrated using UML. The diagram is not a complete representation of the GML hierarchy it is purely illustrative.

sitionType hold the coordinates for a position within some CRS. However, since direct positions will often be included in a larger context (e.g., within a geometry type) the CRS is implicitly assumed from the containing element. A DirectPositionListType holds the coordinates for a sequence of direct positions within the same CRS. Similarly to the DirectPositionType, the CRS of a DirectPositionListType will typically be specified at a higher level. In addition, the DirectPositionListType contains an optional attribute, count, that specifies the number of direct positions encoded within the list. The primary instance of the DirectPositionType is the element pos and the primary instance of the DirectPositionListType is the element poslist. Figure 6.2 provides an overview of the GML 3 geometry schema hierarchy, including the AbstractGeometry root, both direct position types and a selection of geometry types (point, line and polygon) that are discussed in the following sections. The diagram in Figure 6.2 is not a complete representation and certain inconsistencies have been introduced. For example, the Polygon element has two properties, interior and exterior, however, these are not directly inherited from the DirectPositionListType, as suggested in the diagram. The properties are actually LinearRingTypes, which themselves are AbstractRingTypes. The decision to omit these types was in the interest of retaining simplicity. Nonetheless, Figure 6.2 provides an illustration of the concepts and types discussed in this section.

### GML Point

The GML Point element is the simplest geometry provided, completely described by a single coordinate tuple. The gml:pos element specifies the direct position of a Point. Listing 6.1 provides an example of a GML 3 Point element.

```
<gml:Point gml:id="1312" srsName="urn:ogc:def:crs:EPSG:6.6:4326">
  <gml:pos>52.01 0.56</gml:pos>
</gml:Point>
```

Listing 6.1: A GML 3 Point element, the direct position of a Point is encoded within the pos property.

The example in Listing 6.1 demonstrates that the srsName attribute of a pos element is inferred from its parent element if left blank. GML provides a MultiPoint type to aggregate multiple Point elements. When a MultiPoint element is defined with an srsName attribute, the srsName attribute of the constituent Point elements may be omitted. The GML MultiPoint type is an implementation of the ISO 19107 GM_MultiPoint.

### GML LineString

Conceptually, a `LineString` element is perceived as an `AbstractCurveType` within GML 3. A `LineString` is defined as "a special curve that consists of a single segment with linear interpolation", i.e., it contains two or more direct positions with a straight line between each coordinate tuple. There are many other types of curves detailed in OGC 07-036 (2007), however, they are not covered here. The direct positions of a `LineString` can be encoded in two ways. Firstly, as depicted in Figure 6.2, they may be encoded in a single `poslist` element. Alternatively, it is possible to encode a `LineString` as a series of `DirectPositionTypes`, or, `pos` elements. The second method of encoding can be seen in Listing 6.2.

```
<gml:LineString gml:id="4121" srsName="urn:ogc:def:crs:EPSG:6.6:4326">
   <gml:pos>52.01 0.56</gml:pos>
   <gml:pos>51.72 0.56</gml:pos>
   <gml:pos>52.01 0.42</gml:pos>
</gml:LineString>
```

Listing 6.2: A GML 3 `LineString` element. This example demonstrates how the direct positions of a `LineString` may be encoded using a series of `pos` elements.

GML provides a `MultiCurve` element for aggregating multiple curves, which includes `LineStrings`. The same principles apply to the `MultiCurve` as the `MultiPoint` element with regards to the `srsName` attribute. The GML `MultiCurve` element is an implementation of the ISO 19107 `GM_MultiCurve`.

### GML Polygon

A GML `Polygon` inherits from the `AbstractSurfaceType` (Figure 6.2). OGC 07-036 (2007) defines a surface as a continuous region of a plane. A `Polygon` is a special surface, defined by a single surface patch whose boundary is coplanar. The interior of a `Polygon` uses planar interpolation. The implementation of the surface boundary for a `Polygon` is through the `interior` and `exterior` elements. Both the `interior` and `exterior` elements are `LinearRings`, which must contain a minimum of 4 direct positions. A `Polygon` may have any number of `interior` boundaries, but may only have a single `exterior` boundary. Listing 6.3 provides an example encoding of a GML `Polygon`.

There are more surface types defined in OGC 07-036 (2007), however, they are not described in this thesis. Collections of polygons and other surfaces may be aggregated using the GML `MultiSurface` element, an implementation of ISO 19107 `GM_MultiSurface`.

```
<gml:Polygon gml:id="5311" srsName="urn:ogc:def:crs:EPSG:6.6:4326">
   <gml:exterior>
      <gml:LinearRing>
         <gml:posList>
            52.01 0.56 52.01 0.42 51.72 0.42 51.72 0.56
         </gml:posList>
      </gml:LinearRing>
   </gml:exterior>
</gml:Polygon>
```

Listing 6.3: A GML 3 `Polygon` element. This polygon is constructed with an `exterior` boundary, defined by the `DirectPositionListType`, `poslist`.

### 6.2.2 Uncertainty within GML geometry schemas

With the importance of quantifying the uncertainties relating to spatial data outlined in Section 2.4, the lack of scope for expressing uncertainty within the GML geometry schemas is disconcerting. The most direct mechanism for quantifying positional uncertainty within GML is via the `GenericMetadata` element, inherited by each geometry from the `AbstractGML` type. However, this solution has a number of substantial flaws. Firstly, the `GenericMetadata` element is now deprecated, it may still be used but is liable to be removed from GML in future versions. Secondly, the content of the `GenericMetadata` is XML anyType, effectively allowing any data to reside within it. This could cause problems, for example, if a user encodes just a single value. This value may be intended to represent the standard deviation of a variable, yet the lack of a structured representation means that it could be interpreted as any number of things. Listing 6.4 provides an example of how a GML Point may look with a `GenericMetadata` element.

```
<gml:Point gml:id="1312" srsName="urn:ogc:def:crs:EPSG:6.6:4326">
   <gml:metaDataProperty>
      <gml:GenericMetaData>
         <un:Statistic definition="std_dev">
            <un:value>0.0012</un:value>
         </un:Statistic>
      </gml:GenericMetaData>
   </gml:metaDataProperty>
   <gml:pos>52.01 0.56</gml:pos>
</gml:Point>
```

Listing 6.4: A GML `Point` with a `GenericMetadata` to encode the positional uncertainty. The content of the `GenericMetadata` is a cut-down example of UncertML, encoding the standard deviation.

With `GenericMetadata` now deprecated, the inclusion of metadata within GML features has become the responsibility of application schema designers, i.e, GML users. OGC 07-036 (2007)

states that to associate metadata with a GML object, a property element should be defined whose content model is derived (by extension) from the `AbstractMetadataPropertyType` element. This approach provides greater flexibility to GML producers, but is inadequate to fully quantify positional uncertainties for a number of reasons. Firstly, the metadata must be added to a GML feature; GML geometries are not features, but properties of features. This restriction stipulates that one would create a feature (e.g, a road) which has a geometric property but which also has some metadata pertaining to the uncertainty about that geometric property. This presents a semantic problem which has been touched on earlier — is positional uncertainty data, or metadata? Section 2.4.1 argues that if a value is gathered from a derived observation (e.g., a GPS device) then that value is intrinsically uncertain. Therefore, the uncertainty of a position recorded in this way should be encoded directly and not as metadata. While this criticism is purely semantic, a further problem may also hinder interoperability. When the responsibility of defining metadata is delegated to GML application schema producers, there is a significant risk that two application schemas may use different mechanisms to describe the same ideas. To maintain interoperability for concepts as fundamental as positional uncertainty it is imperative that they be standardised at the highest level, i.e. within the core GML specifications. This requirement means that the current procedures available within GML for quantifying positional uncertainty are inadequate and there is scope for improvement.

## 6.3 uGML implementation

Section 2.4 provided discussion about the substantial literature on understanding and quantifying positional uncertainty. A number of common techniques, or frameworks, were introduced including simple epsilon bands, fuzzy set theory and probability theory. Each framework has merits, yet it was concluded that probability theory provided the most complete quantification of uncertainties regarding geospatial data. Section 6.2 described GML as the *de facto* standard for representing geospatial features and concepts within an interoperable framework. Surprisingly, little or no standard procedure for representing the uncertainty of geospatial data exists within the current version of GML. The contrast between the necessity of quantifying positional uncertainty, and the inability to do so within GML provides the motivation for the work in this chapter.

This section provides a discussion on existing extensions to allow positional uncertainty within GML, highlighting their limitations. Possible solutions are then analysed before a final solution is suggested and demonstrated via a use case.

### 6.3.1 Existing work

Despite the substantial literature on positional uncertainty little research has been conducted into extending GML to provide such functionality. Perhaps the primary reason is that, until now, no interoperable (XML) standard existed to allow the quantification of uncertainty. Any extension of GML to allow positional uncertainty would therefore have to provide a standard for describing uncertainty; a non-trivial task. Another contributing factor to the dearth of GML extensions for positional uncertainty might be the complexity of the GML schemas, discussed in Section 6.2. When considered together, these two factors dictate that the creation of a GML extension for positional uncertainty is a substantial body of work. Nonetheless, a couple of attempts have been made. The following two sections analyse the work carried out by Morris and Petry (2006) and Donaubauer et al. (2008), respectively.

#### A fuzzy approach

Although Section 2.4 concluded that probability theory provided the most robust framework for quantifying positional uncertainty, a substantial amount of research has been conducted on the use of fuzzy set theory. Morris and Petry (2006) outline an extension of GML to provide support for geographic objects with uncertain boundaries. This extension, also titled UGML (herein denoted by the use of a capital 'U'), allows a series of alpha-cut representations to be included within a GML object. Alpha-cuts are defined by Buckley (2004) as "slices through a fuzzy set producing regular (non-fuzzy) sets".

While UGML provides a valuable extension to GML, there are several limitations, or flaws, that prevent it from fully satisfying the requirements of a positional uncertainty extension to GML. Primarily, this implementation of fuzzy set theory, unlike of probability theory, does not allow for complex relationships between locations, i.e. spatial autocorrelation (Section 2.4). This can be seen clearly in Figure 6.3 which depicts 3 realisations of a polygon (core, maximum and an alpha-cut). The simplicity of this approach is a clear contrast to the flexibility of the approach shown in Figure 2.6 and Figure 2.7. Also, the mechanism adopted by Morris and Petry (2006) to allow the inclusion of alpha-cuts within GML is sub-optimal. The key idea is that by using existing GML concepts (`DynamicFeatureType`) the overheads of UGML implementers are reduced. However, by including the uncertainty information at the feature-level, and not the geometry-level, assumptions must be made. For example, if a feature has multiple geometries, a phenomenon that is supported by GML, it is unclear which geometry the alpha-cuts refer to. The use of `DynamicFea-`

`tureType` also promotes the repetition of duplicate feature properties.

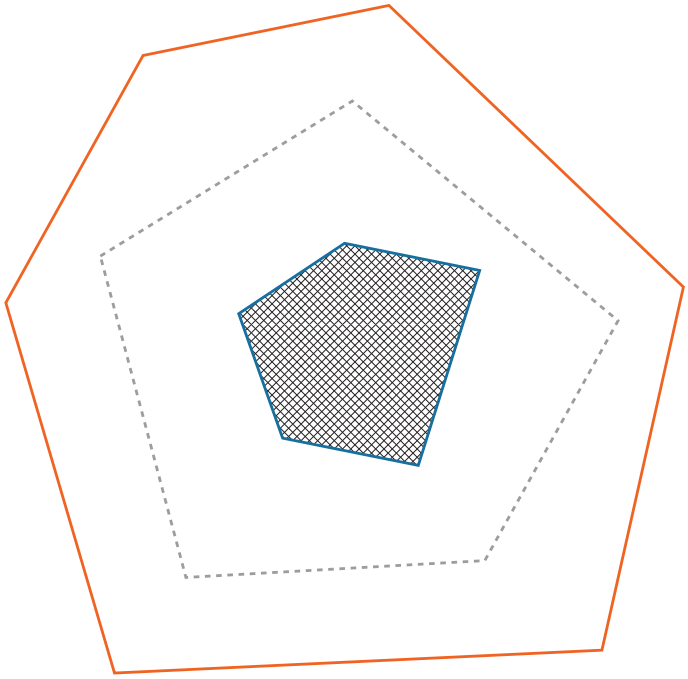


Figure 6.3: An illustration of how alpha cuts can represent positional uncertainty for a polygon. The dashed grey line represents an alpha cut of the polygon and the orange line is the maximum.

Morris and Petry (2006) clearly identified the need for a GML extension to cater for positional uncertainty. However, the oversimplification of the fuzzy membership function and poor design choices culminated in a profile of GML that has limited usability. One further point is that this approach loses the exact shape of the continuous membership function by discretising it into alpha-cuts. The 'shape' of the PDF can always be implicit in UncertML either through the use of a well-known distribution of through the use of quantiles to approximate user-specified PDFs. The solution presented by Morris and Petry (2006) can only approximate the membership function in the latter manner by storing many alpha-cut representations, and the authors do not discuss this strategy in any detail.

#### An ISO 19115 approach

Donaubauer et al. (2008) also identify the need for a GML extension to allow for positional uncertainty. Their work, however, focusses on the use of the ISO 19115 and ISO 19139 metadata standards to provide the quantification of uncertainty. While the use of these ISO standards arguably improves interoperability with existing standards there is a significant flaw. The metadata concepts outlined in ISO 19115 and realised in ISO 19139 are conceptual, or abstract (Kresse and Fadaie, 2004). While the 400+ elements defined within ISO 19115 may be used as metadata,

often there is not enough information for practical use. This is especially apparent within the Data Quality (DQ) clause. Donaubauer et al. (2008) provide an example use case of ISO 19139-encoded uncertainty within a GML `MultiCurve` aggregate (Listing 6.5).

```
<app:area gml:id="idfafe29f9-8992-44c8-9718-c100a7bffdca">
   <gml:multiCurveProperty>
      <gml:MultiCurve>
         <gml:curveMember>
            <gml:Curve>
               <gml:metaDataProperty>
                  <mdl:GeometryLevelMetadata>
                     <mdl:absoluteExternalPositionalAccuracy>
                        <gmd:DQ_AbsoluteExternalPositionalAccuracy>
                           <gmd:result>
                              <gmd:DQ_QuantitativeResult>
                                 <gmd:valueUnit xlink:href="#m"/>
                                 <gmd:value>
                                    <gco:Record xsi:type="gml:LengthType" uom="#m">2.0<
/gco:Record>
                                 </gmd:value>
                              </gmd:DQ_QuantitativeResult>
                           </gmd:result>
                        </gmd:DQ_AbsoluteExternalPositionalAccuracy>
                     </mdl:absoluteExternalPositionalAccuracy>
                  </mdl:GeometryLevelMetadata>
               </gml:metaDataProperty>
               <gml:segments>
                  <gml:LineStringSegment>
                     <gml:posList>0 0 10 0 5 10</gml:posList>
                  </gml:LineStringSegment>
               </gml:segments>
            </gml:Curve>
         </gml:curveMember>
      </gml:MultiCurve>
   </gml:multiCurveProperty>
</app:area>
```

Listing 6.5: A GML `MultiCurve` element using ISO 19115 and ISO 19139 elements to encode the positional uncertainty (Donaubauer et al., 2008).

The example in Listing 6.5 clearly identifies the problems of using the ISO 19115 metadata DQ elements to quantify positional uncertainty. Within the `MultiCurve` element exists a metadata property of type `DQ_AbsoluteExternalPositionalAccuracy`. This property contains a `DQ_QuantitativeResult` with a value of 2 metres. The problem is that the semantics of this value are not well defined, so that it is not clear what the 2 metres actually refers to. Also, using this extension does not provide a mechanism for quantifying spatial autocorrelation, or other complex phenomena such as covariance between specific sets of points.

However, the method of extension employed by Donaubauer et al. (2008) is superior to that

employed by Morris and Petry (2006). The uncertainty has been delegated to a geometry level, via the proposed `GeometryLevelMetadata` property, and is therefore more explicit. It is interesting to note that both Donaubauer et al. (2008) and Morris and Petry (2006) have adopted the view that positional uncertainty is metadata and not data. The reasons for this are explored in the following section.

### 6.3.2 Approaches to extending GML in order to enable uncertain geometries

One of the main benefits of XML is that it is extensible, which means that there are many ways in which to extend an existing XML vocabulary such as GML to allow additional functionality. The requirement to integrate UncertML into GML restricts any extension to the use of elements, rather than attributes. The key decision then becomes whether positional uncertainty should be considered as data or metadata. The two previous examples of integrating positional uncertainty into GML both concentrated on a metadata perspective (Morris and Petry, 2006; Donaubauer et al., 2008). The following section provides an alternative, whereby positional uncertainty is considered to be data.

#### Data approach

Listing 6.1 outlines a typical GML `Point` element. The 'data' is considered to be the coordinates, i.e. the values that reside within the `pos` element. The current GML specification (3.2.1) states that a `pos` (`DirectPositionType`) is an extension of a `doubleList`, i.e. it is a space-separated list of floating point values. Typically, this list contains one value per coordinate so a two-dimensional point will have two values within the `pos` element. Perhaps one method of encoding positional uncertainty would simply be to state, within the specification, that the `pos` element now contains the expected value and variance of each coordinate. This extension would not require any physical changes to the schema. A two-dimensional point would now have the following values: `mean x, variance x, mean y, variance y`. A major disadvantage of this proposal is that it is not explicit enough. For example, while the specification states the desired order and meaning of values within the schema, there are no physical restrictions in place to allow validation. Furthermore, the quantification of uncertainty is limited to a single data pattern (normal distribution); no mechanism is in place for other distribution types or spatial autocorrelation. A more sophisticated extension can be achieved by altering the `DirectPositionType` and `DirectPositionListType` (`poslist`), replacing the space-separated values with an UncertML type.

This modification allows the data of a GML geometry to be upgraded to any UncertML type. In the simplest form this could be a series of expected, or mean, values for the coordinates. However, the range of available types within UncertML allows for a far more complex representation using probability distributions for both marginal and jointly distributed coordinates. Listing 6.6 provides an example of how a GML point is encoded using an UncertML extension to the `pos` element.

```
<ugml:uPoint srsName="EPSG:4326">
  <ugml:upos>
    <un:Distribution definition="Gaussian">
      <un:parameters>
        <un:Parameter definition="mean">
          <un:value>52.01</un:value>
        </un:Parameter>
        <un:Parameter definition="variance">
          <un:value>0.0002</un:value>
        </un:Parameter>
      </un:parameters>
    </un:Distribution>
    <un:Distribution definition="Gaussian">
      <un:parameters>
        <un:Parameter definition="mean">
          <un:value>0.56</un:value>
        </un:Parameter>
        <un:Parameter definition="variance">
          <un:value>0.0002</un:value>
        </un:Parameter>
      </un:parameters>
    </un:Distribution>
  </ugml:upos>
</ugml:uPoint>
```

Listing 6.6: A GML `Point` element extended to use UncertML to encode the positional uncertainty.

The example given in Listing 6.6 locates a point within a quantified uncertain region by giving the marginal distributions of its coordinates. Worth noting is that the extended `pos` element (`upos`) contains two UncertML `Distribution` elements. However, it could contain a `Distribution-Array` instead. Care must be taken to ensure that there are enough distributions to quantify each coordinate of the point. Correlation between error at different coordinates, discussed in Section 2.4.1, can be encoded using the `MultivariateDistribution` type (Listing 6.7).

The main benefit of encoding positional uncertainty as data, rather than metadata, is that all uncertainty is expressed explicitly. This stipulation means that it is impossible to encode a geometry without giving thought to the underlying uncertainty. While the semantic benefits of this system are obvious i.e. all positions are observed indirectly and are therefore inherently uncertain syntactically it has several problems. Primarily the GML schema has in fact been altered,

```
<ugml:uPoint srsName="EPSG:4326">
   <ugml:upos>
      <un:MultivariateDistribution definition="Gaussian">
         <un:parameters>
            <un:ParameterArray>
               <un:elementType>
                  <un:Parameter definition="mean"/>
               </un:elementType>
               <un:elementCount>2</un:elementCount>
               <swe:encoding>
                  <swe:TextBlock decimalSeparator="." blockSeparator=" "
tokenSeparator=","/>
               </swe:encoding>
               <swe:values>
                  52.01
                  0.56
               </swe:values>
            </un:ParameterArray>
            <un:ParameterArray>
               <un:elementType>
                  <un:Parameter definition="covariance"/>
               </un:elementType>
               <un:elementCount>4</un:elementCount>
               <swe:encoding>
                  <swe:TextBlock decimalSeparator="." blockSeparator=" "
tokenSeparator=","/>
               </swe:encoding>
               <swe:values>
                  0.0002,0.0015
                  0.0015,0.0002
               </swe:values>
            </un:ParameterArray>
         </un:parameters>
      </un:MultivariateDistribution>
   </ugml:upos>
</ugml:uPoint>
```

Listing 6.7: A GML `Point` element using the `MultivariateDistribution` type to encode spatial autocorrelation between coordinates.

rather than extended. The alteration to `pos` to allow it to encode UncertML types rather than space-separated numbers propagates through the whole GML geometry schema. The effects of this single alteration are that each geometry that should support uncertainty must also be altered. While the semantics of a `Point` and `uPoint` element are very similar (i.e., they describe a point location via a direct position) the slight syntactic changes (e.g., `upos` instead of `pos`) break all compatibility with existing GML software. For example, a GML 3.2.1 parser could process a `Point` element but would not be capable of processing a `uPoint` element. A secondary effect of this alteration is that the changes to both `pos` and `poslist` are not automatically inherited by the GML geometry types. This requires alterations to be made to every GML geometry, i.e. a specific

'uncertain' geometry type must be defined for each existing GML geometry. The result of this is a schema that contains many elements from the GML geometry schema that have been altered slightly, but enough to render them incompatible with existing GML software.

Extending, or altering, GML to allow positional uncertainty to be encoded as data, rather than metadata, provides a semantically sound solution, but at what cost? The need to change the base position types within the geometry schema forces changes upon all geometry elements, breaking the compatibility with existing GML software. Technically, all geospatial positions are inherently uncertain, therefore they should be encoded as such. However, when such an encoding hinders interoperability it comes at too high a cost.

#### Metadata approach

The problems encountered by the data-centric extension, discussed above, are resolved by employing a metadata extension. Extending GML to add an additional 'metadata' property is a classical approach of XML extension and, typically, does not break compatibility with existing software. While the extension will not validate against the stock GML schemas, any GML parser can choose to ignore a property if it does not understand it. There are implications to ignoring uncertainty in geospatial data, especially when the criticality of uncertainty in decision making is considered. However, the decision to ignore any uncertainty is at the software level and should not be considered a negative aspect of a metadata-focussed extension, provided that users of the data are aware that it is not being exploited to the full. In the previous section the need to create separate elements for each uncertain GML geometry was discussed. Using a metadata approach this can be avoided. In fact, by extending the GML `AbstractGeometry` element to add an `uncertainty` property, every GML geometry automatically inherits the changes. Essentially, this results in a single change to the root of the hierarchy which is adopted, automatically, by all child elements, in stark contrast to the previously-discussed method. The tangible benefits of adopting an extension such as this clearly outweigh the detrimental side-effects of a metadata-centric approach. Therefore, the decision to implement uGML as a metadata extension was made.

The physical changes required to GML are small. Listing 6.8 depicts how a single element (uncertainty) has been added to the `AbstractGeometryType` complex type. The uncertainty element references the `AbstractUncertainty` element from UncertML, allowing any UncertML type to be encoded within a GML geometry. The result of this extension can be seen in Listing 6.9, which shows the same `Point` as Listing 6.6 but with a metadata extension.

```
<element name="AbstractGeometry" type="gml:AbstractGeometryType" abstract="true"
substitutionGroup="gml:AbstractGML" />

<complexType name="AbstractGeometryType" abstract="true">
   <complexContent>
      <extension base="gml:AbstractGMLType">
         <sequence>
            <element name="uncertainty">
               <complexType>
                  <sequence>
                     <element ref="un:AbstractUncertainty"/>
                  </sequence>
               </complexType>
            </element>
         </sequence>
         <attributeGroup ref="gml:SRSReferenceGroup"/>
      </extension>
   </complexContent>
</complexType>
```

Listing 6.8: Extension to GML `AbstractGeometryType` to allow the addition of positional uncertainty.

On first glance, the two examples in Listings 6.6 and 6.9 appear identical, however, there are differences. Primarily, the root element is a GML `Point`, not `uPoint`, and the data is encoded within the GML `pos`, not `upos`, element. These slight changes mean that an existing GML parser can process and understand the `Point` element, choosing to ignore anything between, and including, the `uncertainty` property. This will result in a point with coordinates `52.01,0.56`, however, the actual meaning of these coordinates must be consistent across all users of uGML. When the positional uncertainty is quantified within metadata as it is here, it is not clear what the actual coordinates (data) values mean. Therefore, within uGML, it is specified that the values within a `pos` or `poslist` element must always be the expected value (statistical mean). While this may seem obvious to the reader in situations where the positional uncertainty is quantified using a normal distribution, care must be taken when using other concepts to quantify the uncertainties. For instance, when using the UncertML `Realisations` type to quantify a coordinate, it is important to remember that the value within the `pos` element is the mean value and not a single realisation. Adherence to this rule should ensure that both users who wish to ignore and those who wish to process the positional uncertainty are aware of exactly what is encoded in the data and what is encoded in the metadata.

With the extension made to the root of the geometry substitutability chain, all geometries inherit the benefits. Thus far all examples of positional uncertainty within GML have focussed on a

```
<gml:Point srsName="EPSG:4326">
  <gml:uncertainty>
    <un:Distribution definition="gaussian">
      <un:parameters>
        <un:Parameter definition="mean">
          <un:value>0.0</un:value>
        </un:Parameter>
        <un:Parameter definition="variance">
          <un:value>0.00002</un:value>
        </un:Parameter>
      </un:parameters>
    </un:Distribution>
    <un:Distribution definition="gaussian">
      <un:parameters>
        <un:Parameter definition="mean">
          <un:value>0.0</un:value>
        </un:Parameter>
        <un:Parameter definition="variance">
          <un:value>0.00002</un:value>
        </un:Parameter>
      </un:parameters>
    </un:Distribution>
  </gml:uncertainty>
  <gml:pos>52.01 0.56</gml:pos>
</gml:Point>
```

Listing 6.9: A GML `Point` element extended to embed UncertML within an uncertainty, metadata, property.

simple point. However, much more complex geometries and their uncertainties can be represented using uGML. Listing 6.10 gives an example of how a GML `Polygon` can be encoded with uGML. This example demonstrates the ability of uGML to encode the correlation between the individual points of an object using a multivariate Gaussian distribution. The values in this example have been truncated for brevity. However, the covariance matrix structure is capable of storing the correlation between the uncertainty in coordinates of individual points as well as the correlation between uncertainties at all points within the polygon. In the example in Listing 6.10 there are 144 values in the covariance matrix. This is because there are 6 points in the polygon, each with 2 dimensions. This gives a matrix of size 12x12. This encoding is scalable, for polygons with more points and with higher dimensions.

Listing 6.10 is an example of how uGML can be used to encode a deformable object, discussed in Section 2.4.1. This ability to encode the likely shapes and positions of deformable objects provides a more functional solution than that proposed by Morris and Petry (2006). The benefit of integrating UncertML, a well-defined standard for encoding probabilistic representations of generic uncertainty, is also an improvement over the solution discussed by Donaubauer et al.

(2008). While the solution discussed here is syntactically similar to that proposed by Donaubauer et al. (2008), UncertML ensures a more explicit, and arguably, more interoperable package.

### 6.3.3 A uGML use case

The concepts covered in this chapter can be difficult to understand and the nature of XML, while human-readable, renders it hard to visualise the uGML encodings. These issues motivated the development of a small prototype web application to allow users to create and visualise uGML instances. Developed using the Google Maps API [4], the prototype is a platform to showcase the capabilities of uGML and to provide users with some ideas about potential uses of uGML. The web application interface, mapping, uGML parsing and encoding is a contribution of this thesis. The code to generate realisations from a given set of marginal distributions, or multivariate Gaussian distribution was provided by a third party.


Figure 6.4: uGML web application demonstrating how uncertainty can be added to individual points of an object.

Built around the latest web technologies, the application allows users to draw lines and polygons on top of a map layer. The individual points of the line or polygon can then be inspected and, if required, can have some uncertainty added. Currently, the uncertainty of each point is normally

[4] http://code.google.com/apis/maps/index.html

distributed. However it is possible that the application will be extended in future to allow other types of probability distributions. The user adds uncertainty to points, either individually or as a group, by adjusting a slider to set the standard deviation. Figure 6.4 shows a point with some added uncertainty. The concentric circles represent 1, 2 and 3 standard deviations, thus representing 68%, 95% and 99.7% confidence intervals given it is normally distributed. Unfortunately, due to limitations in the Google Maps API, the coordinates of a point can not be assigned correlated uncertainties. However, an overall spatial correlation can be set, using a slider, to quantify the correlation between uncertainties at the points of an object. All of the objects viewable in the web application are encoded as uGML. At any stage of the web application, a user can switch between the map and uGML views to investigate how the uGML changes as they alter the uncertainty properties. When a polygon or line has no correlation, each point of the shape is represented as a marginal distribution (Listing 6.6). These `upos` distributions are parsed in JavaScript and the mean values are taken as the $x$ and $y$ coordinates. Using this list of coordinates a polygon, or line, is drawn to the map canvas using the Google API. The uGML web application then calculates the standard deviation of each point as the square root of the specified variance, this is then used to draw the concentric circles around each point. If, however, there exists some correlation between the points then a `MultivariateDistribution` is used within a `posList` property (Listing 6.7). The process for rendering the uncertain shape is similar to the marginal case, however, the extraction of the variances from the covariance matrix requires more processing (they form the diagonal of the matrix). The uGML that is generated can be uploaded to a server and retrieved at a later date, where it is parsed and converted back to a visual representation.

Once the uncertainties at each point have been elicited from the user, a series of realisations of the object are generated. These realisations are presented to the user in a controllable animation, to demonstrate the effect that uncertainty has on the position of the object. Figure 6.5 shows the object created in Figure 6.4 with one possible realisation. The relatively large uncertainties of the points result in a realisation of an object that is significantly different to the initial one. The web application allows for many variables to be altered, some with drastic effects on the generated realisations. Unfortunately, the current version of uGML does not support the description of true rigid objects. Providing a correlation value of 1.0 generates realisations that translate from the user-specified polygon, but they do not rotate. The reason for this omission is that the addition of a number of required rotation angles requires a major structural change to the GML schema that would break backwards compatibility with existing GML software. It should be mentioned that


Figure 6.5: A realisation (orange) of an object (blue) created using the uGML web application. Large uncertainties have resulted in a substantially different geometry.

the inclusion of true rigid objects does not require a change to the UncertML schema.

The web application is freely available to use [5]. However, it should be noted that as UncertML and uGML inevitably evolve the code seen in the web application may differ from that discussed in this thesis. The basic concepts of positional uncertainty and UncertML, however, will remain unchanged.

## 6.4 Conclusions

The work in previous chapters focussed on using UncertML to quantify attribute uncertainty. This chapter investigated the use of UncertML to quantify positional uncertainty. Section 2.4 provided an overview of positional uncertainty including current methodologies for quantifying it. There are many ways of quantifying positional uncertainty ranging from simple epsilon bands to fuzzy and probabilistic methods. However, it was decided that positional uncertainty is best quantified using probability theory. Particular attention was paid to the work carried out by Heuvelink et al. (2007) and their categorisation of objects as points, rigid or deformable. While the description of a rigid object is statistically simpler than a deformable object, it proved difficult to implement within uGML. The omission of true rigid objects from uGML is not a reflection on the expressive-

[5] http://www.uncertweb.org/accuracy

ness of UncertML, but rather the difficulty of implementing them within GML while maintaining backwards compatibility. A discussion followed as to whether positional uncertainty should be considered data or metadata. It was concluded that while both schools of thought provided the same information, uncertainty should not be considered metadata for any data that are not directly observed. In the case of positional uncertainty it can be argued that all positions undergo the scrutiny of some form of processing and are therefore not directly observed. However, Section 6.3 shows that treating all data as derived, and hence uncertain, is not always desirable.

Section 6.2 investigated the GML specification. An overview of the history of GML shows that the evolution from version 2 to version 3 drastically increased the complexity of the standard. The increased complexity of GML 3 had led to various software tools continuing to only support version 2. However, despite this complexity the individual components of GML 3 (e.g., geometry, topology etc) are simple enough to use in isolation. Within the GML 3 geometry schemas there is no current mechanism for encoding positional uncertainty. The only means to encode positional uncertainty is via the 'GenericMetadataProperty', however this is far from satisfactory and has, in fact been deprecated. This omission from the GML schemas motivated the work in Section 6.3.

Despite the lack of functionality in GML and the discussed importance of quantifying positional uncertainty, few GML extensions exist for this purpose. Section 6.3 detailed two approaches by Morris and Petry (2006) and Donaubauer et al. (2008) respectively. The method outlined by Morris and Petry (2006) used alpha-cuts from fuzzy set theory but was found to be insufficiently expressive. Donaubauer et al. (2008) utilised the ISO 19115 metadata standard to quantify the uncertainty as metadata. This method proved more complete, but the ISO 19115 standard is too abstract to accurately quantify uncertainties in a way that makes the data easy to interpret and use. The shortcomings of the two examined methods further motivated the work in Section 6.3.2.

Two possible methods of extending GML to allow UncertML to quantify positional uncertainty were discussed in Section 6.3.2. Firstly, a method whereby the positional uncertainty is considered to be data rather than metadata was examined. This method was in-line with the conclusions in Section 2.4, but encoding positional uncertainty as data within GML resulted in too many significant changes to the schema, which in turn led to serious incompatibilities with existing software. This breach of interoperability forced the development of a metadata model. The metadata method proved that with only a simple change the entire GML geometry schema could be enhanced to allow positional uncertainty to be encoded. Various assumptions were identified and made explicit, namely that the data within a `pos` or `poslist` element should always be the

expected value. However, this limitation is insignificant when set against the benefits provided by a metadata model. Finally, a web application was discussed that allows users to draw and visualise uGML shapes via a simple interface.

The work in this chapter has clearly demonstrated the flexibility of the UncertML specification. Prior to this, all examples of UncertML had focussed on attribute uncertainty. The ease with which UncertML can integrate with GML to encode positional uncertainty demonstrates the benefits of designing UncertML in a domain-agnostic way. It can be concluded from the research in this chapter that UncertML can be used to model positional uncertainty, thus providing an answer to the question in objective 2 (Section 1.3.1).

```
<gml:Polygon srsName="EPSG:4326">
  <gml:uncertainty>
    <un:MultivariateDistribution definition="multivariate-gaussian">
      <un:parameters>
        <un:ParameterArray>
          <un:elementType>
            <un:Parameter definition="mean"/>
          </un:elementType>
          <un:elementCount>12</un:elementCount>
          <swe:encoding>
            <swe:TextBlock blockSeparator=" " tokenSeparator=","/>
          </swe:encoding>
          <swe:values>0,0,0,0,0,0</swe:values>
        </un:ParameterArray>
        <un:ParameterArray>
          <un:elementType>
            <un:Parameter definition="covariance"/>
          </un:elementType>
          <un:elementCount>144</un:elementCount>
          <swe:encoding>
            <swe:TextBlock blockSeparator=" " tokenSeparator=","/>
          </swe:encoding>
          <swe:values>
            <!-- 144 values -->
          </swe:values>
        </un:ParameterArray>
      </un:parameters>
    </un:MultivariateDistribution>
  </gml:uncertainty>
  <gml:exterior>
    <gml:LinearRing>
      <gml:poslist>
        38.834004 -77.300227
        38.826114 -77.308810
        38.823439 -77.302458
        38.819159 -77.302458
        38.818892 -77.299197
        38.831463 -77.294047
      </gml:poslist>
    </gml:LinearRing>
  </gml:exterior>
</gml:Polygon>
```

Listing 6.10: uGML `Polygon` element demonstrating how an UncertML `MultivariateDistribution` can be used to quantify the correlation between the points of an object.

# 7 Conclusions

CONTENTS

## 7.1 Thesis summary

### 7.1.1 Development of an interoperable framework for quantifying uncertainty

Chapter 2 reviewed current work in the fields of interoperability and uncertainty. Emphasis was put on XML as the current *de facto* standard for building interoperable frameworks. The increasing popularity of XML has seen a number of supplementary standards evolve. The concept of a Web service as a platform-independent interface to complex processing algorithms provides the foundations for a software engineering architecture, SOA.

In comparison to the relatively new technologies in interoperability, the techniques used to quantify uncertainty are far more established. The basic principles of probability theory have been understood since the sixteenth century. Chapter 2 focussed on providing a solid understanding of probability theory, and how it can be used to quantify uncertainty. Other techniques for quantifying uncertainty were acknowledged, including fuzzy set theory and Dempster-Shafer theory, however, due to requirements dictated by the INTAMAP project and consortium, probability theory was used.

The chapter contained a strong emphasis on the application of both interoperability and uncertainty within the geospatial domain, as this was seen as the main use case for the work in this thesis. The primary efforts of developing interoperability standards within the geospatial domain occur within the OGC. Established in 1994, the OGC is a consortium that has successfully developed numerous geospatial XML standards and services. Several of the key OGC standards identify the importance of quantifying uncertainty, an integral part of geospatial data, yet do not specify how. The lack of a recognised interoperable standard for quantifying uncertainty provided the motivation for the work in this thesis.

In Chapter 3 a solution to the absence of a standard for quantifying uncertainty was provided in UncertML. In response to the research by Gokhale et al. (2002) and Chung et al. (2003), a Web service approach was adopted over other distributed technologies. A conceptual model was outlined that followed a weak-typed model of uncertainty, although arguments were provided for both weak and strong-typed designs. The implementation of the UncertML conceptual model was provided by a set of XML schemas rather than a DTD due to the reasons specified by Bex et al. (2004). The UncertML schemas were illustrated by a series of XML examples, showing how they can be used to describe uncertainty using realisations, summary statistics and probability distributions. The XML examples demonstrated the benefits of a weak-typed design by allowing

multiple statistics to be encoded using a single `Statistic` type. However, a reliance on identifiers to provide semantics is introduced. UncertML uses the URL scheme for identifying resources and a preliminary implementation of a dictionary of uncertainty terms was provided. The provision of UncertML as an XML-based language allows it to be used within SOAs, which currently are a popular modelling paradigm (Erl, 2004, 2005; Josuttis, 2007). The success of UncertML as an interoperable language for describing probabilistic uncertainty can be measured by a few key metrics: flexibility, usability and interoperability. The remaining chapters in this thesis identified varying applications where UncertML can be used thus implicitly demonstrating a degree of success.

### 7.1.2 Applications of UncertML

Chapters 2 and 3 provided the motivation and implementation of an interoperable standard for describing uncertainty using the concepts of probability theory. Chapters 4 – 6 discuss applications of UncertML in a variety of scenarios, using a combination of existing standards. Chapter 4 began with a critical analysis of the SOS specification — an interface to allow querying and retrieval of observation data, encoded in accordance with the O&M standard. The goal of the SOS, to provide an interface for managing deployed sensors and retrieving sensor data consistent for all sensor systems is ambitious. The difficulties faced by the SOS can be attributed to the standard upon which it is based, O&M. The problem is that O&M is an abstract conceptual model of an observation and is not meant to treated as an implementation. It is stipulated that profiles of O&M should be developed by the user prior to implementing it. However, the development of profiles by users, rather than the OGC, may result in duplicate profiles for similar observations. This duplication of profiles is not interoperable.

Despite the problems with both SOS and O&M specifications, a reasonable amount of interest has grown in the Sensor Web community. The lack of an existing standard for encoding uncertainty within the OGC, despite the fact that all sensor-observed data is inherently uncertain is a major oversight. Chapter 4 provided an extension to the 52° North implementation of a SOS to allow the uncertainty of observation results to be encoded using UncertML. This extension was illustrated by a use case, providing data gathered from Weather Underground, with estimated uncertainties, via a SOS interface. This use case clearly shows UncertML being used to quantify observation errors, an objective of this thesis outlined in Section 1.3.1. Furthermore, the use of UncertML within a Sensor Web context will no doubt contribute to the ongoing research in this field, in

particular to the work of Havlik et al. (2009), Liang et al. (2005) and Terhorst et al. (2006).

Chapter 5 introduced the INTAMAP project. The development of UncertML occurred as a requirement of the INTAMAP project. The interpolation techniques utilised in INTAMAP are an extension of the work carried out in Dubois and Galmarini (2005). The contribution of this thesis was to provide an interoperable framework around these automated algorithms. This framework consisted of a WPS interface, integrated with O&M, GML and UncertML — including the API.

Since the process of interpolating observation data identifies and introduces uncertainty, a language capable of describing this uncertainty was required. The WPS interface provides a loosely-coupled, autonomous method for requesting the interpolation of observation data; two key requirements for a SOA (Erl, 2004). The output of an interpolation is inherently uncertain. However, INTAMAP also allowed for uncertainties on the observation inputs to be encoded, this meets objective 2 in Section 1.3.1. When the user quantifies the uncertainties on inputs they are propagated through the interpolation process via novel techniques (developed by other INTAMAP researchers). The true potential of open standards, and UncertML in particular, was demonstrated by the ability to seamlessly link an UncertML-enhanced SOS as an input to the INTAMAP WPS. This chaining of services demonstrates how UncertML can be used in data processing workflows. The chapter concluded with a philosophical discussion about interoperability and what it means to be 'interoperable'. Simply implementing standards-based interfaces is often not enough to ensure interoperability and often supporting tools and APIs should be developed.

The previous two chapters both utilised UncertML to describe attribute uncertainty. However, a framework for describing uncertainty needs to be able to quantify positional uncertainty as well. Chapter 6 provided a critical analysis of existing interoperable standards for encoding geometries, namely GML. The simplicity of the strong-typed GML 2 saw a rapid adoption, which prompted the development of a far larger set of schemas: GML 3 (weak-typed). However, the additional complexities within GML 3 proved difficult for software implementers, who chose to support version 2.

There is no suitable mechanism for describing uncertainty within GML and this motivated the development of uGML. Quantification of positional uncertainty was implemented according to initial work by Heuvelink et al. (2007), who state that uncertain geometries belong to one of three categories: point geometries, rigid geometries or deformable geometries. UncertML was demonstrated to be capable of describing all three categories, including cases where spatial auto-correlation was present; however, rigid objects were only capable of translations and not rotations

due to potential issues with existing GML software. Using the `Distribution` type within GML allows the use of various probability distributions to encode the positional uncertainty. While the examples in Chapter 6 were all based on the Gaussian distribution, because of the weak-typed design it is possible to use any probability distribution. This allows the concepts discussed in Ripley (1977) to be encoded in UncertML.

Two possible implementations of uGML were discussed. One where the uncertainty was considered to be the data and one where it was considered metadata. It was argued that logically, perhaps it is better to consider positional uncertainty as the data. Unfortunately, due to complications in the development of uGML it was proved that a schema that treated uncertainty as metadata was far more extensible. The potential of uGML was illustrated by a prototype web application that allowed users to build lines and polygons with uncertain points. Any correlation between these uncertainties could also be defined, demonstrating the flexibility of the uGML and UncertML schemas. The contribution of this chapter was the demonstration that UncertML can be used to encode positional uncertainties, as stipulated in objective 3, Section 1.3.1.

### 7.1.3 Thesis aims and objectives

To summarise, the conclusions of this thesis are given below, specifically how the work in each chapter relates to the objectives outlined in Section 1.3.1:

- UncertML is an XML language capable of expressing uncertainty using probabilistic techniques. Uncertainty can be quantified using summary statistics, probability distributions or a set of realisations. The design uses a weak-typed approach which provides flexibility at the cost of usability. The work in Chapter 3 concluded that UncertML is a flexible and usable solution for representing probabilistic uncertainty. This was a primary objective set out by this thesis in Section 1.3.1. However, the degree to which this objective has been met is difficult to accurately quantify. Chapters 4– 6 each provided a unique application within which UncertML was integrated. The diversity of these applications domains is testament to how interoperable UncertML is as an information model.

- Designing a data model requires a delicate balance between expressiveness (i.e., it must contain a large enough vocabulary to be of use) and restrictiveness (i.e., it must not be so vague that it is not clear what terms belong to the vocabulary). Many OGC standards fall into the ‘too expressive’ category and require the development of profiles (restrictions) to prove useful. The OGC should govern the creation of these profiles.

- The INTAMAP project provided an interoperable interface to allow complex interpolation algorithms to be utilised by naïve users. UncertML was used throughout INTAMAP and provided a mechanism for propagating uncertainty through the interpolation process. The propagation of uncertainty coupled with the integration of various OGC standards demonstrates that UncertML can be used for processing of data in uncertainty-enabled workflows (objective 3, Section 1.3.1).

- A SOS extension was developed that allowed the SOS to serve observations with characterised uncertainty. This was used by the INTAMAP service, demonstrating a primitive form of service chaining. The integration with the O&M standard is proof that UncertML can be used to quantify observation errors, thus answering the first part of objective 2.

- uGML was developed to use UncertML to describe positional uncertainty. Designed as an extension to the GML schema, uGML provided the ability to describe uncertainty on point, rigid (partly) and deformable objects, including any spatial autocorrelation present. The conclusions of Chapter 6 state that UncertML can be used to quantify some forms of positional uncertainty, in answer to the second part of objective 2.

## 7.2 Directions for future research

The work in this thesis has provided the foundations of an interoperable language capable of describing complex uncertainties using a probabilistic framework. This thesis has demonstrated the flexibility that UncertML provides by integrating it into a series of varying scenarios and standards. There is huge potential for UncertML, and some of the directions that it can be taken forward are listed below:

**Remodel UncertML** — UncertML was modelled on a weak-typed design methodology. This allowed UncertML to encode uncertainty with any statistic or probability distribution. It is this flexibility that is UncertML's greatest strength — but also its greatest weakness. The ability to describe any distribution or statistic relies on an identifier that references a term in a dictionary. It is argued that if all users of UncertML understand that identifier A refers to a specific statistic then meaningful processing of that can ensue. However, the problem arises when users start referring to their own dictionaries, a valid and reasonable option. The use of non-standardised vocabularies can result in two `Statistic` types referring to the same statistic, but with different identifiers, which breaks interoperability. Coupled with

the fact that the majority of users would only use a small subset of possible statistics and distributions a completely hard-typed design would be beneficial. Another criticism of the UncertML schemas is the reliance on the SWE Common standard. Specifically, it is the implicit reliance of SWE Common on the GML schemas that causes the problems. Often users of UncertML are required to write 'parsers', pieces of software that convert the XML implementation of UncertML into a native programming language object. Usually this can be acheived automatically through the use of existing software tools. However, due to a problem with the GML schemas (which is propagated through SWE Common, and into UncertML), these tools do not work with UncertML and so a user must create the parsers themselves. While the reuse of code has obvious benefits, the problems associated with the abstraction of GML and the propagation of these problems through to UncertML far outweigh the benefits. At the time of publication (September 2010) an initial draft of this remodelled UncertML (titled UncertML 2) has just been released that is purely hard-typed and removes all dependencies of SWE Common.

**Different encodings** — The work in this thesis provided a set of XML schemas for implementing UncertML. XML has been established as the primary interoperable method of sharing data. However, various other languages exist. One such language that is seeing a rise in popularity is JavaScript Object Notation (JSON). Due to the separation of the conceptual model from the implementation of UncertML it is possible to provide UncertML implementations in different languages, e.g. JSON. While the focus of this thesis has been to provide an interoperable method for exchanging uncertainty, it may be beneficial to provide more efficient (in terms of verbosity) encodings. For instance, a Network Common Data Form (NetCDF) encoding could prove useful, especially when quantifying uncertainties over a large number of domain points. This is made possible by the abstraction of a dictionary of uncertainty terms which can inform various implementations via referencing. The initial prototype of the dictionary has received a positive response from a number of users.

**Develop an API** — A side effect of developing a hard-typed schema is that creating software for it becomes easier. An UncertML API was developed as part of the INTAMAP project, however, due to the weak-typed design of UncertML, the API had limited use. For instance, it was able to parse the `Distribution` element into a corresponding `Distribution` class in Java, but this class was generic and therefore could not do any meaningful processing on this distribution. With a hard-typed design it would be possible to parse a 'Gaussian

distribution' into a corresponding class. This explicit definition of distributions allows the API to then provide much more functionality. For instance, it would be possible to generate realisations from a distribution, calculate the moments of a distribution, or find the probability of exceeding a threshold. Tools could also be included to allow visualisation of the distributions (and statistics where possible). The API could also support the conversion between the different UncertML encodings.

**Extend the UncertML-enhanced SOS** — The SOS prototype in Chapter 4 provided a solid prototype to build upon. This implementation could be improved in two ways. First, an extension to the Filter Encoding Specification (OGC 04-095, 2005) could allow a user to query observations based upon uncertainty criteria. For example, all observations with a probability of exceeding a particular value could be returned, or all observations with a variance less than a given value. Combined with the existing spatio-temporal filters, this would provide users with an extremely powerful filtering tool. The second improvement to the UncertML SOS could be to integrate the work in Chapter 6. The locations of the sensors within a given SOS could be uncertain, and therefore encoding this uncertainty within the SOS would be beneficial. This could be achieved using uGML. The integration of uGML could also be combined with the filter encoding.

**Create a uGML profile** — uGML is capable of describing uncertain geometries by extending the GML geometry schemas. The number of geometries within GML is vast, and creating software that supports all of these is a complex operation. Creating a profile of uGML which only supports a subset of geometries could allow the development of supporting tools. For example, a uGML 'viewer' application could be developed that allowed users to visualise the geometries, and associated uncertainties, which uGML describes. This is only possible when operating on a small number of geometries from the GML schema. The extension to uGML could also see support for true rigid objects (i.e., support rotation as well as translation), which is currently not possible.

**Develop supporting schemas** — UncertML adopts a 'separation of concerns' philosophy. This stipulates that any information that is not directly relevant to the quantification of uncertainty should not reside within the UncertML schemas. For this reason UncertML does not provide methods for allowing users to specify concepts such as units of measure, location or temporal attributes. These concepts should be encoded by supporting schemas such as

GML. While this separation works well, ensuring UncertML remains uncluttered, delegating the creation of supporting schemas to users creates the same problems faced by O&M. A simple 'random variable' schema that provides elements to describe units of measure, space and time would improve interoperability.

**Allow for other methodologies** — UncertML describes uncertainties quantified using probability theory. However, as discussed throughout this thesis, there are other ways to quantify uncertainty. Future work could investigate the possibility of adding a fuzzy section to UncertML, as well as Dempster-Schafer or Bayes linear sections.

**Uncertainty-enabled Model Web (UncertWeb)** — The UncertWeb project[1] is a continuation of the INTAMAP project. However, rather than only providing a single interpolation service, UncertWeb is researching the possibility of developing a framework for chaining models using Web service technologies. UncertML plays a vital role in UncertWeb as the majority, if not all, inputs and outputs to these models are uncertain. UncertML must be capable of describing all the different uncertainties present within a host of models including air quality, weather forecasting and biodiversity models.

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

# Bibliography

Keith Ballinger, David Ehnebuske, Christopher Ferris, Martin Gudgin, Canyang Kevin Liu, Mark Nottingham, and Prasad Yendluri. WS-I Basic Profile Version 1.1. The Web Services-Interoperability Organization, 2006.

Geert Jan Bex, Frank Neven, and Jan Van den Bussche. DTDs versus XML schema: a practical study. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 79–84, New York, NY, USA, 2004. ACM.

Mike Botts, George Percivall, Carl Reed, and John Davidson. OGC Sensor Web Enablement: Overview and high level architecture. *GeoSensor Networks*, pages 175–190, 2008.

Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (Second Edition). W3C recommendation, World Wide Web Consortium (W3C), August 2006a.

Tim Bray, Jean. Paoli, C. M. Sperberg-McQueen, Eve Maler, and Yergeau François. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C recommendation, World Wide Web Consortium (W3C), August 2006b.

James J. Buckley. *Fuzzy statistics*. Springer Verlag, 2004.

P. A. Burrough. Development of intelligent geographical information systems. *International Journal of Geographic Information Science*, 6:497–513, 1992.

Jen-Yao Chung, Kwei-Jay Lin, and R.G. Mathieu. Web services computing: advancing software interoperability. *Computer*, 36(10):35 – 37, oct. 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1236469.

S. De Bruin. Querying probabilistic land cover data using fuzzy set theory. *International Journal of Geographical Information Science*, 14(4):359–372, 2000.

Edsger W. Dijkstra. *Selected writings on computing: a personal perspective*. Springer-Verlag New York, Inc., New York, NY, USA, 1982. ISBN 0-387-90652-5.

A. Donaubauer, T. Kutzner, and F. Straub. Towards a Quality Aware Web Processing Service. In *GI-Days*, 2008.

Gregoire Dubois and Stefano Galmarini. Introduction to the spatial interpolation comparison (SIC) 2004 exercise and presentation of the datasets. *Applied GIS*, 1:1–11, 2005.

M. Duckham, K. Mason, J. Stell, and M. Worboys. A formal approach to imperfection in geographic information. *Computers, Environment and Urban Systems*, 25:89–103, 2001.

Thomas Erl. *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, April 2004. ISBN 0131428985.

Thomas Erl. *Service-Oriented Architecture : Concepts, Technology, and Design*. Prentice Hall PTR, August 2005. ISBN 0131858580.

David C. Fallside and Priscilla Walmsley. Xml schema part 0: Primer second edition. Technical report, World Wide Web Consortium, October 2004.

R.. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. Technical report, The Internet Society, June 1999.

Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

P.F. Fisher. Probable and fuzzy models of the viewshed operation. In *Innovations in GIS: selected papers from the First national Conference on GIS Research UK*, pages 161–75, 1994.

P.F. Fisher. Models of uncertainty in spatial data. *Geographical information systems*, 1:191–205, 1999.

Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley Professional, October 1996. ISBN 0201895420.

S. Fritz and L. See. Comparison of land cover maps using fuzzy agreement. *International Journal of Geographical Information Science*, 19(7):787–807, 2005.

A Gokhale, B Kumar, and A Sahuguet. Reinventing the wheel? corba vs. web services. In *in The Eleventh International World Wide Web Conference. 2002*, 2002.

Denis Havlik, Thomas Bleier, and Gerald Schimak. Sharing sensor data with sensorsa and cascading sensor observation service. *Sensors*, 9(7):5493–5502, 2009. ISSN 1424-8220.

G. Heuvelink, J. Brown, and E. van Loon. A probabilistic framework for representing and simulating uncertain environmental variables. *International Journal of Geographic Information Science*, 21(5):1–11, 2007.

Gerard B. M. Heuvelink. *Error Propagation in Environmental Modelling with GIS*. Taylor & Francis Ltd, 1998.

Gerard B. M. Heuvelink and James D. Brown. Uncertain environmental variables in gis. In Shashi Shekhar and Hui Xiong, editors, *Encyclopedia of GIS*, pages 1184–1189. Springer, 2008. ISBN 978-0-387-30858-6.

G.J. Hunter and M.F. Goodchild. Dealing with error in a spatial database: A simple case study. *Photogrammetric Engineering and Remote Sensing*, 61(5):529–537, 1995.

ISO/FDIS 19115. *19115:2003 Geographic information – Metadata*. ISO, Geneva, Switzerland, 2003.

ISO/IEC 11404. *11404:1996 Information technology – Programming languages, their environments and system software interfaces – language-independent datatypes*. ISO, Geneva, Switzerland, 1996.

ISO/IEC Guide 98:1995. *Guide to the expression of uncertainty in measurement (GUM)*. ISO, Geneva, Switzerland, 1995.

ISO/TC 211 19101. *19101:2001 Geographic information – Reference model*. ISO, Geneva, Switzerland, 2001.

ISO/TC 211 19107. *19107:2003 Geographic information – Spatial schema*. ISO, Geneva, Switzerland, 2003.

ISO/TC 211 19109. *19109:2003 Geographic information – Rules for application schema*. ISO, Geneva, Switzerland, 2003.

ISO/TC 211 19113. *19113:2002 Geographic information – Quality principles*. ISO, Geneva, Switzerland, 2002.

ISO/TC 211 19114. *19114:2003 Geographic information – Quality evaluation procedures*. ISO, Geneva, Switzerland, 2003.

ISO/TC 211 19123. *19123:2004 Geographic information – Schema for coverage geometry and functions*. ISO, Geneva, Switzerland, 2004.

ISO/TS 19138. *19138:2006 Geographic information – Data quality measures*. ISO, Geneva, Switzerland, 2006.

Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.

E. T. Jaynes. *Probability Theory: The Logic of Science (Vol 1)*. Cambridge University Press, 2003.

Nicolai M. Josuttis. *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, 1 edition, August 2007. ISBN 0596529554.

G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, NJ, USA, 1st edition, May 1995. ISBN 0131011715.

W. Kresse and K. Fadaie. *ISO standards for geographic information*. Springer Verlag, 2004.

Steve H.L. Liang, Arie Croitoru, and C. Vincent Tao. A distributed geospatial infrastructure for sensor web. *Computers & Geosciences*, 31(2):221 – 231, 2005. ISSN 0098-3004. Geospatial Research in Europe: AGILE 2003.

Y. Liu, Q. H. Guo, J. Wieczorek, and M. F. Goodchild. Positioning localities based on spatial assertions. *International Journal of Geographical Information Science*, 23(11):1471–1501, 2009.

Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, April 2005. ISBN 0470870001.

A. Lucieer and M.J. Kraak. Interactive and visual fuzzy classification of remotely sensed imagery for exploration of uncertainty. *International Journal of Geographical Information Science*, 18 (5):491–512, 2004.

Ole Lehrmann Madsen, Boris Magnusson, and Birger Mølier-Pedersen. Strong typing of object-oriented languages revisited. *SIGPLAN Not.*, 25:140–150, September 1990. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/97946.97964. URL `http://doi.acm.org/10.1145/97946.97964`.

Noah Mendelsohn and Stuart Williams. The use of metadata in uris. World Wide Web Consortium, TAG Finding, January 2007.

R. Moats. Urn syntax. Internet Engineering Task Force RFC 2141, 1997.

Deshendran Moodley and Ingo Simonis. A new architecture for the sensor web: The SWAP framework. In *5th International Semantic Web Conference ISWC*, Athens, Georgia, USA, 2006.

Ashley Morris and Frederick E. Petry. UGML: an extension of GML to provide support for geographic objects with uncertain boundaries. In *7th International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences*, pages 794–801, 2006.

Michael Z. Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson. Developing web services choreography standards: the case of rest vs. soap. *Decision Support Systems*, 40(1):9–29, July 2005.

OGC 02-069. *OpenGIS Geography Markup Language (GML) Implementation Specification, version 2.1.2*. OpenGIS Implementation Specification. Open Geospatial Consortium Inc., Wayland, USA, September 2002.

OGC 03-003r10. *Level 0 Profile of GML3 for WFS*. OpenGIS OGC Interoperability Program Report-Engineering Specification. Open Geospatial Consortium Inc., Wayland, USA, October 2003.

OGC 03-014. *OWS 1.2 SOAP Experiment Report*. OpenGIS Discussion Paper. Open Geospatial Consortium Inc., Wayland, USA, January 2003.

OGC 03-028. *OWS 1.2 UDDI Experiment*. OpenGIS Interoperability Program Report. Open Geospatial Consortium Inc., Wayland, USA, January 2003.

OGC 04-049r1. *WCS Change Request: Support for WSDL & SOAP*. OGC Discussion Paper. Open Geospatial Consortium Inc., Wayland, USA, April 2005.

OGC 04-050r1. *WMS Change Request: Support for WSDL & SOAP*. OGC Discussion Paper. Open Geospatial Consortium Inc., Wayland, USA, April 2005.

OGC 04-060r1. *OWS 2 Common Architecture: WSDL SOAP UDDI*. OGC Discussion Paper. Open Geospatial Consortium Inc., Wayland, USA, August 2004.

OGC 04-094. *Web Feature Service Implementation Specification*. OpenGIS Implementation Specification. Open Geospatial Consortium Inc., Wayland, USA, May 2005.

OGC 04-095. *OpenGIS Filter Encoding Implementation Specification*. OpenGIS Implementation Specification. Open Geospatial Consortium Inc., Wayland, USA, May 2005.

OGC 05-007r7. *OpenGIS Web Processing Service*. OpenGIS Standard. Open Geospatial Consortium Inc., Wayland, USA, 2007.

OGC 05-094r1. *GML 3.1.1 CRS support profile*. OpenGIS Implementation Specification Profile. Open Geospatial Consortium Inc., Wayland, USA, 2005.

OGC 05-095r1. *GML 3.1.1 common CRSs profile*. OpenGIS Implementation Specification Profile. Open Geospatial Consortium Inc., Wayland, USA, 2005.

OGC 05-096r1. *GML 3.1.1 grid CRSs profile*. OpenGIS Implementation Specification Profile. Open Geospatial Consortium Inc., Wayland, USA, 2005.

OGC 05-099r2. *GML 3.1.1 simple dictionary profile*. OpenGIS Implementation Specification Profile. Open Geospatial Consortium Inc., Wayland, USA, 2005.

OGC 06-009r6. *Sensor Observation Service*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, October 2007.

OGC 06-010r6. *OpenGIS Transducer Markup Language Implementation Specification*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, December 2006.

OGC 06-028r3. *OGC Sensor Alert Service Candidate Implementation Specification*. OGC Best Practices. Open Geospatial Consortium Inc., Wayland, USA, May 2006.

OGC 06-049r1. *Geography Markup Language (GML) simple features profile*. OpenGIS Implementation Specification Profile. Open Geospatial Consortium Inc., Wayland, USA, 2006.

OGC 06-095. *Draft OpenGIS Web Notification Service Implementation Specification*. OGC Best Practices. Open Geospatial Consortium Inc., Wayland, USA, 2006.

OGC 06-121r3. *OGC Web Services Common Specification*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, 2007.

OGC 07-000. *OpenGIS Sensor Model Language (SensorML) Implementation Specification*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, July 2007.

OGC 07-002r3. *Observations and Measurements – Part 2 – Sampling Features*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, 2007.

OGC 07-014r3. *OpenGIS Sensor Planning Service Implementation Specification*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, August 2007.

OGC 07-022r1. *Observations and Measurements – Part 1 – Observation schema*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, December 2007.

OGC 07-036. *OpenGIS Geography Markup Language (GML) encoding standard: version 3.2.1*. OGC Standard. Open Geospatial Consortium Inc., Wayland, USA, 2007.

OGC 08-122r2. *Uncertainty Markup Language: UncertML.* OGC Discussion Paper. Open Geospatial Consortium Inc., Wayland, USA, 2008.

A. Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables, and Stochastic Processes*. Mc-Graw Hill, fourth edition, 1984.

Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful Web services vs. "big" Web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 805–814, New York, NY, USA, 2008. ACM.

George Percivall and Carl Reed. OGC Sensor Web Enablement standards. *Sensors & Transducers*, 71(9):698–706, September 2006.

J Perkal. On the length of empirical curves. In *Discussion Paper 10 (Ann Arbor, MI: Michigan Inter-University Community of Mathematical Geographers).*, 1966.

Erik T. Ray. *Learning XML.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, January 2001.

Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, Sebastopol, CA, USA, 2007. ISBN 0-596-52926-0.

B. D. Ripley. Modelling Spatial Patterns. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):172–212, 1977. ISSN 00359246. doi: 10.2307/2984796. URL `http://dx.doi.org/10.2307/2984796`.

D.J. Russomanno, C. Kothari, and O. Thomas. Building a sensor ontology: A practical approach leveraging iso and ogc models. In *The 2005 International Conference on Artificial Intelligence*, 2005.

W. Shi. A generic statistical approach for modelling error of geometric features in gis. *International Journal of Geographic Information Science*, 12(2):131–143, 1998.

Aaron Skonnard and Martin Gudgin. *Essential Xml Quick Reference: A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP, and More*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 4th edition, 2001. ISBN 0201740958.

Alain Tamayo, Carlos Abargues, Arturo Beltran, and Carlos Granell. Gathering statistics of XML Schema Usage in OGC Web Services. In *Second Open Source GIS UK Conference*, 2010.

Andrew Terhorst, Deshendran Moodley, Ingo Simonis, Philip Frost, Graeme McFerren, Stacey Roos, and Frans van den Bergh. Using the sensor web to detect and monitor the spread of vegetation fires in southern africa. In Silvia Nittel, Alexandros Labrinidis, and Anthony Stefanidis, editors, *GeoSensor Networks*, volume 4540 of *Lecture Notes in Computer Science*, pages 239–251. Springer, 2006. ISBN 978-3-540-79995-5.

T. L. van Zyl, I. Simonis, and G. Mcferren. The sensor web: systems of sensor systems. *International Journal of Digital Earth*, 2(1):1–14, 2008.

Steve Vinoski. Rest eye for the soa guy. *IEEE Internet Computing*, 11(1):82–84, 2007. ISSN 1089-7801.

Eric Van Der Vlist. *XML Schema*. O'Reilly Media, Inc., 1st edition, June 2002.

P. Walley. *Statistical Reasoning With Imprecise Probabilities*. Chapman & Hall, 1991.

Peter Walley. Inferences from multinomial data: Learning about a bag of marbles. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):pp. 3–57, 1996.

M. Williams, D. Cornford, L. Bastin, and B. Ingram. UncertML: an XML schema for exchanging uncertainty. In *GISRUK*, Manchester, UK, 2008a.

M. Williams, L. Cornford, D. Bastin, and B. Ingram. A conceptual model for uncertainty — UncertML. In *GI Days*, Muenster, Germany, June 2008b.

Matthew Williams, Dan Cornford, Ben Ingram, Lucy Bastin, Tony Beaumont, Edzer Pebesma, and Gregoire Dubois. Supporting interoperable interpolation: the INTAMAP approach. In *International Symposium on Environmental Software Systems*, Prague, May 2007.

Matthew Williams, Dan Cornford, Lucy Bastin, and Ben Ingram. Describing and Communicating Uncertainty within the Semantic Web. In *Uncertainty Reasoning for the Semantic Web Workshop*, Karlsruhe, Germany, October 2008c. 7th International Semantic Web Conference.

Matthew Williams, Dan Cornford, Lucy Bastin, and Ben Ingram. Exchanging uncertainty: Interoperable geostatistics? In P. M. M. Atkinson and C. D. D. Lloyd, editors, *geoENV VII — Geostatistics for Environmental Applications*, volume 16 of *Quantitative Geology and Geostatistics*, pages 321–331. Springer Netherlands, 2010.

Matthew Williams, Dan Cornford, Lucy Bastin, Richard Jones, and Stephen Parker. Automatic processing, quality assurance and serving of real-time weather data. *Computers & Geosciences*, 37(3):353 – 362, 2011. ISSN 0098-3004.

Hung-chih Yang and D. Stott Parker. Lightweight model bases and table-driven modeling. In *Proceedings of the 12th international conference on Database systems for advanced applications*, DASFAA'07, pages 740–752, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-71702-7. URL `http://portal.acm.org/citation.cfm?id=1783823.1783904`.

Dale L. Zimmerman, Jie Li, and Xiangming Fang. Spatial autocorrelation among automated geocoding errors and its effects on testing for disease clustering. *Statistics in Medicine*, 29(9): 1025–1036, 2010. ISSN 1097-0258.