THE UNIVERSITY *of* EDINBURGH

Enabling Scientific Data on the Web

Raymond Alexander Miłowski

Doctor of Philosophy

Institute for Language, Cognition, and Computation

School of Informatics

University of Edinburgh

2014

Abstract

Scientific data does not exist on the Web in the same way as the written word; reviews, media, wikis, social networks, and blogs all contribute to the interconnected nature of ordinary language on the Web. Network effects create additional value from seemingly minor contributions to the Web. But nothing such as this exists for scientific data. Simply put, within the Open Web Platform, we cannot currently turn and apply similar mechanisms for scientific work without great effort. Thus, the Web has not so far enabled Science as well as it has enabled dissemination and interconnection for the written word: to truly enable Science on the Web, we must endeavor to make data and its semantics first-class Web constituents.

This thesis focuses on solving this problem by enabling scientific data to exist on the Web in such a way that it can be processed both as viewable content and consumed data. Starting from the principles on which the Web has so far thrived, we propose solutions to enable complex data exchanges while preserving the Web as it stands. We introduce the Partition Annotate Name (PAN) methodology, which relies upon embracing the core architectural principles of the Web: name things with URIs; process common data formats; use common rules under a shared contract between publisher, developer, and consumer.

Acknowledgments

I dedicate this thesis to my mother and father, Carol and Raymond Miłowski, who instilled and encouraged a love of knowledge throughout my life. I have been fortunate to have encountered many others who added to their chorus. My long-standing friendship and support of my advisor, Henry S. Thompson, is no exception. Without your insight and encouragement, I wouldn't have started this research and I cannot express my gratitude enough.

To David Robertson and Michael Rovatsos, your support has been essential. At key moments, your reviews kept us grounded whilst providing welcomed insights.

To my friend Murray Maloney, at many different times you provided your superb editing skills in support of this endeavor. Your opinions and skill have been invaluable to me.

To my friend Vasilis Karaiskos, thank you for providing me a home away from home and wonderful company. It made the process of trotting across the globe so much more sane.

To my colleagues at Informatics, Sarah Luger, Annette Leonhard-MacDonald, Amy Isard, and many others, you were my essential social and cultural support. When traveling so far, it was always good to be welcomed into your fold.

Finally, and most important, to my family, Anya and children Max and Veronika, thank you for supporting me and enduring my long trips away from San Francisco. It was hard to be away so much, but it was even harder in reverse. I am grateful and indebted for the support and understanding; you have all my love for doing so.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own, except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification, except as specified.

Raymond Alexander Miłowski

October 24th, 2014

Table of Contents

List of Figures

Chapter 1

Science and the Ordinary Web

1. *Scientific data must exist in a usable form on the Ordinary Web to enable, for scientific data, that which is already enabled for the written word.*

2. *Web Architecture serves as our guide for a general solution.*

The Ordinary Web is that which we, as people who are capable of operating a Web browser, encounter every day; the typical Web pages, whether hand-crafted or machine-generated from megalithic database stores through complex data mappings; content organized by systems or on an ad-hoc basis; resources hosted by Web servers or large-scale consumer services; and, information, possibly discovered or disseminated by search engines and social networks. Over time, using the Ordinary Web has become increasingly easy. Tools and technologies have advanced to enable the ordinary person to publish information in their blogs, as comments or reviews, or to build their own Web sites with relative ease.

As the Web has evolved, scripting has evolved into widgets as user interface components, control or data processing semantics, and mashups as combinations of different Web resources; these facilities allow authors and publishers to create rich interactive content. As a consequence of these enhancements, the Web has enabled the deployment of replacements for desktop applications in the form of browser-hosted applications (e.g., e-mail clients, spreadsheets, etc.). These developments have resulted in a rich ecosystem of publishers, consumers, and data resources that is easy to discover, navigate, use, and explore.

When one endeavors to make the transition from being a consumer to being a publisher, the Ordinary Web extends to meet their needs. One can employ a simple hosted service to publish information on their own, use administrative interfaces to enable widgets, and embed scripts (third-party or ones of their own making) to enable "outside of the box" features. As users and publishers move from hosted services to their own servers and services, the need for technological sophistication may increase, but probably only modestly. Service providers and technology developers often tune their systems to make installation and administration straightforward; the backing systems of the Web scale up to the next level without massive efforts.

When the consumer-turned-publisher becomes wildly successful, the Ordinary Web extends with more technological sophistication in the form of clustering, load-balancing, and the consultants necessary to put them together. More importantly, the required processes are well-known, so the publisher can make sophisticated choices in the quality of the services they wish to provide, all at a certain economy of scale.

The Web ecosystem thrives, not just because of the "heavy lifting" by talented developers and businesses to provide the underlying technology. Those aspects are necessary to create a stable environment, but are not sufficient

to induce a thriving ecosystem. Rather, it is the architecture of the Web that makes this ecosystem possible and enables it to thrive. That is, a basic and enabling principle of the Web is the fundamental ability to name and link resources via URIs and to re-compose documents on the Web via the Web browser, as a standards-based platform.

For the written word, ordinary people (i.e., non-technical) can create information on the Ordinary Web that can be immensely useful. For example, there are 4,247,911 entries as of June 4th, 2013 in the English version of Wikipedia [1]; not only can anyone enhance or update Wikipedia, but they can also link to Wikipedia as a reference. Referencing or embedding snippets from Wikipedia enhances the information they are publishing.

The publisher's act of creating relationships between pages by linking declares an informal relationship on the Web between the referrer and the linked page (i.e., Wikipedia). A lot of information about the reference can be inferred from the information contained within the newly authored page. At a minimum, we can observe a simple referring relationship between or among pages, typically in the context of the author's native language. Moreover, within the actual markup that encodes the reference there may be more information that explicitly defines the relationship with useful semantics that a receiving application can process.

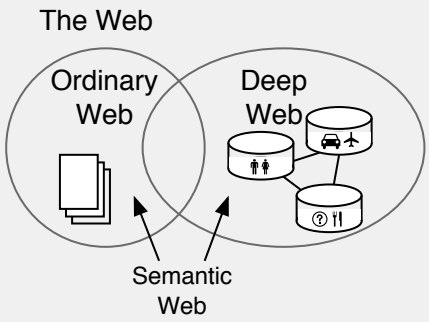
Search engines and other semantic inference machines derive knowledge from the Web by understanding the relationships between the written words, their context in the document's markup, and the linked structure of the Web. These systems are able to apply pipelines of processing to extract meaning, from the context of the markup and the native language, to build massive Deep Web databases [2] employed by applications of search, relevance, or mapping. By participating in the Web in a seemingly minor way, the ordinary

user is creating value on the Web; that value is packaged into services that also operate on the Ordinary Web.

How Deep Web services are provided is hidden behind a complex pipeline of massive annotation and enrichment processes that can only be orchestrated by sufficiently large and well funded entities. That is, these Deep Web services use extensive infrastructure that is not commonly available on the Ordinary Web, nor provided to ordinary individuals. Yet, although these Deep Web services are not available to the ordinary user, the information artifacts they contain can be accessed through the Ordinary Web via "portals" (e.g., Google Maps).

In general, people have benefited enormously from this combination of the Ordinary Web and Deep Web services. In fact, the amount of knowledge of the written word, crowd-sourced opinions, social networks, and other human generated data is enormous, distributed, and well partitioned. Deep Web services, such as mapping services that expose mash-ups of the

One Web, Three Labels



The Ordinary Web is the collection of resources we interact with through browsers and originally implemented as simple files. As services evolved, databases were increasingly used to publish resources on the Web. The Deep Web is a further evolution where information is harvested from the resources on the Web and used to create deep databases of information and metadata that surfaces again on the Web as resources. In among all of this is the Semantic Web [16] that provides annotations or other information resources useful to both viewers and consuming Deep Web services.

There is only one Web. All these different kinds of resources participate in the same Web that is linked together by naming (URIs), protocols (HTTP), and common formats (HTML).

Web and GIS data, could not exist unless they were embedded in and harvested from information that already exists on the Ordinary Web. That is, there exists an enormous economy of scale in allowing people to publish information in simple ways so that it can then be easily harvested by some set of criteria that the publisher may not have even anticipated. The information published on the Web is informal, often unreliable, possibly incorrect, but, when taken as a whole, very valuable.

A large part of the value demonstrated by Deep Web services is derived from architectural principles outlined in *Architecture of the World Wide Web, Volume One* as published by the *World Wide Web Consortium (W3C)* [3]:

1. We name resources on the Web with URIs.

2. We interact with resources via URIs and over protocols such as HTTP.

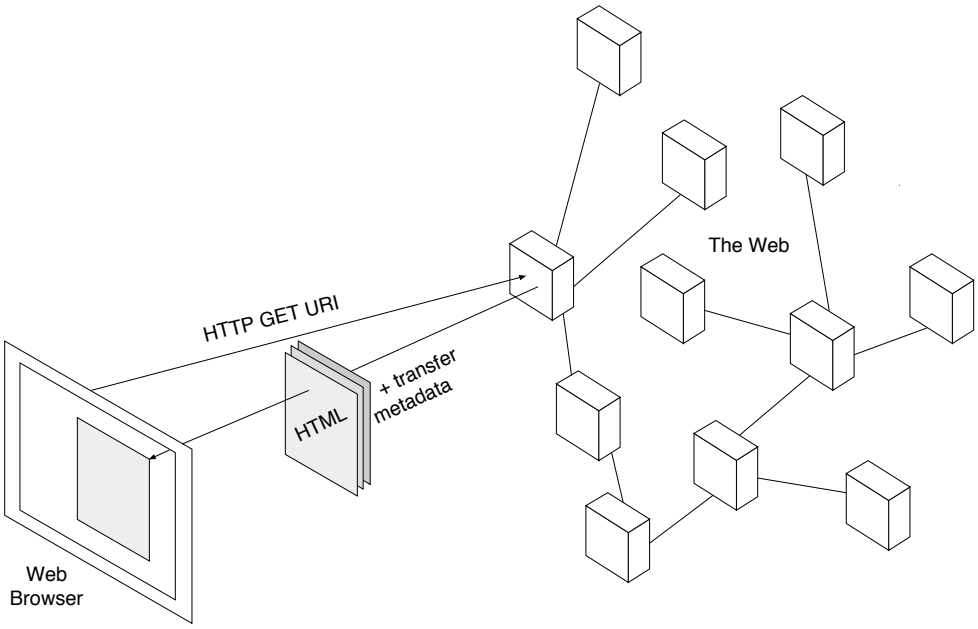
3. We use common data representations such as HTML or XML.

4. We use metadata, from both the protocol and data representation, to understand how to process representations.

5. We use annotations, encoded in markup and links, to discover new information such as related resource locations on the Web.

The conventional view of Web architecture is a simple relationship shown in *Figure 1.1, Conventional Web Architecture*, which illustrates the basics of principles (1) through (3). We interact with resources on the Web via names, a URI, that the publisher has assigned to them. Using protocol requests, such as an HTTP GET, a common representation, such as HTML, is retrieved by the Web browser. The metadata packaged within the transfer protocol provides sufficient information for the Web browser to recognize the document as something it can process.

Figure 1.1 Conventional Web Architecture

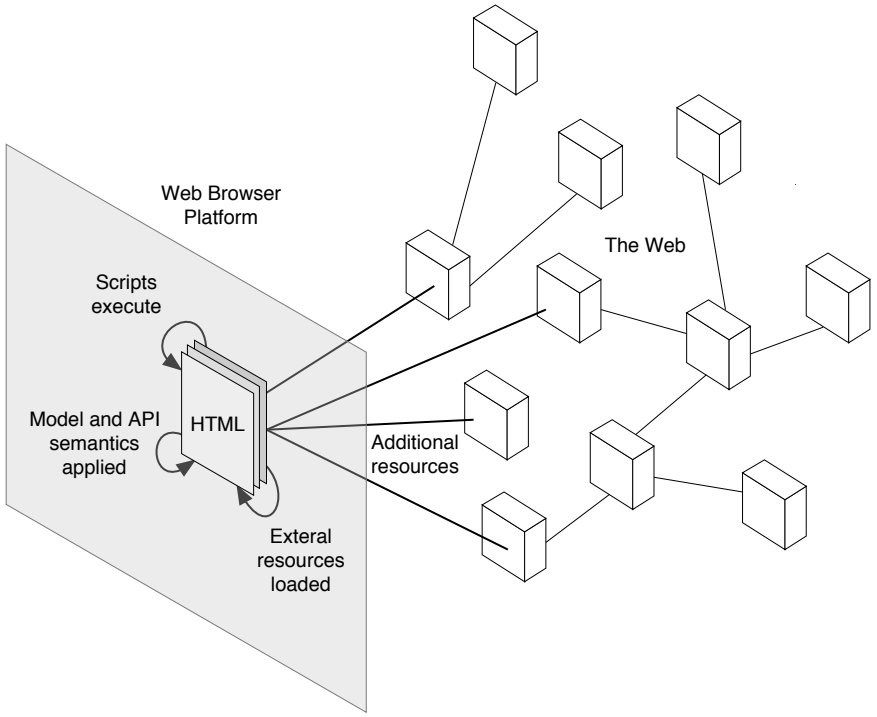


What happens within the Web browser (or Web crawler, or any other user agent) is a bit more complicated. By inspecting the transfer protocol metadata, the receiving system is able to decode the entity body into the correct sequence of bytes (and possibly the correct sequence of characters depending on the media type received). The browser uses the metadata about the type of content of a resource representation to determine its registered media type, which then allows the browser to understand how to process it. Most browsers support the well-known media types, such as HTML, SVG, and various image and video formats. In the normal use case, the intended action is to render the document for presentation to the user. Many other media types may result in useful but degraded renderings, such as presenting a tree-view of XML or displaying text.

We can consider the HTML vocabulary to be the intrinsic language of the Web [4]; a specific processing model is applied that extends what we can do with principles (3) through (5) as shown in *Figure 1.2, The Intrinsic Web*. When a Web browser loads an HTML document, it applies the processing model for

HTML to the markup received. The document is ultimately loaded into a local structure called the DOM (Document Object Model) [5] where specific HTML-oriented APIs are available, in addition to the regular DOM API.

Figure 1.2 The Intrinsic Web



During HTML document processing, certain markup causes different resources, such as images, scripts, or stylesheets to be discovered and loaded for further processing. Depending on the documented semantics of the markup, these resources may cause embedded media objects to be displayed, affect the general processing of the document, or affect the rendering of the document. As each resource becomes available, and assuming that the transfer rate is sufficient, the browser adjusts the rendering presented to the user such that the user barely notices the change.

Scripts are among the resources that may be retrieved in the course of processing an HTML document; scripts are executed in the order in which they are encountered. That is, if the script is completely embedded within

the document, it executes immediately. Otherwise, the scripts execute after their representations are loaded from the Web, and generally in the order they occur in the document.

These various processing actions within the Web browser operate together to provide a shared model of a platform that users and developers have come to rely upon. It is the common representation of principle (3) which allows principle (4) and (5) to be extended by discovering additional metadata and resources. Each link to an additional resource (e.g., a script or stylesheet) is named by URI and associated with metadata from the markup within the document. The role that the referenced resource plays is discovered by traversing the document and inspecting known element names (e.g., `script` or `link` elements) where the information within that sub-tree, at that context, provides sufficient metadata or information to proceed (e.g., execution of an embedded script or loading and executing an externally referenced script resource).

Using the architectural principles of the Web and the processing models of intrinsic vocabularies, the Web has extended from the written word successfully into the world of commerce, mapping, and social networking. While there are many innovations that have driven these adaptations, the connections among the collections of written information remain the fundamental underlying mechanism. Even so, the Web has failed to reach into scientific data in the same way. The publication and use of scientific data over the Web is the focus of our interest in this thesis.

1.1 Specific Outcomes

The PAN Methodology presented in this thesis addresses the critical issue of publishing scientific data as "Web-native" resources to satisfy the scientific method's requirement for a traceable and computable record of the data used in reportable scientific research and analysis. The Web is used as a both a mechanism and representation of the various data artifacts produced during the process of scientific endeavors. Subsequently, we demonstrate how this technique is usable for both dissemination and computation at the same time.

The methodology addresses, enables, and results in specific outcomes that can be summarized as:

- application of architectural principles of the Web that enables dissemination, partitioning, naming of potentially large data sets;
- enhancement of interchange by novel application of annotations within Web resources that enables local processing of information;
- computation and network effects over the Web by use of the browser as a platform.

These outcomes combined enable the standard Web browser to become a powerful computational tool in addition to its more traditional role of browsing, navigating, and rendering information for human consumption.

1.2 An Exemplar of Science on the Web

As we have already observed, science on the Web has not yet penetrated into the deep silos of data that are *de rigueur* in scientific analysis. Lacking

are reliable and robust mechanisms for naming, retrieving, analyzing, and annotating data down to its individual data elements. Limited linking capabilities and a lack of semantic relationships has hindered scientific exploration and analysis on the Web.

Considering such scientific endeavors, we return to Wikipedia, where one can discover a plethora of entries about scientific subjects. This seems only natural for a free encyclopedia. As Wikipedia's reputation has evolved, science-oriented organizations have considered using Wiki entries as a primary tool for making databases of scientific information available to the general public.

For example, WikiProject RNA [6] is a successful project which uses the same wiki infrastructure as Wikipedia to maintain a database of RNA families, their categorization, and other related information about RNA structures. The result is an open database of information which has been endorsed by a number of scientists and organizations [7].

There is a problem with this approach as it is currently implemented. While a human can access the information provided via a simple Web browser, a data processing application can do little more than follow links from one page to another. In fact, applications have trouble distinguishing between links to RNA information and links to elsewhere on Wikipedia or the Web. Moreover, what appears to be structured data within the entry pages is actually not much more than regular text or images. Often, the result is that an individual data element is not identifiable and so is not processable by an application; there is no assumption of the context of a written native “scientific” language that an application can use to process the entry and glean additional information.

The domain of science has not benefited from the success of the Ordinary Web to the same extent. While there are specific stories of successful Web-based scientific endeavors, the amount of "heavy lifting" necessary to get the data

out onto the Web, and usable by a broad range of individuals, is enormous and rarely generalizable. That is, there is currently no economy of scale.

1.3 Web Principles and Scientific Data

We are led to ask: Are the architectural principles of the Web somehow wrong for Science? Are we attempting to make the Web do something it cannot? The answers to these questions come from understanding what principles (3) through (5) actually mean.

A large part of the problem for Science on the Web is a lack of specificity. It is customary in writing to make indirect and fuzzy references to information; humans rely upon visual cues like adjacency of columns of data in a table to infer relationships; it turns out that these cues can be sufficient for Deep Web services to provide useful information later on. However, the job of Scientists is to take known facts and new information, and to use insight and the Scientific Method to produce new data, observations, and possibly generate new facts. The scientific gathering and publishing of data needs to be done in a repeatable way that is reasonable to replicate along with requisite precise semantics; natural language text is rarely suitable for Science on the Web.

From the perspective of the Web, the process of scientific endeavors produces a trail of data artifacts and some of these data sets may be very large. For any scientific process to be repeatable, those data artifacts must be available to be browsed, navigated, and interrogated by automated systems. The collection of the various aspects and information content of these kinds of data artifacts is what is meant by *scientific data*.

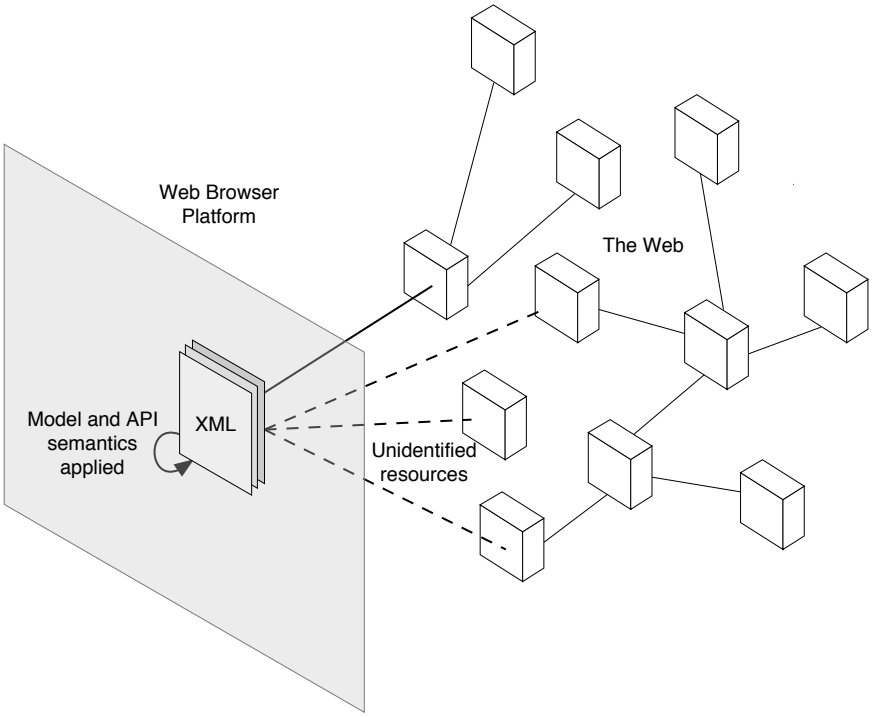
These scientific data collections contain artifacts of the acts of the carrying out the scientific method that are often collections of observations or experimental data themselves. These sub-collections contains measurements of quantities, details about how they were measured, and other annotations critical to interpreting the data correctly. Further, they may also contain important metadata that describes the subject, methodologies, and mechanisms involved. Consequently, the data must maintain its context to be fully processed correctly.

Yet, when scientific data is published on the Web, HTML alone, as the *lingua franca* of the Web, is insufficient to provide a way to uniquely identify data, and the terms used with the data, for human readers and applications. While XML could be used to help solve this, it is not really an intrinsic language of the Web in the same way as HTML because it lacks operational semantics in a typical Web browser.

The processing model for XML, shown in *Figure 1.3, XML on the Web*, is quite limited within the Web Browser. When an XML document is loaded by a typical Web browser, there is little to rely on within the shared platform between developer and user. The DOM is limited to generic markup semantics; particular XML vocabularies are not specifically recognized. As a result, scripts are not automatically executed, embedded objects are not automatically loaded or rendered, and neither are links identified. Even though there are standards for some of these semantics, they have not yet been implemented across commonly deployed browser platforms [4].

As it stands, applications have a hard time processing XML representations of resources. Failing to consider the XML resource representation's semantics and expected processing rules serves no purpose and obfuscates new information by making it undiscoverable. In the end, XML alone represents a poor choice for initiating a rich, shared experience over the Web.

Figure 1.3 XML on the Web



There are alternate representation choices, such as JSON [8], that suffer from similar issues. When any semi-structured data is transferred to the Web browser, it must have intrinsic semantics to operate upon. Even though the chosen markup may be human readable with natural language labels, this does not help the format become machine readable. As such, JSON suffers from the same pitfalls as XML on the Web and within the Web browser, but with much less standardization.

When we choose humans over applications, we can use current methodologies on the Ordinary Web as demonstrated by WikiProject RNA. We pass through principles (3) through (5) as the browser renders the HTML, understands how to process it, and new locations are made available to the reader within the rendered text. The user's experience is only limited by time and their willingness to explore.

If we want to alleviate limitations on time and make better connections for people, while also enabling applications to discover and use scientific information by applying the architectural principles of the Web, we need something more. The information that is presented to the user needs more underlying metadata and context so that local applications, embedded within the browser, can enhance their experience. At the same time, for information to be utilized locally, we need the same harvesting of information outcomes that enable discovery and processing by Deep Web services that currently exist on the Ordinary Web.

1.4 The Hypothesis

This leads us to a hypothesis:

> *The Web has not so far enabled Scientific Data as well as it has enabled dissemination and interconnection for the written word: to truly enable Science on the Web, we must make data and its semantics first-class Web constituents.*

Some may counter that there are "big science" activities on the Web (e.g., the Human Genome Project, weather modeling/forecasting, planet hunting, etc.) and that their data is available *en masse*. There are large biological databases, large amounts of astronomical data, geospatial data sets, meteorological data of all sorts, etc., that are all available for download, and offline processing. Some of these data sets have scientifically useful derivatives on the Web, but their data is not really on the Web as individual resources named with URIs. Instead, the data is packaged in archives that are accessible via the Web and intended for offline processing.

The mere size of the data sets involved is daunting to an individual and requires large databases or storage systems to hold and process. The data is often encoded in uncommon formats, violating principle (3), and often requires sophistication in both domain knowledge and technical skills, violating principles (4) and (5). Moreover, software capable of processing the data may be unavailable to the average user. In spite of these impediments, there remains tremendous value in the ability to "paddle around in the data" in the same way that one can explore the written word on the Web. Professional and amateur scientists alike would benefit from exploring data as easily as we can explore the written word today.

There is plenty of small-scale science that is enormously important. For example, field biologists often traipse through the woods recording observations of species along with annotations of climate, environment, geospatial coordinates, etc. Their data sets are small and akin to those of a person who publishes a blog entry or single Web page. Yet, this information is not easily discoverable and used on the Web. If it was on the Web and at scale, massive amounts of ecological information could be harvested from the Web about their activities and observations.

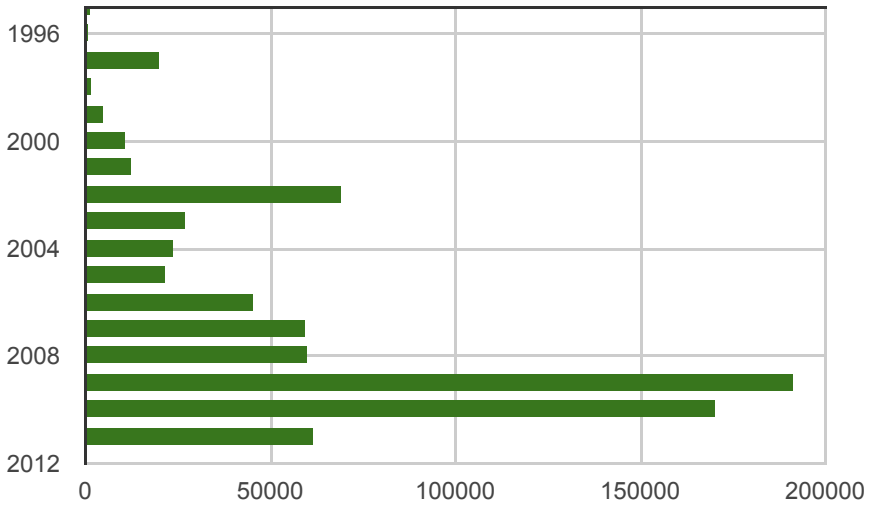
We observe that Science often involves annotation of data and reproducibility. Our goal is to enable semantically rich annotations in the Ordinary Web to enable principles (3) through (5) for scientific data sets. Ideally, we would do so with a clear path for scaling, from the miniscule to massive, by using the distributed nature of the Web. The resulting data sets will be identifiable, categorized by domain, processed by people and tools alike, and results can be directly reproduced on the Web. This gives a direct benefit to both the publisher and consumer in scientific domains, while enabling others to build Deep Web services for more complex analysis.

1.5 Existing Approaches to Data

When successful, current forays to find and use scientific data sources often end at a website hosted by an affiliated organization where the original producer of the data has deposited a file or archive for download. Others have taken these artifacts, severing them from their original provenance, and produced useful Web applications. In all cases, the data formats used are very common within the specific scientific community but often predate the advent of the Web. These formats are intended to be downloaded and processed by more traditional desktop or server applications.

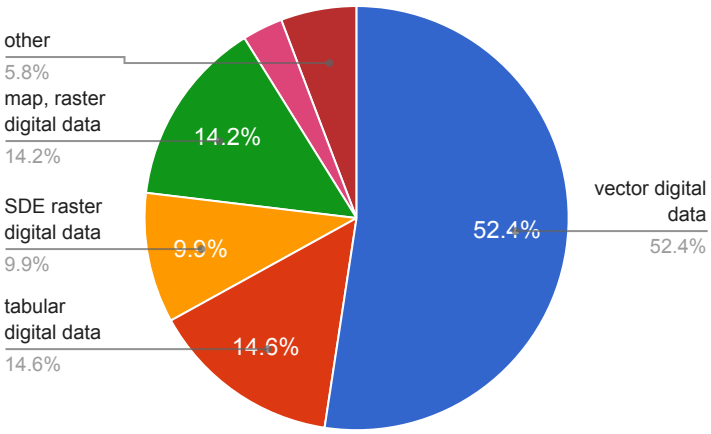
For example, the US Government created `data.gov` as a single portal for sharing data with a distinct subcategory for geospatial data. There are 441,360 distinct catalog entries for geospatial data, as of May 8, 2012, with the earliest published date of 1870, and with the number of data sets per year growing as shown in *Figure 1.4, Data Sets Published by Year*. Each catalog entry describes geospatial data collected by a government agency, much of which is map data (vector or raster), and points to a repository, maintained by the responsible agency, where you may be able to find the original data.

Figure 1.4 Data Sets Published by Year



As an experiment, we downloaded the entire catalog of metadata for each record. Each catalog entry contains some essential core metadata and each has a link to another metadata document, encoded in XML using a proprietary ESRI profile of ISO 19115 (Geographic information - Metadata) [9]. This ancillary metadata has a `geoform` element that contains a loose categorization of the data that is referenced by a URI contained in an `onlink` element. The contents of the `geoform` element consists of free form text. A summary of values found for the 441,360 catalog entries is shown in *Figure 1.5, Geospatial Data Formats*. Of these values, the top two categories (vector digital data and tabular digital data) represent 67% of the catalog entries followed by 27.2% for the various raster images, and 5.8% for other formats.

Figure 1.5 Geospatial Data Formats



Geospatial vector image formats, such as the common ESRI Shapefile format [10], often contain embedded tabular data (e.g., related dBase tables for shapes in ESRI format). As such, there is overlap between the top category of "vector digital data" and "tabular digital data". What percentage of the data is purely vector data (e.g., map boundaries, etc.) versus actual tabular data (e.g., data pinned to a position on a map) is uncertain, but it is reasonable to assume that some portion of the 52.4% of vector digital data also contains useful tabular data. One can infer that there are "hidden treasures" of scientific tabular data

within at least two thirds (i.e., tabular + vector data) of these resources, yet it can be very difficult to find the actual data due to our inability to follow from the catalog entry to an actual data resource.

Further complicating matters, when a data resource can be found, it is most likely not available in a Web-compatible format. As a result, the casual user's typical tool, the Web browser, cannot process the data format. This inhibits exploratory data analysis by "paddling around in the data." Instead, the user must download a possibly unknown data format and then switch modes to search for tools that read and display the data. Having broken the direct usability of the data, the user may discontinue exploration.

In contrast to `data.gov`, the International Virtual Observatory Alliance (IVOA) [11] develops standards to enable astronomers to exchange information directly across the Web in standardized formats. The basic method of exchange is the *Simple Cone Search* service that is designed to operate on cones defined by a celestial coordinate and a radius (see *Section 2.1.1, Simple Cone Search*). An astronomer invokes the service by constructing a URI consisting of a service base URI and query parameters that define the cone. A simple retrieval request results in an XML document with tabular data.

The design pattern used by IVOA has been successful in allowing tools to exchange data over the Web. The invocation method is simple and well-defined; the results are regular and encoded in a flexible Web-oriented format. Yet, for the casual user of these services, the fact that the browser cannot directly do much more than display a tree of rendered XML hinders exploration. More importantly, the layered semantics of the table columns cannot be used directly by this casual user.

The approach used by IVOA is worthy of further exploration and serves as an inspiration for this thesis. Their various services and formats are discussed in

Section 2.1, Astroinformatics and the IVOA Approach, where the VOTable format has some interesting ideas in terms of annotation and exchange of tabular data. Our view is that the approach used by IVOA needs to be generalized for use outside of astronomy.

1.6 Existing Common Formats

In examining principle (3) of the Web, we see that there are a variety of various formats for exchanging scientific data on the Web which fall roughly into three categories:

1. Proprietary or custom formats produced by tools commonly used within a specific domain.

2. Standardized formats utilizing syntax and encodings that are uncommon on the Web.

3. Standardized formats whose basis is HTML, XML, or another common syntax, along within encodings that are recognized by common Web browsers.

Given these three categories, in *Figure 1.6, Categorization of Common Formats*, we present a non-exhaustive list of formats common to scientific data repositories [12]. The domains chosen are exemplars and not exhaustive; it should be noted that "Geospatial" is more of a property of specific endeavors than an area of study. Nevertheless, the table shows a number of different attempts at data exchange and format definitions.

Formats that fall into Category 1 offer little hope of processing data on the Web until there is some agreement to use a standardized format. Such formats

proliferate within certain domains, such as Molecular Biology, often due to the fast pace of software innovation. In contrast, there are other areas, such as various uses of geospatial data, where proprietary formats, such as ESRI Shapefiles, have been used as a primary method of data exchange for many years. Formats that are proprietary are not necessarily publicly documented; this limits access to those who license specialized software or are willing to trust tools that have reverse-engineered the format.

Figure 1.6 Categorization of Common Formats

Domain	Category 1	Category 2	Category 3
Astronomy		FITS, TIFF	VOTable, PNG, JPeg, GIF
Biology	Too many ...	FASTA, ASN.1, et. al.	
Chemistry		CTab, SMILES	CML, JCAMP
Geospatial	ESRI Shapefiles		GML, KML
Meteorology		GRIB, BUFR	

In domains where there is recognized need for standardized exchanges of data, formats such as those in Category 2 are used. These formats are often based on simple textual representations, but some are binary such as the FITS format for image data [13]. While many of these formats are very successful and widely used, the common Web browser will not recognize them as formats that it can process.

Often there are also common alternatives for formats in Category 2. For example, FITS image data can be exchanged using a variety of common pixel image formats. Meanwhile, the metadata contained in the FITS image may not have a common mapping. It is these mismatches between generic formats and specific uses within a domain that drives the existence of formats within Category 2.

Also, it is often the case that formats in Category 2 have been designed for packaging and transporting large amounts of data between systems. That is, they represent whole data sets rather than specific partitions. As such, these formats may be inappropriate for the Web because the resulting resource is too large to be retrieved and processed by the Web browser.

Finally, there are formats in Category 3, which are distinguished by the fact that a Web browser has some default interpretation of the data received. For example, any XML format received by the browser has a default processing model applied to it. By definition, such processing is generic. That is, it knows nothing about the specific semantics of the domain vocabulary but it does allow the receiver a minimal ability to browse the raw data. As such, Category 3 often fails principle (3) of the Web as our expectations of intrinsic behaviors within Web browsers have changed.

It is in Category 3 where the use of browser-aware syntax can be applied to bring in domain semantics. If a format that is already understood by the browser is used, the same mechanisms used to bootstrap Web applications, within the browser as a platform, can be leveraged to incorporate domain semantics. These semantics can then enable domain specific processing and rendering of the data held within the format.

Scripting is a common mechanism for associating behaviors with markup constructs. Currently, browsers only provide scripting support in a limited number of intrinsic vocabularies (e.g., HTML/XHTML and SVG). As scripting is the main mechanism for bootstrapping Web applications within the browser, the semantics that can be defined beyond simple rendering or linking may be very limited. That is, there is much room for improvement in how browsers understand that a specific use of markup actually represents a specific kind of data with semantics that require loading and using additional resources from the Web.

1.7 Metadata and Assumptions

Turning to principle (4) of the Web Architecture, we observe that when a document is received by the browser, inspecting the metadata, and using the knowledge of the format discovered via the metadata, enables the browser to understand how to process the data. Initially, the browser relies upon the transport protocol to retrieve the `Content-Type` metadata of the document with a value of a media type (e.g., `text/xml`). If the value is a recognized media type, for which the browser has intrinsic knowledge of processing semantics, the browser invokes a chain of processing that leads to an anticipated set of behaviors.

Unfortunately, this is where formats in Category (3) often fall short. For example, while the IVOA VOTable format is an XML format and receives the media type `text/xml` or `application/x-votable+xml`, the browser lacks embedded semantics to distinguish it from among different XML dialects. The result is that the browser offers a generic treatment of XML rather than a display of tabular data.

A publisher can partially fix this problem by providing a link to a CSS [14] or XSLT [15] stylesheet. However, not all the capabilities inherent to rendering and providing the behaviors of HTML are provided by CSS. Not only are there limitations in rendering tables, but CSS lacks the ability to identify simple links within documents. Although XSLT can be used to provide partial solutions to these problems, it is a much less widely understood technology and its future in the Web browser is uncertain.

Finally, Category 3 formats such as VOTable and KML contain embedded tabular data. These table columns are labeled by identifiers that are authored by the data producer; these labels attempt to convey the role and identity of the column of information. Such labels (e.g., "Name" or "Location") are rarely

sufficient to serve as identifiers to be uniquely attributed to an owner on the Web, as they are not URIs.

All of these issues lead to a violation of principle (4) in so much as a processing application must make assumptions and process additional out-of-band metadata to understand the received format. The extensibility mechanisms of many of the formats do not use URI naming, in violation of principle (1); so, finding additional information on the Web is nearly impossible. The end result is that even when a common syntax is used, since the processing semantics are specific to the format and producer, the data remains more-or-less inaccessible to the ordinary Web user.

1.8 Implicit Linking

Once a segment of data has been received and processed to the best of the Web browser's ability, the browser (or layered application) finds more information by locating links with particular roles. A typical Web document contains links to many resources, each of which may play different roles. Some may be links to scripts, stylesheets, etc., that help provide application semantics; others are links to alternate representations. The remaining links are those that are often presented to the user for navigation within the resource or to other resources, or for some other use.

For example, in Category 3, the VOTable format contains a row for each observation. Often there are multiple columns whose content is a link to resources such as an image of the observation in various formats. The roles of these links can only be deciphered by inspecting and understanding the column header; this is not something intrinsically identified by the browser via the default processing model for XML.

When a link is identified by annotations rather than within markup as an HTML link, the browser or other receiving application has no ability to detect the links without specific knowledge of the VOTable format. This violates principle (5) because the link is implicit to the format. Ultimately, the ability to link to another resource needs to be a primitive of the syntax such that the syntactic identification of a resource link is inherent rather than a layered semantic.

1.9 Towards the Web

We have introduced the principles of Web Architecture; those principles will give us what we need for science on the Web. In the chapters that follow, we discuss current and historical attempts to use the Web for scientific endeavors. We also explore the technology of the Web to provide a grounding for our solution.

Based on the principles of the Web, we will be looking towards the Semantic Web [16] to provide additional support for scientific data. The Semantic Web uses the first principle of the Web (that we name things with URIs) extensively. Naming turns out to be critical not only for identifying and locating resources, but also for making assertions about them, and drawing inferences from them. The Semantic Web provides us with standard technologies for providing annotations and using them for inference.

The Semantic Web seeks to address the essential issue of knowledge representation and encodes information using graphs, where subjects and predicates are labeled with URIs. A subject is an abstraction of a resource (e.g., an author) and nodes in the graph are various subjects or literal values related by predicates (e.g., who authored a particular document). Recent

advancements allow such knowledge graphs to be intermingled as annotations within other representations on the Web as well as independent Web resources.

Rather than merely endorsing the use of the Semantic Web, we will describe a methodology for using semantic annotations in conjunction with the Ordinary Web. This methodology provides a way to bridge between the deep technologies of the Semantic Web as a processing platform and other commonly used techniques on the Web. Crucially, we attempt to re-purpose common formats for data and exploit the duality of data for viewing and processing, all within the Web viewed as a platform.

1.10 Overview

The PAN Methodology for publishing scientific data is formally presented in Chapter 5 and the intervening chapters provide the necessary background and motivation for the methodology. The following chapters (*Chapter 6, An Exemplary Implementation for Evaluation* and *Chapter 7, Comparisons with Alternatives*) evaluate the methodology by first examining what the methodology enables users of the Web to do within browsers with scientific data. Subsequently, there is a comparison of the implementation of the same interpolation process in two existing science work flow tools against doing so within the browser.

The next chapter, *Chapter 2, Existing Science on the Web*, provides an examination of various approaches previously used for publishing scientific data in the areas of astronomy, ecology, and various geospatial data sources from government agencies. These areas share the common property of often having time-series spatial data and these lend themselves well to partitioning.

As will be shown, partitioning data is a critical property of sharing data on the Web. Other areas of scientific data, such as genomic databases, could have been considered but do not necessarily have innate partitionable facets.

As this thesis is, in part, about the Web, we describe its development and principles in *Chapter 3, The Architecture of the Web*. In this chapter, we pay special attention to annotations formats and derive basic qualities of identifiability, extensibility, flexibility, and durability. We return to these qualities in *Chapter 4, Foundations for Science on the Web* where we apply them to specific existing formats and propose using common formats of the existing Web. The key insight is that we need annotations within these formats to process the scientific data being represented.

When we present and evaluate the PAN Methodology, we imagine being in two worlds; one of the professional scientist and one of the citizen scientist:

- Professional scientists, who often perform and act out the scientific method in various fields of research, organize their research around the collection, annotation, and analysis of experimental or observational data. Due to the nature of their endeavors, their data sets are often large, complex, and difficult to manage.

- Citizen scientists, often amateurs and non-professionals, often participate in scientific endeavors that are their own or via professional scientists. Individually, their view on data sets and the data they generate is much smaller in nature but, taken as a whole group, their data processing needs can also be large and complex to orchestrate.

For the professional, accessing and re-using existing data is already a difficult task that often requires special tools and skills; we wish to de-specialize the processing of data. The citizen scientist is in an even worse position in that

they many not have the resources to acquire the specialized tooling and they may also lack the skills necessary to deploy them; we wish to enable them to process data. The Web has proven itself successful in bridging such gaps for other communities and we seek to show how this can be done for scientific endeavors as well.

The final evaluation demonstrates that the Web browser is an effective tool for both the professional and citizen scientist to perform complicated tasks over data. An interpolation surface is shown to be computable in real-time over large geospatial areas using a reasonably straightforward combination of scripting in the Web browser. The result is the promise, for both the professional and citizen scientist, that libraries of functionality can be built and shared; much like how the Web, as a community, has done so in other non-scientific areas.

Chapter 2

Existing Science on the Web

3. *For scientific data, annotation enhances or replaces the role that natural language plays for the written word on the Web.*

4. *New representations for information create new interchange problems for existing tools.*

What are the roles that natural language plays for the consumer on the Web? The human consumer visiting a Web page is able to read the text and gain some information in that very specific context. For a computational agent, individual passages of text can be processed using Natural Language Processing (NLP) techniques to gain any number of specific records (e.g., named entities). For both consumers, the information is gained in a hypertext context and links to other resources (i.e., media, other Web pages, or sites) can be associated with both the information gained and the context.

Deep Web services leverage information gained in this way to associate vast databases of knowledge with the link structure of the Web as a whole. This

allows their services (i.e., search, maps, etc.) to surface newly synthesized views of information with links back to original sources. Most would agree that these services are useful to the average consumer on the Web.

While some scientific data sources can be found via this method, as mentioned previously in *Section 1.5, Existing Approaches to Data*, their representations are troublesome. The data themselves are rarely in Web-readable formats and so negates the ability of the human consumer to casually browse the data to gain any information. Further, applications discover very little information that they can process, either in syntax or via natural language descriptions, and so rarely traverse or process whatever data may be available.

Without replacing, or significantly augmenting, the role of language on the Web for scientific data, this might remain the *status quo*. Deep Web services are not able to differentiate between links to unidentified data from links to scientific data sets. Further, the metadata necessary to understand what is available at such links is not necessarily available to view or index.

There are numerous approaches for distributing data so that it can be automatically processed to some degree; these generally sort into two distinct categories:

1. The data resources themselves.

2. Metadata about data sets that includes information such as ownership, descriptions, and links to entry points for data.

The metadata category is very important and helps consumers, the casual interested individual, the professional scientist, or automated tool, to locate and understand data sets of interest. The natural language titles, descriptions, and other metadata provide the consumer with the ability to apply the same

search techniques as for other Web searches. This is certainly the approach that catalogs such as `data.gov` have provided.

Even though providing catalogs of information resources is a worthy endeavor, the metadata they contain tends to go stale. As previously discussed, entries in catalogs such as `data.gov` often have incorrect pointers or descriptions of data sets, especially in those data sets that are being actively maintained. It might be better that the metadata have provenance from the actual data set representations rather than as separately maintained entries, but the technical feats necessary to make this happen across government and scientific organizations would seem to be beyond reach, in the early 21st century.

Considering such issues, this thesis is more concerned with the data once it has been found. That is, the focus is on the first category and how the data resources are accessed and understood. There are enormous challenges in making data sets available and effectively useable once one has found an interesting description.

When a consumer visits a scientific data resource, the following is a minimal set of information that is useful to be able to accurately and precisely determine:

1. *To which category/domain of science does this information belong?* For various reasons, the data resource may be visited without the context that led the consumer to the resource. As such, it is very useful if the data resource declares some basic metadata about itself. Specifically, we would like to know in general terms (e.g., this is astronomical data) and then more specific terms relevant to professionals within that field (e.g., exoplanet data).

2. *What are the basic overall spatial, temporal, or other dimensions for the data?* Especially if the resource is a response to a query, the resource needs to be self-contained enough to describe what subset of possible data is represented. This gives applications heuristics as to what to process for display, what to store for later, and whether any specific queries have been completely satisfied.

3. *What does each property of each record mean in "scientific terms"?* Once a set of records and their properties are available for processing, the receiving application needs to know a variety of information about how it is represented. At the basic level, it needs to know how the data is encoded (e.g., an integer or date/time?). More specifically, the data properties need to be grounded in scientific units and semantics. For example, it is not sufficient to say a column of data is "temperature". Instead, it must be identified as "air temperature measured in Celsius with an accuracy of a tenth of a degree". That is, properties must be annotated with a subject of the measurement combined with the unit and other relevant information (e.g., statistical error).

As previously mentioned in *Section 1.5, Existing Approaches to Data*, and *Section 1.6, Existing Common Formats*, much of scientific data is represented in tabular form, of which there are many existing formats. The first two categories (non-standard and standardized/non-web) described are formats that are uncommon on, or incompatible with, the Web. The third category (standardized/web) uses technologies that can be easily used within the Web browser. We shall consider two generally-used formats for tabular data exchange from Category 3: VOTable for astronomical observations and KML for geospatial data.

2.1 Astroinformatics and the IVOA Approach

While astronomers had long been using the Internet to exchange data, primarily through FTP services, the lack of common interchange formats meant that systems had to deal with custom formats in addition to managing provenance and updates. Around 1998, the Strasbourg Astronomical Observatory developed the VizieR system [17] as an online database of astronomical catalogs. In *The VizieR database of astronomical catalogs* [18], the challenges of interoperable exchanges of information without a common format were discussed with the primary observation being that, while most data was tabular, the formats in which they were distributed varied widely. Even when the data was available in FITS format, the usefulness of that format outside of certain tools was limited. As conventions were developed for documenting each format in "Readme files", the increasing amount of data available from automated systems made this approach unworkable.

The VizieR system developed a set of metadata divided into three categories: descriptions of catalogs, descriptions of tabular data, and descriptions of columns. This set of "meta tables" allowed the system to consume data from various sources in an automated fashion and store it in a common format. The result was made available over the Web in a common format that included one of the first uses of XML for scientific data (i.e., at the time, XML had just become a W3C Recommendation [19]).

In conjunction with the VizieR system, two important ideas were developed: a proposal to standardize tabular data using XML and the concept of a Universal Content Descriptor (UCD). An XML format called *Astrores*, described in *Using XML for Accessing Resource in Astronomy* [20], is the direct predecessor to IVOA's VOTable format. It shares the idea of describing columns of data and then using a simple tabular format to encode the actual data. In the original proposal, the data was encoded as "comma separated

values" in a `CSV` element. The combination of the column definitions and quasi-tabular data allowed tools to interpret columns based on basic metadata such as units, data types, labels, and UCD labels.

Around the same time as the Astrores XML format development, the concept of a UCD was developed in *ESO/CDS Data-mining Tool Development Project* [21]. The idea was to have a controlled vocabulary of descriptors, in the form of simple text strings composed of a single word or "token", that systems could use to label columns regardless of the descriptive names, units, or data types used to encode the actual data. For example, two systems might label a column with "POS_EQ_RA", yet in one system use decimal degrees for the data and in the other use hours, minutes, and seconds. A receiving system attempting to aggregate the two data sources can now map these values at a higher level and then use encoding specifics to translate each datatype and syntax used. Also, a UCD allows systems to use human-readable labels for columns of data while providing machine-readable mappings.

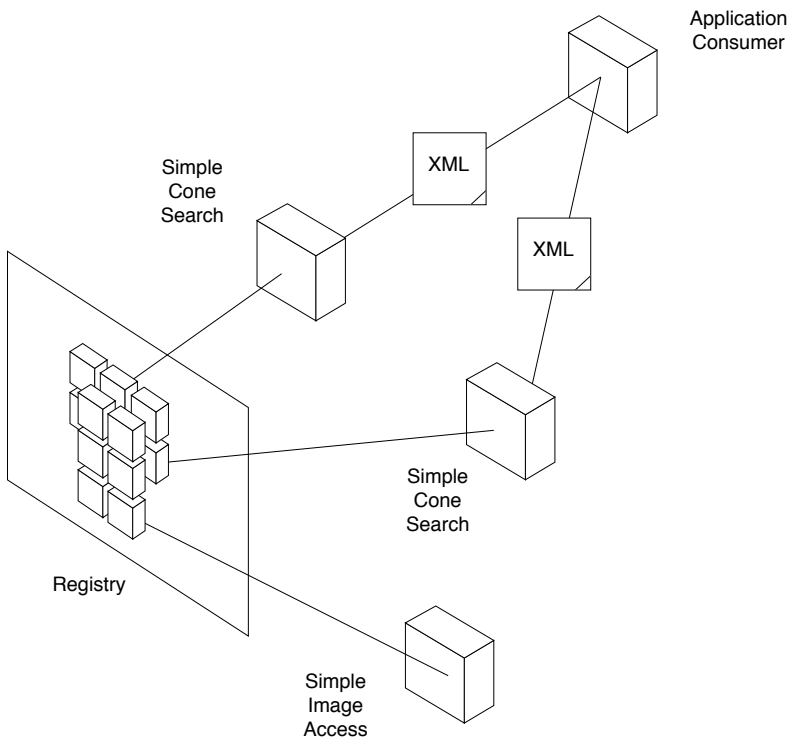
Unfortunately, the problems inherent in developing and agreeing to a common vocabulary proved to be too much for this approach. In *UCD in the IVOA Context* [22], the idea of getting agreement upon a single, non-extensible controlled vocabulary was described as "a nearly impossible task". This realization led to the development of the UCD1+ (called UCD2 in the reference), which maintained the notion of a UCD, but turned it into a mechanism for composing descriptors built from "atoms" and "words" derived from a well-defined and controlled vocabulary. This approach lessened the need for ultimate agreement and provided "some level between the fuzziness of natural language and the accuracy of attributes of data models" [22].

These varied efforts seemed to have led to the establishment of IVOA to develop the myriad of standards and architectures for exchanging

astronomical data that now exist. At their core is a simple Web-enabled service (see *Section 2.1.1, Simple Cone Search*) that provides the ability for applications or scientists to search for data by specifying properties of a cone. The data returned is a standard tabular format defined by IVOA called *VOTable* [23].

The architecture of the suite of standards that help define Simple Cone Search is show in *Figure 2.1, IVOA Architecture*. These basic components are: a registry of services, instances of actual services, and VOTable representations of astronomical data. A typical exchange starts at the registry and ends with retrieval and processing of VOTable XML documents.

Figure 2.1 IVOA Architecture



When an astronomer wants a particular kind of data, they can go to well-known portals (registries) and query for data sources. The tools they use may also have programmatic access to these portals. Contained within the registries are entries for services for specific kinds of data.

Each entry contains pointers (URIs) to a service and other descriptions of the service, including which specific protocol the service implements (typically HTTP). A user or application can search to find endpoints for specific kinds of astronomical data (e.g., observations, star charts, etc.). Once they have a set of endpoints for services, they can query them for data based on specific input parameters.

The registry contains different kinds of service information, many of which return their results in the VOTable XML format. As mentioned previously, some of these services are *Simple Cone Search* services while others return information about images or spectral lines. The user of these services must be knowledgeable of the service type and how the results are returned.

As a result, a user typically employs a tool to find data sources via the registry and aggregate results from a variety of sources for a particular region of the universe. The individual results come back in a similarly structured VOTable syntax. This allows the tool to use common infrastructure to either present the information to the user or otherwise process it locally.

Although tools have been built to integrate this registry model for finding services, such as Astrogrid, which came online in May of 2005 [24], alternative methods exist for finding and using IVOA services. Since many of the service protocols rely on a simple HTTP GET request with query parameters, users can directly access these services once they have a URI for the endpoint; they can simply construct a URI from the rules specified in the relevant specification.

Moreover, many providers of Simple Cone Search services provide their users with direct Web interfaces to the data. They provide a simple format that allows the user a more convenient way to input criteria via a Web browser and then invoke the service for them. Often, the Web form offers a control through which the user can specify the output desired, where, by default,

some HTML rendition will be provided for the data. Irrespective of whether the user requests the data in VOTable XML, the Web interfaces provide a professional or citizen scientist with an easier way to access the data.

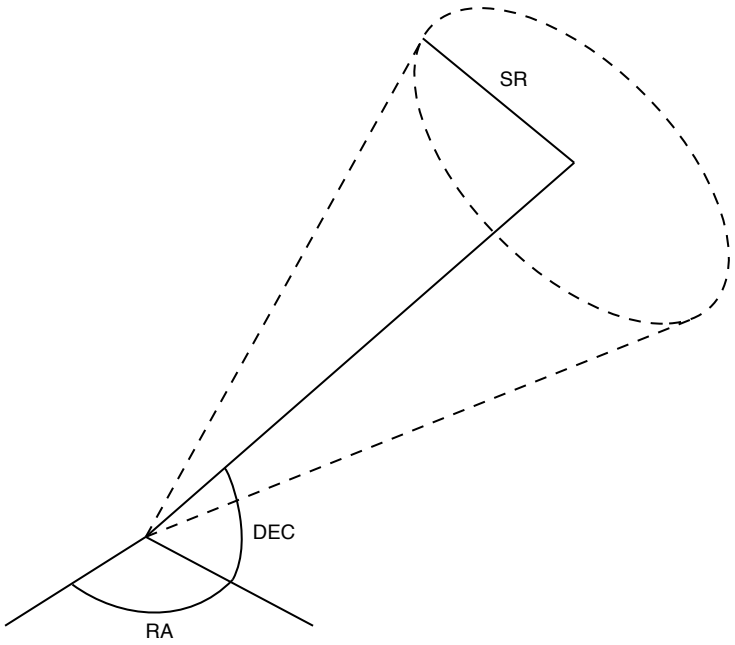
Unfortunately, there is a drawback to providing Web interfaces as forms: they hide the data service from potential application data uses as the consumer may only know of the form and not the backing service. In practice, the Web form and the service are often hosted under different names or path segments. Inspecting the form (i.e., using the infamous "view source") does not necessarily yield the raw URI of the data service. Instead, one is more likely to discover the address of the resource that processes the form and talks to the data service on one's behalf. As such, without first having found the service in the registry, the Web form may obscure the service endpoint URI from use.

2.1.1 Simple Cone Search

An application can access astronomical data over a Simple Cone Search [25] service via a simple HTTP GET request. By formulating the right set of query parameters, data can be retrieved for a specific area of the universe. These request parameters describe an open cone as shown in *Figure 2.2, Cone Query Parameters*, where the `RA` and `DEC` parameters (right ascension and declination) are equatorial coordinates and the `SR` is the radius of the cone in decimal degrees.

For example, the US Navy provides a number of star catalogs as IVOA Cone Search services. The USNO-B1 catalog [26] contains over one billion stars. The endpoint for the service is the URI `http://www.nofs.navy.mil/cgi-bin/vo_cone.cgi?CAT=USNO-B1` and the parameters `RA`, `DEC`, and `SR` can be appended to retrieve specific data about stars contained with the corresponding cone (e.g., `http://www.nofs.navy.mil/cgi-bin/vo_cone.cgi?CAT=USNO-B1&SR=0.25&RA=0&DEC=0` for a cone of radius

Figure 2.2 Cone Query Parameters



0.25° centered at [0°,0°]). The result is an XML document in the VOTable vocabulary.

Many services limit the radius to constrain both the size of the returned XML result and the elapsed response time (i.e., to avoid long-lived connections). For example, the USNO catalog is restricted to a one degree radius. As such, retrieving data for larger areas will require multiple requests.

2.1.2 VOTable for Data

While the VOTable XML vocabulary is most often used in response to a Simple Cone Search or other similar search service request, it can also be used to represent whole data sets. For example, services such as Galaxy Zoo provide their whole database as a downloadable set of VOTable XML documents. We will examine this format against the three criteria outlined at the beginning of the chapter (i.e., domain identification, dimensions, semantics of properties).

The VOTable format provides very little within the markup to identify itself as belonging to Astronomy or any specific area of study. The `VOTABLE` element may contain a single `DESCRIPTION` element that is string valued and typically contains a natural language description; while one can assume the subject is Astronomy by the mere fact that VOTable is in use, the specific subject within astronomy is generally not available.

The VOTable format does provide the ability to encode the request parameters in the response with a set of `PARAM` or `INFO` elements but there is no guarantee the response will contain these elements. Each of these elements has both name (`name`) and type (`datatype`, `xtype`, `unit`, `ucd`, `utype`, and `type`) attributes that can be used to identify the dimensions of the results and/or the query that invoked the response. These elements are not required and there is quite a bit of flexibility in their representation but in common usage they often provide the region of space in which the results were observed.

Finally, VOTable has extensive support for defining what each column of data represents, their underlying data types, domain usage, and grouping. Each column has a `FIELD` element that defines the column's metadata; one of the more interesting components is the UCD [27]. These UCD values provided detailed domain-specific information about the kind of data contained in the column.

The UCD is a sequence of "word tokens" separated by semicolons (;) and made up of "atoms" separated by periods (.). For example, the value "`phys.temperature;instr`" represents the temperature of an instrument and is composed of the words "`phys.temperature`" and "`instr`" and the atoms "`phys`", "`temperature`" and "`instr`". The sequence of words, taken together, describe the data type and provide the ability for systems to create new types from existing vocabularies.

The list of standard words has been defined by IVOA [28] and covers many areas of measurement, physical attributes, and statistical measures. For example, the "maximum temperature of an instrument" can be described using standard words as "`phys.temperature;instr;stat.max`". The words give a receiving application concrete information about the column of data: a temperature measure, of an instrument, a maximum.

For example, in *Figure 2.3, VOTable Example*, taken from the VOTable specification, the document lists 3 galaxies with their position, velocity and error, and their estimated distance. Each column has a definition in the header, each with the UCD, datatype, and unit information. Within the table itself, only simple string values are contained within each table cell.

Figure 2.3 VOTable Example

```
<?xml version="1.0"?>
<VOTABLE version="1.3" xmlns="http://www.ivoa.net/xml/VOTable/v1.3">
  <RESOURCE name="myFavouriteGalaxies">
    <TABLE name="results">
      <DESCRIPTION>Velocities and Distance estimations</DESCRIPTION>
      <FIELD name="RA"   ID="col1" ucd="pos.eq.ra;meta.main"
             datatype="float" width="6" precision="2" unit="deg"/>
      <FIELD name="Dec"  ID="col2" ucd="pos.eq.dec;meta.main"
             datatype="float" width="6" precision="2" unit="deg"/>
      <FIELD name="Name" ID="col3" ucd="meta.id;meta.main"
             datatype="char" arraysize="8*"/>
      <FIELD name="RVel" ID="col4" ucd="spect.dopplerVeloc"
             datatype="int" width="5" unit="km/s"/>
      <FIELD name="e_RVel" ID="col5" ucd="stat.error;spect.dopplerVeloc"
             datatype="int" width="3" unit="km/s"/>
      <FIELD name="R" ID="col6" ucd="pos.distance;pos.heliocentric"
             datatype="float" width="4" precision="1" unit="Mpc">
        <DESCRIPTION>Distance of Galaxy, assuming H=75km/s/Mpc</DESCRIPTION>
      </FIELD>
      <DATA>
        <TABLEDATA>
        <TR>
          <TD>010.68</TD><TD>+41.27</TD><TD>N 224</TD>
          <TD>-297</TD><TD>5</TD><TD>0.7</TD>
        </TR>
        <TR>
          <TD>287.43</TD><TD>-63.85</TD><TD>N 6744</TD>
          <TD>839</TD><TD>6</TD><TD>10.4</TD>
        </TR>
        <TR>
          <TD>023.48</TD><TD>+30.66</TD><TD>N 598</TD>
          <TD>-182</TD><TD>3</TD><TD>0.7</TD>
        </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

The field with identifier `col5` has the UCD "`stat.error;spect.dopplerVeloc`" and the column entries contain the statistical error for `col4`. These two columns can be associated as a group by the addition of the markup in *Figure 2.4, VOTable Group Example*. An application can now read the association between the columns and thereby infer information from the UCD labels provided (e.g., `col5` is the statistical error for `col4`).

Figure 2.4 VOTable Group Example

```
<GROUP name="Velocity">
   <DESCRIPTION>Velocity and its error</DESCRIPTION>
   <FIELDref ref="col4"/>
   <FIELDref ref="col5"/>
</GROUP>
```

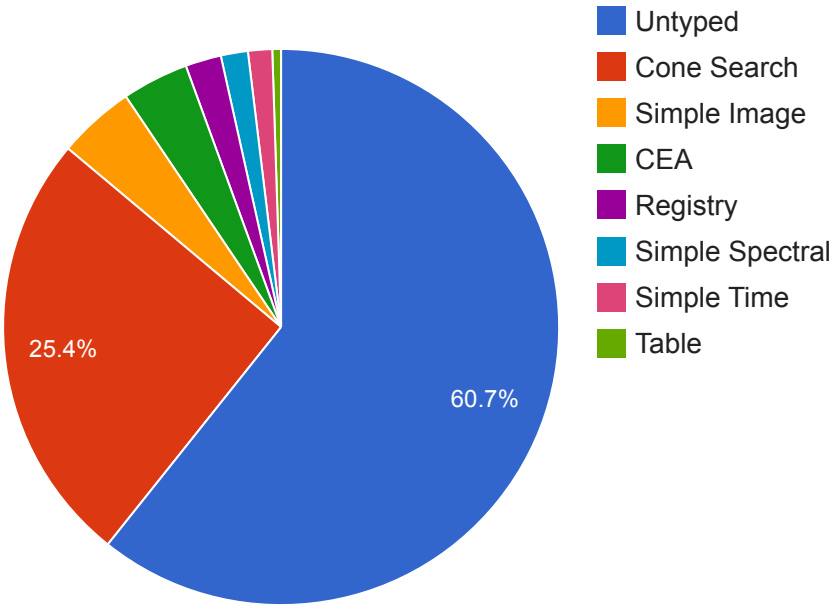
2.1.3 IVOA in Practice

IVOA maintains a "registry of registries" at `http://rofr.ivoa.net` which provides a starting point for finding IVOA services. As of October 2013, there were 20 registries, of which one was not found at the IVOA registry of registries. In total, these registries described 1232 resources with 2352 capabilities. Of these resources, 370 were merely informational and did not describe any Web-based capability.

A "capability" describes a Web-oriented service that is typically serviced by an endpoint interface (a URI) upon which a particular protocol can be applied. For example, a Simple Cone Search service is typed within the registry as the capability `{http://www.ivoa.net/xml/ConeSearch/v1.0}ConeSearch` and typically uses the `{http://www.ivoa.net/xml/VODataService/v1.0}ParamHTTP` interface protocol (or newer versions) for the endpoint. Other capabilities include Simple Image Access, registry services, etc., and many are tied to specific IVOA standards.

As shown in *Figure 2.5, IVOA Resource Capabilities*, 1428 (60%) of the resources have untyped capabilities even though they have interfaces that are partially described within the registry. These services are, presumably, unusable by automated tools without further instruction by a knowledgeable user. For example, many of these services use the "`ParamHTTP`" interface protocol, and so, once the query parameters are known, the service may be very valuable to a consuming scientist.

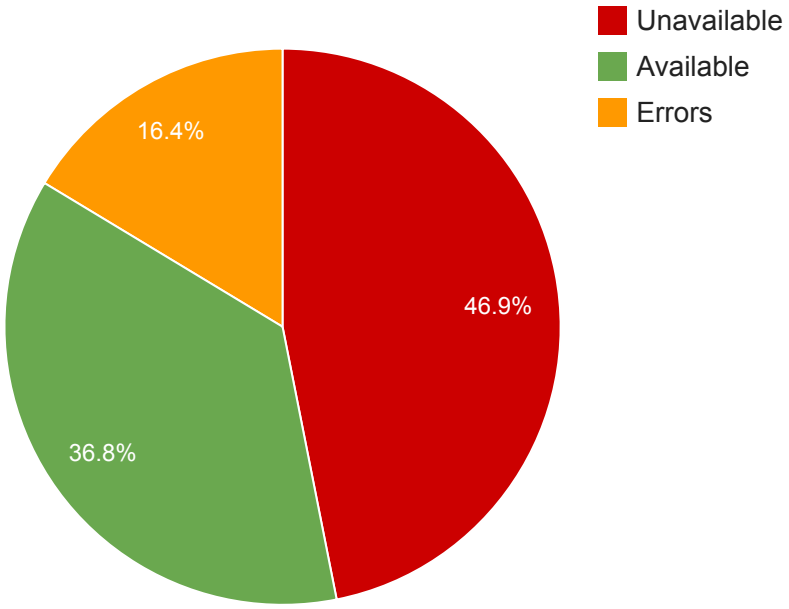
Figure 2.5 IVOA Resource Capabilities



Inspecting these capabilities at the next level down (the interface protocol) reveals that the next largest grouping is Simple Cone Search services. Within the registry, the description of each of these services contains a test query. An automated process was used to test each of the 593 registered "`ParamHTTP`" interfaces for each of the Cone Search capabilities with the results shown in *Figure 2.6, Service Availability*, where the final number of available services was limited to 218 (37%) of the 593 tested. It is worth noting that some of the services returned only error information indicating that the Simple Cone

Service was available via the protocol but unable to actually return any data for various reasons.

Figure 2.6 Service Availability

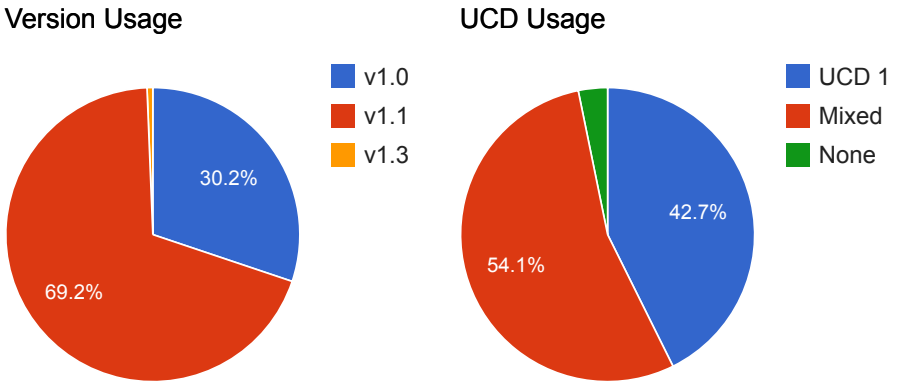


Most importantly, for the services that returned data, a variety of versions of VOTable XML markup was returned. As shown in *Figure 2.7, VOTable Usage*, three different versions of VOTable are returned by the various services. Unfortunately, each version has its own XML namespace; consequently, services must contend with processing similarly structured markup with different names.

In addition, the use of the more advanced UCD+ syntax was limited in that a majority of the services provided a mix of UCD1 and UCD1+ annotations. The consequence being that a receiving application must be able to recognize each syntax and then, hopefully, be able to normalize to UCD1+ words.

Following criticisms from within the IVOA community related to the shortcomings of UCD1, the lack of adoption of UCD1+ since 2005 (9 years later) is unfortunate. We take this to mean that either consumers do not need

Figure 2.7 VOTable Usage



the concept of UCD annotations or they have found ways to work around and use services that still use UCD1.

Further, the existence of successful Web-based projects such as Galaxy Zoo demonstrate that there are astronomical data sources, some of which use the VOTable format, but are not available via IVOA services, such as Simple Cone Search. In fact, the Galaxy Zoo project provides their data as a downloadable compressed archive of VOTable XML. Consequently, the reliance on common registries is likely to limit users' access to data sources.

Meanwhile, as a custom XML format, the VOTable format is not recognized by the browser as anything more specific than an XML document and the browser, by default, provides minimal and uninteresting renderings of its content. These XML results are intended to be processed by specialized tools and not by a Web browser. Yet, many of the services that were encountered attached an XSLT transformation intended to render the results for viewing purposes. As the future of XSLT within the browser is uncertain, this technique may become unavailable to users.

In the end analysis, the VOTable format, despite being aliased under a variety of versions of namespaces, has provided durability for tool-oriented consumers of astronomical data sets regardless of how it is provided directly

on the Web to consumers. This has allowed many commonly used data sets to be re-purposed into a variety of different projects. As we discuss later, this durability stems partially from the proper use of markup languages for encoding information.

2.2 Galaxy Zoo

With respect to astronomical data on the Web, Galaxy Zoo [29] is an often cited example of a successful use of the Web as a platform for crowd-sourcing the processing of scientific data. The Galaxy Zoo project used the opinions of the casual layperson to help categorize the shapes of different galaxies. The project developers applied statistical analysis to the user-generated categorizations to either find consensus or to focus user attention on objects that needed more opinions. The result was the categorization of more than 900,000 galaxies between July 2007 and February 2009 [30].

This project used data from the Sloan Digital Sky Survey [31],[32]. The input contained 15.7TB of FITS image data, 26.8TB of other data, and 18TB of catalogs. The output data set is available for download in compressed archives, smaller in size, in a variety of tabular formats that includes VOTable XML. The output data is not available via any search service.

The consequence of not being able to access the output data, even via a simple cone search service, is that it severely limits the ability to casually browse the data or use only portions of the data. Anyone attempting to build a new Web-based derivative has the immense task of first downloading and processing many terabytes of data. That is, they have to do all the "heavy lifting" themselves because the originators of the data set have deferred all processing to the consumer. The result is that different consumers will need

to perform similar processing over and over, duplicating each other's efforts, and impeding any possible network effect.

The team that developed Galaxy Zoo has been extending the platform to address the needs of other scientific endeavors with similar data processing needs; this partially addresses the issue of duplication of efforts. They developed the Zooniverse platform [33] to enable researchers to setup systems similar to Galaxy Zoo. While there have been a handful of projects outside of astronomy, only a few have published successful results as of mid-2014. These projects also face the problem of not having a commonly accepted data format, as astronomers do, with which they can disseminate their results.

2.3 Geospatial Data

A great variety of geospatial data is tied up in proprietary or custom formats. As mentioned previously, many scientific or government agencies exchange information using the ESRI Shapefile format. This format is both proprietary and publicly undocumented. As such, there is a large opportunity for standardization of the exchange of geospatial data.

The Open Geospatial Consortium (OGC) is a international standards organization that has attempted to fill this void; it consists of "volunteer" members with origins dating back to 1994. They have produced a number of data formats for geospatial data that are beginning to be found in common use for government, scientific, and other open data exchanges of geospatial information. While they provide a comprehensive architecture, their standards have had a varying degree of success.

The OGC provides over 30 standards relating to the exchange of geospatial information, including the popular KML data format [34] and WMS (Web

Map Service) [35]. Their architectures are often a complex orchestration of technologies, often Web-oriented, for use within map-oriented applications such as publishing of government or scientific data sets. Some of these standards (i.e., KML or GML) can be used as replacements for ESRI Shapefiles for data exchange.

Many of the various specifications, including KML and GML, are feature oriented. A *feature* is an abstraction of a real world object or phenomenon that has some geometric aspect (e.g., a point, line, polygon, etc.). Such features often have some number of additional properties that include rendering information and metadata or other data.

2.3.1 GML

Geography Markup Language (GML) [36] is an OpenGIS / OGC standard for expressing geographical features. The core model is based on *ISO 19100 Geographic information/Geomatics* and encoded in a set of XML Schemas [37]. The intent is that applications will embed or derive from the GML schemas to provide application-specific uses.

GML has important base concepts for embedding scientific data in terms of units of measure, measurements, quantities, and modeling of values. Within the GML instance, units of measure can be defined relative to other previously defined units. These units can be used within measurements, quantities, and other composite values. While these might form the basis for encoding tabular scientific data, the specification itself notes:

> *"This schema is primarily intended to serve for "simple" observations. Schemas for scientific, technical and engineering observations and measurements will typically require the development of a GML application schema for such observations. See, for example, the Observations and Measurements specification from the Open Geospatial Consortium. "*

When following the above recommendation, the *Observations and Measurements XML Implementation v2.0* [38] results in markup that looks similar to a table of data with a few bits of GML embedded. That makes the choice of using GML to encode the tabular database suspect given the limited amount of GML used. That said, this is the recommended model for using GML: applications use custom schemas that embed or extend GML.

Because of the heavy use of schema extension mechanisms, and the focus on delivering a core extensible model rather than an end-user model, GML is hard to use directly. Instead, consumers must first become experts with a variety of XML technologies (e.g., XML Schema), develop their own extended data models, and can only then use, encode, and exchange data. The result is something that is not suitable for the uninitiated and has directly limited the use of GML by the average application developer.

2.3.2 KML

The OGC makes this statement about KML in many places:

> *"The OGC KML Standard is an XML grammar to encode and transport representations of geographic data for display in an earth browser, such as a 3D virtual globe, 2D web browser application, or 2D mobile application. Put simply: KML encodes what to show in an earth browser, and how to show it. " [39]*

The history of KML is a bit suspect in that it was originally called *Keyhole Markup Language* and was developed by a company called Keyhole, Inc. that was bought by Google in 2004 [40]. It was developed for use in the Keyhole Earth Viewer visualization software that was later to become Google Earth. In 2008, mostly because of the weight of Google behind KML, and the high-profile use of it within Google Earth, the OGC approved KML 2.2 as an OGC standard relatively unchanged [41].

The primary use of KML is to describe geospatial features, often for presentation within mapping tools such as Google Earth or within a Web-based mapping application. While the markup is oriented towards describing points, polygons, etc., some of the markup allows for "extended data" to be embedded. As such, KML has been used to exchange scientific geospatial data sets such as ecology data, sensor data, etc., by government repositories such as `data.gov`.

Embedding data is only allowed on KML feature elements (i.e., `NetworkLink`, `Placemark`, `GroundOverlay`, `PhotoOverlay`, `ScreenOverlay`, `Document`, and `Folder`) using an `ExtendedData` child element; this element may contain any number of `Data` elements, each of which represent a name/value pair.

KML supports describing a simple schema for embedded data. The typing of data is restricted to typing values with simple data types. While there is some flexibility in the location of the schema description, there is little within the markup to identify ownership or domain semantics. As such, attempts at inference over this data is purely by assumption.

Figure 2.8 KML Descriptions for Data

```
<Placemark>
   <name>  Pell City 5.8 SSW, AL</name>
   <Snippet maxLines="0">empty</Snippet>
   <description><![CDATA[<table border="1" padding ="1">
      <tr><td><B>STN ID</B></td><td><B> AL-SC-3</B></td></tr>
      <tr><td><B>OBS DATE</B></td><td><B>2013-08-08</B></td></tr>
      <tr><td><B>OBS TIME</B></td><td><B> 07:00 AM</B></td></tr>
      <tr><td><B>TOT PRECIP AMT</B></td><td><B> 0.65</B></td></tr>
      <tr><td><B>NEW SNOW DEPTH (IN)</B></td><td><B> NA</B></td></tr>
      <tr><td><B>TOT SNOW DEPTH (IN)</B></td><td><B> NA</B></td></tr>
   </table>
   ]]></description>
```

In practice, KML is more often used for displaying data rather than exchanging data. The example in *Figure 2.8, KML Descriptions for Data*, is from a NOAA data feed for snow observations. Rather than make use the `ExtendedData` facilities, they have focused on making the display of the

data within the place marker a priority. The table markup shown is actually escaped as a string using a `CDATA` marked section, and so a receiving system is required to parse the contents again to, hopefully, obtain the data.

While the VOTable format was developed to exchange tabular data, the KML approach to data feels like an afterthought. Although users have successfully employed KML to provide geospatial annotations, it is uncertain whether it is capable of extending very well beyond the visual display of information. The lack of anything further than basic data type annotations constrains its use for scientific data considerably.

2.3.3 Non-OGC Alternatives

Given the complexity of GML and the trends towards JSON as a data representation format, the Web developer community developed GeoJSON [42] as a more palatable format. GeoJSON is also feature oriented with basic constructs (i.e., points, lines, and polygons) that can be combined in various ways to produce complex features. As a whole, a data exchange consists of a collection of features.

Given that JSON objects are untyped, each object contains a single "`type`" property with a string value containing a type label. The specification defines a fixed set of "Geometry Objects" types for describing feature regions that can be organized into collections of features. Each feature is essentially a pair of two attributes: "`geometry`" whose value is a Geometry Object and "`properties`" whose value is any JSON object.

The interesting characteristic for data exchange is the `properties` attribute, where the value can be any JSON object. As such, any kind of data can be associated with a feature as long as it is serialized as JSON. This gives the application developer a great deal of flexibility and results in broad appeal to the various Web developers using mapping technologies within the browser.

Figure 2.9 GeoJSON Feature Example

```
var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "AT&T Park",
        "amenity": "Baseball Stadium",
        "description": "This is where the SF Giants play!"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [-122.389283, 37.778788 ]
    }
};
```

For example, in *Figure 2.9, GeoJSON Feature Example*, the feature has the geometry of a simple point. The properties describe the feature (a baseball stadium) for a consuming application. Yet, it should be noted, that from a GeoJSON perspective, "name", "amenity", and "description" are merely properties on an object of unknown type. Typically, these properties are either defined by the consuming mapping technology or given semantics by part of the application developed using a mapping library.

From a tabular data perspective, GeoJSON has the same problem as other feature oriented data formats. Iterating over the tabular data becomes a process of iterating over the features and then inspecting the "properties" attribute. Within that attribute must be a JSON object that contains a single row of tabular data; this object has no standard representation that is defined by GeoJSON.

There are many mapping toolkits (mostly JavaScript-based) that provide direct use and rendering of GeoJSON data. Each toolkit has specific definitions of what it will do automatically with the "properties" attribute. Most of these toolkits provide straightforward ways to map custom attributes into interaction features (e.g., map markers with pop-ups).

In terms of usage, GeoJSON is used in various business applications; the US Geological Survey (USGS) provides their earthquake feed as GeoJSON with well-defined properties for each feature (essentially an epicenter point) [43].

There are two different feeds, where a summary feed provides the list of events, and the detail feed gives the details for a specific event. This same data is provided in a variety of other formats including KML and custom XML but these two levels of detail are not separated for these formats.

2.3.4 Failures with Government Data

Information has a tendency to "hang around" in the format it is originally published. For example, marine specimen data was collected from 1975 to 1981 by various U.S. agencies. This data is now archived by the National Oceanic and Atmospheric Administration (NOAA), which makes it available via `data.gov`. The format of these data resources is a flat-file text database format called *F025 (File Type 025 - MARINE MAMMAL SPECIMEN)* and is defined in *NODC Standard Format Marine Mammals of Coastal Alaska Data (1975-1981): Marine Mammal Specimens (F025) (NODC Accession 0014150)* [44]. Any scientist seeking to analyze these historical records of marine populations must first be able to process or decode this file type.

Further, while it is fortunate that the format definition has not been lost, other records fail to exhibit essential properties for archives of information on the Web. Consider the record *Marine mammal specimen and other data from the SURVEYOR and other platforms as part of the Outer Continental Shelf Environmental Assessment Program (OCSEAP) from 20 March 1977 to 02 November 1977 (NODC Accession 7900319)* [45]. The data recovered for this data set contains very little information about the F025 format. It takes some effort to find the correct specification syntax (NODC Accession 0014150) and that distraction presents a direct barrier for the uninitiated.

Recognition of this problem led us to identify the desirable property of a *deterministically identifiable syntax* (i.e., the "self-describing" Web [46]). While text files are often very durable, the data shown in *Figure 2.10, NODC 7900319 Sample*, says very little about the syntax (NODC's F025) or semantics (e.g.,

there's a geospatial region in the beginning) of the data, other than what might be done by guessing. Markup languages, such as XML, provide the ability to name properties and associate namespaces, thereby allowing at least the more obvious properties to be discerned by inspection. However, such markup does not completely solve the semantics problem of associating a specific facet with particular structures.

Figure 2.10 NODC 7900319 Sample

```
025TR493910001          1600200N1412130W197710292400       0401
025TR493920001          29221030107012 NN   1  172001530 880          255
025TR493930001          3           24           544001410 1
025TR493950001          51213 331  721181 70  1
025TR493960001          6            12450 12400
025TR493970001         158755030101  8     1                        3
025TR493970001         1487910307    8     4                        3
025TR493970001         1387910306    8     1                        3
025TR493970001         116179180101  8                              2
025TR493970001         108840010201  8     1   100
025TR493970001          98791030701  8     1   100
025TR493970001         16883102      8     1                        3
025TR493970001          76179180101  8   296 12200
025TR493970001         128791030701  8     3                        2
025TR493970001          8            804
```

The NODC provides a KML version of these files, but inspection of a variety of samples shows that they have only converted the locations into KML. As such, the KML only contains a set of points that indicate where the data is located, but the points are not associated with the actual data itself. At a minimum, given the internal knowledge of the F025 format, an opportunity was missed in not converting and embedding the remaining data into a more modern syntax such as KML.

2.4 Crowd-Sourced Ecology Data

Similar to the Zooniverse projects, the iNaturalist site [47] uses crowds to collect ecology data. Interested users typically employ a mobile or desktop application to upload observations, typically in the form of a picture and

metadata. The website then allows others to help categorize and enhance the observation. That is, once an observation is posted, others can help identify or verify the specific species and other facets of the entry.

As of June 11, 2013, there are 836 projects and 16,993 users on iNaturalist who have generated 278,478 observations. A user may contribute observations to a project from which the resulting data can be downloaded. A researcher who chooses to download the data has the choice of Atom [48], KML, or CSV formats. Each of these formats have issues in the way they have been implemented.

In theory, both the Atom and KML formats have the possibility of providing specific geospatial information. Unfortunately, the properties of the entries are mostly encoded as HTML representations. As such, columns of data, such as species or links to images, are obscured from being uniquely identified in the markup.

For example, while there are geo-locations, taxa categorizations, descriptions, etc., all available via the rich content of the Web site, the choices made by iNaturalist to encode this information into the specific data formats of Atom or KML did not translate this information into individual data elements. In the case of KML, instead of using the `ExtendedData` element, the data is encoded into an HTML description that is also escaped markup due to the fact that KML descriptions are simple strings.

Similarly, in the Atom feed, instead of using some set of XML elements, the data is again encoded into an HTML description. Even though Atom allows XHTML content directly, the site again uses escaped HTML via Atom's `content` element. This presents the same technical problems as KML descriptions.

Both of these unfortunate implementation choices leave the CSV format as the only way to get all the data without making assumptions about processing. While CSV is a common way to exchange tabular data, there are no column semantics and browsers currently treat it as plain text. As such, it is not a format that can be processed in a standard way on the Web without additional "out-of-band" information.

2.5 Summary

In summary, we see that even widely available, more-or-less standardized, spatially-organized data formats on the Web share a number of characteristics which make them difficult or impossible to use by Web-based applications:

1. Lack of consensus or adoption of column labels / annotations.
2. Insufficient mechanisms for describing or annotating tabular data.
3. Escaped markup used for data.
4. Lack of discoverable semantics (links, columns, annotations).
5. Complexity is often an artifact of the format and results in simple data being unnecessarily complex.
6. Insufficient investment in transforming data into usable formats for data processing versus visualization.

The technologies of the Web, as a platform, and the principles upon which the Web is based, provide the means to overcome these problems. Various standards, techniques, and good practices can be woven together to provide a general methodology for publishing data on the Web. Yet, before describing

a methodology for doing so, a review of the basics of the Web and its architectural principles is in order.

Chapter 3

The Architecture of the Web

5. *The Web has essential architectural principles upon which it is based.*

6. *The Semantic Web extends this architecture, thereby allowing data to be annotated.*

7. *RDFa annotations enable algorithmically processable information in context on the Web.*

The *Open Web Platform* (OWP) [49] is a "platform for innovation, consolidation and cost efficiencies" focused on those things that happen within, or intersect with, the actions of the Web browser. This platform is defined by the shared behavior expected by the publisher and users of content and services—a type of social contract readable by developer and authors alike. The collection of individual recommendations (standards documents), technologies, practical algorithms, APIs, vocabularies, and their interactions make the OWP a cohesive and motivating platform for publishers and consumers alike.

Yet, the Web had much more humble beginnings as a proposal for information management at CERN [50] by Sir Tim Berners-Lee who was then a physicist. It was originally designed "to meet the demand for automatic information-sharing between scientists in universities and institutes around the world" [51] and then morphed into something much more pervasive. By the early 1990's, the Web had become something of great interest to business, governments, and academia, as witnessed by the establishment of the *World Wide Web Consortium* (W3C) on October 1st, 1994.

The requirements cited in the original proposal still ring true today: remote access, heterogeneity, non-centralization, access to existing data, "private" links (one can create their own links), data analysis, and live links (resources can change dynamically). These features may seem obvious in retrospect, but in 1989, many hypertext systems were closed systems with pre-authored content only accessible by the producer. The real innovation came about with how these requirements were met.

By 1992, the original system [52] had coalesced into three important innovations: a protocol called *HyperText Transfer Protocol* (HTTP), an addressing scheme (eventually labeled a URI), and a document format called *HyperText Markup Language* (HTML). What ties these all together is a browser that understands how to turn an address into a protocol request to get a representation (i.e., an HTML document). In 1993, a team in Joseph Hardin's lab at the University of Illinois at Urbana-Champaign released the Mosaic browser on Windows, Mac and X-Windows. In later years, the browser became the focus of innovation, but it is these underlying parts of the original system that must operate together to make the browser work successfully.

More than a decade later, after the Web had proven itself enormously successful, the W3C established the Technical Architecture Group (TAG) and they attempted to document the design principles that had both governed and

emerged from the standards development and innovation surrounding the Web and the W3C. This process culminated in publication of the *Architecture of the World Wide Web, Volume One* in late 2004 which re-codifies the three architectural bases of the Web as:

1. *Identification* — URIs are used to identify resources.

2. *Interaction* — Web agents communicate using standardized protocols that enable interaction.

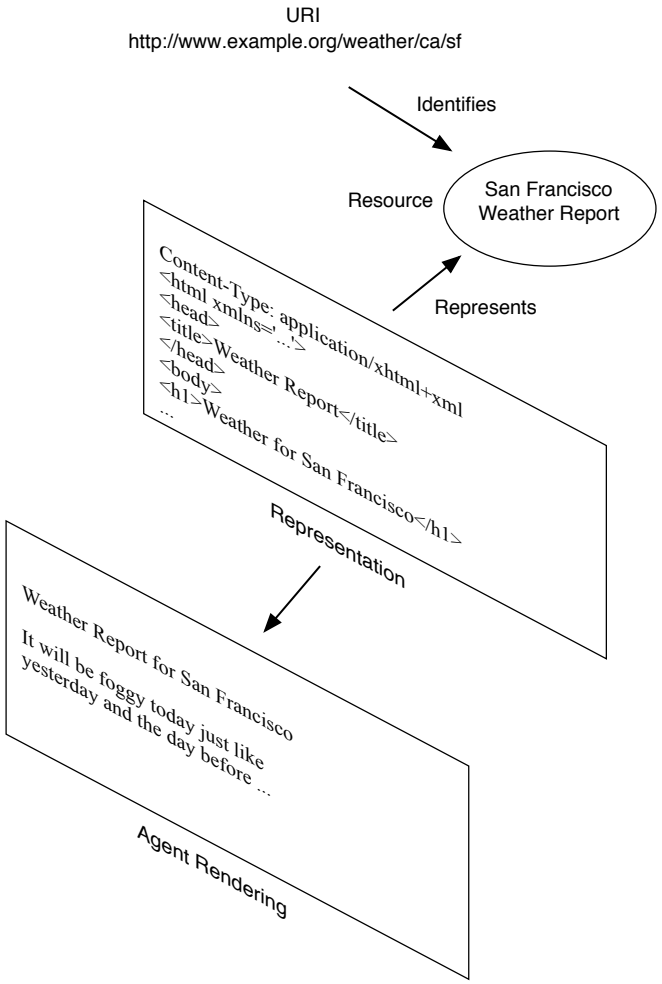
3. *Formats* — Protocols use formats to exchange information which are identified in the payload with metadata (e.g., the `Content-Type` header).

While any kind of data could be transmitted via interactions, the abstract "information space" that participants act upon is populated with items called resources. Resources have representations that are transported over various protocols, and browsers display these representations. These relationships and their ultimate use by a browser are depicted in *Figure 3.1, URI, Resource, and Representation*.

For identification, the Web demonstrates a fundamental principle of hypertext systems: global naming leads to global network effects. The consequence of that principle is that URI naming becomes incredibly important. To increase the value of the Web and to allow more positive effects, publishers should provide URIs as identifiers for resources. Also, distinct URIs should be used for distinct resources to broaden the addressable space of resources.

In practice, URIs are associated with domain owners and some authority which has the ability to construct (mint) new URIs. When the authority assigns URIs to resources, they should ensure they are unique and avoid arbitrarily aliasing the same resource with other URIs. That is, a careful

Figure 3.1 URI, Resource, and Representation



assignment of URIs to resources provides a greater benefit for the Web as a global space of linkable URIs.

Finally, URIs are identifiers that often resolve to locations; they rely upon URI schemes to identify the protocol for accessing the resource. Specifications and authorities should re-use existing schemes (e.g., "`http:`") rather than construct new identifiers (e.g., the "`ivoa:`" scheme violates this good practice). When possible, using resolvable locations such as "`http:`" URIs generally has a preference today due to the fact that you can always attempt to interrogate the URI to gather more information about it.

When agents attempt to access a representation of the resource identified by a URI, they "dereference" the URI. Typically this involves inspecting the URI scheme and attempting the protocol or access method identified by the scheme. If the URI scheme is unrecognized or otherwise not allowed, the agent may simply refuse to access the resource.

Protocols such as HTTP provide many different ways to access a resource via a URI and use different "methods" such as GET, HEAD, POST, etc. These methods differ in their ability to send or retrieve information from the resource. The simplest access methods retrieve a representation of the resource.

When a request is made over a protocol such as HTTP, the URI is deconstructed into a URI path (the portion that follows the domain name and precedes the fragment identifier [#]) and a server address. Only the URI path is sent via the protocol to the server along with whatever additional headers the agent specifies. In the case of certain methods (e.g., POST and PUT), a representation may be sent as well.

The response returns a representation whose type is determined via the metadata returned. Specifically, the "`Content-Type`" header identifies the data format as one of the Internet Media Types that are registered in the IANA registry [53]. The exact response depends on the URI owner and what services they provide. Even if the URI looks as if it might provide a resource of a particular type (e.g., ends with a `.xml` extension so `text/xml` is expected), the `Content-Type` header dictates the response, which may even be in the form of an informational error message in a common format (e.g., HTML). In general, good practice dictates that agents should never ignore the response metadata without the consent of the user.

As a fundamental principle, agents incur no obligations by retrieving a representation. This allows for safe interactions on the Web to be dictated by

the user via their agent. While the agent incurs no obligation, this does not mean that the interaction is inherently safe. A simple retrieval request can have possibly dire consequences as information is always sent in the URI path and otherwise to a service outside of the control of either the user or their agent.

Finally, as a good practice, a URI should have a consistently and predictably available representation. This allows a user to access the representation to gather more information. Yet, just because the representation is available it does not mean that a user should be required to retrieve the representation. That is, in principle, having a reference does not imply dereference.

When a representation is retrieved, there are generally two classes of data formats: binary and textual. A binary format is a sequence of bytes of data whose interpretation is guided by its format specification. In contrast, a textual format is one whose data is encoded as a sequence of characters and represented as a sequence of bytes using some encoding of those characters. These characters and their encoding are affected by the protocol and the metadata embedded within the response.

Textual formats are often more portable and interoperable, and they tend to be more readable by humans. The metadata associated with the format helps the receiving agent decode

Representative State Transfer (REST)

REST [100] is an abstraction of the architecture of the Web in terms of various more-or-less criterial properties such as client-server articulation, statelessness, cacheability, layering, and supportive of uniform interfaces. This abstraction is useful in designing services on the Web as it encapsulates many of the architectural principles of the Web in a way that allows systems to scale. Specifically, it relies on "good" URI naming and stateless resources which are properties extensively used within the solutions provided in this research.

from a sequence of bytes into a sequence of characters via the encoding used in transport. While many modern systems promote and use Unicode, any character encoding and character set can be represented because, at the protocol layer, the representation is sent as sequence of bytes.

In practice, data formats change over time as new versions are developed to handle new requirements. A format should provide a mechanism for identifying version information and extensibility. A good extensibility mechanism does not interfere with the original intent of the specification and also defines how a system should behave when faced with an unknown extension.

Within a particular data format, there are a number of good practices:

1. Separation of content, presentation, and interaction.
2. Identification of links and their roles within the representation.
3. Allow Web-wide and hypertext linking (not just internal).
4. Allow the use of URIs without restrictions.
5. Allow the use of URI references (relative links) to enhance the usability of the data format.

For XML-based formats, there are additional good practices:

1. All element names and global attribute names should be members of a namespace.
2. As namespaces are identifiable by URIs, a namespace document should be one resource format made available, and associated somehow with the URI, that contains material intended for people to read about the namespace, format, and its use.

It is important to note here that many of the XML-related media types (e.g., those that end with `+xml`) are not differentiated within the OWP and are often treated as non-differentiated XML markup. Although it is theoretically and practically possible for there to be different semantics within the processing model for the specific XML markup, the OWP currently invokes the same processing model. This effectively limits what an XML-based application can do within the platform as there is little to bootstrap an application.

From a data exchange perspective, the inherent limitations of XML may be acceptable; from an application perspective, where enabling an extended data model and semantics is helpful, this impoverished processing presents limitations for XML in the browser. The most notable limitations are the inability to recognize links (a fundamental good practice; see Hypertext Driven [55], [56]) or to automatically invoke scripting (a practical reality). If scripting were available for XML, it would enable *extensible* semantics that could be used to add in other behaviors—including linking.

Hypertext Driven

Hypertext driven, as encapsulated by Hypermedia as the Engine of Application State (HATEOAS), is the approach that adds an additional constraint on REST architectures where hypermedia (e.g., HTML) is the interface through which the application navigates or discovers additional services. Specifically, data representation formats can link to one another and form a chain of information upon which an application can act. This allows applications to inspect the instance and find useful links to follow for more information or interactions. A specific use of this mechanism is described in *Chapter 5, The PAN Methodology*.

Notably absent from the Web Architecture document is any mention of Resource Description Framework (RDF) [54] or the Semantic Web as a layered and principled approach to data on the Web. While the Web conceptually supports any variety of formats, the common intersection of formats supported by the various implementations of the OWP is relatively small and this limits interoperable interchange, thereby reducing the common data formats to those embeddable or linked via HTML representation along with a only a few intrinsic XML-based formats.

What is RDF?

RDF is a framework for describing resources. Resources can be anything from abstract concepts, documents, people, to physical objects [57]. Such resources are named via URIs and statements are made about them using triples. A triple consists of a subject (the resource), a predicate (a URI), and an object value where the predicate provides a relationship between the subject and object value.

The object value may contain another subject URI which allows RDF to describe a graph of information. This graph isn't typical found directly as a single Web resource. Instead, RDF triples are "harvested" from various formats that contain annotations. Sometimes, the formats are direct serializations of the triples themselves.

RDF relies on the fundamental principle of URI naming but extends its use such that both RDF subjects and RDF properties are named via URIs. The items within the information space addressed by RDF are triples that contain a subject, a property, and an object value. The object value can be either a literal or another URI, which can, in turn, be the subject URI in another triple. The resulting connections build RDF triples into a graph.

Conceptually, triples are graph constructs where all the subject URIs or object values can be considered nodes and the edges between them are labeled with property URIs. Given any collection of resources on the Web that somehow

contain triples in their data format, a large graph of "knowledge" can be harvested and processed by an inference engine.

Yet, this process rarely happens on the Web directly. The representation of triples as its own XML data format has limited its adoption in other languages up until recently. As the need for ways to further qualify and extend data in representation formats such as HTML have become important, annotation technologies have been developed to merge the concepts of the Semantic Web into the Ordinary Web that users encounter every day.

3.1 RDFa Annotations

In attempts to re-use information already represented on the Web as regular HTML, several different annotation systems have been proposed for HTML: Microformats [58], RDFa [59], and Microdata [60]. While Microformats have their origins starting around 2005, its popularity has come and gone. It relies on the use of the single HTML `class` attribute to annotate and structure information within an HTML document. It has had some success—especially in the hCard (contacts) and hCalendar (calender entries) formats in various e-mail client and calendaring tools.

In 2004, Mark Birbeck published a W3C Note [61] on a way to express RDF triples in HTML. This note evolved into the RDFa 1.0 specification as published by the W3C in 2008. It was subsequently revised in 2012 and an improved version was published as RDFa 1.1 [62] that provides for more general use within any markup language.

The fundamental mechanism of RDFa is a set of attributes, some of which are already in HTML, that are used to annotate *any* element. This change from RDFa 1.0 allows RDFa to be used in any context with an HTML or

XML document as long as the vocabulary allows the attributes. In the case of HTML, the open extensibility model encourages all browsers to allow RDFa annotations without errors.

About the same time that RDFa 1.0 was published, Microdata was proposed to be part of HTML as part of the HTML5 effort. Microdata also uses a set of global attributes and, as a more controversial part of the HTML5 standardization efforts, it was moved to its own specification. As RDFa 1.1 was completed, the membership of the W3C indicated a strong preference that there not be different specifications that accomplish essentially the same goal. As a result, the Microdata specification has been orphaned, published as a W3C Note (a lesser status), and currently has an indeterminate future with the notable exception of support by several large search engine vendors.

In all of these different syntaxes there is a common theme of attempting to re-use and annotate data in its published context within the HTML document. The natural overlap between structured data and its markup representation on a particular Web page is used to annotate and preserve the rich structure of the data. In the case of RDFa, the data has a direct interpretation as RDF triples, which allows all the Semantic Web technologies to be used once the triples are harvested from the document.

This is not the case for either Microformats or Microdata as they are both oriented towards items with properties. There is no guarantee of a URI name for the subject of each item. As such, mapping these technologies into the Semantic Web is either difficult or relies on deep assumptions.

RDFa has a rather simple syntax that fosters the ability to build complex triples from data embedded in commonly used syntaxes such as HTML. This technique provides a number of desirable qualities: *identifiability*, *extensibility*, *flexibility*, and *durability*. We shall explore these qualities but first a primer on the basic syntax of RDFa is necessary.

3.1.1 RDFa Syntax

To address the needs of the community and bridge the gap between RDFa and Microdata, the W3C produced a simple subset called RDFa Lite [63]. This subset removes many of the superfluous features and focuses on three main attributes: `resource`, `typeof`, and `property`; and two syntax attributes: `vocab` and `prefix`. These attributes are global and so are available to be placed on any element. Their interpretation through a top-down-first-to-last tree traversal derives a set of subjects, properties, and object values associated with identifiable locations within the document.

From a syntax perspective, the values of types and property names are expected to be URIs, which can be lengthy to specify. The `vocab` attribute provides a scoping mechanism for a default base URI against which values can be resolved, while the `prefix` attribute allows specifying a set of prefix mappings to URIs that can also be used when referencing URIs.

An example of RDFa using the `schema.org` vocabulary is shown in *Figure 3.2, Example RDFa*. At the very beginning, the default vocabulary is set to `http://schema.org/` for the whole element sub-tree. The subject is named as the resource `#alex` which will resolve against the base URI of the containing document.

Figure 3.2 Example RDFa

```
<p vocab="http://schema.org/" resource="#alex" typeof="Person">
   My name is
   <span property="name">Alex Miłowski</span>
   and you can drop me an e-mail at
   <span property="email">alex@milowski.com</span>.
   <img property="image" src="http://www.milowski.com/images/alex.png" />
</p>
```

The type of this subject is identified as "`Person`" and this value will resolve to `http://schema.org/Person` against the default vocabulary. For all the contained property names, each will also resolve against this same default

vocabulary to produce a URI. For example, the property "name" resolves to the URI http://schema.org/name and so on.

Finally, a value for each property is constructed by examining the element. In the case of simple inline elements, such as span, the content is taken from the text descendants. For example, the "name" property has the value "Alex Miłowski". The result is the set of triples shown in *Figure 3.3, Example RDFa Triples*, in Turtle syntax [64].

Figure 3.3 Example RDFa Triples

```
@prefix schema: <http://schema.org/> .
<#alex> a schema:Person;
        schema:name "Alex Miłowski";
        schema:email "alex@milowski.com;
        schema:image <http://www.milowski.com/images/alex.png> .
```

It is also important to note that RDFa captures link relations within the document. In the example, the image src attribute is used as the object value for the http://schema.org/image property. In general, RDFa will consider any src or href attribute within the document as an object link value and also the start of a new subject for the element's descendants.

3.1.2 Identifiability

Identifying RDFa-based annotations of data benefit from a two-fold syntax. Each application of RDFa has both a "host language" and the vocabulary used within the RDFa annotations. As such, both represent opportunities for deterministic identification of the kind of information being represented.

The first aspect of recognition is the "host language", which is often HTML but could be any markup. A specific host language may have desirable properties such as ubiquitous processing and rendering by tools and applications. As such, the data encoding shares all the benefits (and pitfalls) of the host language.

In the specific case of HTML, data can be structured such that it is always viewable by a potential consumer because it is identified as HTML. This aspect allows casual browsing of data that might otherwise be difficult for the novice or uninitiated consumer. The value of a default rendering of data cannot be overstated.

The second aspect of being *identifiable* is that the basic syntax of RDFa ensures that to get the proper annotations either a default vocabulary is declared or full URIs are used for types and names. This means that a simple inspection of the document, even without fully processing the RDFa, can yield information about the ontology and types used to annotate the data.

There is huge value in being able to look at the source, as viewable plain text, to inspect the encoding and find links, namespaces, and other URIs. These provide the future consumer with clues as to where to find the semantics for unrecognized vocabularies. Since an active domain owner may likely exist, the consumer can consult the owning entity for further information or just visit their website. This benefit points towards a corollary: what a human can see in the *source* of a Web resource is also what a data-oriented application, which may know little or nothing about the host language, can exploit based only on knowing the semantics of the *annotation* vocabulary. The vocabulary used is at least, in part, accessible via the ability to dereference the URIs which are used.

3.1.3 Extensibility

Once an application has started expressing its data using standardized RDFa annotation and markup, it is likely that new requirements for the application will tug at the standardization in unexpected ways. Being able to meet new requirements in a timely fashion, identifying the extensions, and providing fallbacks, is a necessary feature and easily handled with RDFa via several different approaches, as we shall see.

First, subjects can have multiple types. This form of "mixing" allows both unions and derived types to be specified within the single `typeof` attribute. The syntax of the `typeof` attribute allows a space delimited set of types to be specified and each type name is considered independently. The subject consequently has multiple types within the derived annotation graph.

It might seem a bit odd to allow unions and derived types within the same syntax but the type triples are merely assertions involving the subject. Whether two types are disjoint or one is a derivation of another is often defined externally; perhaps available in an XML schema, stylesheet, transformation, pipeline, or namespace representation by dereferencing the vocabulary or namespace URI. In the document itself, the annotations merely identify that the subject is asserted to be of both types.

This kind of flexibility allows extensions (whether unions or derivations) to be identified directly by the syntax. A processing system can identify a known type and understand that there is also other information encoded as well. Thus, when additional properties are encountered, the application can understand whether this is a possible error or part of an additional type.

For example, in *Figure 3.4, RDFa Extension Example*, the annotations use the `schema.org` standard type `http://schema.org/Person` and the extension `http://example.org/Scientist`. The subject `<#marie>` is associated with both types. The property `http://example.org/discovered` is readily distinguishable from the `schema.org` types by inspection, signaling additional information. A consuming application can decide whether the presence of a foreign type should halt processing or invite further investigation; an RDF-aware application could decide to de-reference the URI to seek further information about the foreign type.

In a larger context, the identification of extension with the overt types in the document allows systems to know that new type definitions are required.

Figure 3.4 RDFa Extension Example

```
<div resource="#marie" vocab="http://schema.org/"
     prefix="my: http://example.org/"
     typeof="my:Scientist Person">
   <span property="name">Marie Curie</span>
   <span property="my:discovered">polonium</span>
   <span property="my:discovered">radium</span>
</div>
```

This requirement can surface to the managers of the knowledge repository or to other parts of the system that are capable of finding and loading new ontologies.

The key point is that extensibility is built into the syntax and tied directly into being identifiable. New type URIs that are not already known represent potentially new information structures. They can simultaneously be new discoveries and safely ignored. Systems that choose to process only known types can choose from a spectrum of processing models: from signaling an error, to searching and retrieving new semantics, to ignoring the new type and its associated triples. Such systems also directly know when they are ignoring information that they do not recognize and this knowledge is critical in determining data quality.

3.1.4 Flexibility

Because RDFa uses global attributes, annotations can be embedded anywhere within a document. As data is expressed within markup for other purposes (e.g., a table of data in HTML), annotations can also be associated with this markup to further identify, structure, and otherwise enhance the quality of the information contained. Where these annotations are applied is up to the data provider and not dictated by the format (e.g., by HTML). However, the host language provides some structure; it offers choices such as whether to annotate table headers, a paragraph, or a text phrase.

Figure 3.5 Same Data — Two Ways

First encoding:

```
<p vocab="http://schema.org/" resource="#alex" typeof="Person">
   My name is
   <span property="name">Alex Miłowski</span>
   and you can drop me an e-mail at
   <span property="email">alex@milowski.com</span>.
</p>
```

Second encoding:

```
<h1 vocab="http://schema.org/" resource="#alex" typeof="Person">
<span property="name">
<span property="email" content="alex@milowski.com">
<a href="mailto:alex@milowski.com">Alex Miłowski</a>
</span>
</span>
</h1>
```

Same triples:

```
@prefix schema: <http://schema.org/> .

<#alex> a schema:Person;
   schema:name "Alex Miłowski";
   schema:email "alex@milowski.com" .
```

As requirements for user interfaces change, the markup structures may change. Yet, the information provided via the annotations may be exactly the same. Because the RDFa annotations produce an annotation graph, their exact locations within the document may not really matter to a consuming application. As such, there is a great deal of flexibility in what markup is used and how it is allowed to change and evolve.

By using RDFa annotations, applications can provide open extensibility mechanisms for additional new behaviors. These mechanisms allow users to innovate upon the platform without prior consensus on necessary changes in data formats. That is, the use of RDFa builds in flexibility that allows new information structures to be overlayed without requiring changes to consuming applications.

Finally, two different data producers may have different requirements for how their information is displayed. When information is conveyed via annotations, the same information can be structured in very different ways but produce the same information graph. In *Figure 3.5, Same Data — Two Ways*, the two different markup structures produce the same set of triples. The example shows the resulting triples in Turtle format at the end.

3.1.5 Durability

HTML, the intrinsic markup language of the Ordinary Web, has built an indelible record of durability. Documents that were published on the original website provided at CERN (circa December 1990) are still browsable today. Investments made in publishing information on the Ordinary Web have proven themselves viable for over a decade. This is, in part, due to the durability of the markup in general.

HTML's simple and default rendering models allow information expressed within it, at minimum, to be viewable. While their specific information content may be hard to determine programmatically, the human viewer can often gain a lot of information. For information to remain durable, it must remain consumable by humans without massive decoding efforts; the default rendering model for HTML provides a basic level of durability.

RDFa provides the ability to annotate and qualify data contained within regular Web pages and helps make such viewable pages also interpretable as data. Even though portions of the data may be visually disjoint, the flexibility in the annotations allows them to be directly linked. This lets one Web page serve multiple purposes and prevents data from being separated from its context.

As trends change, the same information encoded in the same annotations can be presented in different ways. This allows the same information content to be

gained from accessing and harvesting the triples. As such, consuming systems accessing the data annotations may not need to change.

Finally, information expressed in regular Web pages can be backed by more extensive information models. The data harvested from the Web page as triples of information fit neatly back into this larger model. The same information can transcend several sources and formats without being impoverished. Thus, the representation on the Web can be as complete as the "original data source" that was used to produce the Web page.

The result of achieving durability is that the Web page can be considered the canonical source of the data for the consumer on the Web. References to the "human readable" version are just as processable as data as other formats. This lessens the need for alternate formats for many data sources and widens access for more complex data sets.

3.2 Bridging the Gap

As the Semantic Web has evolved, the community has come to derive its own set of principles, layered onto the Web, called Linked Data [65], and their own platform, called the Linked Data Platform (LDP) [66]. While these specifications provide interoperability within certain communities, outside these communities and within the OWP platform they have little direct support. Yet, their underlying principles are not so different:

1. *Use URIs as names for things,*

2. *Use HTTP URIs so that people can look up those names,*

3. *When someone looks up a URI, provide useful information, using the standards (**RDF***, ***SPARQL**),*

4. *Include links to other URIs, so that they can discover more things.*

The difference is that the LDP often expects that the returned resource representation (data format) is RDF/XML or some other Semantic-Web-oriented format. This divides the users between those who want a usable format for direct display (HTML) versus those who are interested in the data in the greater context of the Semantic Web. Again, this is where RDFa, as an annotation technology, can help bridge the gap between these currently distinct user communities.

There is simply a divide in thinking between user communities, where some believe the point of accessing the Web is to harvest triples of information that you then store in your larger graph of knowledge, and others believe that you access the Web to directly display the representation you receive. As such, the representation needs to be somewhat self-contained.

Obviously, neither is really true. No one can actually store a complete graph of all the knowledge simply because it hasn't all been linked to each other, let alone considering the time to access, download, and process it. At the same time, a single instance of data, sized for retrieval over the Web, is often insufficient for medium to large-scale problems within scientific domains.

What is true is that the Web persists as a middle ground between information stores and consumers. As a middle ground, the durability of the data is essential, and markup, especially HTML, has shown great durability. RDFa annotations allow data to be encoded via simple markup to expose practical semantics to consumers and processors.

The approach used in this research uses the middle ground to enable the Open Web Platform to process information. Rather than rely upon extensive infrastructure to provide the browser just what it needs, we provide annotations in common formats that enable the browser to compute locally over data. This results in a new class of computations that utilize the Semantic Web without fully operating on either just markup or just graphs of triples.

The use of RDFa annotations allows the browser to inspect the origins of each triple. That is, the triples are expressed in particular locations within the structure of the document and often upon an HTML element. Knowledge of these locations can be useful to enable scripts to process information that is only partially annotated (e.g., column headers of a table of data). In turn, this lessens the burden on the script author to fully understand how to process the whole knowledge graph.

There is an interesting space between these communities where annotation technologies such as RDFa can help scale-up smaller-scale science and provide access to data for non-professionals; these technologies fit within the larger goals of the LDP community. As such, it is not a zero sum game as markup is uniquely positioned to provide data structured for presentation and annotated for automated consumption, all while providing an "on Web" processing model for the Open Web Platform.

Chapter 4

Foundations for Science on the Web

8. *Four desirable qualities for science on the Web are: identifiability, extensibility, flexibility, and durability.*

9. *Re-using existing representation formats along with annotations, rather than creating new formats, enables the existing platform, the Web browser, to be applied to scientific purposes.*

To have science on the Web, the scientific data sets must be directly accessible from within the OWP using the same standards that make up the existing platform. Specifically for scientific data, we will consider the data formats (HTML), augment them with annotation technologies (RDFa), and partition them into Web-sized resources with proper names (URIs). The Web browser is the most obvious starting point for this exploration. At this point, the modern Web browser is more than just a page viewer. Instead, it is a platform (OWP) for deploying applications and viewing content.

Within this platform we can observe certain qualities as manifested by HTML, RDFa, and other formats. They already demonstrate essential desirable qualities of *identifiability*, *extensibility*, *flexibility*, and *durability* for human-readable content. Within particular domains, these formats can also be applied to provide the same qualities for scientific data.

These qualities work in mutually supportive and sometimes co-dependent ways, so that as the needs of users have changed, the OWP has had the unique ability to adapt and evolve. New technologies can often co-exist with older versions, allowing the OWP to preserve investments in technologies and services deployed by businesses and other publishers. In many cases, this coexistence was a conscious choice, by those who developed the new standards, to include mechanisms to preserve backwards compatibility.

This ability to co-exist provides one of the important overarching benefits of using the OWP: your content, application, and services are all based on something that will have *durability*. While a desirable quality in itself, durability does not manifest without the support of the other qualities and without them it is never truly present. In this sense, durability may seem like a derived quality, except that to achieve it one must carefully execute on the delivery of the other qualities with durability in mind, and so crafting towards durability is an overarching goal in itself.

The OWP is designed to be agnostic to the kind of device upon which it is received. The relationships between the information and their semantics, knitted together by a common declarative language (typically HTML), provides the ability for content and applications to outlive their existing devices. This idea of supporting heterogeneity has been there from the beginning of the Web as a requirement for the platform.

Specifically, durability is derived from a number of technical properties generally exhibited by the platform. First, content is *identifiable* in both name

and syntax. Content is received over the Web by dereferencing names (URIs) and the protocols (HTTP et. al.) have well-defined mechanisms for determining the *syntax*. Representations (or formats) that work well within the platform are those that the browser can *uniquely identify* via combinations of protocol or syntax metadata and being formats that are intrinsically supported by the platform (e.g., HTML, XML, various image formats). Specifically, identifiability is built into mechanisms that layer to enhance the ability of the application to decode transport formats and provide specific support for different versions of information to invoke different processing rules.

The ability of the OWP to allow different versions of technology to co-exist on the Web is an example of *extensibility*. Such, extensibility operates on at least two dimensions: versions within technology/standards and user or vendor defined extensions within technology/standards. The platform demonstrates this quality via successful strategies that enable browsers to both view very old content while at the same time provides scripting to allow implementing new features for new types of content.

Finally, formats such as XML, XHTML [67], HTML [68][69], or JSON provide a degree of controlled *flexibility* within the representation through the use of annotations of information. These formats provide publishers and application developers the ability to represent specialized information and transport it to applications hosted within the browser. These applications can use the flexibility within the representation to transport data with specific semantics within the bounds of the chosen syntax. This flexibility contributes to supporting both the *extensibility* and *durability* of the OWP.

These qualities are all desirable for scientific endeavors. As a government or non-profit endeavor, the continually increasing cost of science, often means that economies at scale are very desirable. As a counter balance, ensuring

the durability of information is equally important. Accessing and archiving scientific data as the raw data and results, while preserving the ability to re-purpose the same information over long periods of time, is just as important to science as it is for other Web-based endeavors. As scientists participate on the Web, their research and data can be preserved, accessed, and processed well into the future. This will allow the existence of scientific Deep Web services to harvest new discoveries from data and enable replication of existing research by inspecting the data. In the end, other scientists can use these services to find new information within their domain or related areas.

4.1 Applying RDFa to Existing Vocabularies

If the OWP platform and specific data formats and techniques can provide desirable qualities, what happens when they are applied to existing vocabularies? Are the four key qualities of *identifiability*, *extensibility*, *flexibility*, and *durability* directly realized and any limitations removed? Let's consider the two major examples discussed previously: IVOA VOTable and KML.

Both VOTable and KML are XML-based formats, so using RDFa for annotation is a natural approach. As a set of attributes, RDFa can easily be added to any markup vocabulary to provide annotations. Whether or not the resulting markup is acceptable to various tools depends on how they process the markup and what they do with unknown additional attributes; a feature of processing markup with extensibility in mind.

IVOA could certainly revise VOTable XML to use RDFa directly and this might give them more flexibility within the same markup constructs. By doing so, the various semantics wrapped up within specific element constructs could then be shared across various domains while, at the same time,

vocabularies from outside IVOA could be directly usable. Specifically, the annotation of columns via the `FIELD` element could be generalized as an annotation vocabulary useful for other areas of astronomy or other scientific domains.

The various collections of properties that make exchange actually work within the IVOA framework could be reduced to a specific type and property vocabulary. With an agreed core, ontology, and object values, these annotations could be reused in other data contexts such as regular Web pages encoded in HTML. This might allows similar semantics to VOTable XML for data exchange without specialized vocabularies or tools.

Along the same lines, NASA developed a base ontology for quantities, units, and dimensions called Quantities, Units, Dimensions and Data Types Ontologies (QUDT) [70]. This ontology provides the basic types and properties for specifying scientific observations as measurements along with their units of measure. It also has the added benefit that it is grounded in SI units (International System of Unit) with additional relation and conversion information for different units of measure (e.g., Celsius and Fahrenheit map to Kelvin).

This means that IVOA can take advantage of any general standardization within the greater scientific community without having to develop everything on their own. At the same time, their contributions for solving hard data interchange problems could be re-used in other domains. This all comes as a consequence of using an enabling, common annotation format, such as RDFa.

Within the VOTable vocabulary, the various datatype attributes (`datatype`, `xtype`, `unit`, `ucd`, `utype`, and `type`) most likely reduce to a single semantic label and a specification of a unit. While the markup becomes simpler, the ontology most likely receives some of the complexity. The real benefit is

that ontology structures and vocabularies can be shared from other scientific endeavors.

Those using KML can also benefit from using RDFa. The data structures such as `ExtendedData` seem to be rarely in use—possibly because they have little effect on the user interface. Instead, data is embedded in descriptions attached to features as escaped HTML. This data in descriptions can also be annotated with RDFa without any change to KML.

The use of RDFa annotations within KML allows typing and mapping of columns of data (e.g., snow fall shown in *Figure 2.8, KML Descriptions for Data*) to known ontologies for units of measure. Instead of just having a table of data to show within the map, the application can inspect the annotations to understand that a particular column is a measurement, its unit of measure (inches), and the subject that is being measured (snow fall, last hour). A very simple example is shown in *Figure 4.1, RDFa in KML*, where it should be noted that the description is no longer escaped HTML and, as such, it is no longer valid KML under current schemas.

Figure 4.1 RDFa in KML

```
<Placemark>
   <name>  Pell City 5.8 SSW, AL</name>
   <Snippet maxLines="0">empty</Snippet>
   <description>
   <table vocab="http://weather.gov/" about="http://weather.gov/snow/AL-SC-3">
      <tr><td>STN ID</td><td property="id">AL-SC-3</td></tr>
      <tr><td>OBS DATE</td><td property="date">2013-08-08</td></tr>
      <tr><td>OBS TIME</td><td property="time">07:00 AM</td></tr>
      <tr><td>TOT PRECIP AMT</td><td property="totalPrecipitation">0.65</td></tr>
      <tr><td>NEW SNOW DEPTH (IN)</td><td property="snowDepth">NA</td></tr>
      <tr><td>TOT SNOW DEPTH (IN)</td><td property="totalSnowDepth">NA</td></tr>
   </table>
   </description>
</Placemark>
```

The consequence is that the application can perform intelligent operations on the data being displayed. In the simple case of units of measure, the application can display locale-specific variants (e.g., inches vs centimeters,

Fahrenheit vs Celsius) of the quantity. More complicated operations might be calculating average values for all data within the map view area.

Unfortunately, neither of these approaches are currently practical. Both the VOTable and KML markup vocabularies are designed as closed vocabularies. Their various XML Schema definitions are closed and do not allow annotation markup in almost every instance. The only extension is the KML `ExtendedData` element which does allow any markup construct. As such, there is no way to use RDFa for extensibility as described because any receiving tool that validates data will fail and reject the data received.

4.2 Using RDFa in HTML

While formats such as VOTable or KML are accessible within the OWP as XML data formats, the OWP does not provide any automated intrinsic semantics. As we have observed previously, XML languages have an impoverished default processing model. As such, invoking scripts to process annotations or render complex displays becomes much more difficult.

The alternative to augmenting a specific domain or scientific data format is to use HTML, the working language of the Web, as a mechanism for transmitting data. The benefit is that there is a default interpretation which provides both rendering and operational semantics within the OWP. These facilities can be used to bridge the gap between rendering for display and data processing.

A lot of scientific data is organized in a tabular format as described in *Section 1.5, Existing Approaches to Data*, and there exists a well-used table construct in HTML. With RDFa annotations, tabular data can be exchanged without inventing a new markup language, as IVOA did with VOTable. Instead, a

single HTML page can encode the context of the tabular data and use the HTML `table` element in combination with RDFa annotations. The end result is that the data is both viewable by casual browsing while, hidden from view, deeper underpinnings are available that describe the domain, context, and specifics of each column of data.

Yet, HTML alone does not provide other mechanisms for structuring, annotating, naming, and partitioning complex data. By just adding in RDFa, data can be contextualized to identify it and provide precise semantics.

But the use of annotations requires more planning with regards to the resources that are provided on the Web. What is needed is a way to apply HTML and RDFa to data sets that are potentially large and unending. Once the data can be annotated, it must be carved up into a usable set of Web resources that are accessible via URIs. A methodology that incorporates all of these parts is what is needed and will be presented in the next chapter.

Chapter 5

The PAN Methodology

10. *Data must to be partitioned into "Web-sized" representations to enable interaction with data over the Web.*

11. *Rational and regular URI naming schemes must be applied to partitions to enable interaction with data over the Web.*

12. *Quadrangles and sequence numbers are useful mechanisms for partitioning and referencing geospatial data.*

In *Section 4.2, Using RDFa in HTML*, we presented a technique for using HTML tables with RDFa annotations for tabular data sets. Applying this approach to potentially large data sets reduces to three basic problem areas:

1. The data sets are typically too large to be processed by the typical OWP implementation as one large Web resource.

2. HTML table markup lacks the necessary constructs to convey all the information encoded within typical tabular data sets.

3. A naming strategy must be developed so that information is usable on the Web, such that it can be both identified and easily retrievable by a common mechanism (i.e., over HTTP GET requests).

In solving these problems, it is essential that we return to the principles of the Web so that URI naming is used to access the data set, common formats are used, and an appropriate-sized representation is returned when the URI is accessed. We want to avoid the pitfalls encountered by prior attempts to disseminate scientific data, where large packaged archives (e.g., compressed tar files) of data files in a variety of formats are distributed for offline processing. Instead, we intend to expose this data in "Web sized" portions that are usable within the OWP.

In this chapter, a new methodology, called *PAN (Partition, Annotate, and Name)*, is described; it provides both interactive (e.g., within the browser) and offline processing of data sets. As the acronym's definition suggests, the essential steps of the methodology are to partition data sets into small portions that can be retrieved by regularly named resources whose representations are of a reasonable size for the Web. Such resources must be named by well-defined URIs and contain sufficient annotations to enable semantic processing within the OWP. Each of these components of the methodology serve essential roles in bridging the gaps between existing Web formats or data processing techniques and the needs of scientific data processing; they each contribute to extending our four key qualities of identifiability, extensibility, flexibility, and durability to scientific data.

The basic tenets of the methodology are:

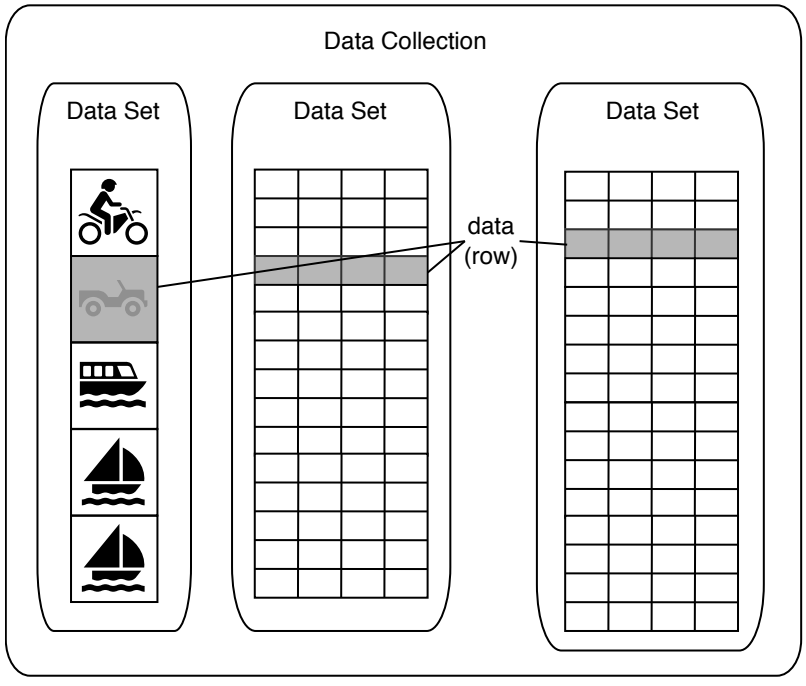
1. **P**artition the data set along properties inherent in the data (e.g., subject, temporal, or geospatial coordinates, etc.) into reasonable sized subsets suitable to Web applications.

2. **A**nnotate the data according to some identifiable ontology and encode in a common syntax using RDFa.

3. **N**ame each data partition with a unique URI using a consistent naming scheme that can be traced back to your partitioning scheme from (1).

As parts of the PAN methodology pertain to how data is represented, we must provide some general abstract model against which we will operate. In general, the terms "data", "data set", and "collection" can be used to mean different concepts or can often refer to the same kind of information object. For the purposes of this methodology, these terms will take on specific meanings and their definitions are used throughout this section and following chapters.

Figure 5.1 Abstract Data Collection Model



Our abstract model for data collections is shown in *Figure 5.1, Abstract Data Collection Model*. In broad terms, a *data collection* is a set of related *data sets* as

subject-oriented collections of data. These data sets may share some facets as keys between themselves. For example, a data set containing weather reports may reference station identifiers that are also used within another data set of metadata about the stations themselves.

A *data set* contains a singular kind of composite data structured as necessary. In relational terms, a data set maps to the concept of a data table (or tightly related set of tables). In object terms, a data set maps to instances of a single class and its instance variables. In terms of conceptual data, data sets map to sets of related observations, such as a set of weather reports.

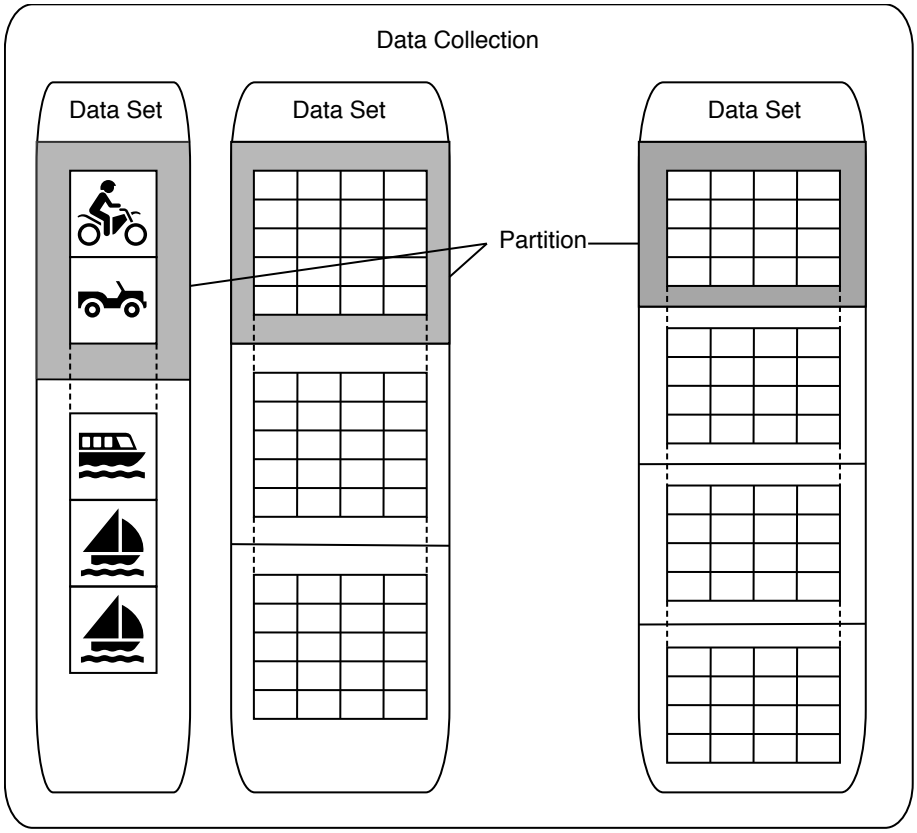
Data within the data set is often organized into common tabular structures where columns of information share the same facet definition; a row (or tuple) is used to represent the actual data (e.g., the weather report itself). Information contained within these rows is often useful in navigating the data set (e.g., observation time, geospatial locations, etc.). As such, certain facets may be used for partitioning the data.

In the abstract, *data* is encoded within records of information within the data set. Some information may not necessarily be tabular data (e.g., images) and so data within may simply be sequences of information whose facets are not easily accessible. Meanwhile, records that do exhibit some kind of regular structure can expose their facets as a tuple of information. This allows for a broad concept of columns of data without necessarily tying the actual data to a tabular model; this allows representations to be used that are not tables. The result is that data is accessible by traversing record instances within the data set and accessing the facets of these records, however they are structured.

When the size of a data set is too large to be consumed as a whole, data is grouped into partitions. A *partition* is merely a grouping of data within a data set as shown in *Figure 5.2, Abstract Data Collection Partitioning*. As each data set within a data collection may have different properties, each data set is likely to

have its own partitioning based on choices that are appropriate for the data it contains. Partitions are choices made by the publisher and are not an inherent property of the data.

Figure 5.2 Abstract Data Collection Partitioning



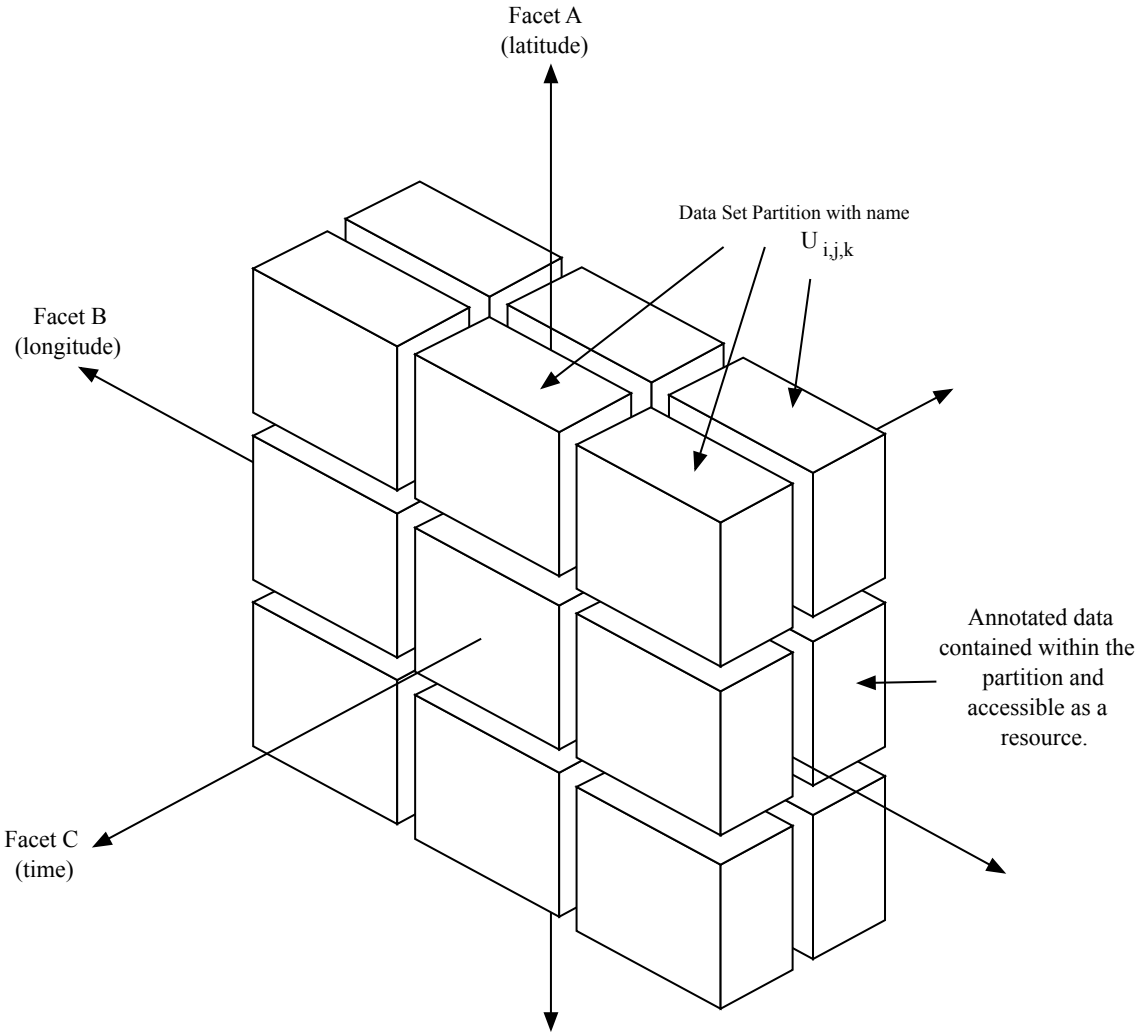
Partitioning is based on the facets of the data (e.g., time or location of observation) rather than arbitrary divisions (e.g., first 100 records). The facets necessary for partitioning are called *basic facets* and are expected for each row of data. Each *basic facet* is an atomic value which is typed such that it belongs to a specific value space.

Finally, a *partitioning* is defined by a set of *basic facet ranges*, where each individual range is associated with a specific value space partition. If the ranges are regular, it may be the case that no data matches and the partition is empty. Given that the basic facet ranges represent a choice of how the value

space is divided, a data set may be subject to more than one partitioning based on different choices (e.g., a time facet can be partitioned by different time durations).

When a data set is multidimensional, it may be partitioned by multiple basic facet ranges at the same time, as shown in *Figure 5.3, PAN Methodology Partitioning*. Each partition is shown as blocks in the diagram and can be addressed by a specific range for each of the partitioned basic facets. For example, in the diagram, the basic facets are latitude, longitude, and observation time, and each block has a specific range for each of them.

Figure 5.3 PAN Methodology Partitioning



While any data set is likely to contain a finite amount of data, it should be noted that the set of partitions can be open-ended (infinite). That is, particular basic facets, such as observation time, whose value space is infinite, lend themselves to a potentially infinite number of partitions. At any point in time, the actual set of partitions that contain data is likely to be finite and the data accessible for that particular partition is stable, but the addressable space of partitions is infinite.

The actual data contained within a data set is accessed by accessing partitions. Within each partition (blocks in the diagram) is the subset of data (rows) from the data set that has facet values in the basic facet ranges. The result is a resource for the partition contains an encoding of the subset of the data in some common format (e.g., HTML with RDFa annotations).

A consuming application accesses that data partition by a fixed URI assigned via publisher-chosen encoding rules that translate the basic facet ranges into a URI. Accessing that URI returns a representation that contains a subset in the choice of common syntax. If the data set remains unchanged, the data returned remains stable for that URI, which allows for caching of partitions.

5.1 Partitioning Data Sets

The first step in the methodology is to choose a partitioning method for the data set. The essential task is to reduce the data set into a number of smaller partitions that can be used on the Web. Each resulting partition can then be considered its own resource, available for consumption.

There are a number of reasons that this first step is necessary on the Web. First, if the whole data set were to be represented by a single resource, as is often currently done, the resulting resource will most likely be “very large”

for sufficiently rich data sets. While this may be reasonable for users who wish to download and process the data set offline, for those who wish to process the data within the OWP, the relatively large size presents a problem. Most data on the OWP is processed by in-memory processors and stored as data structures within the browser. As such, the actually representation size in terms of bytes received often results in a multiple (roughly 3-5 times in practice) in memory consumption. If the resource size is large, the memory needed to hold the data is correspondingly larger and often outside of the practical means of the browser.

Second, large resources take longer to transport. Systems that wait until the final byte is transferred to hand over data to applications will necessarily wait longer for large data representations. The consequence for the OWP is that the browser will wait longer to receive and process resources. Smaller sized resources reduce this wait and allow an application to inspect the resource representation for interesting data to process or display.

Third, when data is used by live systems built on the OWP, small resource representation sizes often increase the responsiveness of the using application's user interface. That is, requests for data that result in small resource representations are completed faster and increase the parallelism of the application such that while data is being received, other previously completed requests are being rendered. Also, other non-data requests can be serviced. The consequence is that the user interface can be continuously updated as data is being received.

Finally, while not a required feature of the OWP, but as a matter of designing for stability, most browsers limit the number of concurrent connections to the *same host* and the number of concurrent connections overall to prevent overloading the network, host, or simply running out of resources within the browser. As such, only a certain amount of parallelism is available. The

architecture of the receiving application is often better served by allowing different requests to be sequenced by the application without queuing behind other long-lived requests due to the large size of the received resource representation.

How all of these factors reduce to practical constraints is discussed and demonstrated further in *Chapter 6, An Exemplary Implementation for Evaluation*, and indicates a preference for partitioning data sets into "small enough" units of information that are still useful to the receiving application. Often, data sets, such as time-series data, have dimensions in which they continuously expand (e.g., new observations are continuously received) and so choices must be made to limit the response size. In that discussion, the tradeoffs between size, time period duration, utility of the data received, and application responsiveness are shown via an example. For any particular class of receiving application, a particular optimal partition size can be determined.

Further, the choice of partitioning criteria should not be arbitrary (e.g., first 100 observations). An application using the data set needs the ability to set criteria related to the interface controls, such as location, dates, time durations, and other criteria to select the partitions to request. For example, if the application is rendering data into a particular map view, the geospatial boundaries of the map are essential criteria for choosing which partitions to request. The partition size is dictated by the criteria used for dividing the data and the size of representation of that data partition in some particular syntax. The overall goal is a balance between a useful partition of the data and the size and complexity of the representation that must be transported over the Web and processed.

These partitions should not be considered dynamic queries used to satisfy a particular application's need for data. Instead, each partition should have some inherent aspect of the data set. That is, the choice of facets for

partitioning data should be independent of application, regular, fairly static, and useful for navigating the data set.

The stability of the choices of partitions results in a regular set of requests that can easily be mapped onto resource representations. This allows the data service to optimize the pathways for producing the representation and lowers the overall elapsed time for receiving the data. This also leads to the ability for applications to cache responses as necessary and allowed. Again, this all leads to better responsiveness for the application, of which some is shown in *Chapter 6, An Exemplary Implementation for Evaluation*.

More formally, a *partitioning* of a data set is a partitioning of data by basic facets such that:

1. Each basic facet considered has a total ordering whose relation can be used to define the partition.

2. Every partition can be uniquely identified by a set of basic facet ranges.

3. Ideally, every data (row) instance within the data set belongs to one and only one partition to prevent duplicate retrieval.

4. Any data (row) instance belongs to a partition when the normalized value of its basic facets are within the defined ranges.

5. A partitioning may result in a possibly large or infinite number of partitions, many of which may be empty.

The result of partitioning is a set of data set partition resources whose representations contain the data (rows) matching the basic facet ranges. Each resource representation should be such that the maximum size over all the possible partitions is sufficiently below practical maximums for any

consuming applications. As such, there is an interaction between the partitioning and the representation choices.

As many data sets tend to be multidimensional (e.g., weather reports have both geospatial coordinates and the date / time of the observation), the choices of partitions for basic facets are not independent. If the goal is to create data set partition resources whose maximum amount of data is below some practical threshold, the partitioning of the basic facets must consider their effects on this maximum together. As such, certain choices of partitioning may only make sense when considered in the larger context.

Also, some data set partitions may end up being empty given sparse data sets. This is a direct consequence of choosing fixed partitioning schemes. While it makes addressing and identifying the partition easier, it also allows applications to request data set partition resources that have no data in them. Navigation metadata or partition summaries can be used to give applications better choices for navigating the resources, and particular solutions are discussed in both *Section 5.3, Annotating Data Sets*, and *Section 5.2, Naming Data Sets*.

In practice, partitions are not typically realized (e.g., stored) as separate resources. Instead, they are the results of queries against some database. As such, the penalty for requesting an empty partition is the execution of a null query and a minimal amount of processing. The cost is experienced more within the receiving application that spent time and resources requesting and subsequently processing an empty partition. While this may impact performance, applications can avoid empty partitions by using navigation aids as described in *Section 5.3.3, Summaries*.

5.1.1 Value Partitioning

A totally ordered value space (e.g., integers with the > operator) is easily partitioned along the ordering relation. Any method for segmenting the possible values into subsets is sufficient as long as the result is universal and should be non-overlapping. The use of non-overlapping partitions is a useful quality as this guarantees that data will belong to a single partition. Conceptually, such relations are typical expressed as range pairs involving operations such as “greater than” and “less than or equal”.

A universal method for partitioning a totally ordered value space simply means that, given any actual possible value, there is always a partition to which the values belong. That is, the method cannot be limited to data values on-hand but to the innate properties of the value space. This prevents unexpected consequences when new data is observed outside of normal ranges.

Also, some data sets may have multiple value spaces that can be partitioned simultaneously. For example, spatial coordinates can be partitioned into partially-closed subspaces, such as grids over two-dimensional coordinates or other such tilings. A simple application of this is dividing geospatial coordinates into quadrangles.

While partitioning data sets considers the set of value spaces together to define a partition, a value space itself must first be properly partitioned. Each value space partitioning should be sufficiently capable of being considered on its own. A value space partitioning must also be independently verifiable without considering another value space. For example, latitude and longitude values can be partitioned independent of each other even though, when partitioning a data set, they are usually considered together.

Also, some values spaces map oddly to independent relations. For example, spatial coordinates for locations on the surface of an ellipsoid can be measured in a number of different systems but they all have the problem that rotations eventually overlap (e.g., 370° is the same as 10°). A partitioning of the surface of an ellipsoid can be universal only if the values are first normalized.

5.1.2 Date/Time Partitioning

Data sets that contain time-series data (e.g., observations over time) have dates and times associated with each observation which may be very important to the consuming application. For example, weather data is often received as a sequence of discrete weather reports recorded at specific times. A consuming application typically filters the weather data for the past hour, past day, or past week. As such, navigating along date and time for certain durations is important for the consuming application.

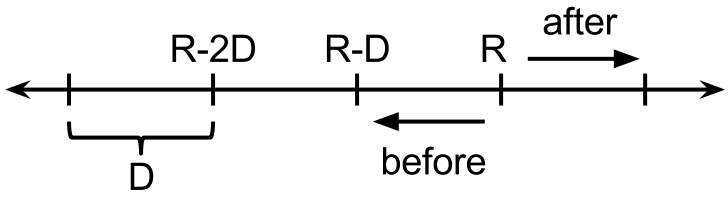
While allowing a client to specify a start and end time may be the most generic approach, the resulting resource may be very large given a particular choice by the client. The result of such large queries is that the response must be broken into separate "pages" where the pagination control is specific to the request. As such, the use of pagination for user-specific queries creates a management problem for handling temporary resources that lack permanence beyond the duration of the client's interaction.

By partitioning against fixed time durations (e.g., 30 minutes), the resource size can be constrained by the service. As noted above, the duration time will depend on data density within the time-series data, record size, and other properties. Once the duration is fixed, clients can compute the specific time period for the partition required and, depending on URI assignments, the fixed duration will result in stable naming and caching. Employing fixed durations results in better overall performance over the Web for both the client and data service provider.

Moreover, a client need not completely understand the fixed durations used for partitioning. Instead, if a client application can request from a default starting point (e.g., the latest time partition) or from a specific date/time the server can map their request via a redirect to a correctly named resource for that time period. Once the starting point is received, the application can navigate back and forth along the time line by following links that should be provided within the partition's data annotations.

More formally, each data set is partitioned via a date and time facet (e.g., observation date and time). The value space for the date and time is partitioned into segments of time of a fixed time duration D (e.g., PT30M) starting with a reference time R, typically given in a UTC time. Data belongs to a partition if the date and time facet is after the starting time and before or equal to the ending time. That is, the date and time dimension is divided into segments of time duration D. This segmenting shown in *Figure 5.4, Date and Time Partitions*.

Figure 5.4 Date and Time Partitions



5.1.3 Geospatial Partitioning

Quadrangles (rectangular regions mapped onto the reference ellipsoid) are a common way to organize map-oriented data; they provide a familiar way to partition geospatial data sets. A requesting application need only refer to the correct quadrangle, via some reference mechanism, to retrieve data for a

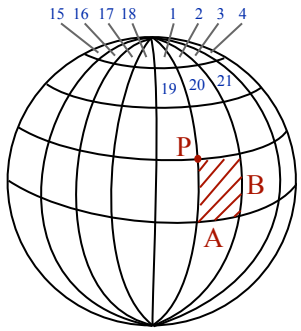
particular region. The data located within the extent of the quadrangle is then returned to the application.

Such geospatially-oriented data sets can be easily partitioned by quadrangles into resources; the result is a uniform covering of spatial coordinates. While some quadrangle partitions may be empty, others contain varying amounts of data. By reducing the size of the quadrangle, the size of the data within a specific quadrangle can be reduced; the choice of quadrangle size is one factor in controlling the resulting representation size.

Also, the set of non-empty quadrangles gives a simple overview of geospatial coverage of the data set. Each quadrangle represents a specific set of coordinate boundaries. The amount of data within those boundaries combined with the set of quadrangles that have data is a very useful overview of the data coverage of a specific data set.

Finally, using an appropriate fixed quadrangle size results in a uniform and unique set of quadrangles. Each quadrangle can be identified with a unique *sequence number* by a simple enumeration starting at the north pole and working towards the south pole (see *Figure 5.5, Quadrangles*). This number has the nice property that it is easily calculated, as detailed in *Appendix A, Sequence Numbers*, from any given latitude and longitude coordinate. As a result, locations are easily hashed into a sequence number.

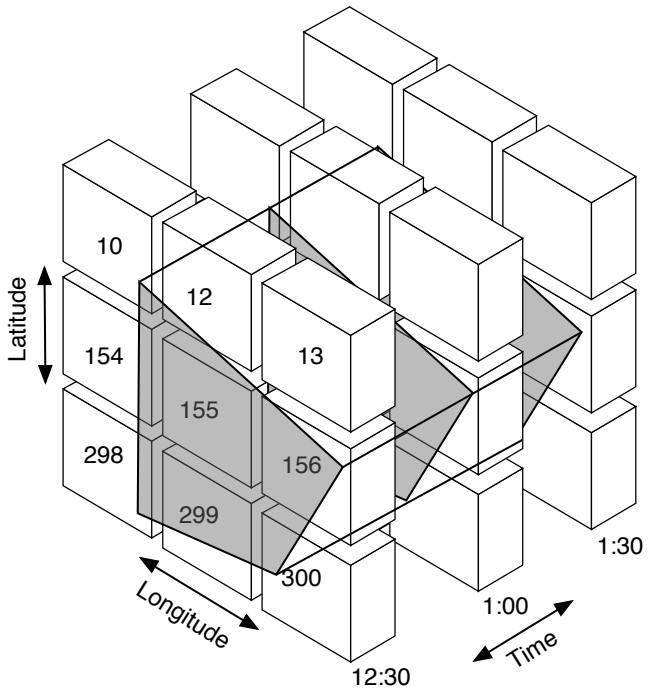
Figure 5.5 Quadrangles



The result of using a fixed quadrangle size is that the sequence number can be used as part of the name of the partition. This can be used by client applications to quickly calculate, from a given position, the sequence number of the quadrangle that contains relevant data. Combining the sequence number with other facets allows a client application to retrieve a specific quadrangle data resource.

For example, given time-series geospatial data and assuming a quadrangle size of 2.5°, there are 10,368 (360 / 2.5 * 180 / 2.5) quadrangles, and each of which can further be divided by half-hour time durations. A client with specific questions about a particular geospatial region over a specific time period can calculate the data partition resources necessary to retrieve a complete set of data covering the region. This calculation of the partitions required for a geospatial region is shown in *Figure 5.6, Quadrangle Coverings*, where each time partition requires a different data partition for each quadrangle that covers the region.

Figure 5.6 Quadrangle Coverings



Formally, a quadrangle partitioning is defined by two pairs of coordinates, one for each of latitude and longitude. The range values are always defined as a interval $a \ b$, where $a \quad b$. The upper bound is not included in the range except for longitude, where -90 must be in the last set of quadrangles. Any coordinates must first be normalized before the quadrangle is assigned. This partitioning is also used in the calculation of the sequence numbers as shown in *Appendix A, Sequence Numbers*.

5.2 Naming Data Sets

Following one of the most important basic principles of the Web, all the major constructs of the PAN ontology must be named. While data collections and data sets are more easily named as they typically lack in number, their partitions and summaries are plentiful. A carefully crafted naming scheme must be developed that aligns with the partitioning choices made so that each resource has a distinct and sensible URI.

The *Architecture of the World Wide Web, Volume One* enumerates a number of different principles, "good practices", and constraints for naming. The one principle that applies to URIs directly is that of "Global naming leads to global network effects". That is, a URI that has global scope in terms of its usability allows other applications and representations to point to that resource via the URI.

When inspecting URIs, as a good practice, it is recommended that:

> *"Agents making use of URIs SHOULD NOT attempt to infer properties of the referenced resource [from the URI itself]."*

Time has shown that, in practice, this is not necessarily how good systems evolve or are used. Instead, how implementers and systems use metadata in URIs is both inferred by consumers through inspection or via explicit documentation. As consumers come to depend on certain URI structures to access information (e.g., constructing a URI for weather for a particular region), these structures are documented by service providers to enhance their usability. The result is that both applications and users inspect URIs and extract metadata to use in either the application or in the construction of new URIs.

Along the way, the Technical Architecture Group (TAG) at the W3C revised their position on the issue of metadata encoding in URIs and drafted a new finding [71] that softens their view on opaque URIs. In general, there is a difference between metadata used to inform the processing of a resource versus metadata that is used in formulating URIs to new resources. In the case of the correct processing, a receiving application should never depend on the correctness of information that is inferred (e.g., assuming that a URI ending with a particular extension [`.xml`] is of a particular media type [`text/xml`]).

While URIs should often be treated as opaque identifiers, there is an implicit social contract in a well-formed URI that encodes useful metadata. For example, if a user encounters a resource identified by `http://www.example.org/weather/San-Francisco` they might infer that the weather for other cities could be retrieved simply by changing "San-Francisco" to "Montreal" or "Edinburgh." Whether or not the retrieval succeeds and does return weather for the requested region is a matter of social experimentation. The social contract between the naming authority (the domain owner) and the consumer (a user who accessed that URI in their browser) may or may not be broken depending on whether the user's experiment delivers the desired result.

The TAG's new position is codified into the following recommended good practice:

> *URI assignment authorities and the Web servers deployed for them may benefit from an orderly mapping from resource metadata into URIs.*

In the previous example, this reduces to codifying whether there is a URI template `http://www.example.org/weather/{location}` with the variable `location` upon which both the system and user operate. This allows explicit documentation of the social contract, via the URI, between the user and the server. Such a URI template may be explicitly documented, inferred from examples, or discovered by inspecting a Web form or other application use.

The usefulness of embedding metadata into URIs does depend on the implementation choices. For example, if a system exposes an internal identifier using a non-descriptive mapping such as `http://www.example.com/node/3452` for the weather for San Francisco, there is very little in the segments of the URI path that a user might recognize to indicate either the location or that weather information is to be expected. While the documentation might explicitly state that a code (e.g., "`3452`") is to be used to express weather for a particular location, the usability of the URI suffers and the social contract is less effective.

Such descriptive and orderly mapping also allows the system to evolve over time as URI structures that are no longer in use can be mapped to new structures via redirection or other techniques. As the system evolves over time, descriptive metadata has an advantage over other metadata because it is easier to map. In contrast, if the internal identifiers change, the previous URIs may become very hard to maintain or map; the system then tends to become brittle and the *durability* of the system may be placed in jeopardy [72].

While the choices for creating an "orderly mapping" might seem like more of an art than a science, such decisions reduce to discovering an appropriate set of metadata that can uniquely identify a resource without resorting to arcane measures. Many domains of science have data with properties that can describe portions of their data that make obvious choices for metadata to encode into URI structures (e.g., coordinate systems, genus/species, regional identifiers, etc.). Other choices may be more subtle and require use-case scenarios to understand how a user might access the data and what expectations for *durability* of the URI are needed.

We want to name data collections and data set partitions such that we maximize their network effects on the Web. At its simplest, we want a data partition to be able to be referenced and shared with a stable URI that returns a reasonably stable representation of the data subset. In more complex situations, we want a URI naming schema that maximizes our ability to reference and share the important aspects of the data collection.

In crafting names, there are a number of good practices to consider. These practices, taken together with expected behaviors within the OWP, result in a set of guidelines for publishing data collections on the Web. Considering the PAN Methodology, we want a set of guidelines that work specifically for the methodology.

The following are specific good practices from Web Architecture:

1. *Identify with URIs* — To benefit from and increase the value of the World Wide Web, agents should provide URIs as identifiers for resources.

2. *Reuse URI schemes* — A specification SHOULD reuse an existing URI scheme (rather than create a new one) when it provides the desired properties of identifiers and their relation to resources.

3. *Ownership / Authority* — The domain owner provides a default authority over naming of resources. Systems should allow references to URIs without dictating the authority or ownership.

4. *Avoid Aliases* — Avoid arbitrary URI aliases for the same resource. That is, a single resource should have one and only one URI.

To these good practices we add:

5. Use URI path segments to encode basic facets and other metadata.

This practice is very useful for scientific data sets that are partitioned in that certain parameters or choices of partitioning can be encoded with the URI. When a reference is made to a partition, these path segments can be preserved via the algorithms for resolving relative URI references. If metadata is encoded in query parameters, they will not be preserved and an application may need to understand how to reformulate the necessary query parameters.

As will be shown in *Chapter 6, An Exemplary Implementation for Evaluation*, using path segments is helpful for enumerating data. In examples from that chapter, the URI path encodes the information such as the choice of quadrangle size (e.g., `/data/q/2.5/` for 2.5° quadrangles). When a resource path is made relative to that (e.g., `./n/2976/`) it resolves to produce a full path that still includes the choice of size (e.g., `/data/q/2.5/n/2976/`). The property of subsumption has been applied in that the subsequent path inherits the metadata of the parent path. In the case of geospatial data with quadrangle partitions, this becomes very useful so that the choices of quadrangle size are preserved along with the sequence number (e.g., 2.5° quadrangles with sequence number 2976).

This brings us to the last added good practice:

6. Metadata choices encoded in URIs should be preserved; URI naming that uses subsumption can preserve some metadata for relative references.

5.3 Annotating Data Sets

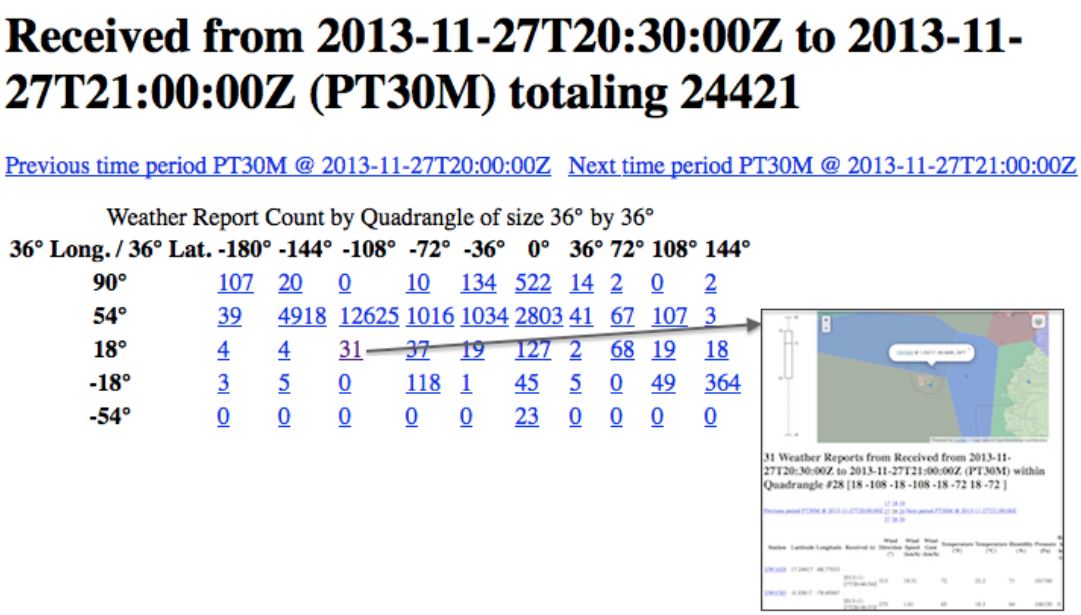
There are many possible choices for Web representations of data but in *Section 4.2, Using RDFa in HTML*, we made the argument that some combination of HTML markup and RDFa annotations offers the right balance between casual access and machine processing. We observe that annotated Web pages serve a dual purpose: they can both be rendered for display and consumed as data for processing. The question that remains is how this approach makes the data displayed in tables consumable as data.

For example, a summary table of weather reports per quadrangle for a weather-related site is shown in *Figure 5.7, Quadrangle Summary Page*, where each entry displays a count that links to a page that contains the data set partition of weather reports within that particular quadrangle. A thumbnail of this page is shown for the quadrangle with sequence number 28 that contains 31 weather reports.

The summary page and the linked pages both contain tabular data, annotations that label columns or rows for those tables, and information about the page itself. The process of turning these pages into processable data requires that both the context of the table of data and the table itself receive annotations. We will start by looking at the markup behind the first summary table shown in *Figure 5.7, Quadrangle Summary Page*.

The quadrangle summary page has a number of properties associated with the resource that is typed and identified with a simple wrapper as shown in

Figure 5.7 Quadrangle Summary Page



Received from 2013-11-27T20:30:00Z to 2013-11-27T21:00:00Z (PT30M) totaling 24421

Previous time period PT30M @ 2013-11-27T20:00:00Z Next time period PT30M @ 2013-11-27T21:00:00Z

Weather Report Count by Quadrangle of size 36° by 36°

36° Long. / 36° Lat.	-180°	-144°	-108°	-72°	-36°	0°	36°	72°	108°	144°
90°	107	20	0	10	134	522	14	2	0	2
54°	39	4918	12625	1016	1034	2803	41	67	107	3
18°	4	4	31	37	19	127	2	68	19	18
-18°	3	5	0	118	1	45	5	0	49	364
-54°	0	0	0	0	0	23	0	0	0	0

31 Weather Reports from Received from 2013-11-27T20:30:00Z to 2013-11-27T21:00:00Z (PT30M) within Quadrangle #28 [18 -108 -18 -108 -18 -72 18 -72]

Figure 5.8, Partition Summary Markup. Within the various markup constructs, we attached properties, via RDFa attributes, that define aspects of the partition summary, such as the basic facet range that is being summarized. This information was displayed to the user within the header (`h1` element) and there is no need to duplicate that information. Instead, the various pieces of information are wrapped with `span` elements with the appropriate property name.

Figure 5.8 Partition Summary Markup

```
<div typeof="PartitionSummary"
     resource="http://www.mesonet.info/data/q/36/2013-11-27T20:30:00Z">
<h1>
<span property="range" typeof="FacetPartition">
   Received from
   <span property="start">2013-11-27T20:30:00Z</span> to
   <span property="end">2013-11-27T21:00:00Z</span>
   (<span property="length">PT30M</span>)
</span>
totaling
<span property="count">24421</span>
</h1>
...
</div>
```

These RDFa attributes, the various elements, and their contents result in the graph shown in *Figure 5.9, Partition Summary Annotations* with the triples listed in *Appendix D: Figure D.1, Partition Summary Triples*. Some are simple value associations, such as the total number of weather reports within the time period (i.e., the `count` property with value "24421"), while others contain more complex structures (i.e., the `range` property). In the case of the `range` property, the value describes the basic facet range in terms of start, end, and length (duration).

Graph Notation

< . . . > is a URI

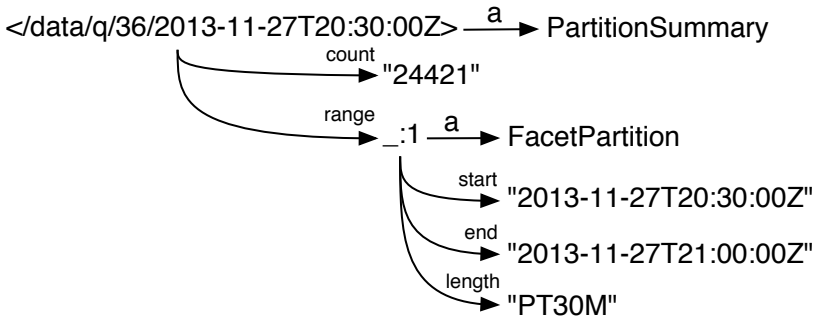
start → is a property labeled with a CURIE or URI

a → is a type property

_:x is a blank node label

Each object value is either a literal, a URI, or a CURIE.

Figure 5.9 Partition Summary Annotations



Within the partition summary shown in *Figure 5.7, Quadrangle Summary Page*, is a table of data whose row and column labels are the North-West coordinate of the quadrangle. This is an example of a labeled table, where the row and column labels can be used to find data, and the entry typically contains a single kind of data (e.g., weather report counts per quadrangle). For this kind of table, at minimum we must describe the entry and the column and row labels.

To avoid excessive annotation, the annotations just describe what the entries contain and not each of the entries themselves. This markup is shown in *Figure 5.10, Labeled Table Markup*, where the caption contains most of the complexity. There are many possible properties but the minimum of the value type and the row and column label types are shown.

Figure 5.10 Labeled Table Markup

```
<table typeof="LabeledTable">
<caption property="title"
   <span property="entry" typeof="Entry">
      Weather Report Count
      <span property="valueType" resource="xsd:int">
      by Quadrangle of size 36° by 36°
      <span property="columnLabel"
            resource="http://www.w3.org/2003/01/geo/wgs84_pos#lat"/>
      <span property="rowLabel"
            resource="http://www.w3.org/2003/01/geo/wgs84_pos#long"/>
   </span>
</caption>
...
</table>
```

The markup uses a small set of annotations, shown in *Figure 5.11, Labeled Table Annotations*, with the triples listed in *Appendix D: Figure D.2, Labeled Table Triples*. The table subject (i.e., `_:4`) has a small number of properties and the blank node for the `entry` property contains the necessary information to understand the table cells. For this example, from the `valueType`, `columnLabel`, and `rowLabel` an application can understand that an integer value, longitude, and latitude can be found for each table cell. Other properties can be used to allow applications to understand further information contained within the table cells.

Finally, the labeled table is expected to have a single row of column labels and a single table cell to label each row, as shown in *Figure 5.12, Labeled Table Row Markup*, where the first row corresponds to the table header shown in *Figure 5.7, Quadrangle Summary Page*. Each subsequent row has a single row label followed by each table cell entry that contains the link to the data set partition. The value for the table cell entry is not marked up but instead obtained by traversing the table.

Figure 5.11 Labeled Table Annotations

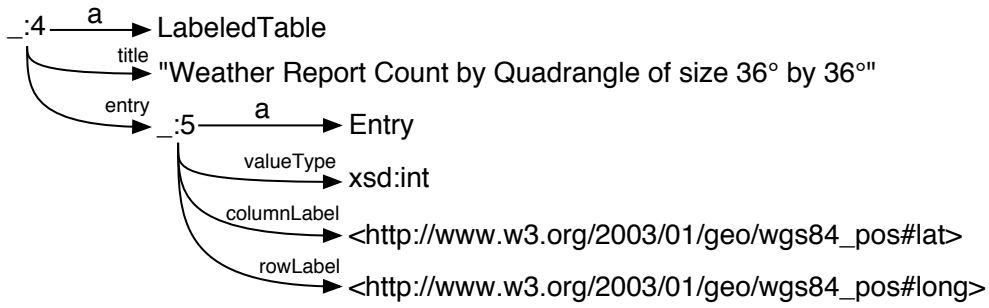


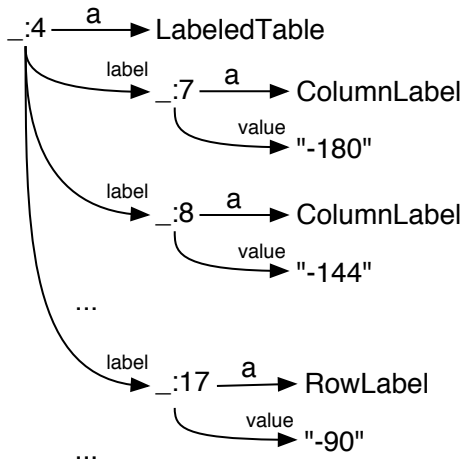
Figure 5.12 Labeled Table Row Markup

```
<tr>
  <th>36° Long. / 36° Lat.</th>
  <th property="label" typeof="ColumnLabel"><span property="value">-180</span>°</th>
  <th property="label" typeof="ColumnLabel"><span property="value">-144</span>°</th>
  <th property="label" typeof="ColumnLabel"><span property="value">-108</span>°</th>
  ...</tr>
<tr>
  <th property="label" typeof="RowLabel"><span property="value">90</span>°</th>
  <td><a href="n/6/2013-11-27T20:30:00Z">107</a></td>
  <td><a href="n/7/2013-11-27T20:30:00Z">20</a></td>
  <td><a href="n/8/2013-11-27T20:30:00Z">0</a></td>
   ...</tr>
```

The result of processing the table annotations is shown in *Figure 5.13, Labeled Table Row Annotations* with the triples listed in *Appendix D: Figure D.3, Labeled Table Row Triples*, where an additional property `label` is added to the table subject (_:4) that enumerates the row and column label subjects. Each of these subjects has a single property of the value of the label. Using a typed blank node allows the client to lookup the specific row or column element by type for processing within the client.

The table can now be traversed to find specific entries, either by enumerating each table cell or by locating specific cells via the labels. The markup corresponds to a matrix of entries whose values are typed as described by the `Entry` node in the annotation graph. An application can process the table as data to gain additional inferred triples or process the data directly; this is an exemplar of a hybrid model of data processing, where both the annotations as triples and the markup are used to process the data.

Figure 5.13 Labeled Table Row Annotations

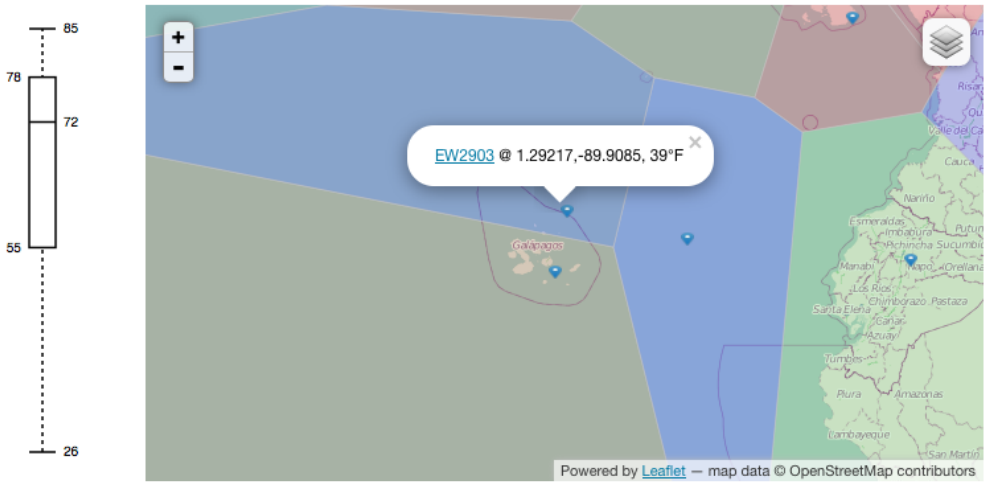


When an application accesses a particular quadrangle's data partition (shown in *Figure 5.14, Quadrangle Data Partition Page*), a similar annotation scheme is applied. Essential information about the quadrangle is shown to the user along with a visualization and a table of data containing the weather reports. Each page similar to this contains links to other partitions, allowing a user to navigate to other partitions for nearby quadrangles or different time periods.

The partition starts as before, with the essential information about the partition being described in the heading markup, as shown in *Figure 5.15, Data Partition Markup*. The count and basic facets of the partition are described via various properties. In this example, describing the basic facet labels allows an application to understand the semantic label associated with the aspect of the data used in partitioning. The fact that quadrangles are used for partitioning is described as well.

The resulting annotation graph is shown in *Figure 5.16, Data Partition Annotations* with the triples listed in *Appendix D: Figure D.4, Data Partition Triples*, where the `Partition` typed resource has `count` and `range` properties. The range properties can be used to determine the basic facet

Figure 5.14 Quadrangle Data Partition Page



31 Weather Reports from Received from 2013-11-27T20:30:00Z to 2013-11-27T21:00:00Z (PT30M) within Quadrangle #28 [18 -108 -18 -108 -18 -72 18 -72]

17 18 19
Previous period PT30M @ 2013-11-27T20:00:00Z 27 28 29 Next period PT30M @ 2013-11-27T21:00:00Z
37 38 39

Station	Lattitude	Longitude	Received At	Wind Direction (°)	Wind Speed (km/h)	Wind Gust (km/h)	Temperature (°F)	Temperature (°C)	Humidity (%)	Pressure (Pa)	Ra la ho (c
DW1029	17.24617	-88.77033	2013-11-27T20:46:54Z	313	19.31		72	22.2	71	101740	
DW4785	-0.35817	-78.49567	2013-11-27T20:36:37Z	275	1.61		65	18.3	64	100150	0

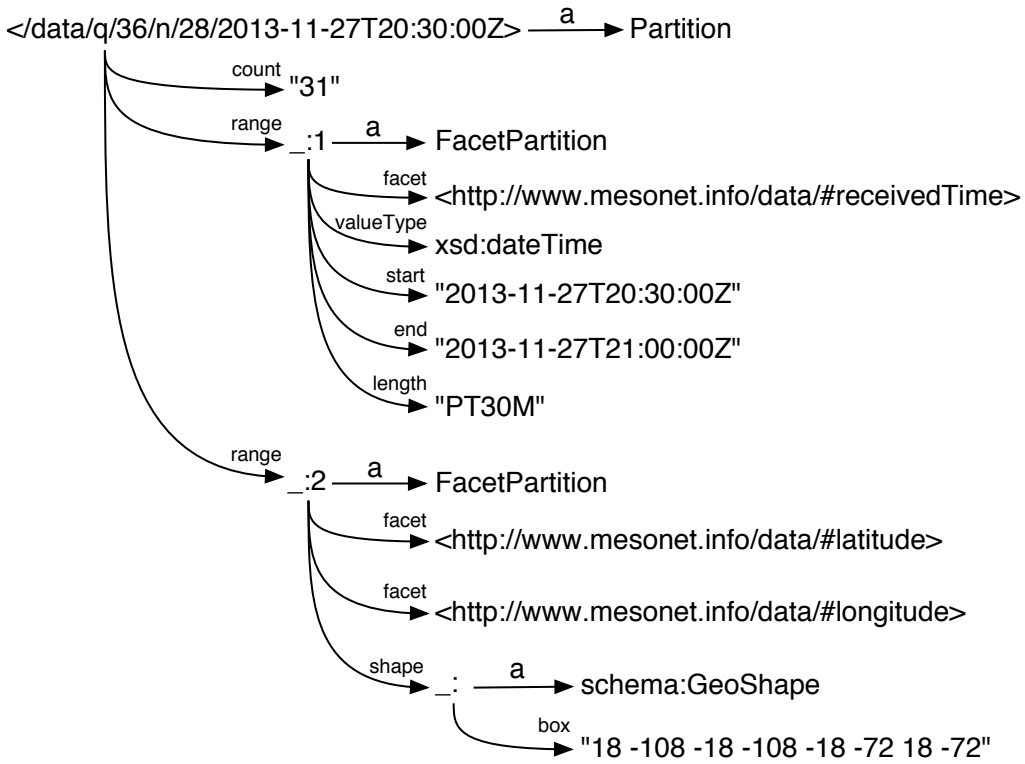
ranges of the partition by solely looking at the annotation graph. A quick check of the `count` property reveals whether the partition is empty.

When the partition contains tabular data, a simple table construct is used to encode the information, where the columns of data are annotated within the header, as shown in *Figure 5.17, Data Partition Table Markup*. Each column of data is described within the `thead` element and typed as a column. The properties describe a title, semantic label, and value space. The value space

Figure 5.15 Data Partition Markup

```
<div typeof="Partition"
     resource="http://www.mesonet.info/data/q/36/n/28/2013-11-27T20:30:00Z">
<h1><span property="count">31</span> Weather Reports from
<span property="range" typeof="FacetPartition">
   <span property="facet" resource="/data/#receivedTime"/>
   <span property="valueType" resource="xsd:dateTime"/>
   Received from
   <span property="start">2013-11-27T20:30:00Z</span> to
   <span property="end">2013-11-27T21:00:00Z</span>
  (<span property="length">PT30M</span>)
</span>
within Quadrangle #28 [
<span property="range" typeof="FacetPartition">
   <span property="facet" resource="/data/#latitude"></span>
   <span property="facet" resource="/data/#longitude"></span>
   <span property="shape" typeof="schema:GeoShape">
      <span property="schema:box">18 -108 -18 -108 -18 -72 18 -72</span>
   </span>
</span>]</h1>
```

Figure 5.16 Data Partition Annotations



is further described to enumerate the quantity being measured and various other aspects of the column's data.

The data itself is contained in a sequence of table bodies (i.e., a `tbody` element); one for each weather station in the example markup. The cells in each column contain plain text values. An application can align the information contained in the column header with the value found in the table cell to understand how to interpret the text value.

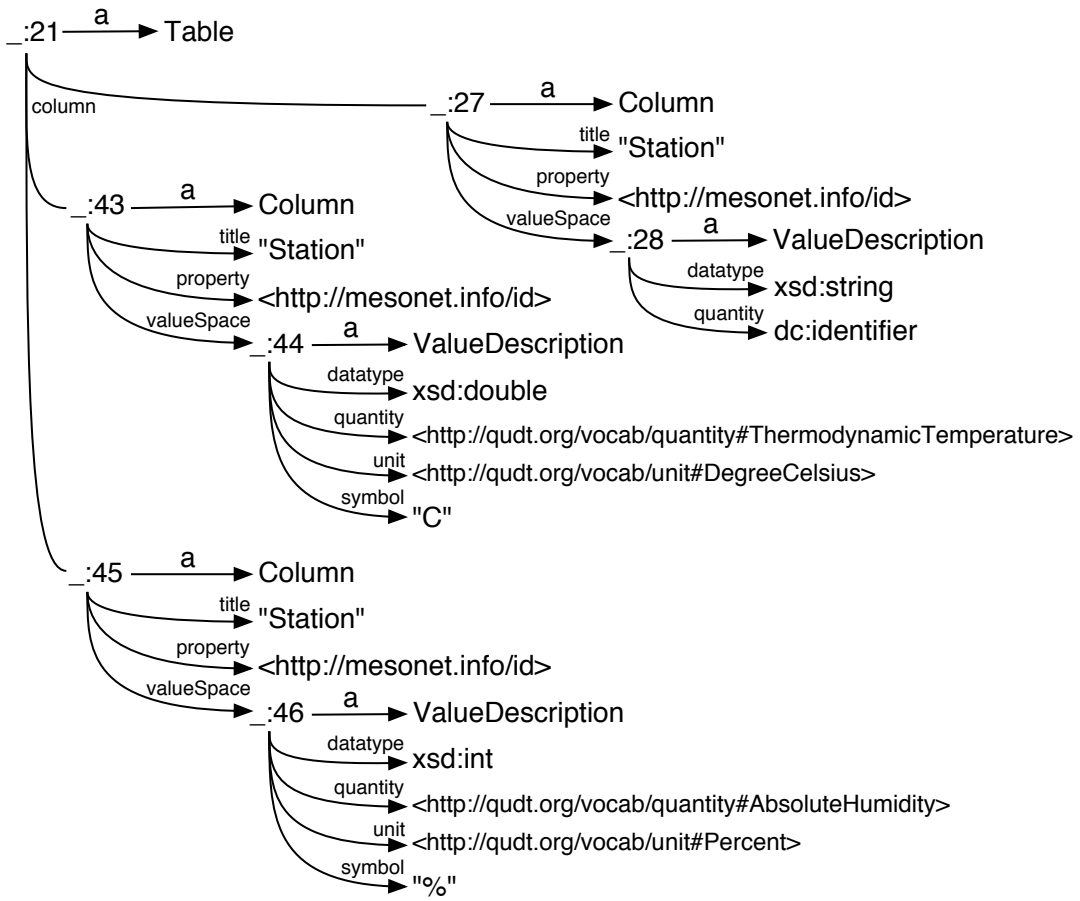
Figure 5.17 Data Partition Table Markup

```
<table typeof="Table">
<thead>
<tr>
  <th property="column" typeof="Column">
     <span property="title"Station</span>
     <span property="property" resource="w:id"/>
     <span property="valueSpace" typeof="ValueDescription">
        <span property="datatype" resource="xsd:string"/>
        <span property="quantity" resource="dc:identifier"/>
     </span>
  </th>
...
  <th property="column" typeof="Column">
     <span property="title">Temperature</span>
     <span property="property" resource="w:airTemperature"/>
     <span property="valueSpace" typeof="ValueDescription">
        (°<span property="symbol">C</span>)
        <span property="datatype" resource="xsd:double"/>
        <span property="quantity" resource="quantity:ThermodynamicTemperature"/>
        <span property="unit" resource="unit:DegreeCelsius"/>
     </span>
  </th>
  <th property="column" typeof="Column">
     <span property="title">Humidity</span>
     <span property="property" resource="w:airHumidity"/>
     <span property="valueSpace" typeof="ValueDescription">
        (<span property="symbol"%</span>)
        <span property="datatype" resource="xsd:int"/>
        <span property="quantity" resource="quantity:AbsoluteHumidity"/>
        <span property="unit" resource="unit:Percent"/>
     </span>
  </th>
...
</tr>
</thead>
<tbody>
   <tr>
     <td>DW1029</td>
...
     <td>22.2</td>
     <td>71</td>
...
   </tr>
</tbody>
...
</table>
```

The resulting annotation graph for the table is shown in *Figure 5.18, Data Partition Table Annotations*, with the triples listed in *Appendix D: Figure D.5,*

Data Partition Table Triples, where the table columns are enumerated on the table subject (i.e., `_:21`) `column` property. Each table column is completely defined with the triples but an application must track the location of the column subject to be able to associate the column definition's triples with the column in the markup. This is a straightforward process of locating the `th` element by the `Column` type and examining the annotation graph for the related subject.

Figure 5.18 Data Partition Table Annotations

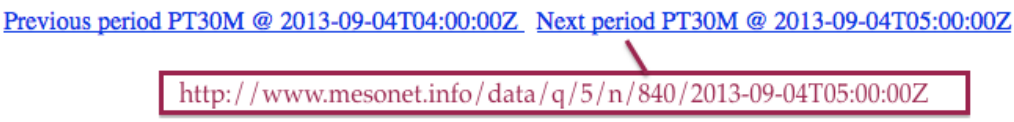


5.3.1 Partition Links

Each link from partition to partition needs to be typed and qualified so that applications can distinguish the link from others and understand what the resource may provide before retrieving the linked resource. For example, a

rendered set of previous and next links, by time period, is shown in *Figure 5.19, Rendered Partition Links*, where the previous and next 30 minute durations are accessible. If an application assumes the partition navigation is by time, all that is needed is to look for the property `next` or `previous` in the annotation graph (e.g., the first triple in *Appendix D: Figure D.6, Partition Link Triples*).

Figure 5.19 Rendered Partition Links



There are some assumptions being made with this kind of link navigation:

- the target resource is another partition or partition summary,
- the basic facet being navigated is known and time related,
- the duration or start time of the linked partition is predetermined.

In general, we want the current partition to provide more information to remove the need for these assumptions. As such, we want to provide annotations that allow an application to inspect the current annotation graph and understand whether a partition link is worth visiting in advance of retrieval. This facilitates understanding whether the link traverses in the right direction and when "enough data" has been received in comparison to some application need.

For any link to a partition, we use the `typeof` attribute as shown at the very beginning of *Figure 5.20, Partition Link Markup*, to indicate the resource type; just as the partition resource would assert about itself if retrieved. This link establishes a new subject node and any descendant element properties will be associated with the link resolved from the `href` attribute. In this particular

case, we will use this feature of RDFa to associate properties with the target resource.

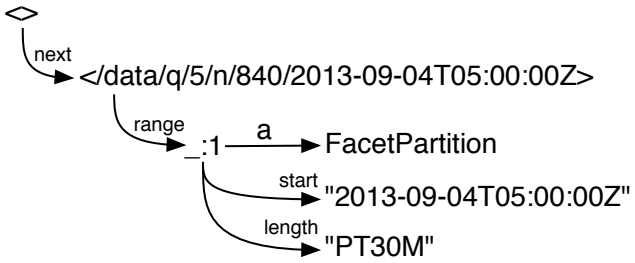
Figure 5.20 Partition Link Markup

```
<a rel="next" typeof="Partition"
   href="http://www.mesonet.info/data/q/5/n/840/2013-09-04T05:00:00Z">
   Next period
   <span property="range" typeof="FacetPartition">
     <span property="length">PT30M</span> @
     <span property="start">2013-09-04T05:00:00Z</span>
   </span>
</a>
```

The resulting annotations for the link example is shown in *Figure 5.21, Partition Link Annotations*, with the triples listed in *Appendix D: Figure D.6, Partition Link Triples*. The `range` property contains a node typed as `FacetPartition` that contains a number of properties intended to help the consumer understand the facet range of the partition. In the example, the start time and length (duration) are defined via additional properties. Again, the parent element in the markup (i.e., the `span` with the type `FacetPartition`) defines a new subject (a blank node) and descendant elements assign property values to the range's node.

In the annotation graph that contains the triples we can follow from the current partition via the `next` property to the subject of the linked resource. That resource has a `range` property that allows us to get information about the facet partition. While the example shows the `length` and `start` properties, there can also be `facet` and `valueType` properties for defining the domain value and data type, respectively, of the partition range. This allows an application to understand what basic facet the range is over (e.g., received observation time) and a particular encoding (e.g., time specified via `xsd:dateTime`).

Figure 5.21 Partition Link Annotations



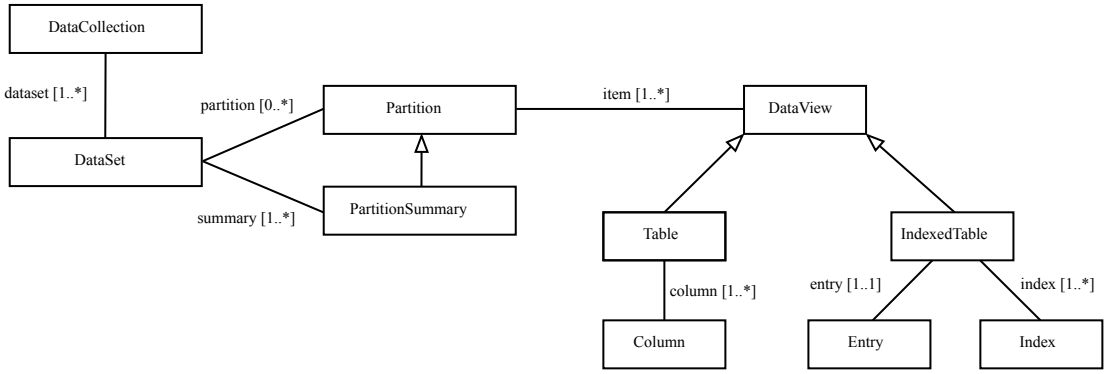
5.3.2 Ontology Overview

The previous sections used an ontology we developed for data sets that can be easily used to annotate HTML structures with RDFa. An attempt was made to keep the ontology as small as possible without including complexity from related ontologies that may have been developed for other purposes. The resulting class structure is shown in *Figure 5.22, Pantabular Ontology Class Structure*.

In the ontology, there are classes for the basic structures: `DataCollection`, `DataSet`, `Partition`, and `DataView`. At the very root, a data collection is just a container that is typed as `DataCollection`. A data collection has a set of `dataset` properties that holds the subject node (URI) of each contained data set; each typed as `DataSet`. Both of these classes have an extensive set of Dublin Core Metadata properties [73] that can be used to further describe the data collection or data set.

A `DataSet` instance has a number of important properties. First, specific partitions of the data set can be associated by the `partition` property. The target object is a instance of `Partition` or one of its subclasses.

Figure 5.22 Pantabular Ontology Class Structure



The `Partition` instance has a set of `item` properties which associates the data set partition with the data subset. Each item is currently limited to tabular data (`Table`) or a labeled table (`LabeledTable`) in the PAN ontology but other item types are possible and allowed.

Each `Table` item type is further described by a set of column descriptions. These descriptions describe the subject, type, and other facets of the data within the column. The properties can be used to find specific columns of data by units, subject, or other facets, but the table cell entries themselves are not annotated.

VoID

The VoID Vocabulary [74] is an existing ontology that can be used to describe data sets but not encode the data itself. VoID addresses how the data set is accessed. Many of the properties are similar in functionality to what is described in *Appendix B, Resource Schemes and Discovery* but that is considered out of scope. VoID does not address how columns of information within the data set are described and so does not help with the exchange of tabular data. As VoID helps in describing a data set at a higher level, it is compatible with the Pantabular ontology and can compliment its deployment.

The `LabeledTable` item type is a matrix representation with cells of data and row and column indices. Each `LabeledTable` item type is described by

a singular `Entry` instance that describes the entry contents and a set of `Label` instances. The entry description can be used to understand what each table cell represents. The indices can be used as coordinates for finding specific entries within the matrix.

The choice of item type depends on what kind of data is being represented. While there may be other factors, a basic set of rules is as follows:

- If the data set contains a sequence of records or tuples (e.g., weather reports) and the resource represents a data set partition, the `Table` type should be used for the item. This allows description of each type column of data and a simple enumeration of the records.

- If the resource is a summarization of data partitions (e.g., counts of records per quadrangle) over two basic facets (e.g., latitude and longitude), a `LabeledTable` should be used where the column and row indices are the two basic facets. This representation only works if a matrix with a finite number of entries can be constructed.

- A `LabeledTable` can always be represented as a `Table` whose rows are a set of tuples containing: the row label, column label, and table cell value.

- If the data is not a finite matrix or set of tuples, a custom representation outside of the PAN ontology must be used. For example, each item could be a link to an image resource.

The `LabeledTable` representation is akin to a statistical marginal. Typically, the table cell value has a single value. For example, that value might contain the number of records or the average temperature per quadrangle.

While it is conceivable, with multidimensional data sets, that the marginal would contain a tuple of data, such a use would be an extension of its

intended use. Originally, the `LabeledTable` was developed to allow a simple overview of data density that maps well onto quadrangles (e.g., summary of record counts per quadrangle). Within such a context, the matrix is labeled by location and the table cells contain a singular count of data records.

The main complexity comes in navigating between the data set and its partitions. While there is a property called `partition` on the `DataSet` class that contains the subject URI of a partition, it is impractical to merely enumerate all the partitions. Depending on kind or size of data set, there could be a very large or countably infinite number of partitions.

Instead, partitions are discovered by following links within their Web representations. As links are traversed and resources are processed, new data set partitions are discovered. An application can then collect these links into their own annotation graph for later processing.

The relationship between these properties, classes, and actual Web resources is shown in *Figure 5.23, Pantabular Resource Structure*. A

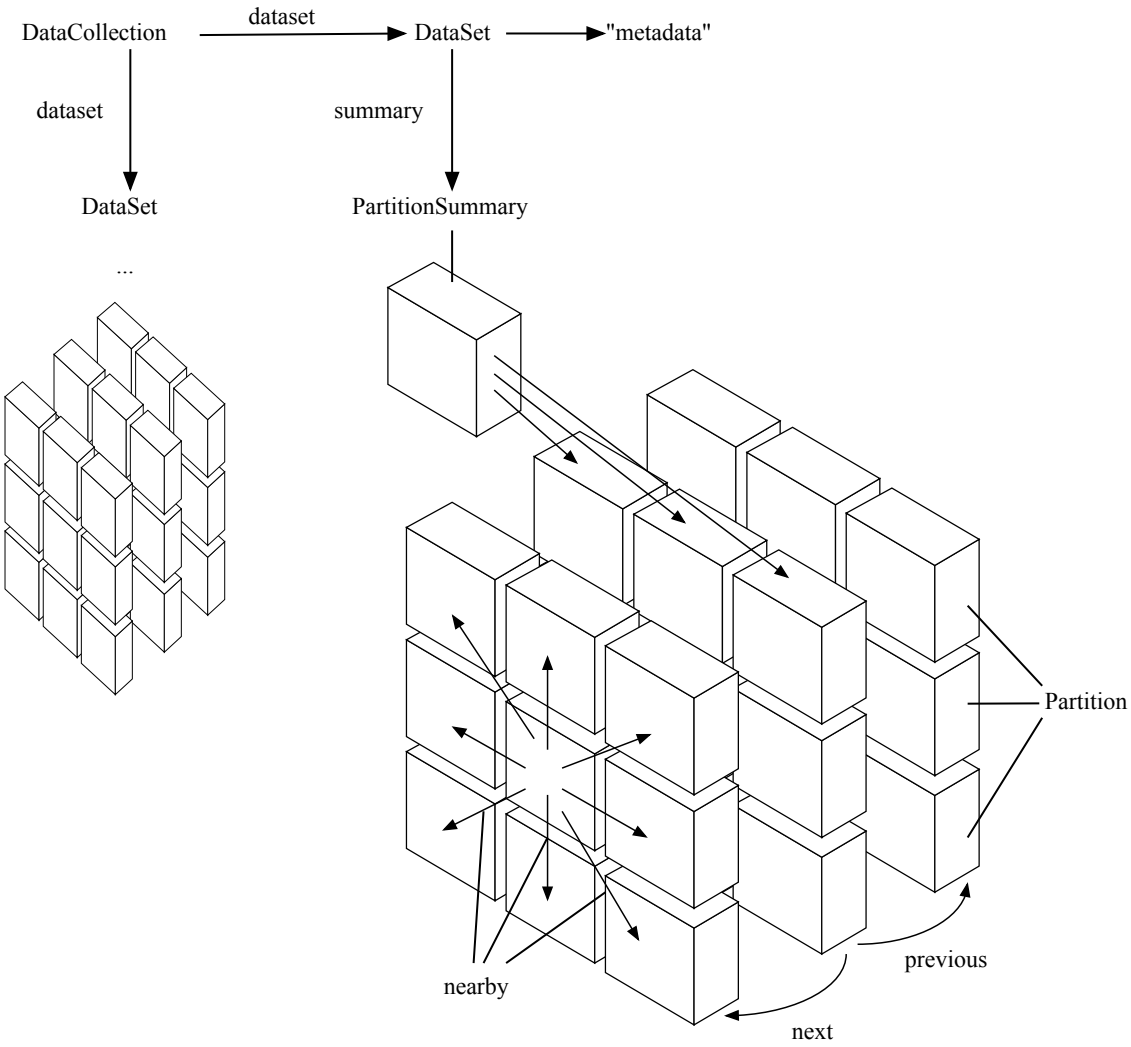
RDF Data Cube

The RDF Data Cube Vocabulary [75] is a late development in relation to this research. The vocabulary takes the approach of annotating every observation (i.e., every table cell in every table row). While the approach used for describing observations could be applied to describe the columns of tabular data, the organization of observations is different. Whether this difference is significant is unclear at this time.

Specifically, Data Cube uses slices, which would partition a data set into possibly large subsets based on fixed ranges, rather than multiple facet ranges defining many partitions of a data set. It is unlikely that these approaches are compatible. Even so, partitions as described in this methodology are not incompatible overall with Data Cube as a replacement for slices and could be added to it in the future.

typical discovery process starts by examining the `summary` property of a data set. This property is expected to be found on the `DataSet` instance when it is processed.

Figure 5.23 Pantabular Resource Structure



The result of examining the object value of the `summary` property and processing the related resource is an instance of `PartitionSummary`, which is a subclass of `Partition`. The typical item contained within this partition is an instance of `LabeledTable` that provides a set of entries that summarize and link to specific partitions.

For example, a partition summary of a set of weather reports might contain a set of counts of weather reports within the last hour with links to partitions defined by specific quadrangles. This easily fits within the structure of a matrix whose entries are the counts, and the indices are latitude or longitude ranges. This information can be annotated with the `LabeledTable` class and related properties.

Once an application navigates from the summary to a particular partition, the partition can contain certain properties such as `nearby`, `next`, or `previous`. These properties define related partitions by some adjacent basic facet range. For example, "next" might be the following time period (future) while "nearby" might be an adjacent quadrangle in the same time period.

By properly providing a starting summary and links within each partition, a whole data set can be enumerated by just examining the annotation graph and the links contained within it. The basic facet dimensions along which the link extends should be available within the current partition to allow applications to make decisions about whether a new partition resource should be retrieved. By doing so, an application can satisfy a particular query by determining the exact set of partition data resources that will satisfy a particular basic facet range.

How an application discovers these links or understands how metadata is encoded into URIs was considered but is not necessarily critical for the use of data within applications. Yet, having some mechanism for describing the link structure would tend to make using applications less brittle. An approach to this beyond what is described here is discussed in *Appendix B, Resource Schemes and Discovery*.

5.3.3 Summaries

Partition summaries are typically encoded with labeled tables. The columns and rows are labeled with annotations that allow a particular table cell to be located by a particular pair of basic facet ranges. For example, the cell entry can correspond to a geospatial quadrangle and so the column and row indices relate to longitude and latitude ranges, respectively.

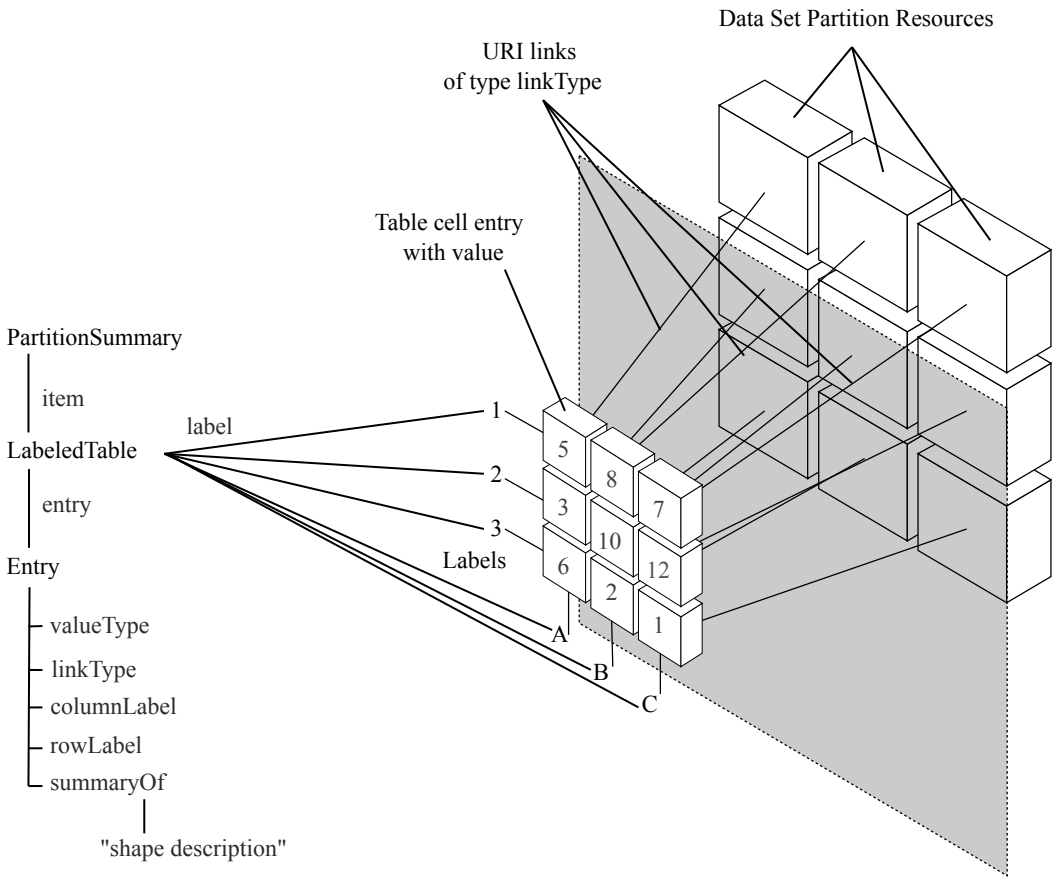
The structure of a `PartitionSummary` instance and how it maps to markup and structure is shown in *Figure 5.24, Partition Summaries via Labeled Tables*. The partition summary often contains a single item of type `LabeledTable` which describes the summarization.

The `LabeledTable` class has two major properties of `entry` and `label`. The singleton `entry` property describes the table cell entries as follows:

- *valueType* — the datatype of the table cell entry (e.g., an integer).
- *columnLabel* — the type of the column label value (e.g., longitude).
- *rowLabel* — the type of the row label value (e.g., latitude).
- *summaryOf* — a description of the object being summarized by the entry (e.g., a quadrangle). In RDFa terms, this property is often a typed blank node that has its own properties (e.g., typed as http://schema.org/GeoShape to describe the quadrangle's box).
- *linkType* — the expected type of the subject referenced in a link within each table cell.

Each column or row label is a `label` property of the `LabeledTable` instance. A `value` property provides the label value. Indices must be typed as `ColumnLabel` or `RowLabel` to allow applications to easily distinguish between rows and columns. Each of these types is a subclass of `Label`.

Figure 5.24 Partition Summaries via Labeled Tables



Finally, the table cell contains a single value and link to a data set partition resource. The value is a singular value such as "number of records" that is useful for assessing whether the partition should be retrieved. The link provides the actual Web resource that contains the data set partition.

A rendered example of this summary table was shown in *Figure 5.7, Quadrangle Summary Page*, with the previous and next time period links. The column and row indices are shown to the user, allowing them to navigate to the appropriate entry. Each entry links to a data set partition for the same time period; its URI is shown as an annotation in the figure.

The previous entry markup example (*Figure 5.10, Labeled Table Markup*) only showed a few of the table entry descriptions; *Figure 5.25, Summary Entry Example*, shows additional information such as a textual description of the

entry value and the `summaryOf` property for describing the quadrangle. Also, the target type of the partition link contained in each cell is also given.

Figure 5.25 Summary Entry Example

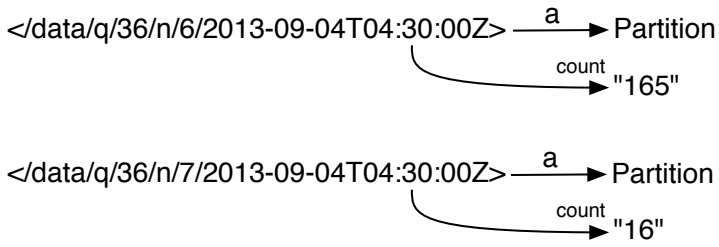
```
<table property="item" typeof="LabeledTable">
<caption>
<span property="entry" typeof="Entry">
 <span property="description">Weather Report Count</span>
 <span property="valueType" resource="xsd:int"/>
 by
 <span property="summaryOf" typeof="schema:GeoShape">
    <span property="schema:description">Quadrangle</span>
    of size
    <span property="schema:box" content="0 0 36 0 36 36 0 36">36° by 36°</span>
 </span>
 <span property="linkType" resource="Partition"/>
 <span property="columnLabel"
       resource="http://www.w3.org/2003/01/geo/wgs84_pos#lat"/>
 <span property="rowLabel"
       resource="http://www.w3.org/2003/01/geo/wgs84_pos#long"/>
</span>
</caption>
...
</table>
```

Just as described in *Section 5.3.1, Partition Links*, the data set partition link in the table cells can be further qualified with additional properties. In *Figure 5.26, Summary Link Markup*, a `count` property is associated with each partition link subject. This allows a receiving application to get information directly from the annotation graph shown in *Figure 5.27, Summary Link Annotations* with the resulting triples listed in *Appendix D: Figure D.7, Summary Link Triples*.

Figure 5.26 Summary Link Markup

```
<table property="item" typeof="LabeledTable">
...
<tr>
<th property="label" typeof="RowLabel"><span property="value">90</span>°</th>
<td><a href="n/6/2013-09-04T04:30:00Z" typeof="Partition">
    <span property="count">165</span></a></td>
<td><a href="n/7/2013-09-04T04:30:00Z" typeof="Partition">
    <span property="count">16</span></a></td>
...
</tr>
...
</table>
```

Figure 5.27 Summary Link Annotations



5.3.4 Tabular Data

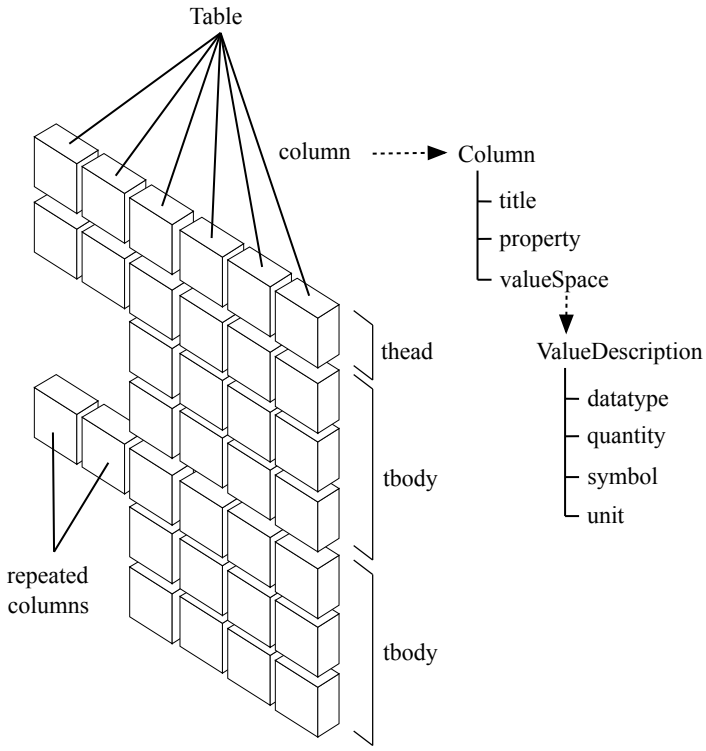
Tabular data contained within data set partitions can be annotated with the `Table` class properties. Rather than encode each cell of data as specific properties, the column headers are annotated with descriptions that apply to whole columns of data. The result is that a receiving application can process the table cells as simple values or use the column definitions to construct particular properties from the cell's value.

As shown in *Figure 5.28, Table Annotations*, a `Table` instance has a set of `column` properties associated with each column header that are each of type `Column`. This class has three properties: `title`, `property`, and `valueSpace` that are used to describe the column. Of these properties, the `title` property is just a human-readable label.

The other two properties, `property` and `valueSpace`, are intended for application processing and give more specific meaning to the column's values. The property named `property` contains a URI of an application-specific property specific to the domain. This value is reserved for scientific domain users to specify a domain specific label (e.g., "air temperature" or "star magnitude") and takes a similar role to the UCD+ annotation in the VOTable model.

The property named `valueSpace` contains a `ValueDescription` instance that is typically a blank node that describes the datatype, quantity, and unit of

Figure 5.28 Table Annotations



measure encoded in the table column. The `datatype` property describes the lexical encoding of the value that is expected for each table cell of data within the column. The remaining properties describe the measured quantity.

The description of the value space has `quantity` and `unit` properties that typically use QUDT vocabulary terms while the `datatype` property uses XML Schema simple types. In the case of values such as latitude and longitude, the commonly recognized Geo vocabulary from the W3C can be used [76].

To improve interoperability, the `quantity` and `unit` properties are URIs that qualify the specific observation (e.g., thermodynamic temperature) and the unit of measurement (e.g., Celsius vs Fahrenheit). While the values can be any URI, values from the QUDT vocabulary will provide maximum interoperability at this time.

Finally, for display purposes, `title` and `symbol` properties are provided. These provide the ability for systems to read the data and display information. In the case where the URI labels used for the `quantity` or `unit` properties are not recognized, the symbol and labels provide a fallback mechanism in a receiving application. While vocabularies such as QUDT provide symbols for each unit of measurement, an application may wish to display unit symbols without necessarily loading the vocabulary. Also, the `symbol` property provides a way to supply alternate symbols (e.g., °C versus `degC`).

Figure 5.29 Table Columns

```
<table property="item" typeof="Table">>
<thead><tr>
  <th property="column" typeof="Column">
    <span property="title">Station</span>
    <span property="property" resource="w:id"/>
    <span property="valueSpace" typeof="ValueDescription">
      <span property="datatype" resource="xsd:string"/>
      <span property="quantity" resource="dc:identifier"/>
    </span></th>
  <th property="column" typeof="Column">
    <span property="title">Latitude</span>
    <span property="property" resource="w:lat"/>
    <span property="valueSpace" typeof="ValueDescription">
      <span property="datatype" resource="xsd:double"/>
      <span property="quantity"
            resource="http://www.w3.org/2003/01/geo/wgs84_pos#lat"/>
    </span></th>
  ... longitude, receivedAt ...
  <th property="column" typeof="Column">
    <span property="title">Temperature</span>
    <span property="property" resource="w:airTemperature"/>
    <span property="valueSpace" typeof="ValueDescription">
      (°<span property="symbol">C</span>)
      <span property="datatype" resource="xsd:double"/>
      <span property="quantity"
            resource="quantity:ThermodynamicTemperature"/>
      <span property="unit" resource="unit:DegreeCelsius"/>
     </span></th>
    ...
   </tr></thead>
<tbody>...</tbody>
</table>
```

When the table representation is processed, a row consists of a set of cells with simple values. These values can be parsed into type instances by using the datatype specified in the corresponding type headers. Similarly, the property described by the column header can be associated with this value. This allows more complex conversions into triples within an annotation graph as desired.

Since some data sets repeat cell values within columns, certain table representations can be sectioned into a set of table bodies. Within these table bodies, missing cells can repeat table values. This allows simple compaction of data sets with repeated values (e.g., identifiers or locations).

All these annotations can easily be represented by an HTML table with RDFa annotations as show in both *Figure 5.29, Table Columns*, and *Figure 5.30, Table Bodies*. A partial listing of table columns is listed in the example, where the first two reference known URIs for terms for quantities. The last example defines air temperature using QUDT quantities and units.

In the table body examples, the first row is typed as `StaticColumns` and enumerates the fixed data showing the repeated identifier and location. In subsequent rows, this information is not repeated and the cells remain empty. The intent is that the repeated data will be assumed for the scope of the table body.

Note how the identifier table column (the first column) also has additional RDFa annotations. Further annotations of table cells may provide additional information. In this case, a link is provided to a specific weather station's data by identifier and time period. The table cell will still be interpreted as a simple string value using the text content of the cell (i.e., the concatenation of the text descendants).

5.4 Summary

In some respects, the PAN Methodology may seem simple: we provide reasonably named, Web-size portions of data, in regular HTML tables, with RDFa annotations to enrich the content. Yet, in this simplicity we can recognize the principles that have made the Web successful. Providing simple

Figure 5.30 Table Bodies

```
<table property="item" typeof="Table">
<thead>...</thead>
<tbody>
  <tr typeof="StaticColumns">
    <td><a rel="related"
           href="/data/station/CW6499/2013-09-04T04:30:00Z"
           typeof="Partition">
        <span property="identifier">CW6499</span></a></td>
    <td>34.47583</td>
    <td>-120.2045</td>
  </tr>
  <tr>
    <td/><td/><td/>
    <td>2013-09-04T04:58:35Z</td>
    <td>28.9</td>
    <td>33</td>
    ...
 </tr>
 <tr>
    <td/><td/><td/>
    <td>2013-09-04T04:53:41Z</td>
    <td>28.9</td>
    <td>33</td>
    ...
 </tr>
 </tbody>
 <tbody>...</tbody>
 ...
</table>
```

ways to access data enables a basic level of operation on the Web that is not found in other methodologies currently used to get scientific data onto the Web.

The PAN methodology allows small and large data sets to be exposed via a rational mechanism using the Ordinary Web. It represents a middle ground between unstructured and highly-structured, single purpose data. As a vast majority of scientific data is not accessible directly within the OWP due to representation problems, enabling data access means that operating within this middle ground can have profound impact on many areas of science.

In the early 21st Century, mechanisms for publishing scientific data are largely out of reach for both citizen and professional scientists due to cost, scale, and complexity. We can bridge that gap by re-purposing existing technologies so that we can employ existing tools in novel ways. It is in this space where the PAN methodology fits well as it obviates the need for

additional tools to understand anything more than existing HTML markup structures and current mechanisms for adding behaviors.

PAN treats data sets as interactive resources with which you can navigate, explore, and process. The use of facet-based partitions enables Web-sized navigation through everyday facets understood by the consumer (e.g., date/time, location, etc.). At the same time, doing so requires nothing more than a browser, even though more sophisticated processing is certainly possible. At the core of the methodology is the partitioning, annotation, and naming mechanisms of the methodology.

We have arrived at the proposition that we can best realize the full potential of the OWP by embracing its technologies and philosophies; we hope to demonstrate how the PAN methodology exposes scientific data and enables computation with common Web browser technology in *Chapter 6, An Exemplary Implementation for Evaluation*. Moreover, we compare tools for scientific workflows in *Chapter 7, Comparisons with Alternatives* and show how the PAN Methodology is compatible within data services for such tools and that the OWP platform excels at workflow tasks.

Chapter 6

An Exemplary Implementation for Evaluation

13. *PAN enables simple methods for open access to open data.*

14. *PAN enhances the ability to compute over data within the OWP; it is possible, practical, and useful.*

15. *Annotation provides a useful middle-ground between casual browsing and use by programmatic consumers for computation.*

16. *PAN scales-down and enables smaller data sets to be published with simple methods and so helps enable citizen science on the Web.*

17. *PAN scales-up by enabling harvesting of information with unique source attribution.*

We have claimed that the OWP and RDFa offer a path to achieving the four necessary qualities of identifiability, extensibility, flexibility, and durability

for scientific data on the Web. Following that path, the PAN Methodology provides a roadmap for how data can be published on the Web. We now need to test the methodology against a real data set and evaluate its use in a number of different scenarios.

Experiments to evaluate the PAN methodology against these claims require a sufficiently large scientific data set whose interpretation and use will not be controversial. While there are many existing data sets, and some have their own Web portals or services, finding data that is not Web-oriented and does not have an existing acceptable use presents a bit of a conundrum; if it is not on the Web, it is difficult to find. Fortunately, there exist automated sensor networks, such as weather station networks, that produce data that can be accessed via the Internet but are not well represented on the Web.

6.1 mesonet.info: Weather Data via PAN

We chose to conduct experiments with a data set of weather observations from the *Citizen Weather Observation Program* (CWOP) [77], a loosely associated network of automated weather stations hosted by citizens, local governments, and businesses throughout the world. These weather stations provide their data through a peer-to-peer network that communicates over the APRS-IS protocol [78]. This line-oriented protocol can be received from servers that aggregate the feeds of weather and position reports from all the various weather stations; an example is shown in *Figure 6.1, Example APRS Feed*.

Figure 6.1 Example APRS Feed

```
DW3904>APRS,TCPXX*,qAX,CWOP:@090158z5132.18N/00043.53W_061/000g001t030r000p000P000h87b10389L000.DsVP
CW1604>APRS,TCPXX*,qAX,CWOP:@090158z4444.70N/06531.17W_204/004g009t027r000p000P000h80b10204.DsVP
DW6741>APRS,TCPXX*,qAX,CWOP:@090158z3749.55N/08000.08W_296/005g...t036r...p...P008h74b10188.DsVP
DW6916>APRS,TCPXX*,qAX,CWOP:@090158z4310.23N/10818.40W_238/001g002t027r000p000P000h58b10189.DsVP
DW6011>APRS,TCPXX*,qAX,CWOP:@090158z4307.07N/08756.60W_261/002g006t028r000p000P000h55b10249.DsVP
```

The APRS feed contains weather reports encoded according to the US National Weather Service NWS APRS standard [79]; they may also contain date, time, and location information. Complicating this however, some weather stations report their positions independently as separate reports. Also, mixed within this feed is other data output by various systems or sensors connected to the CWOP network of servers.

To make the format easier to handle, we developed a program (server daemon) to receive the APRS feed and transform it into the XML syntax shown in *Figure 6.2, Example APRS XML*. The data in the XML attributes represents the choices of the APRS specification and, in some cases, non-SI units are used for measurements. Because we consider the time reported by weather stations to be unreliable, we add a time-stamp to reflect the time at which the data was received.

The program has been deployed on a server and receives more than 78,000 weather reports per hour from more than 14,000 weather stations. Data is received in a continuous feed, chunked by the program into 5 minute segments, and stored as a sequence of XML documents. The result is a time-series data set that is represented by a sequence of XML documents, each of which contains about 6500 weather report records.

Figure 6.2 Example APRS XML

```
<aprs xmlns="..." source="..." start="2013-03-...">
<report from="EW0938" type="weather"
        latitude="39.55917" longitude="-84.1155"
        received="2013-03-29T05:25:00Z" at="2013-03-29T05:39:00Z"
        wind-dir="0" wind-speed="0" wind-gust="0" temperature="30"
        rain-hour="0" rain-24hours="0" rain-midnight="0" humidity="86"
        pressure="10244" />
<report from="IW4EMA-3" type="weather"
        latitude="44.86133" longitude="11.6465"
        received="2013-03-29T05:25:00Z" at="2013-03-29T05:38:00Z"
...
```

The CWOP APRS feed generates more than 13GB of XML data per month, containing an average of over 100 million weather reports from throughout

the world (although, unevenly distributed and highly concentrated in North America and Europe). As a scientific data set, it has three distinct qualities: geospatial orientation, measurement of observed phenomena, and time-series data (observations over time). For the weather measurements, there are a variety of different properties that are necessary to define a "record" in detail. For example, "temperature" needs to convey "air temperature, over land, measured at a particular elevation." As such, the data set provides more than sufficient examples of the complexity of exchanging data with precisely defined semantics.

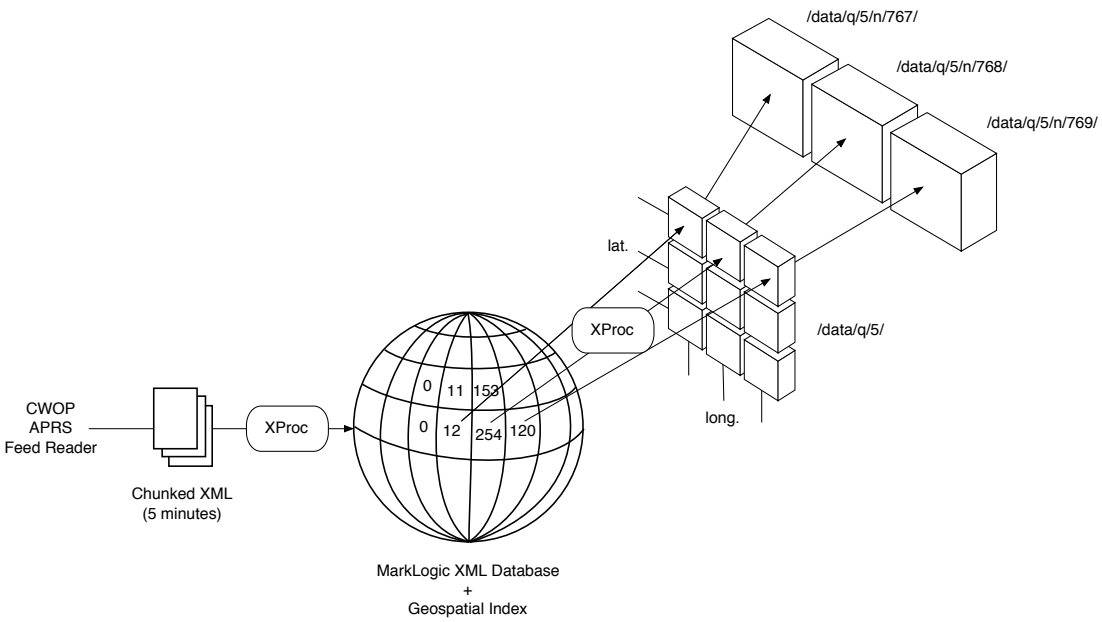
We built and deployed a multi-tiered system at `mesonet.info` that both stores the data set and publishes it via the PAN Methodology. The architecture of this system is shown in *Figure 6.3, mesonet.info Architecture*, where on the left side of the diagram the APRS feed is received as chunked XML and stored into a MarkLogic XML Database [80] via an XProc pipeline [81]. The data is then indexed in various ways with special attention being paid to geospatial indexing.

The data is then summarized and accessible via the PAN Methodology using quadrangles of certain sizes. As the data set contains time-series geospatial data, it is partitioned by time segments and by quadrangles. Internally, this partitioning occurs at the same time, using various indices, within the XQuery [82] executed by the MarkLogic server.

For the sake of experimentation, any size quadrangle or time duration is allowed in the design of the `mesonet.info` system. There are certain rational constraints on the quadrangle sizes (i.e., that it be an even divisor of 360°) and time durations default to 30 minutes. Certain choices may result in poor performance (e.g. too long of a duration) or inconsistent quadrangles (e.g. uneven divisors) and the system does not reject such choices. A production system would likely limit the choices to a small fixed set of quadrangle sizes

and time durations based on both user feedback and ability to support the resulting response sizes.

Figure 6.3 mesonet.info Architecture



A typical interaction with the data starts by retrieving a representation of the summary at the URI path `/data/q/{size}/{period}` where the variable *size* is the quadrangle size in degrees and *period* is the starting point of the time partition in XML Schema's `dateTime` format. As a convenience, the current time period is accessible without the time period being specified in the URI. For example, `/data/q/5/` provides a summary of 5° quadrangles for the current time period by redirecting the request to the appropriate URI with the proper time period start time appended.

Once a choice of quadrangle size (e.g., 5°) is made, the summary (e.g., `/data/q/5/`) contains counts of the number of weather reports received within the quadrangle in the given time period (partition) as shown in *Figure 6.4, Weather*

Reports Quadrangle Summary, where the markup behind a single row is shown in *Figure 6.5, Summary Row Markup*. Each table cell contains an integer value (the count) and a link to the particular quadrangle's data. De-referencing that link returns another document that contains the weather reports for the quadrangle for the given time period.

Figure 6.4 Weather Reports Quadrangle Summary

Received from 2013-11-20T22:30:00Z to 2013-11-20T23:00:00Z (PT30M) totaling 18282

Previous time period PT30M @ 2013-11-20T22:00:00Z Next time period PT30M @ 2013-11-20T23:00:00Z

5° Long. / 5° Lat.	-180°	-175°	-170°	-165°	-160°	-155°	-150°	-145°	-140°	-135°	-130°	-125°	-120°	-115°	-110°	-105°	-100°	-95°	-90°	-85°	-80°	-75°	-70°	-65°
90°	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
85°	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80°	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75°	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
70°	0	0	0	0	0	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
65°	0	0	3	1	0	1	73	1	3	0	0	0	0	2	0	0	0	0	0	0	0	0	2	0
60°	0	0	0	0	0	1	0	0	0	5	0	2	5	1	0	0	0	0	0	0	0	0	0	0
55°	0	0	1	0	0	0	0	0	0	0	4	8	6	35	6	5	4	1	0	0	0	0	1	0
50°	0	0	0	0	0	0	0	0	0	0	1	774	186	76	36	51	39	116	47	18	18	61	48	30
45°	0	0	0	0	0	0	0	0	0	0	0	222	63	197	171	119	133	279	619	589	747	1381	75	15
40°	0	0	0	0	0	0	0	0	0	0	0	599	223	65	258	241	183	320	423	626	801	35	0	0
35°	0	0	0	0	0	0	0	0	0	0	0	10	685	352	104	38	639	206	334	421	55	1	0	0
30°	0	0	0	0	0	0	0	0	0	0	0	0	0	7	1	18	244	34	5	502	1	0	0	0
25°	0	0	0	0	24	0	0	0	0	0	0	0	0	0	11	6	7	0	5	15	0	2	0	0
20°	0	0	0	0	1	5	0	0	0	0	0	0	0	0	0	5	2	0	3	4	6	1	25	9

Figure 6.5 Summary Row Markup

```
<tr vocab="http://pantabular.org/">
<th property="label" typeof="RowLabel"><span property="value">40</span>°</th>
<td><a href="n/757/2013-10-29T21:00:00Z" typeof="Partition">
    <span property="count">0</span></a></td>
<td><a href="n/758/2013-10-29T21:00:00Z" typeof="Partition">
    <span property="count">0</span></a></td>
...
<td><a href="n/768/2013-10-29T21:00:00Z" typeof="Partition">
    <span property="count">217</span></a></td>
<td><a href="n/769/2013-10-29T21:00:00Z" typeof="Partition">
    <span property="count">80</span></a></td>
<td><a href="n/770/2013-10-29T21:00:00Z" typeof="Partition">
    <span property="count">22</span></a></td>
...
</tr>
```

The annotation graph generated from the embedded RDFa describes each link to each different quadrangle page as a separate data set partition typed as `Partition` (as described in *Section 5.3, Annotating Data Sets*) which are

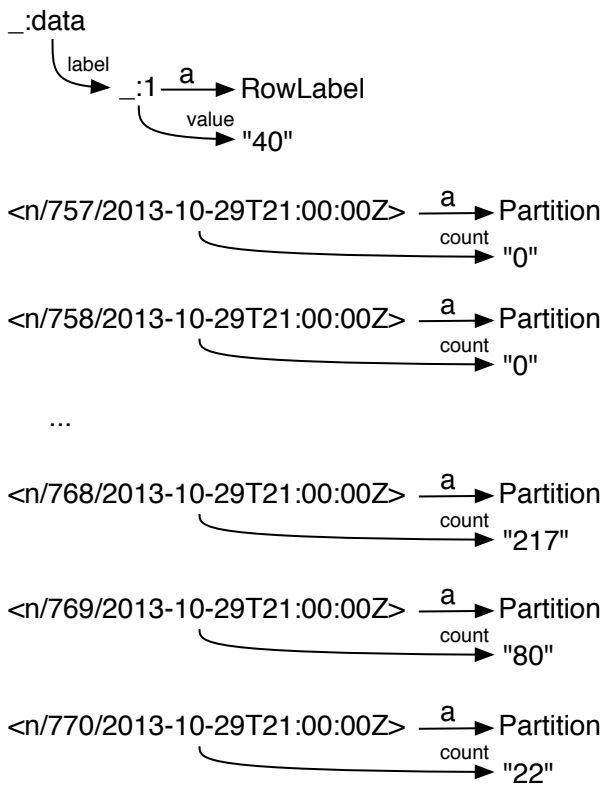
shown in *Figure 6.6, Summary Row Annotations* with the triples listed in *Appendix D: Figure D.8, Summary Row Triples from mesonet.info*. In relation to each partition subject, there is only one property for the total count. The position associated with the quadrangle summarized by the table cell can be computed from the column and row indices.

In the example, only the count is a property of the partition subject, however other properties could be embedded. If additional properties are needed, a consuming application needs to do extra work. For example, if an application needs the sequence number, it has four choices: it could have been provided as an additional property (not available in this example); it can be calculated from the row and column indices; it can be extracted from knowledge of the metadata in the URI of the partition; it can be retrieved by processing the partition resource identified in the link. If the application does not have the knowledge locally, either directly from annotations or via extracted metadata, the cost of retrieving must be incurred by de-referencing the resource link.

From the summary or independently, each quadrangle data partition is accessible at a URI path of `/data/q/{size}/n/{seq}/{period}` where the variable *seq* is the *sequence number* of the quadrangle and the other variables are as before. If an application knows the particular quadrangle of interest, this URI can be constructed directly and the summary document can be bypassed. As was the case with the summary, as a convenience, the current time period is accessible without the time period being specified in the URI and a redirect will inform the application of the full URI.

The quadrangle data partition is more complicated because it contains and describes each column of weather report data. First, the whole set of weather reports for the quadrangle data partition is encoded as a single table. This table is subsequently broken into a number of table bodies, one for each

Figure 6.6 Summary Row Annotations



weather station. These table bodies hold the individual weather reports of the particular weather station for that time period, in chronological order.

The header of the table describes the measurements available within each weather report row and is shown in *Figure 6.7, Quadrangle Table Header*, where each column is described using the PAN ontology (see *Section 5.3.4, Tabular Data*). Each column has a property named `property` with values in the `mesonet.info` namespace and all the value space properties for quantities employ QUDT vocabulary with the exception of the weather station identifier and location. Each `datatype` property uses an XML Schema type label.

Within the table are a number of table bodies, one for each weather station, as shown in *Figure 6.8, Quadrangle Data*. The first table row defines the repeated data of the identifier, latitude, and longitude for each subsequent row. Each of these rows contain the remaining data without these elements repeated.

Figure 6.7 Quadrangle Table Header

```
<thead vocab="http://pantabular.org/">
<tr>
<th property="column" typeof="Column"><span property="title">Station</span>
  <span property="property" resource="w:id"/>
  <span property="valueSpace" typeof="ValueDescription">
    <span property="datatype" resource="xsd:string"/>
    <span property="quantity" resource="dc:identifier"/>
  </span>
</th>
<th property="column" typeof="Column"><span property="title">Latitude</span>
  <span property="property" resource="w:lat"/>
  <span property="valueSpace" typeof="ValueDescription">
    <span property="datatype" resource="xsd:double"/>
    <span property="quantity"
              resource="http://www.w3.org/2003/01/geo/wgs84_pos#lat"/>
  </span>
</th>
...
<th property="column" typeof="Column"><span property="title">Temperature</span>
  <span property="property" resource="w:airTemperature"/>
  <span property="valueSpace" typeof="ValueDescription">
    (°<span property="symbol">C</span>)
    <span property="datatype" resource="xsd:double"/>
    <span property="quantity" resource="quantity:ThermodynamicTemperature"/>
    <span property="unit" resource="unit:DegreeCelsius"/>
  </span>
</th>
...
</tr>
</thead>
```

Figure 6.8 Quadrangle Data

```
<tbody vocab="http://pantabular.org/">
<tr typeof="StaticColumns"><!-- First row with repeated data. -->
<td><a rel="related" href="/data/station/AA6AV-10/2013-10-30T04:00:00Z"
        typeof="Partition">
       <span property="identifier">AA6AV-10</span>
    </a>
</td>
<td>38.32917</td>
<td>-122.319</td>
</tr>
<tr><!-- Second row with repeated data omitted. -->
<td/><td/><td/>
<td>2013-10-30T04:20:10Z</td>
<td>238</td>
<td>0</td>
<td/>
<td>49</td>
<td>9.4</td>
<td>74</td>
<td>101690</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td/>
<td/>
</tr>
...
</tbody>
```

6.2 Evaluation Strategy

Having explored the design of the `mesonet.info` system, we continue by considering the qualities it has attained as a result of applying the PAN Methodology and how those qualities support scientific endeavors on the Web. By examining the qualities `mesonet.info` has attained, we will evaluate the PAN Methodology as a mechanism for disseminating scientific data. That is, we will describe how certain qualities achieve desirable outcomes for scientific data on the Web; validating our hypothesis (see *Section 1.4, The Hypothesis*) and demonstrating our outcomes (see *Section 1.1, Specific Outcomes*).

Specifically, in the sections that follow, we walk through various aspects of the `mesonet.info` system as provided within the PAN Methodology and demonstrate how we intend the various system properties to be used. The demonstration system provides building blocks upon which libraries of functionality are directly built. Subsequently, the ability to overlay visualization, computation, or translation of the data set demonstrates various desirable qualities for scientific endeavors for professional or citizen scientists.

Each of these evaluations is described in the following sections and is categorized as follows:

- *Section 6.3, Data Access Methods and Algorithms* — The necessary algorithms for using PAN-enabled data within the OWP.
- *Section 6.4, Open Access to Open Data* — By re-using OWP technologies, both old and new, data that should be openly accessible is enabled to be so.

- *Section 6.5, Data Navigation via APIs* — Information within resources, encoded as markup with annotations, can be easily navigated via APIs.

- *Section 6.6, Scaling Down for Simple Methods* — The mechanisms of the PAN methodology allows scaling-down to simple acts of publishing, receiving, and using data.

- *Section 6.7, Summaries and Navigation* — Summaries and links provide the ability to navigate data sets for both human and machine consumers.

- *Section 6.8, Computing on the OWP* — Computing over and within the OWP is possible, practical, and useful.

- *Section 6.9, Scaling Up via Compatibility* — Data represented via the PAN methodology is compatible with existing systems using Semantic Web technologies and useful in a larger scope.

These sections serve as one set of evaluations of the PAN Methodology in the context of the `mesonet.info` and CWOP data set whereas another evaluation is the comparison in *Chapter 7, Comparisons with Alternatives*. The intent of the following sections is to explore aspects of accessing and using data within the OWP via services implementing the PAN Methodology. As noted in the follow sections, implementation details can be found in the appendices.

6.3 Data Access Methods and Algorithms

The most common task performed with weather data is accessing reports for a specific region and time period. Whether we are looking up reports on yesterday's snow fall or considering seasonal precipitation and flood statistics, it almost always pertains to a given place over a certain period of time. When such a task uses the CWOP data as provided by `mesonet.info` via the PAN Methodology, the application starts with a geospatial region and a time period and then must conceptually do the following:

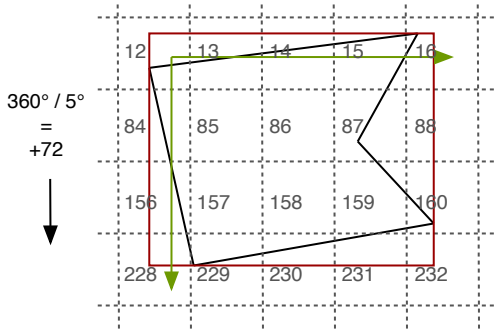
1. Choose a reference quadrangle size.

2. Calculate the number of quadrangles needed for the task's geospatial region.

3. Calculate the number of date and time partitions needed for the task's total time period.

While determining the number of time period partitions necessary is a straight-forward calculation, it might seem that computing the necessary quadrangles could be complex. Fortunately, all that is required is an enumeration of sequence numbers. The computation must first compute a bounding box and then compute the sequence number for each corner of the box; the corners give the extreme values for the sequence numbers. As sequence numbers simply enumerate quadrangles over the whole reference ellipsoid (i.e., the earth's surface), the quadrangles that cover the bounding box can just be listed by counting from a given reference quadrangle in the upper left and then the number of quadrangles that tile the circumference (which is constant) can be used to move to the next row. An example of this process is shown in *Figure 6.9, Bounding Box Algorithm for Sequence Numbers*, where the region is outlined in black and the bounding box is shown in red.

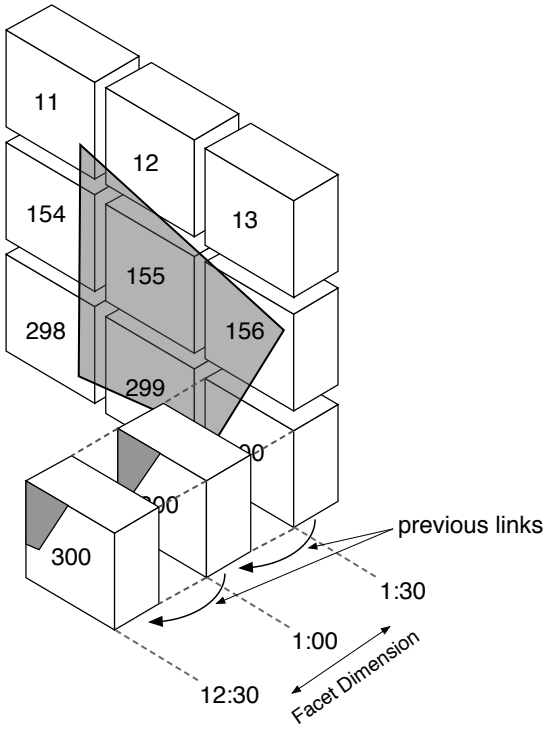
Figure 6.9 Bounding Box Algorithm for Sequence Numbers



Given the naming choices implemented for `mesonet.info`, requesting the data for a polygon reduces to generating a sequence of URIs from the sequence numbers found using the *bounding box algorithm*. Each URI is instantiated by substituting for the variables into the template `/data/q/{size}/n/{seq}/{period}`, where the quadrangle size is generally constant and decided upon in advance to control the resource size or response time. From this URI sequence, the data can be retrieved, either in sequence or in parallel, depending on the capabilities of the OWP platform implementation.

We now have an efficient process for retrieving data over both a geospatial region and time period via the *Backtracking Algorithm* (see *Figure 6.10, Backtracking Algorithm for a Region*). Rather than computing the exact number of time partitions and quadrangles necessary as a complete cross product, an application can backtrack from the most recent time partition that contains the end of the requested time period. Requests are made for just the quadrangle data partitions for the end of that specific time period for each quadrangle identified by the bounding box algorithm. From the annotation graph extracted from within each retrieved quadrangle data partition, the previous link is selected and, if the time period for that newly discovered partition is within the desired range, the link is queued for retrieval. This process continues until all the necessary partitions are visited.

Figure 6.10 Backtracking Algorithm for a Region



This Backtracking Algorithm can be stated as follows, where *G* is the region and t_s t_e

1. The set of quadrangles sequence numbers *Q* is determined by the *Bounding Box Algorithm*.

2. For each quadrangle sequence number in *Q*, the data partition for the time period partition containing t_e is requested.

3. For each subsequent response, the representation is processed as needed and then the follow process is applied:

    a. The `previous` link is located in the annotation graph.

    b. From the `range` property value, locate the `length` and `start` properties.

c. The values from (b) determine the previous partition's time period end date and time. Request the link if the end of the time period of the partition is after t_s.

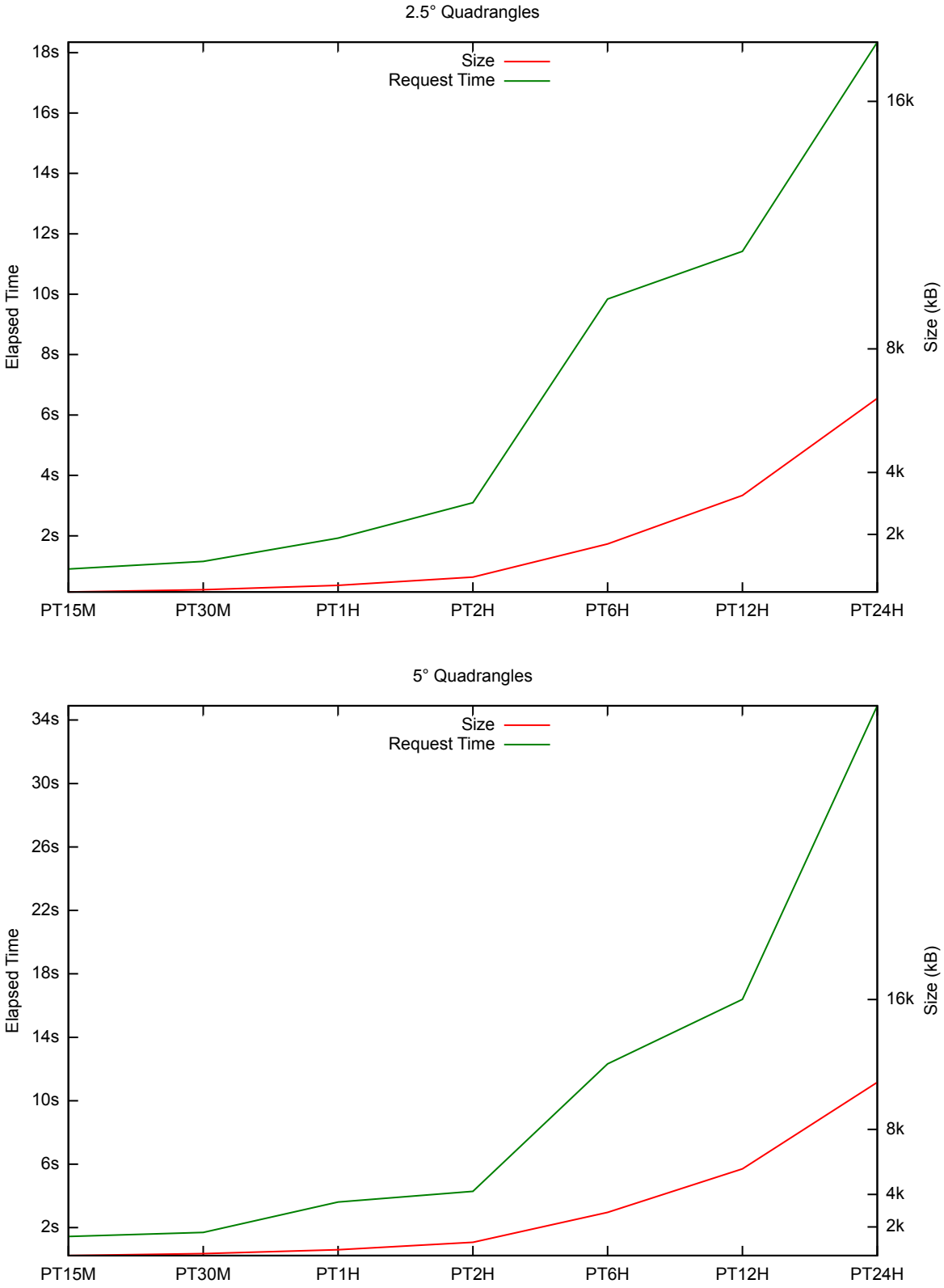
It should be noted that the use of quadrangles over-retrieves data for irregular polygons much the same way that an IVOA cone search query may require overlapping cones to retrieve a wide region of space. A receiving application must test locations on the boundary of the polygons to see whether particular items are within the desired region. Depending on the sophistication of the application and complexity of the polygon, membership may or may not be simple to test.

The choice of quadrangle size and time period are intertwined with both the size of the response and the amount of time to retrieve the response. In *Figure 6.11, Time Duration vs Response Time / Size*, the elapsed response time and size of the response were tested on the `mesonet.info` infrastructure for various time period partition sizes and for quadrangle sizes of 2.5° and 5°. The elapsed response time remains relatively low for time partitions up to 30 minutes, increases slightly for partitions of 1-2 hours, and then radically increases. As the time period partition grows past a practical limit, the cost of retrieving the content, transporting it to the browser, and then processing grows by many factors. The data demonstrates that the typical Web application design pattern of many smaller requests have better elapsed times in comparison to a few large requests.

6.4 Open Access to Open Data

The implementation of PAN methodology on `mesonet.info` provides the ability for any consumer to directly access and interact with addressable

Figure 6.11 Time Duration vs Response Time / Size



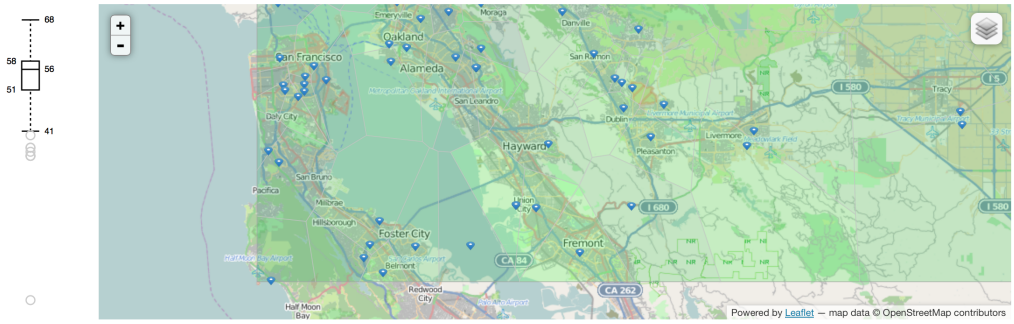
partitions of scientific data with a regular Web browser. This satisfies an overarching goal of open access to scientific data, in contrast with many tabular/spatial/temporal data formats that require special tools to view, search and analyze the data. The Web browser opens access to data that previously required specialized tools that a user might not have access to for a variety of reasons.

For PAN-enabled data sets, the Web browser can play a variety of roles. The browser can receive and render PAN encoded data tables for display to a user through typical means without added effort by the publisher. This enables all the accessibility cues within the browser and allows a variety of consumers and devices to access partition data.

Moreover, if the PAN-enabled data partition is viewed within the browser instead of merely being received as data via some request, all the usual scripting and visualization facilities of the Web browser are available for use. This means a sufficiently-enabled browser client can present a visualization rather than a large set of table columns. On `mesonet.info`, this is demonstrated on the particular quadrangle data partition pages where a default visualization is automatically made available to a consumer as shown in *Figure 6.12, Visualization of Partitions on mesonet.info*. The visualization is built client-side with the data provided within the resource and then a mapping service (via D3 [83], Leaflet [84], and Open Street Maps [85]) is used to show temperature data on a map while a box plot with some basic statistics for temperature is shown on the far left.

The example, as shown in *Figure 6.12, Visualization of Partitions on mesonet.info*, demonstrates an inherent duality exposed by the PAN methodology. In one case, a user agent can load a resource from the Web and treat it strictly as a data format that contains markup whose semantics provide little more than access to the structures they represent. When used in this context, the

Figure 6.12 Visualization of Partitions on `mesonet.info`



731 Weather Reports from Received from 2014-03-04T00:30:00Z to 2014-03-04T01:00:00Z (PT30M) within Quadrangle #2976 [40 -122.5 37.5 -122.5 37.5 -120 40 -120]

quadrangle data partitions are simply HTML data with additional RDFa annotation semantics.

When loaded as a viewable resource (e.g., in a browser tab or iframe), the document executes with additional layered semantics. In this context, the same document will be processed, the same HTML data and RDFa annotations semantics will be applied, but additional scripting and embedded media (e.g., images) processing will also occur. At minimum, this means tabular data is rendered as a table. Yet, in *Figure 6.12, Visualization of Partitions on mesonet.info*, scripts automatically execute within the OWP as they would for any other typical Web page and so also provides an embedded visualization. The result is that a data provider can publish Web resources with embedded default semantics, have those semantics execute locally within the Web browser, and provides the casual consumer with an enhanced user experience. Furthermore, the same script with little or no modification can be used to display any PAN-annotated geo-located data by taking advantage of the column annotations.

Meanwhile, when that same resource is loaded by data access calls with the OWP (e.g., XMLHttpRequest invocations), the scripts will not execute nor will any embedded media be accessed. The data provider has not changed how

the resource was produced to affect this outcome. Instead, the browser enacts a slightly different processing model where standard APIs are applied so that the same interfaces to tabular data are available to the application but the browser will not go any further. For the receiving application, if the primary use case is data dissemination, the cost to consume the dynamic visualizations embedded by the provider is minimal as they are never invoked.

6.5 Data Navigation via APIs

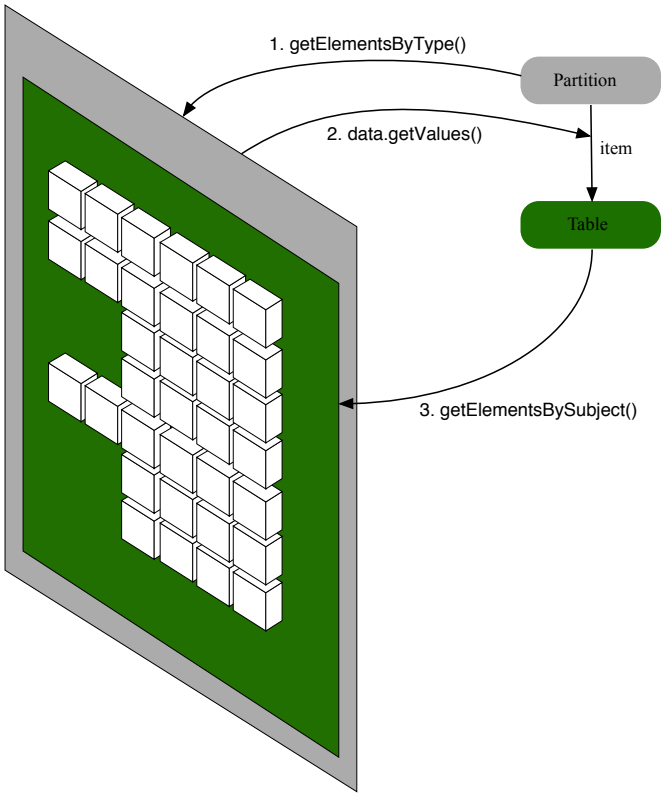
As we noted in *Section 5.3.4, Tabular Data*, use of the PAN vocabulary via RDFa does not annotate each table cell value. As such, the annotation graph only contains explicit information about the columns. Information about individual data cells is only available by inference, via the column (and row) annotations. This results in a requirement that the receiving application process the entire document, including the annotations, to access all the data.

Fortunately, there are APIs to help navigate both HTML and RDFa. First, the DOM has defined interfaces to HTML elements [86] and defines specific interfaces to HTML tables. Every table row and cell can easily be enumerated via a matrix-like API (`rows` and `cells` properties). The advantage is that any row or column spans, etc. are computed by the browser before the API is presented to the consuming script.

More importantly, there is a document-oriented API for RDFa [87] that was published as a W3C Note and defined by the same W3C Working Group that published RDFa 1.1. This API provides the ability to access the RDFa annotations directly from the document and to navigate back and forth between the document and specific properties. The graph can be accessed and processed directly or in parts as constructs are found within the document.

We implemented a conforming RDFa 1.1 processor and the RDFa API, along with some extensions, in Green Turtle [88] as an RDFa processor for browsers. While browsers may provide native RDFa implementations within the OWP in the future, existing Websites can enable RDFa by simply including the Green Turtle script. Other processing environments may have similar capabilities and also enable RDFa via Green Turtle or other implementations.

Figure 6.13 Accessing a Table



A typical application interaction starts with accessing a quadrangle data partition's data using the API shown in *Figure 6.13, Accessing a Table*; the corresponding script is presented in *Figure 6.14, Accessing a Table via the API*. While more complex partitions may contain many items, the `mesonet.info` implementation has only one; so finding the table of data within the Web page is straightforward. In the example, the script navigates the document via the graph, back into the annotation graph to find a subject, and then uses the subject to navigate back within the document as follows:

1. Find the *container element* via type annotations.

2. Use the element returned to get a *list of subject URIs* of the items.

3. Use the appropriate subject URI to find the *table element* (e.g., there is only one in the example).

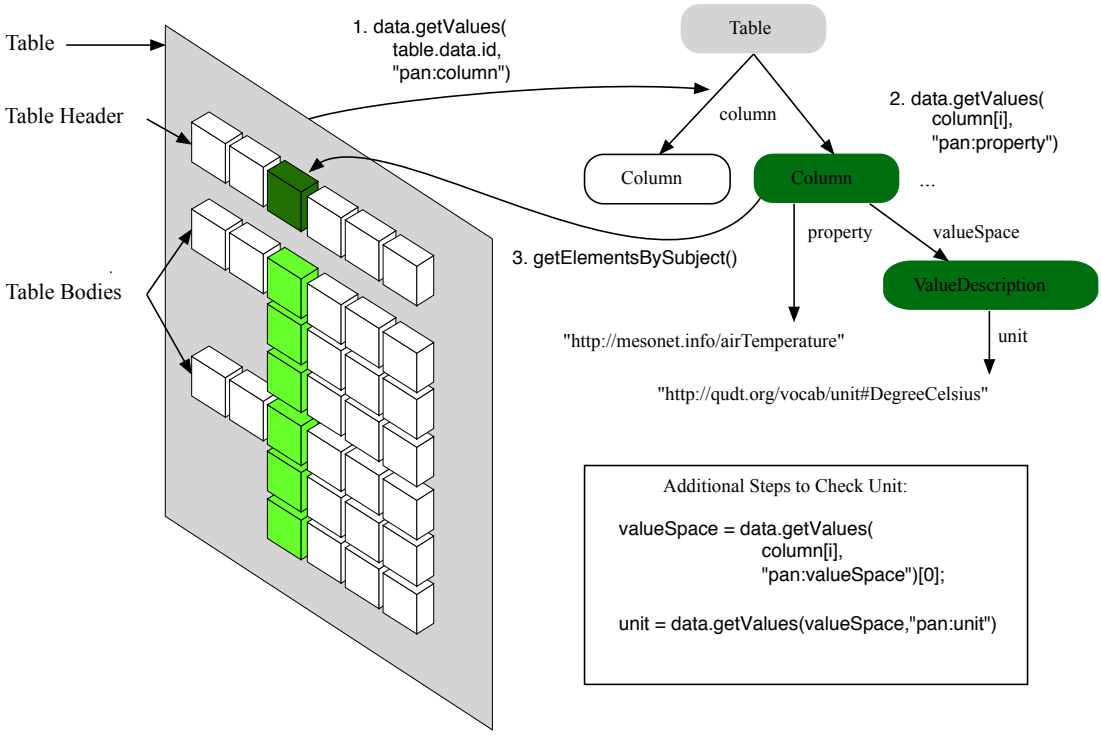
Figure 6.14 Accessing a Table via the API

```
// (1) Find the element that holds the partition
// Note: from the graph to the DOM via the RDFa API
var datasets = document.getElementsByType("pan:Partition");

// (2) Use the subject to find the partition's item subjects
// Note: from the DOM to the graph via the RDFa API
var items = document.data.getValues(datasets[0].data.id,"pan:item");

// (3) Access the first item (a table)
// Note: from the graph to the DOM via the RDFa API
var table = document.getElementsBySubject(items[0])[0];
```

Once the tabular data is located, an application has a direct reference to the table element containing the data and can enumerate the table rows and cells. To understand which columns contain the desired data, the script must first locate the columns by their annotations. The process requires traversing using the table's subject URI into the annotation graph to find the column definition annotations. From these annotations, the particular column can be located by examining further properties and then the column heading element can be located by subject URI within the document.

For example, locating the first air temperature column is shown in *Figure 6.15, Finding a Column by Property*. The assumption is that the table has already been located and we just need to examine the `property` predicate to find a column whose object value is air temperature (`http://mesonet.info/airTemperature`). If a match is found, the subject URI used to retrieve this property value can also be used to locate an element. If further qualification is necessary, the value space of the property can be examined to inspect properties such as unit of measurement (e.g., Celsius versus Fahrenheit).

Figure 6.15 Finding a Column by Property



The result is that an application has a wide range of abilities to locate and consume data within the table. From the column definition annotations, the datatype and other properties of the value space can be extracted and dynamic interpretation of the textual value can be applied. On the other end, once a column is recognized by label, the table cell values can be used directly as textual content without conversion into a specific datatype. This flexibility provides the ability to scale up or down in terms of its complexity for the use of table cell values.

6.6 Scaling Down for Simple Methods

There are two dimensions upon which to evaluate the PAN methodology in terms of its ability to scale down:

1. The ability to work with small Web-sized subsets of a large data set via commonly available methods.

2. The ability to publish smaller data sets via existing methods (e.g., blogs posts, social networks, etc.).

6.6.1 Local Knowledge and *In Situ* Services

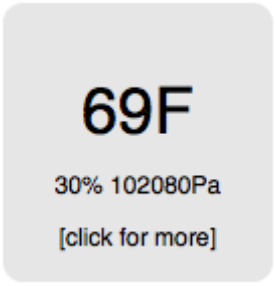
While the `mesonet.info` infrastructure provides access to a reasonably large data set (13+ GB per month), the use of the data scales down due to the quadrangle partitioning. A client application can fine tune the quadrangle size to zoom in on a specific location; as the size of the quadrangle diminishes, so too does the volume of data received and processed by the browser and the application. As such, if the application requires specific data (e.g., temperature closest to a location), the application can tune the quadrangle requests to the minimal number and size to receive the required information.

For example, a typical feature of informational Web sites is to enable other Web sites to embed their data. Using Cross-Origin Resource Sharing (CORS) [89], data services such as those provided by `mesonet.info` can be directly accessed by other Websites without causing security exceptions within the browser. The data resources just need to declare that they are allowed to be accessed by other sites by adding a few headers to the HTTP response when the data partitions or other resources are retrieved.

Using CORS allows the browser to serve as the point of aggregation. Instead of an intermediary service accessing the data and providing renderings of the data, scripts can access the data directly and build the UI within the browser on the receiving Web page. As a result, the data resources (e.g., a quadrangle's weather reports) are directly re-purposed for use as data without any need to change the underlying service.

A simple example of this is shown in *Figure 6.16, Weather Badge*, where the weather for a specific location is shown. This badge is created by declaring the location within regular HTML markup via RDFa annotations as shown on the right in *Figure 6.16, Weather Badge*, using the same technique described in [90] where local services use local knowledge to enhance the document. The badge is identified by type via RDFa annotations and all the related data service information is expressed within the anchor markup. A local service script only has to inspect the annotation graph, find elements of certain badge types, and then access the data quadrangle.

Figure 6.16 Weather Badge




```
<div vocab="http://www.mesonet.info/" typeof="badge/summary">
  <a property="source" href="http://www.mesonet.info/data/q/"
      typeof="quadrangle">
     <span property="quadrangle-size">2.5</span>° Quadrangles
     <span property="template"
           content="{base}{quadrangle-size}/n/{sequence}/"/>
  </a>
  for location
  <span property="latitude">37.74292</span>,
  <span property="longitude">-122.4632</span>
</div>
```

In the case of this particular weather badge, the location information encoded in the badge declaration markup is used to compute the sequence number (see *Appendix A, Sequence Numbers*) that will contain the weather data. The sequence number is then used in the URI template (i.e., in the `template` property value in the example) to construct a URI to the data set partition that contains the weather reports for the desired locations. The rest of the process requires computing the report nearest to the desired location and generating the display. The typical expectation by the Web page author is that the invocation (markup) and badge shown in *Figure 6.16, Weather Badge*, occur in the same location in the Web page and only the badge is displayed to the user.

6.6.2 Publishing Small Data Sets

While the previous section dealt with enabling simple consumers of scientific data, there is also the issue of enabling simple methods to publish small data sets not of the same scale as CWOP or other large-scale endeavors. Ecology data is an example where individual data sets are very small (i.e., observations of species and environmental conditions over a constrained geospatial region). As such, they fit well into modern Web publishing services, such as blogs, where individuals employ easy-to-use authoring tools to publish information, typically in the form of articles or journal entries. Certainly, enabling them to publish their data sets within the same tools at the same time is very desirable.

Current systems require uploading data files as archives, and other non-Web oriented formats as attachments to the entry. This confounds the casual browsing of data and discourages indexing by search engines. Instead, the information can be directly published via the PAN methodology using the following strategy:

1. If the geospatial region is sufficiently large, a `LabeledTable` typed HTML table is authored that enumerates just the quadrangles that cover the bounding box of the region. This index must contain the counts of observations within each quadrangle.

2. For each quadrangle, a partition of the data is encoded in a typed HTML table element (i.e., `typeof="Table"`).

3. All the tables from (1) and (2) are composed in or "pasted" into the authoring tool provided by the blogging system.

Since RDFa annotations are encoded in markup attributes, the blogging system will not reject the extra markup. There is no requirement for additional

scripts and, as such, the content is deemed benign and allowed through. The result is the blog entry page is now also a data resource that can be processed directly by other systems.

In this case, we rely on proper naming within the blogging system to give a useful URI for the data set. Within the data set, the links between the partitions are all local to the same document. A receiving system should always take care to handle such local references as they may confuse systems that assume that each URI is a distinct resource. In this case, they are likely to be locations within the same resource and, while it doesn't matter to the annotation graph, it does to the processing application in terms of understanding when it needs to fetch a new resource.

6.7 Summaries and Navigation

Navigation aids are needed for time-series data sets, such as CWOP weather data, that result in a large and open-ended number of partitions. The PAN Methodology provides two mechanisms to satisfy those needs:

1. A resource akin to a statistical marginal provides the ability to summarize the amount of data within regions such as quadrangles.

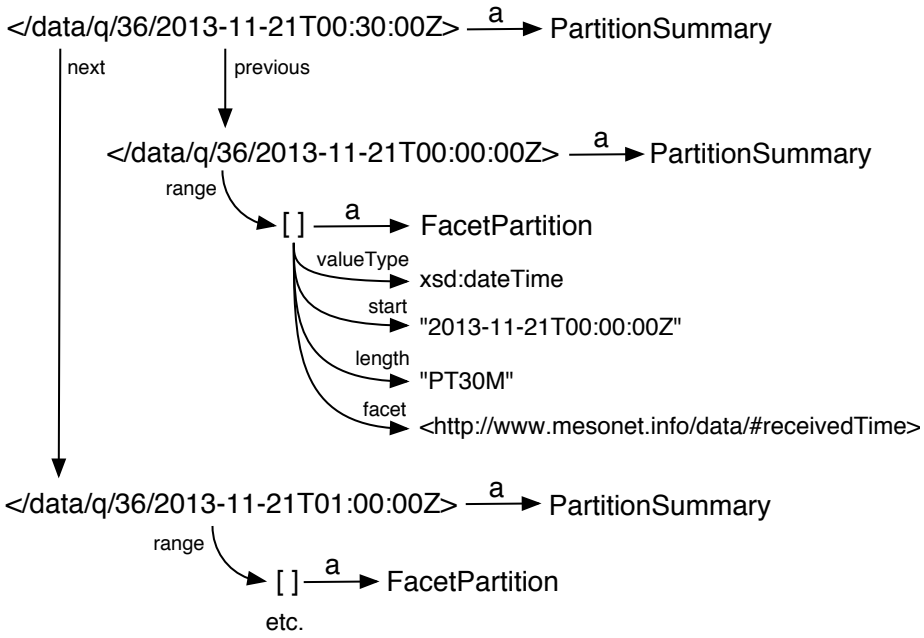
2. Links provide the ability for data partition resources to express relations amongst partitions such as next, previous, and nearby.

For the `mesonet.info` implementation, summary resources are provided as shown in *Figure 6.4, Weather Reports Quadrangle Summary*, as a table of counts per quadrangle and time period. There are various links within this document, of which the simplest are the links typed with the relationship `previous` and `next`. A system could assume that these are different time

partitions and just follow the links but they can also inspect the annotations to understand which basic facet they are navigating.

In *Figure 6.17, Previous & Next Link Annotations*, the `previous` and `next` relations are shown in relation to the partition subject and the full triples are listed in *Appendix D: Figure D.9, Previous & Next Link Annotation Triples*. Each of these properties have an object value that is the subject URI of the related partition summary with an associated type. These related partition summaries have additional annotations that describe the range of their facet partition. By comparing those range values to the current, an application can understand the facet being navigated via the relations. In this case, the value type is `xsd:dateTime` and the semantic label is `http://www.mesonet.info/data/#receivedTime` and so the relationship navigates the summaries in terms of the received time of the weather reports.

Figure 6.17 Previous & Next Link Annotations



Within the `LabeledTable` instance itself are links to each particular data partition from the table cell that corresponds to the quadrangle. As the table

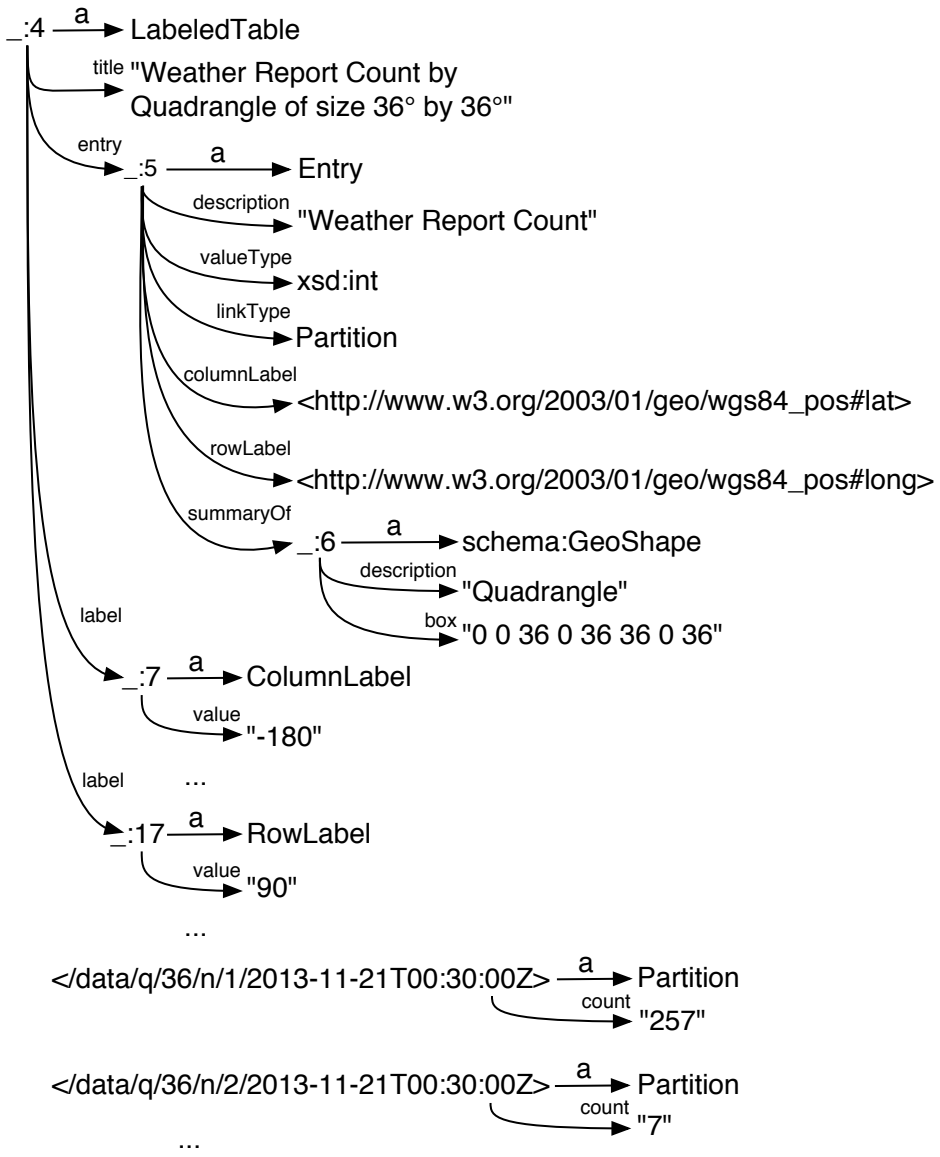
is only partially annotated, the resulting annotations (see *Figure 6.18, LabeledTable Annotations* with the triples listed in *Appendix D: Figure D.10, LabeledTable Annotation Triples*) demonstrate how the partition links are severed in the graph. Starting just after subject `_:17` are a list of partitions that are disconnected in the annotation graph but whose origin is a specific table cell. Each quadrangle data partition link is typed as a partition and has a count property. There is little to indicate to which table cell it belongs within the annotations but an RDFa processor can trace the origin within the document.

An application could provide additional inferencing over the table and add additional relationships to the graph. Fortunately, enhancements to the RDFa API provide a simple way to find this information from a table cell. The `linkType` property (see *Figure 6.18, LabeledTable Annotations*) for the entry provides a URI that can be used to find the link element. A simple look up via the API call `getFirstElementByType()` returns the link element and the application can use the subject URI to find additional properties such as the count of reports. In this case, count of reports is also available as the cell's textual contents.

While summaries are useful for navigating data sets, they can also be useful data resources in themselves. In *Figure 6.19, Quadrangle Summary Visualization*, a map displays summary data over North America and part of Europe using 2.5° quadrangles color coded by data intensity. Each quadrangle shows the data count and is a link that and leads to a visualization of the quadrangle itself.

The visualization is computed by navigating the `LabeledTable` entries and computing the bounds of the quadrangle for each entry via the indices. The count and the link are located within the annotation graph and this information is used to color and label the quadrangle regions within the map

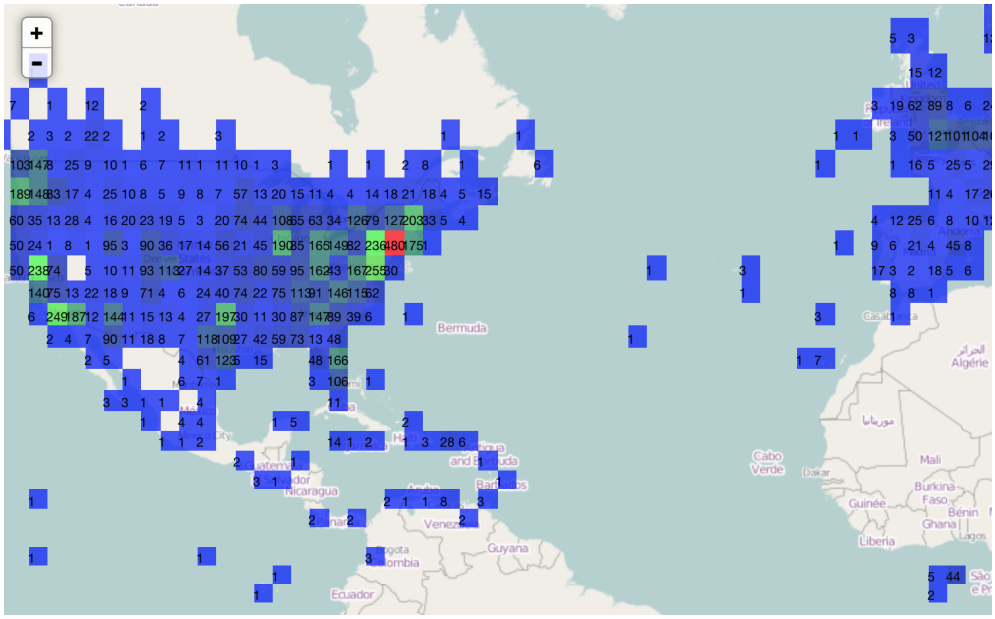
Figure 6.18 LabeledTable Annotations



with links to further information. The summary provides a quick way to access this data and produce a summarization for visual inspection.

Finally, when quadrangle data partitions are accessed, their representation contains links to other partitions. In addition to the `next` and `previous` relations, there is a set of `nearby` relations that is partially shown in *Figure 6.20, Nearby Link Annotations*, and the full triples in *Appendix D: Figure D.11, Nearby Link Annotation Triples*. These links provide access to adjacent

Figure 6.19 Quadrangle Summary Visualization

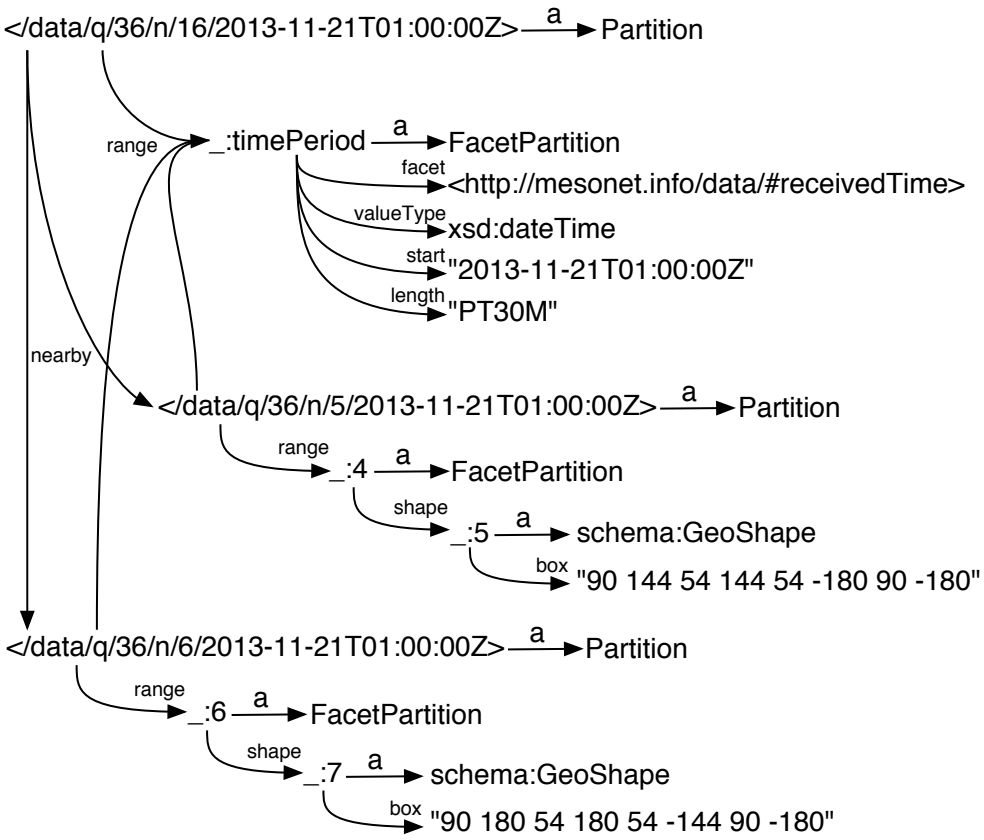


quadrangles whose properties are described in the annotation graph. Each link shares the same time period as the current partition but describes a different quadrangle (a different box via the `shape` property). An application can use this information to determine the geometry and other properties of the linked partition; a necessary precursor to deciding whether to retrieve that partition's data by following the link.

6.8 Computing on the OWP

The browser, as the most common interface to the OWP, has become a surprisingly solid platform for deploying applications with its myriad of advanced features and capabilities. The idea of directly using the OWP and computing against data is captivating. Data sets expressed using the PAN methodology can be accessed, traversed, and operated upon within the OWP using a combination of HTML and RDFa APIs. We are faced with the question

Figure 6.20 Nearby Link Annotations



of what can be accomplished with that ability before other methods become more tractable?

Many data processing tasks require access to large subsets of data and there are existing systems that provide the ability to map tasks over such subsets. We will examine how a Map / Reduce [91] process can be replicated in the browser and then what we can compute with that process. In that examination, we will specifically look at visualization of data set subsets and how the Web browser can be used as a computational tool in contrast to its typical use as a user interface that renders pre-computed images.

6.8.1 Map / Reduce

A Map / Reduce algorithm divides a larger problem into a number of sub-problems (the map step) and then collects the answers of all the sub-problems to produce a single answer (the reduce step). Given the structure inherent in regularly partitioned data sets, it is straightforward to define a Map / Reduce algorithm that operates on partitions. Within the PAN methodology, computing over geospatial regions is facilitated by using quadrangles and sequence numbers that are part of the partitioning of the data set. The process then involves mapping user functions over Web resources (data partitions) and then applying a reduction function to the collected results.

We start with the input of a geospatial region *G*, a time period t_s t_e , and a set of facet types *F*. The process produces an output *O* is as follows:

1. Let *R* .
2. Calculate the quadrangle sequence numbers *S* from the region *G*.
3. For each sequence number in *S*, generate a URI for the latest partition which contains t_e and add the URI to the queue *Q*.
4. While *Q* is not empty:
    a. Remove *U* from *Q*.
    b. Request a representation *W* of *U*.
    c. Locate and harvest the columns *C F* from *W* via the RDFa annotations.
    d. Convert *C F* into native data types defined by the column descriptions in RDFa into an array of data *D*.

e. Apply the user's map function: D M_D.

f. Add M_D to the result R.

g. Locate a link to the previous partition. If time period end for the partition is after t_s , add the URI to Q.

5. Apply the user's reduce function: R O.

We have implemented an example of a Map / Reduce process in about 410 lines of JavaScript [92] code (uncompressed); it relies upon an RDFa API in the browser-based application. The implementation is limited to rectangular regions to simplify generation of the starting quadrangle sequence numbers and testing of membership of returned data locations. The user need only provide the region, time period, columns they are interested in processing, and their Map / Reduce functions. A simple example that calculates the average temperature for a given region and time period is shown in *Appendix C: Figure C.4, Using Map / Reduce to Calculate Average Temperature*.

6.8.2 Barnes Interpolation

Barnes Interpolation [93] is the calculation of a set of data points (interpolated values) across a two-dimensional surface from a set of measurements; it was originally developed specifically for weather forecasting. It can be used to produce renderings of temperature, wind, barometric pressure, etc. that are commonly expected from discrete measurements from weather stations (e.g., a colored gradient). As the interpolation is typically applied to a geospatial region over time, this process is a good fit for the application of the Map / Reduce implementation described in the previous section.

The interpolation process starts with a weighted average and then calculates a refinement using all the grid points:

$$g_k(x,y) = \frac{\sum_i^n w_i o_i}{\sum_i^n w_i} \qquad g_k(x,y) \quad g_k(x,y) = \frac{\sum_i^n w_i \left(o_i - g_i(x,y)\right)}{\sum_i^n w_i}$$

$$g_k^n(x,y) \quad g_k^n(x,y) = \frac{\sum_i^n w_i \left(o_i - g_i^n(x,y)\right)}{\sum_i^n w_i}$$

where:

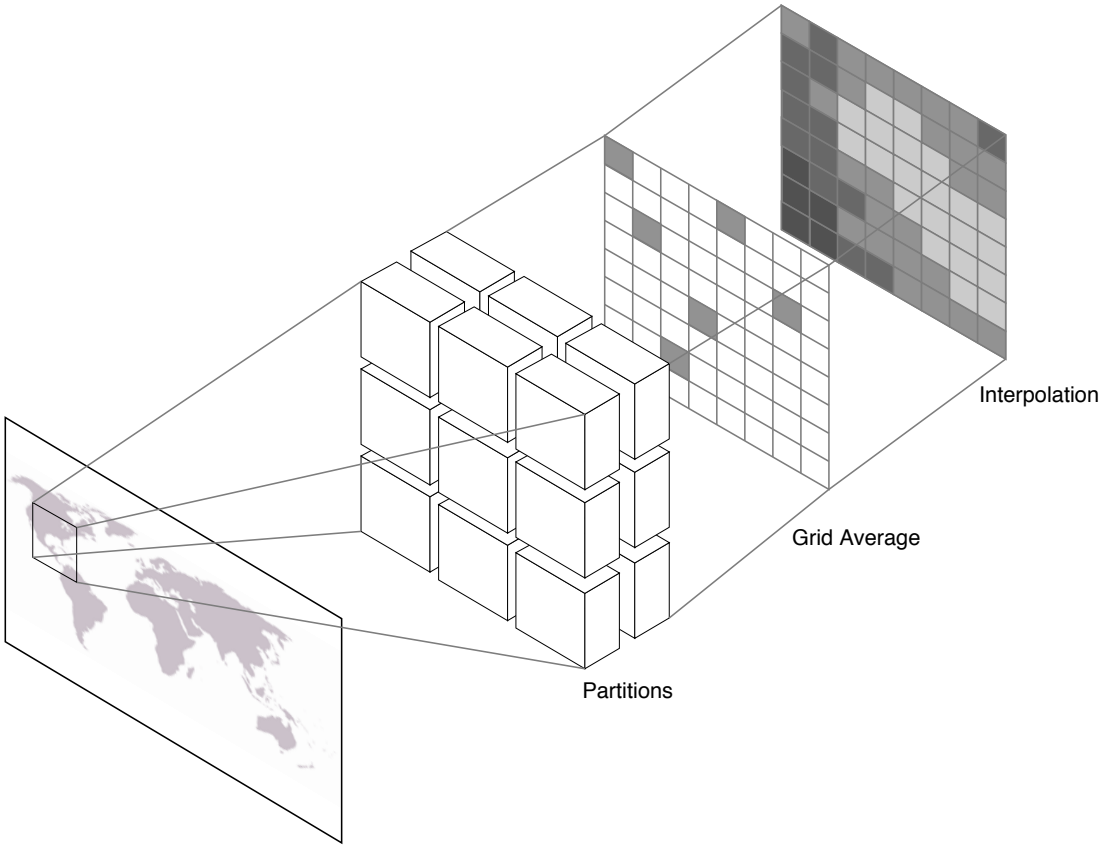
- o_i is the observed value,
- weight for each observed value is w_i $\frac{d_i}{L\,C}$,
- d_i is the distance (km) from the grid point to observation point,
- L is a scalar value relative to the distances between observed phenomena,
- C a convergence factor.

It should be noted that difference between o_i and $g_i^n(x,y)$ in the numerator measures error for the grid cell that contains an observed value. This difference causes the interpolated value for observed grid cells to converge to the observed value after sufficient iterations. Using this observation, an

algorithm can use this convergence as criteria to determine when to stop iterating.

For weather data, certain values are useful in this computation: *L* is set to 111.3, the number of kilometers in a degree at the equator, and *C* varies where 1 is used for the initial pass and 0.3 is used afterward [94]. Using these values, three iterations are sufficient to get reasonable convergence for air temperature. Other measured quantities may require more iterations.

Figure 6.21 Barnes Interpolation Process



This interpolation process can be divided into a number of stages as shown in *Figure 6.21, Barnes Interpolation Process*, where at the beginning (on the left) we have a particular geospatial region and at the end (on the right) we have a visualization of the interpolated grid. At the end of the process, each grid cell contains an interpolated value that can be mapped onto a color gradient.

These results can be used to display a typical rendering of temperature by a colored surface over a map display.

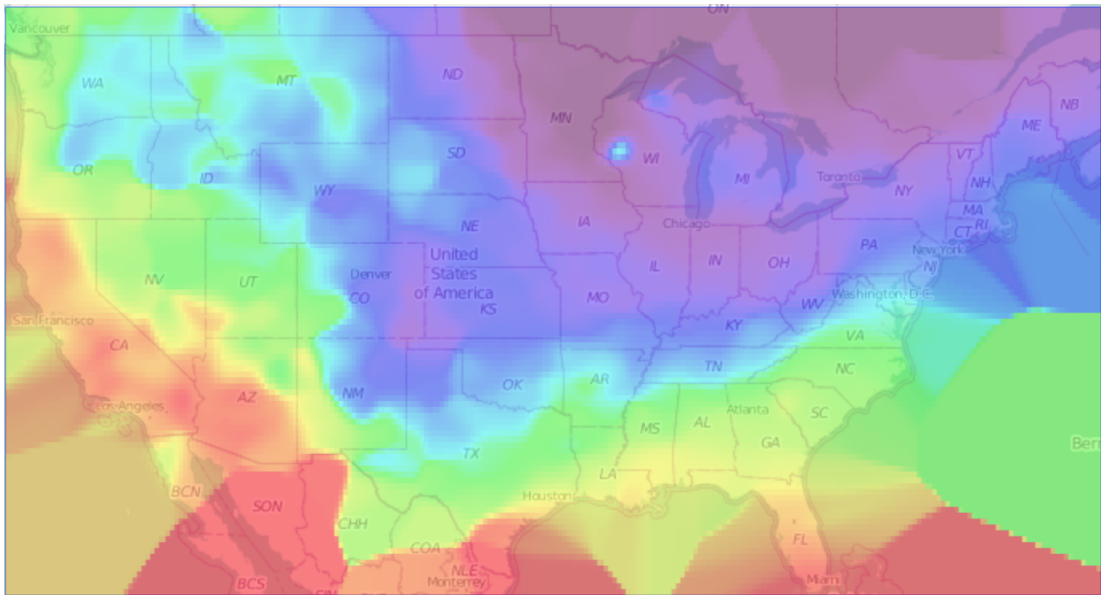
Given a choice of time period and quadrangle size, combined with the geospatial region, a particular set of data partition Web resources that cover the requested inputs can be identified. The Map / Reduce process can be used to process these data partitions into a sparse grid of observed values where each populated cell contains the mean value of the observed values that are located in that cell. Other statistical processes can be applied within this process to filter out bad data values if they are present.

The sparse grid result is then input to the Barnes interpolation process, whose implementation is described in *Appendix F, Barnes Interpolation Implementation*. The penultimate result is a grid of numerical values that correspond to rectangular regions within the input geospatial region. These regions and numerical values can then be color coded to produce an output image or for use in interpretations or visualizations. Thus, the OWP provides all the basics elements for computing and visualizing an interpolation surface for weather data.

As an experiment, an implementation of Barnes Interpolation was developed as a script and is called as a post-process step after the sparse grid is calculated via Map / Reduce. The invocation is shown in *Appendix C: Figure C.5, Example Barnes Interpolation Script*, and the sparse grid computation is shown in *Appendix C: Figure C.6, Example Grid Average Script*. The implementation takes as input a rectangular geospatial region, a time period, and a quadrangle size. The Map / Reduce process calculates and retrieves the necessary quadrangles to produce the interpolation surface after first calculating an average temperature for each station within the geospatial region.

An example visualization of the output of the process is shown in *Figure 6.22, Polar Vortex - 2014-01-23T20:00:00Z - PT30M*, for the region [50°, -125°,

Figure 6.22 Polar Vortex - 2014-01-23T20:00:00Z - PT30M

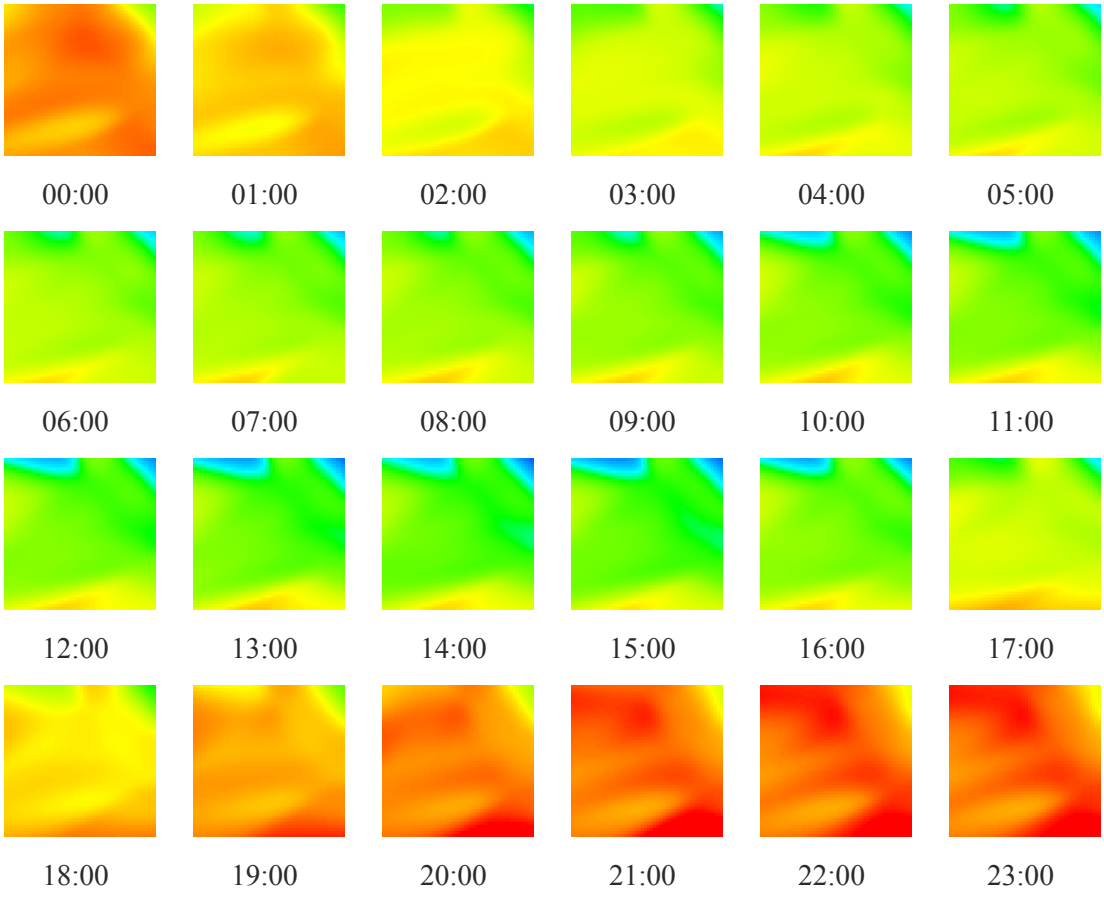


25°, -65°] for 30 minutes of data on an 0.25° resolution interpolation grid. To access the necessary data with 2.5° quadrangles and 30 minute durations, 240 data partitions were retrieved over the Web. On average, the process took 22 seconds to complete with 13.6 seconds needed for the data access on a modest network connection (6Mbps) with a reasonably powerful laptop (2.6Ghz Intel Core i7 MacBook Pro with 16G of memory).

Examining this implementation from a different perspective, an example of interpolating a sequence of images for 24 hour period is shown in *Figure 6.23, Interpolation Sequence*, for the region [40°, -125°, 35°, -120°] over 1 hour segments (twice the time period of the previous example) starting at 2014-01-23T00:00:00Z and which required 8 data partitions resources for 2.5° quadrangles for each frame. The hours of the day for each frame are in the UTC timezone and so, for the Pacific coast, the first frame starts in the late afternoon (warmer) and then cools throughout the night. Each frame took an average of 2.9 seconds to render with a standard deviation of 724ms of which the interpolation was only 52ms on average. Smaller regions render in reasonable enough times for the process to occur in almost real-time such that

animations of data can be generated in response to a user's query. The only real constraint is the transport times for the data.

Figure 6.23 Interpolation Sequence



6.9 Scaling Up via Compatibility

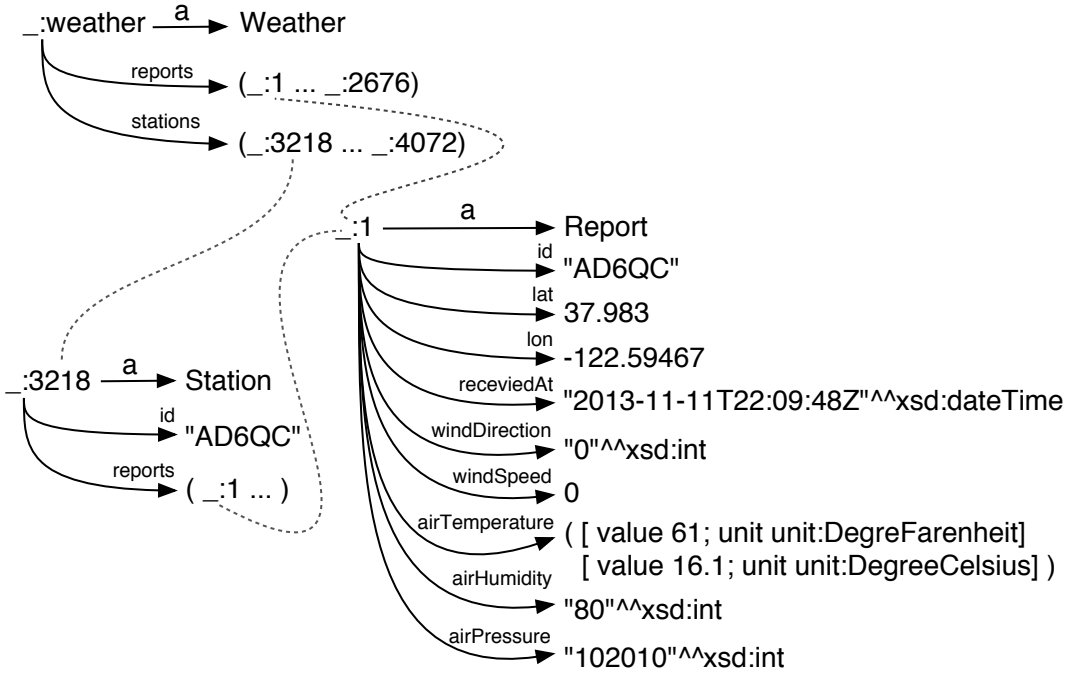
When a data set is published via the PAN methodology, there likely remains a requirement to inter-operate with systems that use other data formats. As the PAN methodology only partially annotates the data, a consuming system must navigate the tabular data and apply some kind of transformation. Given the orientation of PAN towards the Semantic Web, a consumer with an

existing triple database and inferencing system needs to harvest all the data by turning partitions into triples and storing the result for later processing.

PAN-annotated partitions offer a simple transformation of rows into subjects where each column becomes a property of the subject, because each column definition has a property defined as a URI. Ergo, the values in the table cells become the object value in the triple. An example of this simple transformation is diagrammed in *Figure 6.24, Weather Report Graph*, with an extract of the full triples listed in *Appendix D: Figure D.12, Weather Report Triples*.

As a nuance, the report happens to contain the same reported value using different units (Fahrenheit vs Celsius) which adds some complexity to the values. Instead of a single object value for `w:airTemperature`, the value is an object collection of two blank nodes. Each of these blank nodes contains the value and the unit used.

Figure 6.24 Weather Report Graph



Just as with the Barnes Interpolation, the ability to convert data into triples on `mesonet.info` was implemented via simple Map / Reduce process, where the invocation is shown in *Appendix C: Figure C.7, Harvester Implementation*. The process requires URI names for properties and types used; these are passed into the process at initialization where the implementation uses these names to label properties or for the value of subject types.

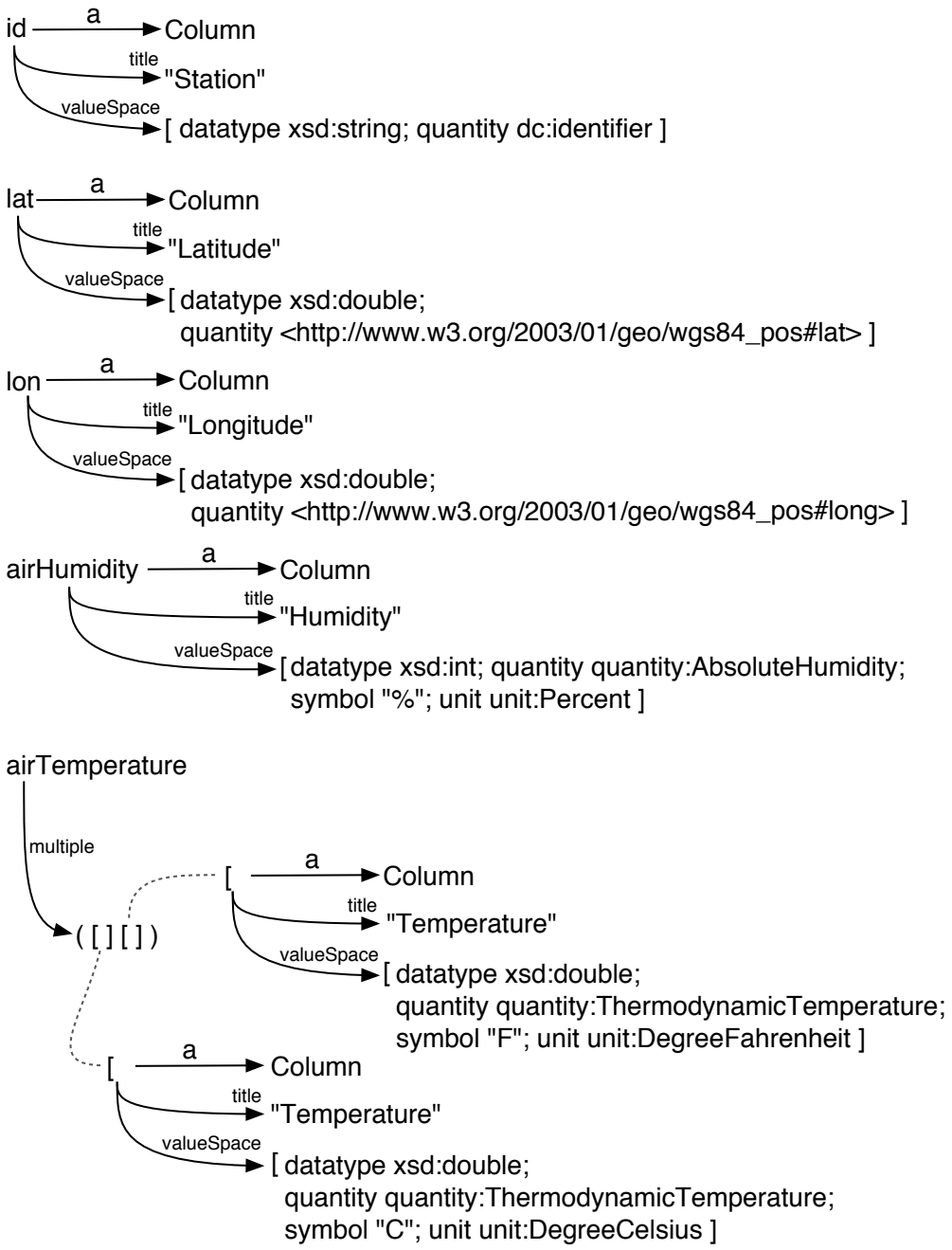
Once a set of reports are converted to triples, a receiving system needs navigation information to help process the information. A consumer can start from the designated subject `_:weather` and navigate weather reports either the via the stations or all the weather reports, as shown at the top of *Figure 6.24, Weather Report Graph*. An extract of the triples for these navigation aids are shown in *Appendix D: Figure D.13, Navigation Triples*.

Finally, as the individual properties of the weather report correspond to specific columns, each property can be defined as well. The property URI is used as the subject of a column definition and a variety of examples are shown in *Figure 6.25, Weather Column Definition Graph*, with an extract of the full triples shown in *Appendix D: Figure D.14, Column Definition Triples*. Each definition has a title and a value space definition, where the latter can be used to understand more about the encoding, quantity, and unit of measurement used for value representations. The value space is defined by datatype, quantity, unit, and symbol. These values are transferred from the PAN column definitions to the resulting graph.

A complexity may arise when there are multiple columns for the same observed value (e.g., temperature in different units). In this situation, a `w:multiple` property contains an object collection with both column definitions. Each column is defined exactly as it would be otherwise. The result is that a consuming application must match the observed value in the

weather report by unit of measurement or symbol, but must do so only when the property has multiple values.

Figure 6.25 Weather Column Definition Graph



The whole conversion process, as just described, was run for a 20 minute time period for the region [(40°,-125°) (35°,-120°)] using 2.5° quadrangles. The

process retrieved a single partition for each of the nine sequence numbers 2975, 2976, 2977, 3119, 3120, 3121, 3263, 3264, 3265. Within these partitions there were 536 reports for 321 stations. The result was 14,947 triples generated for the same data.

While the number of triples may seem large, many production-scale triples databases are quite capable of storing and querying billions of triples. Getting from the PAN structures to these triples is demonstrably easy. The inferencing necessary to generate triples for PAN-encoded data has been demonstrated to be minimal and reduces to little more than enumerating table rows over a set of Web resources (partitions). As such, harvesting information from a PAN-encoded data set is a straightforward process.

6.10 Summary

The PAN Methodology, as implemented by `mesonet.info`, provides a number of direct benefits to consumers of weather data:

- The data is encoded as regular Web pages, so it is directly browsable with links to navigate from one partition to another, taking full advantage of the abilities of the OWP to visualize the data and enhance the user experience.
- Consumers intent on using the information directly can embed the data by employing "widgets" developed for such purposes.
- Developers do not need special permission to use the data and create their own widgets or visualizations.

- Scientists can compute directly over the data set within the browser to accomplish real tasks.

- Developers can use these constructs to deploy computations at a large scale using the OWP as a computing platform (e.g., via "headless" browsers).

- The data provided via PAN is compatible with the Semantic Web and Linked Data Platform.

- The weather data provided by CWOP has now become directly accessible to search engines and other agents on the Web.

To achieve these benefits, consumers and developers need not make an expensive commitment to using the Semantic Web or the Linked Data Platform. While RDFa annotations enable additional semantics and computation, their effects can be considered locally. Real benefits accrue without the burden of handling additional unneeded complexity.

At the same time, the approach is compatible with those systems that need to harvest information as triples for later processing. The partitions are annotated with RDFa to provide column semantics. The information about the columns can be used to generate triples for every row of tabular data.

Finally, real work is possible directly within the OWP. The Barnes Interpolation process demonstrates how information can be processed to generate complex visualizations. The OWP provides the ability to process and calculate upon data and the ability to display complex outputs. All of these demonstrations show how the facilities of the OWP can be brought to bear upon scientific problems by engaging the collective creativity of developers and users of the Web.

Chapter 7

Comparisons with Alternatives

18. *Existing eScience workflow tools can process PAN-enabled data; they are agnostic to data representations.*

19. *Partitioning enables an optimal way to organize data for efficient access.*

20. *The OWP is as competent as eScience workflow tools.*

In the previous chapter, we demonstrated how we compute over PAN-enabled data within the OWP. This platform provides the requisite primitives and processing power sufficient for performing scientific computations and visualizations *in situ*. The result is that scientific applications can be developed and deployed within the OWP as tools for scientists and end users.

Systems for processing scientific data and accomplishing tasks both predate and co-exist with the modern OWP. Consequently, a comparison of other approaches to processing scientific data is warranted. We will consider

scientific workflows for processing data on the Web and tools that have been developed to handle data whose provenance is a Web service.

7.1 Workflow and Tools for eScience

A workflow can be considered an abstract sequence of steps that are chained together to perform some useful process. In computing, a workflow can be used to record the steps in a computation that produces an artifact. Each of these steps consume information, both in terms of "inputs" and "options to control processing" that produce outputs; inputs, options, and outputs are all data represented in some format. Chaining within the workflow matches outputs to inputs and often results in the workflow being data-driven.

Workflows can be broadly categorized as control-flow oriented or data-flow oriented. When control-flow oriented, each step in the workflow is orchestrated by some control language or dependency semantics. Meanwhile, for data-flow oriented, each step in the workflow is invoked when its inputs "arrive" at the step. While either mechanism is viable for scientific workflows, there seems to be a preference for data-flow oriented workflows for scientific processes [95].

Finally, a typical workflow is accompanied by a visual description and tools for building workflows. Tools allow the workflow author to diagram the workflow by picking steps from a variety of predefined possibilities and then configuring their use within the platform. A user draws an association between steps to indicate connection of outputs to inputs of other steps.

While there have been a number of attempts to address data and workflow within eScience, two notable tools are Taverna [96] and Kepler [97]. In both tools, a user develops a scientific workflow through a visual tool and then can

execute the workflow, either through the desktop tool or through command-line variants. Each step has a number of options, distinct and separate from the data inputs, that are used to control the step; these can be setup within the visual editor.

While both Taverna and Kepler are data-flow oriented workflows, Kepler differs from Taverna in that it uses the concept of a "director" to decide when a step executes while Taverna steps generally execute when a sufficient set of data is received on the step's inputs. When the Synchronous Data Flow (SDF) director is used, the Kepler workflow executes in a similar fashion to Taverna.

In both tools, steps are selected from a pre-populated inventory of steps; a user selects a step from the palette and drags it onto the workflow. Once placed, the step can be connected to the appropriate place within the workflow.

The steps provided primitives for executing scripts and accessing data. A typical process might interact with a Web service or simple Web resource to retrieve initial data and subsequent steps manipulate and process that data in various ways. The workflow produces a number of outputs in various formats.

Both Taverna and Kepler provide steps that allow a user to access data in various Web-oriented formats, including XML, and process them with typical XML tools (e.g., XSLT [98]). In addition, a user can write their own scripts in the tool-specific choice of scripting language. The scripts allow the user to extend the workflow tool without necessarily becoming an experienced programmer that can produce and integrate a library into the tool's inventory of steps.

7.2 Workflows for Barnes Interpolation

The Barnes Interpolation example (see *Section 6.8.2, Barnes Interpolation*) seems likely as a perfect candidate for comparing eScience workflow tools as we have a working example within the OWP and data encoded via PAN in `mesonet.info`. The interpolation process starts with a few input parameters (i.e., a geospatial region and time period); the result is a visualization of the interpolated grid. The steps within the process must interact with a data source, typically on the Web, collect the observed values into an input grid, and then run a local computation.

Both of the Taverna and Kepler tools should be able to accomplish this task given their built-in components. They both have steps to access data from Web resources, and steps for manipulating XML markup, and they each accommodate scripting to implement the data gathering and interpolation steps. As such, there should be no need to develop extensive libraries to support the task. Instead, we can act as a typical user of the tool that is using it as intended.

Neither Taverna nor Kepler have a stated preference for how data is accessed or represented. They do provide steps for accessing data over SOAP [99] or REST [100] services and so have a built-in preference via their inventory of steps. Moreover, while their inventory of steps provide ways to directly manipulate XML markup, it is not necessary that the data be represented in any particular methodology or format. This brings into question whether the CWOP data provided via `mesonet.info` via PAN is the right starting place.

In the particular case of using the CWOP data, finding the data from another source is difficult as the CWOP data is only available directly via the APRS feed over the APRS-IS protocol (a non-HTTP protocol). While the NOAA does consume the CWOP data, their output only contains the subset of the data

they consider valid. As a result, obtaining the CWOP data from the NOAA would provide an inconsistent basis for comparison.

Since both Taverna and Kepler are agnostic as to where the data comes from, the use of the PAN-enabled CWOP data should provide an interesting comparison to computing within the OWP. As such, the comparison is not between data representations but, instead, between systems that process the PAN-enabled data. That is, a comparison between the OWP as platform for scientific computing versus Taverna and Kepler.

As such, we implemented the Barnes Interpolation (see *Section 6.8.2, Barnes Interpolation*) workflow for both Taverna and Kepler (see *Appendix E, eScience Workflows*) that use the PAN-enabled CWOP data from `mesonet.info` to accomplish the following tasks shown in *Figure 7.1, Generic Barnes Workflow*:

1. From the input region, time period, and quadrangle size, generate a sequence of URIs for each of the data partition Web resources.

2. Fetch each data partition Web resource.

3. For each data partition Web resource received, extract single column of data of observed values (e.g., temperature) via XSLT.

4. Collect observation data into a sparse grid of the mean value for the observed values that occur within each grid cell.

5. Apply Barnes Interpolation to the grid to produce a full set of interpolated values.

6. Render the interpolated values as an SVG image with appropriate color coding.

There is very little difference between the Kepler and Taverna workflows with regards to what they do conceptually. The implementation technology

for scripting is different (Python vs. BeanShell) and the configuration of each step is a bit different, but otherwise they both follow the same sequence of operations shown in *Figure 7.1, Generic Barnes Workflow*. There are a few minor differences, either between the two, or with PAN+OWP:

- Taverna can only pass strings between steps and so every step that processes XML must parse each input and serialize each result. Kepler does not have this restriction.

- Taverna requires a few extra steps to merge responses from iterating over the list of URIs into a single XML document. For Kepler, it builds a single large array of the Web request responses.

- The PAN+OWP implementation does not employ XSLT and performs its extraction of data by accessing the RDFa annotations.

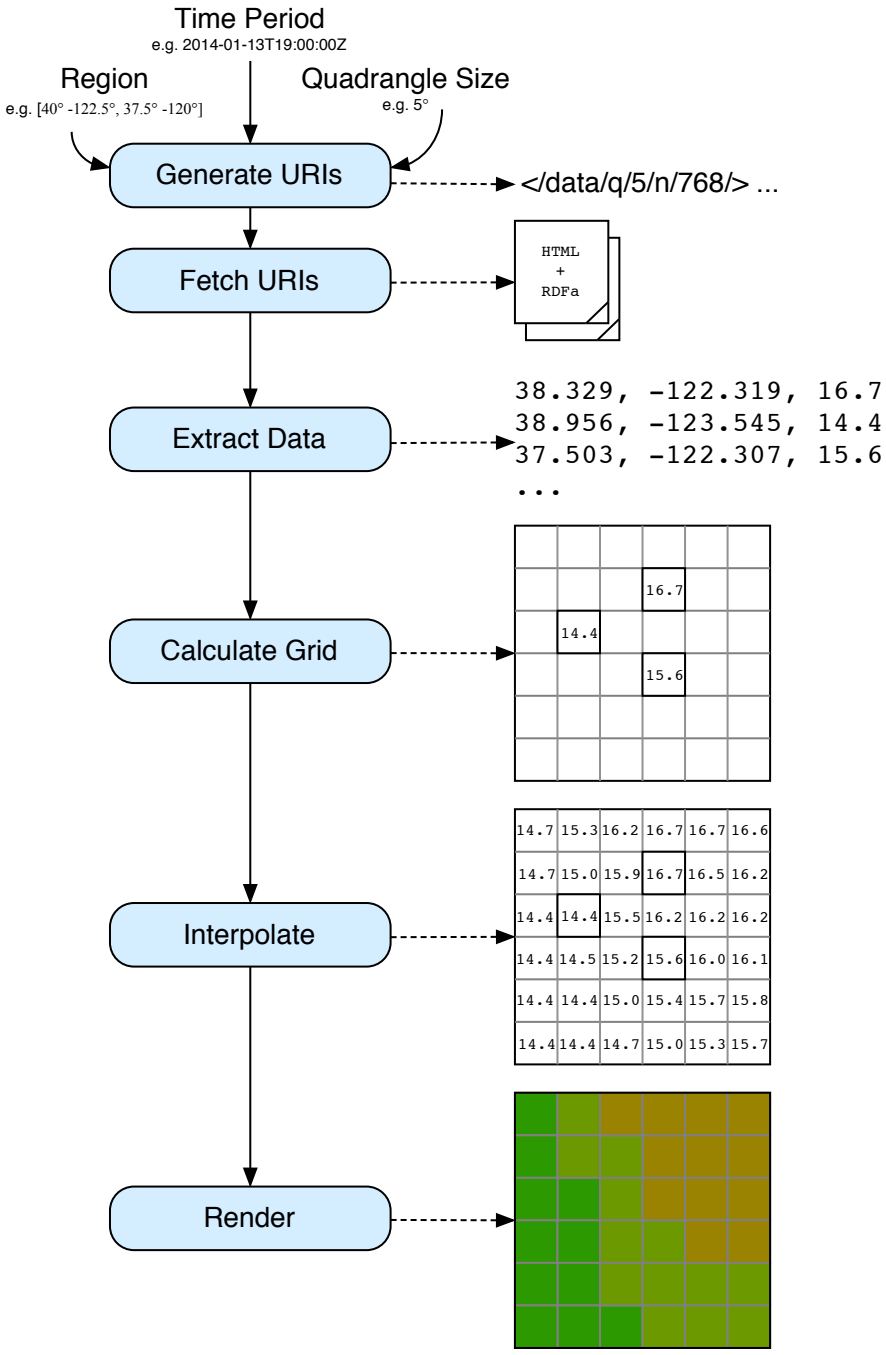
- The PAN+OWP implementation does not use a straight-through pipeline and, instead, builds the grid incrementally by taking advantage of the Map / Reduce process.

7.3 The User Experience

Both of Taverna and Kepler are visual tools, written in Java, and oriented towards a user designing a workflow as an artifact (a file format) that can then be run by a workflow engine as necessary. A user can run the workflow directly within the tool. This provides the ability for the user to iteratively develop the workflow by adding steps and testing the result.

As neither tool is aware of RDFa as a data representation format, it became necessary to develop a simple XSLT library to process the RDFa data to

Figure 7.1 Generic Barnes Workflow



accomplish extraction of a single column of data (e.g., temperature). The intent was to use a standard step to invoke the transformation and run the data extraction from the data partition Web resource. The features necessary to process the data efficiently required the use of XSLT 2.0 [101].

Unfortunately, neither tool supported XSLT 2.0 directly and they both installed a bug-ridden version of an XSLT 1.0 processor. To the novice, this unexpected glitch might be an insurmountable obstacle. For an experienced Java developer, the ability to replace the XSLT processor within the system is known but requires some effort.

In spite of considerable effort to modify the installed software for both Taverna and Kepler to include a new XSLT processor that implements XSLT 2.0, neither Taverna nor Kepler were able to successfully apply a standard XSLT step properly. Instead, in both cases, a simple script was used to apply the transform to the document. While this script was easily developed, the fact that it was a possibility might not be obvious to a user.

The remaining steps, with an exception of fetching the Web resource via the URI, were all custom scripts written in the scripting language preferred by the tool. In both tools, to configure these steps, the script must be embedded within the workflow. There is no provision for referencing an external file, making it awkward to author scripts using typical developer tools.

The tools provided comparable user experiences for the visual editing of the workflows, but executing and inspecting the results was superior within Taverna. It provided the ability to examine the inputs and outputs of the workflow and inspect intermediary results. This is one place where Taverna excelled over all others—including the OWP.

Finally, the workflow file that the tools store is not intended to be accessed by the user, in so much as is easily determined. The format is a serialization of the representation of the workflow in an XML format that is not documented for the user. Presumably, no user is expected to open the file format in another tool and just edit the source. This design decision runs counter to the "view source and modify" mantra of the open Web.

Comparing this experience to the implementation of the same process within OWP requires recognizing that we did not develop a visual workflow authoring tool. While a developer may use the libraries to create their own workflows, such a user has a different skill set. A developer for the OWP is always writing JavaScript code, no matter how small of a task. While a conscious choice, the intended user is expected to be Web savvy and the resulting code and process are slightly different.

The OWP implementation is described in detail in *Section 6.8.2, Barnes Interpolation*, where the process uses a Map / Reduce mechanism to produce the sparse grid of observed values. The subsequent interpolation step is then a single call to a library. This results in an in-memory grid of interpolated values that can be visualized and displayed directly within the browser using any number of techniques (including overlays on a map).

One last usability comparison is that, in each implementation, a set of scripts were developed to accomplish each workflow step and the amount of code is a simple measure of complexity. In *Figure 7.2, Code Complexity*, the total line count for uncompressed code was computed for all inputs to each target platform. Taverna had the most complexity, possibly due to the choice of Java and BeanShell [102] as scripting technologies; Kepler was the most compact by using JPython [103] for its scripting.

A conscious choice was made to use the tools as a typical user might and not develop extensively complex solutions to problems encountered. This meant implementing the steps using the scripting language provided by the tool and any additional supporting features. In the case of Taverna/BeanShell, additional Java libraries can easily be configured within the tool but such option was not readily available for Kepler/JPython.

Of the choices of implementation technology, the determination of which is “easier” to a particular audience of developers is subjective. The choice

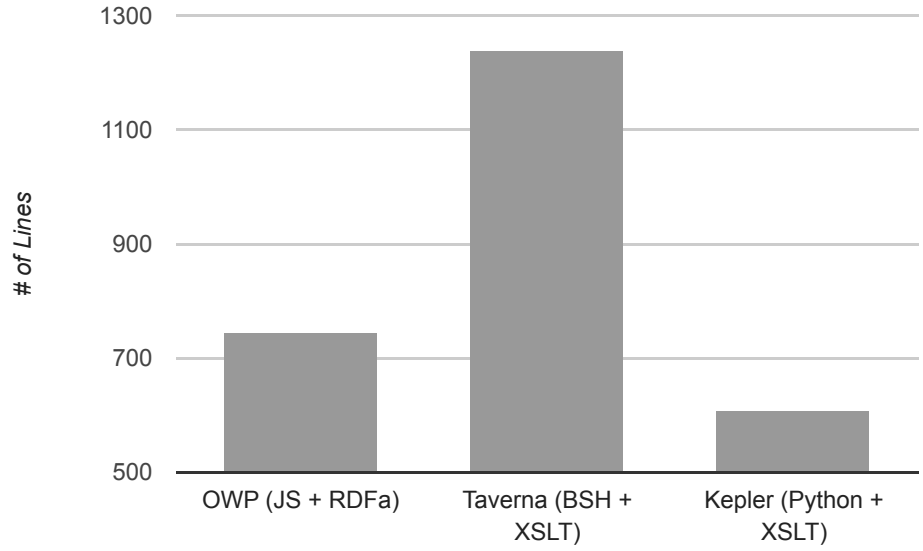
of BeanShell is somewhat obscure (the JSR is dormant [104]) but not objectionable to a seasoned Java developer. Python is widely used within scientific communities, but JPython certainly has its issues (see *Section 7.4, Comparing Workflows to OWP+PAN*). JavaScript is less used for scientific computation but increasingly being used for data visualization.

It became clear that to attain the best performance from either Kepler or Taverna, writing components in Java or a language that compiles to efficient byte code is a winning strategy. In the case of the Taverna workflow, the interpolation was written in Java and used as a library by the BeanShell script. We shall see how this helps the performance of the interpolation step for Taverna. Such an enhancement can be considered for the Kepler workflow, except that the ability to configure supporting libraries is not easily deciphered.

As a final note about workflow implementation, both the Kepler and Taverna tools required quite a bit of "Java trickery" to properly configure an XSLT implementation that was capable of doing the data extraction via RDFa. While both tools shipped with the ability to perform some basic XSLT transformations, the choice of implementation was of poor quality. To make these workflows work, the XSLT implementation had to be replaced within the install and that turned out to be a non-trivial task. For Taverna, further configuration outside of the install was necessary to provide the supporting tools it required (i.e., "graphviz" for generating the workflow diagrams), and that installation process had some additional issues as well.

In comparison, the OWP platform does not require any special modifications. Any reasonably standards-compliant browser should be able to run the interpolation process. The speed at which it can do so is directly related to the quality of the implementation and the performance characteristics of the machine on which it is run.

Figure 7.2 Code Complexity



7.4 Comparing Workflows to OWP+PAN

Scientific workflows can also be codified by libraries and scripts that are embedded within applications whose steps are not necessarily uniquely identified. The previous examples of computing within the OWP are somewhere in between, where the OWP implementation employs scripts that are chained together, and orchestrated by browser events, to accomplish the task. As such, there exists a broad spectrum from tightly integrated code through to abstract workflow steps that pass data between themselves.

With this ambivalence to data sources in mind, we performed a comparison of the ability of these eScience workflow tools to carry out the Barnes Interpolation process over the PAN-enabled data services of `mesonet.info`. Each workflow was to carry out the same interpolation process, retrieving data partition resources for 2.5° quadrangles, over rectangular geospatial regions of increasing sizes to result in retrieval of 1, 4, 16, and 64 separate data partition Web resources. The result of the interpolation is an SVG image

that can be compared to the results from the OWP implementation from `mesonet.info`.

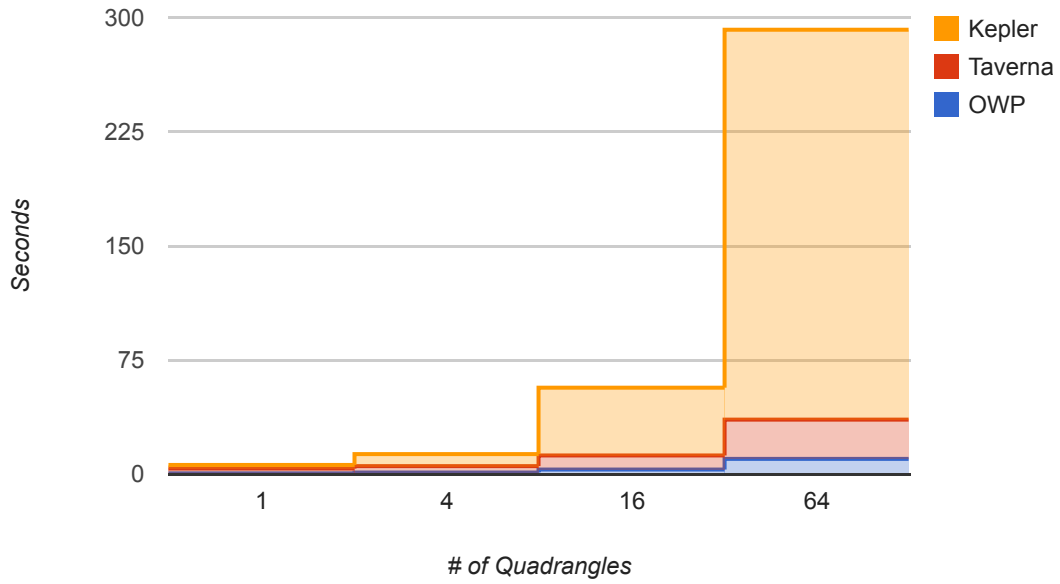
An overall performance comparison of the three implementations is shown in *Figure 7.3, Overall Interpolation Comparison*, that was performed for the sample time period (30 minutes starting at 2014-01-29T20:00:00Z) with four different regions ([40°,-122.5°,37.5°,-120°], [40°,-125°,35°,-120°], [40°,-125°,30°,-115°], and [40°,-125°,20°,-105°]) using 2.5° quadrangles and a 0.1° resolution on the interpolation grid. As is easily seen, the Kepler workflow dominates the chart and performs very slowly on progressively larger regions. The Taverna workflow roughly keeps within the same factor as the OWP implementation, with the OWP being the fastest.

These numbers reflect the choice of implementation technologies. Both Kepler and Taverna are Java-based tools and component steps in the workflows are some combination of Java classes. In the case of Taverna, general scripting of steps is accomplished by using BeanShell. In contrast, Kepler uses JPython for scripting steps.

The surprising result is that the same Barnes Interpolation algorithm implemented in Python for the Kepler workflow and run via JPython is significantly slower and takes up a majority of the processing time (e.g., 87% for 64 partitions). The algorithm has a similar structure to the Java-based implementation used in the Taverna workflow (see *Appendix F, Barnes Interpolation Implementation*). Researching this result reveals that JPython has significant penalties for array processing compared with other Python implementations.

The Kepler workflow also suffers from slower data access speeds (i.e., 1.9 times longer than Taverna and 7.2 times longer than the OWP). The relative performance of the access methods is most likely a function of the quality of implementation of the Web access (e.g., HTTP GET) along with the

Figure 7.3 Overall Interpolation Comparison



	Number of Quadrangles			
	1	**4**	**16**	**64**
OWP	0.49	1.03	3.00	10.00
Taverna	3.18	4.20	9.26	25.82
Kepler	2.35	7.92	44.58	256.43

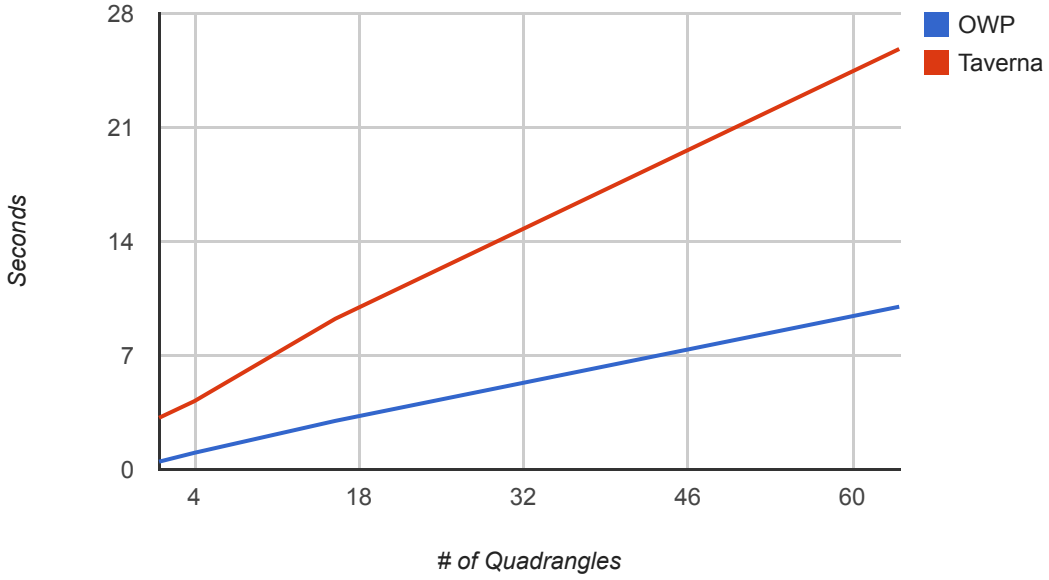
The table of data above corresponds to the graph and all the measurements are in seconds.

implementation's ability to run a sequence of data manipulation steps over a sequence of inputs in parallel. For processing individual requests, the data extraction uses the same Java-based XSLT implementation and transformation used by Taverna and so should be similar, but the result shows it is not.

Looking specifically at Taverna versus the OWP, we can see more detail (see *Figure 7.4, Taverna vs OWP - Overall Comparison*). Both implementations

have linear performance characteristics but the Taverna implementation is increasingly slower as the number of data partitions increases. While the response times are not as fast as the OWP, the overall time for Taverna is not completely unreasonable for offline processing.

Figure 7.4 Taverna vs OWP - Overall Comparison



Separating the Barnes Interpolation from the data access shows where a majority of the cost is being incurred in Taverna. As the number of data partitions increases, the cost of the data access processing increases (see *Figure 7.5, Taverna vs OWP - Data Access*) while the interpolation remains a relatively constant factor of the OWP platform (see *Figure 7.6, Taverna vs OWP - Interpolation*). Significant improvements in overall processing time can be attained by either improving the data access methods within Taverna or the step choices and configuration within the workflow.

As both Taverna and Kepler are Java-based workflow engines, it might be expected that they can take advantage of the advanced threading support of Java to allow data access to occur in parallel. JavaScript within the OWP does not have this ability in that there is only one main thread and computation is

Figure 7.5 Taverna vs OWP - Data Access

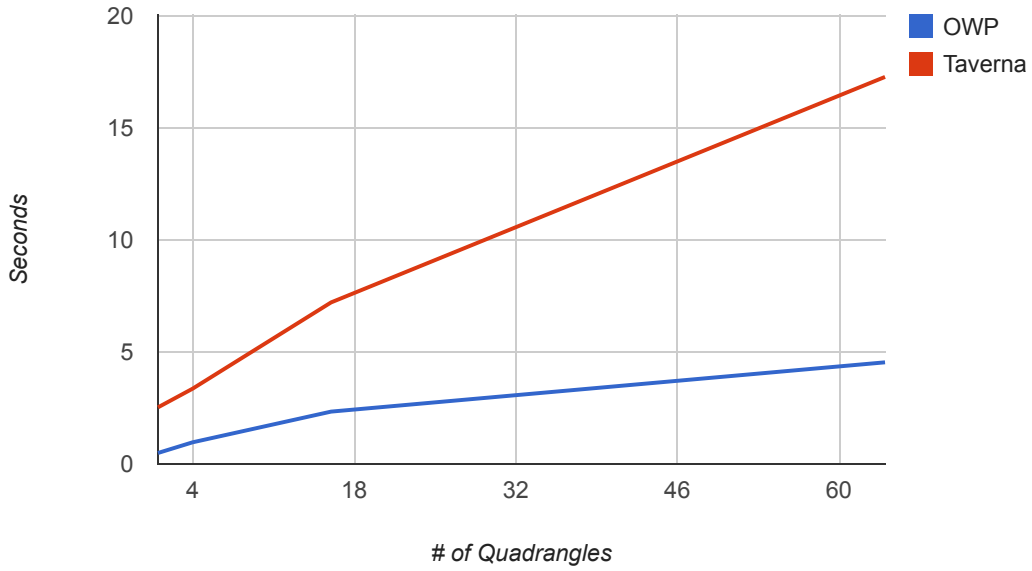
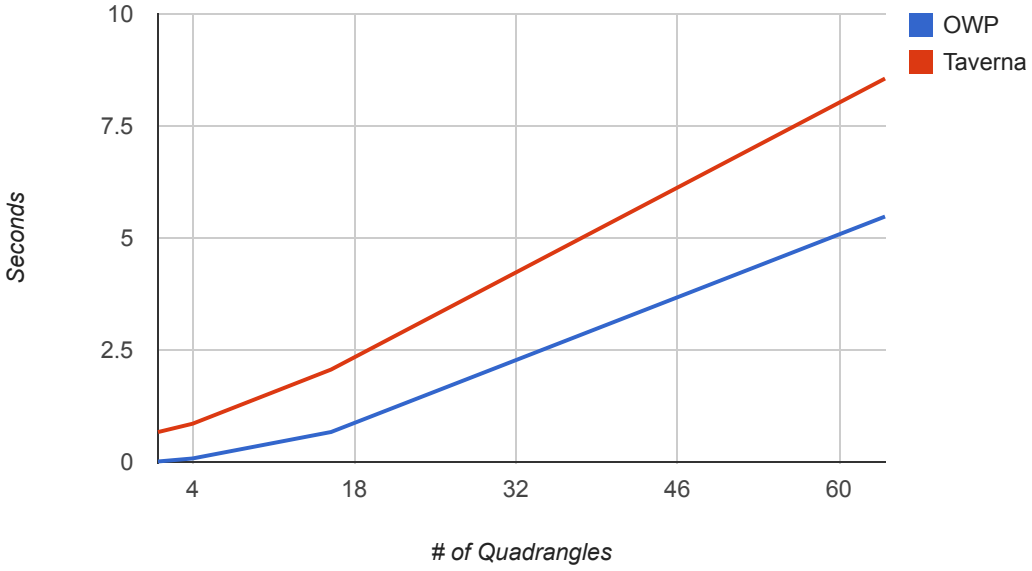


Figure 7.6 Taverna vs OWP - Interpolation



interleaved by the use of callbacks. Yet, the OWP platform implementation is able to get some parallel request processing and neither workflow platform seems to do well in comparison.

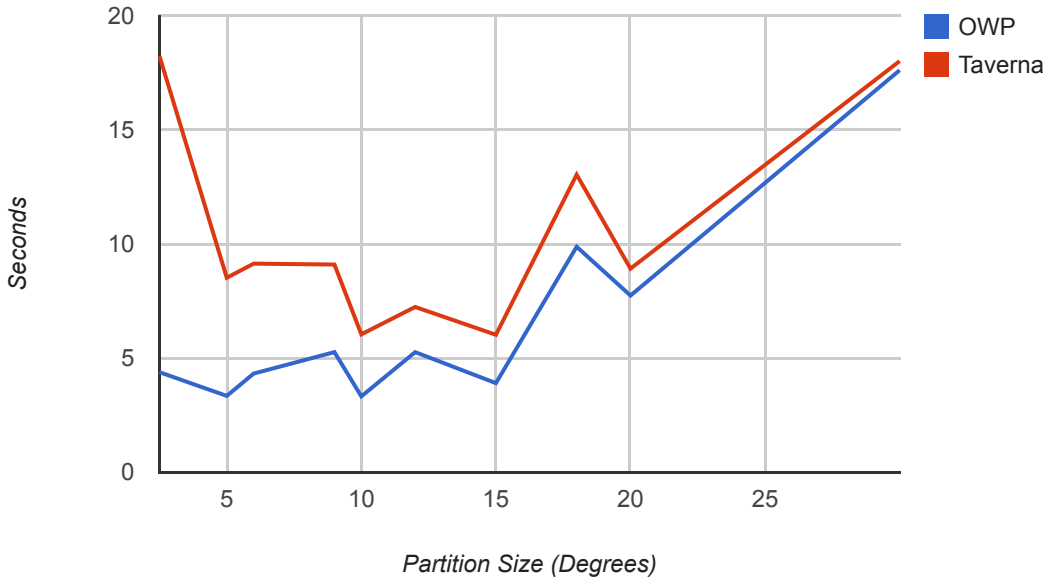
To this end, Taverna supports iteration directly within the workflow and allows for automatic iteration over a sequence of items. In this particular workflow, the iteration feature makes it easy to specify and apply a portion of the workflow to the sequence of data partition URIs to generate a complete result (the sparse grid of observed values). Unfortunately, the iteration does not seem to take advantage of the natural parallelism of these requests and this adds significantly to the processing time for larger regions.

It is unlikely that Kepler will have such a feature as its workflow uses a director to sequence when steps occur. In attempts to get parallel requests, the director needed to know exactly how many items were to be in the sequence. It also lacked the ability to collect the result (i.e., a meet in the workflow lattice) which was necessary for this workflow. Nevertheless, the problems with Kepler in interpolation workflow were not caused by lack of iteration features.

7.5 Does Partitioning Help?

As data access is a significant portion of the computation time, the question arises as to whether partitioning helps tools such as Taverna or scripts written for the OWP. The data shown in *Figure 7.7, Partitioning and Data Access*, is the timing of the data access for the largest region (20° by 20° rectangle) used in the previous timing experiment. The partition size was varied from 2.5° to 30° and the amount of time necessary to produce the sparse grid of observed values was measured for both Taverna and the OWP. The same data is accessed via different sized partitions and the result of the computation is always the same.

Figure 7.7 Partitioning and Data Access



While the OWP remains relatively constant at under 5 seconds through partitions up to size 15°, there is a dramatic drop in processing time for Taverna from 2.5° to 5°. For some undetermined reason, the smallest partition size, which causes the largest number of partitions to be retrieved, causes a problem for Taverna. Note that at 5°, data access and processing times for both systems are similar in relative performance and increase in the amount of time necessary to process larger and larger partitions. There is a notably large spike at 18° partitions which is due to poor intersections between the requested region and the necessary fixed quadrangles.

The access times for Taverna and OWP are remarkably similar but this should not be a great surprise. Both systems incur the same latency between request and response for accessing the same data partition resources via the same URIs. Ergo, the difference in processing time is a combination of how efficiently the post-response processes can be performed and how much parallelism is available.

While the OWP platform performs consistently better for smaller partition sizes, both systems approach the same amount of time as the partition size grows larger. The data seems to confirm that there is a range of partition sizes that are optimal; outside of this range a system will experience more data access and processing time. While there are other factors that may affect the choice of partition size, the choice of size can be experimentally determined.

In the end, data density across geospatial regions matters in terms of picking an optimal partition size. While the optimal value can be experimentally determined, smaller tends to be better for both tools. In case of tools such as Taverna, partition sizes that result in too many requests detrimentally affect the overall processing time. The OWP platform seems to have less issues with many small requests, most likely due to the fact that the OWP has been optimized to handle this case particularly well.

7.6 Summary

In some respects, we took these workflow tools outside of their "comfort zone", so the comparison is a bit unfair. Despite this, in terms of runtime performance, Taverna performed reasonably well and Kepler perhaps a bit less well. That the difference is in favor of PAN+OWP was more pronounced in the area of ease of development is less surprising, given the somewhat awkward fit of the task to Taverna and Kepler's application orientation. A reverse experiment that takes a workflow already implemented in one or both of the workflow tools and implements them within the OWP would also be a useful test for the OWP platform and the PAN Methodology in the future. Such an experiment would require providing data accessible over the Web via services.

The PAN Methodology addresses how data is published and accessed; this is something that workflow tools remain silent about. Workflow tools have preferred methods for accessing data in terms of format and size. As such, there is a hidden assumption that, within eScience workflows, the data will be accessible in a form that is easily processed by the workflow.

The PAN approach provides a new perspective that allows the OWP to succeed in accomplishing some of the same things as eScience workflow tools. Not only is the OWP sufficient and performs well, but it also provides a level of uniformity in processing expectations due to its standardization. This allows components to be packaged and distributed for common operations, such as Map / Reduce or libraries for interpolation and visualization.

At the same time, when data is published via PAN, the use of existing tools is not prevented. The data is accessible on the Web via the same expected mechanisms of REST-oriented services. While a small layer of processing is required to handle the RDFa annotations, existing tools seem quite capable of manipulating and extracting data for their purposes without extraordinary efforts by the user. As such, the methodology provides a way to bridge the gap between past efforts using eScience workflow tools and future efforts on the Web using the OWP.

Chapter 8

Summary, Future Work, and Conclusion

Publishing and sharing scientific data on the Web remains challenging and there is ever an increasing demand for successful mechanisms to do so. There is an increasing amount of data being collected, often by automated systems, that can easily overwhelm both the publisher and consumer. At the same time, there is an ever increasing demand for open and easily accessible scientific data.

We have shown that much of the data being produced by government-related scientific agencies is represented in tabular form. At the same time, a plethora of formats for data sets have been unleashed, some of which only exist within the archives of such agencies. Access to this historical record of scientific data is inhibited by a variety of problems; data is problematic due to formats that are obscure or hard to process and understand.

In looking at existing efforts by the `data.gov`, IVOA, and OGC, we see early adopters of the Web who heavily embraced XML and Web technologies. While their efforts resulted in standards that systems can use to exchange information, aspects of their approach lack direct support within the current

OWP. Consequently, accessing data archives lacks ease of use and requires quite a bit of background knowledge on advanced XML technologies.

Where the data formats were developed to exchange tabular data (VOTable), the lack of adoption of further semantic vocabularies has limited innovation. Further, for data formats where tabular data was possibly an afterthought (KML), the limited ability to precisely define the tabular data or the inconvenient division of the data makes the format less useable. Many of these efforts are limited by being designed for specific use cases (e.g. tool-to-tool exchange or visual display).

The result is we see archives like `data.gov` or applications like Galaxy Zoo that attempt to use these standards but fail to generalize. In some cases, the tabular data has limited conversion potential because of the visual-orientation format and limited knowledge of the "right" visualization. In others, the format either fails to generalize or is used outside its intended context (e.g. large compressed archives of VOTable documents).

To address all of these issues, we identified four essential qualities for scientific data on the Web: identifiability, extensibility, flexibility, and durability. We suggested achieving these qualities by re-using the existing OWP combined with RDFa annotations. The qualities are then derived by exposing data using the technologies of the Web; specifically, naming, simple retrieval, and HTML with RDFa annotations.

Finally, the PAN Methodology describes how to publish data sets onto the Web. The concept of partitioning data into reasonable-sized Web resources is essential for viewing and processing live data on the Web. At the same time, naming becomes important for the ability to access data and to provide the infrastructure for referencing and annotation.

The application of the PAN Methodology to the CWOP data set and its implementation at `mesonet.info` provides the ability to test the methodology on a large and ever-expanding data set. As data services on the Web, the same information can serve both the OWP and other tools such as eScience workflow tools like Taverna and Kepler. Use of this system, as determined by the fact that same workflow is possible in all three systems, demonstrates the openness and practicality of the methodology.

Moreover, not only is the resulting data easily accessible but computation within the OWP over the data it is possible in real-time. The facilities inherent in the OWP provide a stable basis for developing both libraries and applications for processing data. This demonstrates the key outcome of enabling experimentation on the Web by unintended uses; in turn, this enables a network-effect for scientific data.

8.1 Open Questions

With respect to the PAN Methodology, some questions remain open:

1. How would the approaches of the IVOA standards be different and what is the likelihood of adoption?

2. Can the role of HTML be replaced by other markup or data formats?

3. What are the practical limitations of computing over the data?

4. How should non-public data be shared and accessed?

IVOA continues to improve existing specifications and develop new application areas. Recently, they published *Data Access Layer Interface* [105] that "defines resources, parameters, and responses common to all DAL

services." This specification explicitly endorses the use of RDFa annotated HTML as an alternative to an IVOA custom XML vocabulary for describing services. As such, there is a conceptual basis already present to use RDFa as an alternative to encode information that might otherwise be conveyed with custom markup.

Members within the IVOA community have expressed their desire for RDFa annotations and possible applications within existing markup schemes. They have not yet gone so far as to replace the VOTable format. This is most likely due to the current investments in systems and tools that support the VOTable syntax.

Certainly nothing within the PAN Methodology limits the format of a data set partition resource to only HTML annotated with RDFa. While browsers may intrinsically understand HTML and provide additional operational semantics (e.g., executing of scripts), any markup can have RDFa annotations. The problem is not within the browser but the existing receiving applications. As was noted before, KML, VOTable, and other markup vocabularies are defined as *closed* vocabularies, and so, not only are the RDFa attributes not allowed but tools may find them objectionable and fail to process the received data.

Given sufficient adoption of the PAN Methodology by scientific data publishers, with all this data available for processing, brings us back to the question of practical limits and how far can we push the OWP to compute with this data. The Barnes Interpolation example gives some insight; a vast majority of the time is spent retrieving data rather than processing it. As such, the limitations are directly related to the cost of retrieval both in terms of time and network bandwidth.

Many grid and column stores overcome this problem by facilitating data locality. That is, they bring the processing closer to the data, often caching columns of information across a cluster of computing nodes. This allows

a computation to be orchestrated such that when the process required to perform the task and a portion of the data are co-located, everything operates locally.

Data and processor co-locality is a desirable properly for processing large amounts of data quickly and the PAN methodology is designed to leverage this property. PAN provides a basis for retrieving partitioned data without necessarily having to load and process the whole data set. That is, processes can be designed to decide which partitions are needed and only those data set partitions are retrieved and stored locally.

Meanwhile, the idea of computing over live data is still viable and very compelling. The tradeoff between the cost and complexity of a large column storage cluster versus the simplicity of aggregating within browser is a tradeoff of speed versus cost. Enabling such cost tradeoffs also enables computing for small-and-medium-sized science or interdisciplinary applications, where cost and support may represent major barriers.

Finally, there are issues related to sharing data that is not available under open access licensing terms. As the PAN Methodology relies upon the Web, there are many layered applications for security and controlling access to data. Specifically, *The OAuth 2.0 Authorization Framework* [106] provides the ability to use access tokens and other security mechanism to control access to data. This allows the same PAN Methodology to be accessed over secure networks by designated users.

8.2 Future Work

There are a number of possible next steps that fall under several categories:

1. *Technology Enhancements* — scaling or enhancing the infrastructure further to provide new insights such as:

    - Automatic extraction of data into triples, KML, and other common data formats.

    - Scale-up the implementation via clustering and caching as the current system only holds three months of data (300+ million weather reports).

    - Implementation and use of service descriptions (i.e., *Appendix B, Resource Schemes and Discovery*).

    - How would improvements in data partition retrieval times improve or hinder the ability to compute? What are the limitations before the data retrieval overwhelms the computation?

    - How can a Map / Reduce system (e.g., Hadoop) utilize a PAN-enabled archive of data?

    - How could OAuth 2.0 be integrated to control access to data sets?

2. *Verification* — verifying the methodology against further existing data sets or new use cases such as:

    - Do the naming choices used for `mesonet.info` represent a best practice? How can we verify the "social contract" of the URI?

    - How much data can be automatically converted (e.g., KML or F025 marine mammal ecology data) and what additional metadata is necessary?

- How well does the iNaturalist database or an IVOA Simple Cone Search service translate into a PAN-enabled archive?
- How could the use of the QUDT vocabulary be extended or used more effectively to achieve some interoperability goals similar to what IVOA was attempting to do with UCD+ labels?

3. *Extension* — extend the methodology beyond systems with records and coordinate systems such as:
    - Digital humanities applications (i.e., The Pompeii Bibliography and Mapping Resource [107]),
    - Biological databases (e.g., genomic or protein databases),
    - Non-scientific data such as community, government, census data, etc.,
    - Other non-tabular data sets.
4. *Outreach* — facilitate adoption by:
    - Develop and promote a "PAN Starter Kit" for both publishers and consumers.
    - The CWOP archive is coordinated in part by the US NOAA's National Weather Service yet the property labels used by `mesonet.info` are specific to itself. A common set of weather-related properties would be useful to all and there are a number of standards organizations that could help produce such a standard.
    - Ecology is a great example where the data is already crowd-sourced and often "low-tech" but having the results in an

open digital archive would be enormously useful. What are the characteristics of the work flow and data and how does that affect the resulting archive?

- Many journals have attempted or are attempting to provide the ability to archive data sets that are often referenced in published papers (e.g., Scientific Data from Nature, May 2014). How well does a partitioned data set fit within their model? Would an interactive model such as PAN be socially acceptable to their business model?

- Work with IVOA to apply the PAN methodology for partition data as an alternative to VOTable.

8.3 Conclusion

The PAN Methodology was born from a desire to address the needs of both the professional and citizen scientist while providing access to any interested party. The use of the OWP as a basis allows the PAN methodology to scale down to very small data sets and this enables both medium and small scale science to operate on the Web at a lower cost with increased network effects. Compatibility with the OWP is maintained by allowing applications to crawl and harvest data via the annotations.

Some current trends advocate exposing government and scientific data on the "Semantic Web" and for providing a complex stack of semantically-enabled (triple-aware) technologies. These trends ignore the need to enable a user to consume data without complexity. In this respect, the PAN methodology provides a useful middle ground between complex data representations or services and simple expressions of partitioned tabular data. By doing so, PAN

enables the OWP platform to be an active participant; all that is required for real work to be accomplished is a Web browser.

Providing the CWOP data through `mesonet.info` has demonstrated the practical application of the methodology to a large and regular data set. All the evaluations were enacted through the OWP and enabled by the representation of previously unavailable data. The generation of complex resultants (e.g., a Barnes Interpolation image) in real-time demonstrates how enabling data within the OWP allows for unexpected consequences; computations previously done offline are now available within the OWP.

The PAN Methodology provides publishers a straightforward way to provide data on the Web that is computable within the OWP. They can provide one set of data that is able to be explored and viewed by consumers while still preserving the ability for tools and technology to compute over the very same resources. The ability for both the OWP and eScience workflow tools, like Taverna, to easily compute over the PAN-enabled `mesonet.info` data demonstrates the tractability of this approach.

Enabling data access and manipulation within the OWP makes possible the network effect on the Web for scientific data. Data that was once delegated to the "download and process" model can now be made interactive. Tools and applications can easily be deployed on the Web for professionals and citizen scientists and a new set of participants: Web developers. By doing so, a wider range of individuals are enabled to participate in scientific endeavors. Their contributions are yet unforeseen but history has shown that with the open empowerment of individuals comes dramatic innovation.

Appendix A

Sequence Numbers

Each quadrangle can be uniquely identified by a positive integer, called a *sequence number*, by numbering from a preferred pole and continuing around by longitude as shown in *Figure 5.5, Quadrangles* where A=B=20°. Given any point of latitude and longitude , the sequence number *s* can be calculated by first translating the coordinates into range values (between 0 and 180 for latitude and 0 and 360 for longitude) and then applying a specific formula:

$$s \quad \frac{}{s} \quad \frac{}{s} \quad \frac{}{s}$$

where $_s$ is the longitude dimension (A) and $_s$ is the latitude dimension (B) of the quadrangle.

In reverse, the sequence number *s* also defines a unique position *P*, the upper northwest corner, that is calculated as follows:

$$N \quad \frac{}{s} \quad N \quad \frac{}{s}$$

$z \quad s \qquad N\,N$

$$z \quad N_s$$

$$P \quad \frac{z}{N}_s$$

For example, for weather station DW8568 located in San Francisco at coordinates (37.74283,-122.46283) and a choice of quadrangle size of A=B=2.5, the sequence number of the quadrangle that contains the station is 2976. This quadrangle covers almost all of San Francisco, a good portion of the Bay Area, and inland towards Sacramento. The point P for sequence number 2976 is calculated to be (40,-122.5). Using the values for A and B, the whole extent of the quadrangle can be calculated to be [40, -122.5, 37.5, -120].

In summary, sequence numbers map to specific quadrangles of a given size and any coordinate that belongs to one of these quadrangles can be associated with the quadrangle's sequence number. This ability to go back a forth between a sequence number and coordinate is an easy calculation and useful for client applications. These properties make sequence numbers desirable to use in addressing geospatial regions when requesting data over the Web.

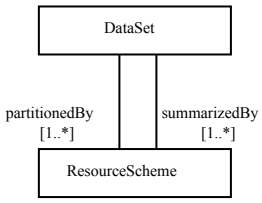
Appendix B

Resource Schemes and Discovery

While the PAN ontology provides specific properties for both starting points (`summary`) and associating partitions with data sets (`partition`), it is not always the case that these properties are directly observable. For example, a set of partitions are only discovered by accessing a summary (`PartitionSummary` typed resources) and then following the links to partitions. Applications and other systems wishing to enumerate partitions directly via metadata need more information about the choices and schemes used to encoded basic facets into URIs.

To address this, a `DataSet` may describe a set of "resource schemes" represented by `ResourceScheme` class instances that are essentially URI templates (see *Figure B.1, ResourceScheme Class*). Each template has a set of variables and parameters that can be substituted to create an actual URI used for retrieving a starting point. This URI can then be used to retrieve a representation that can be processed for annotations.

Figure B.1 ResourceScheme Class

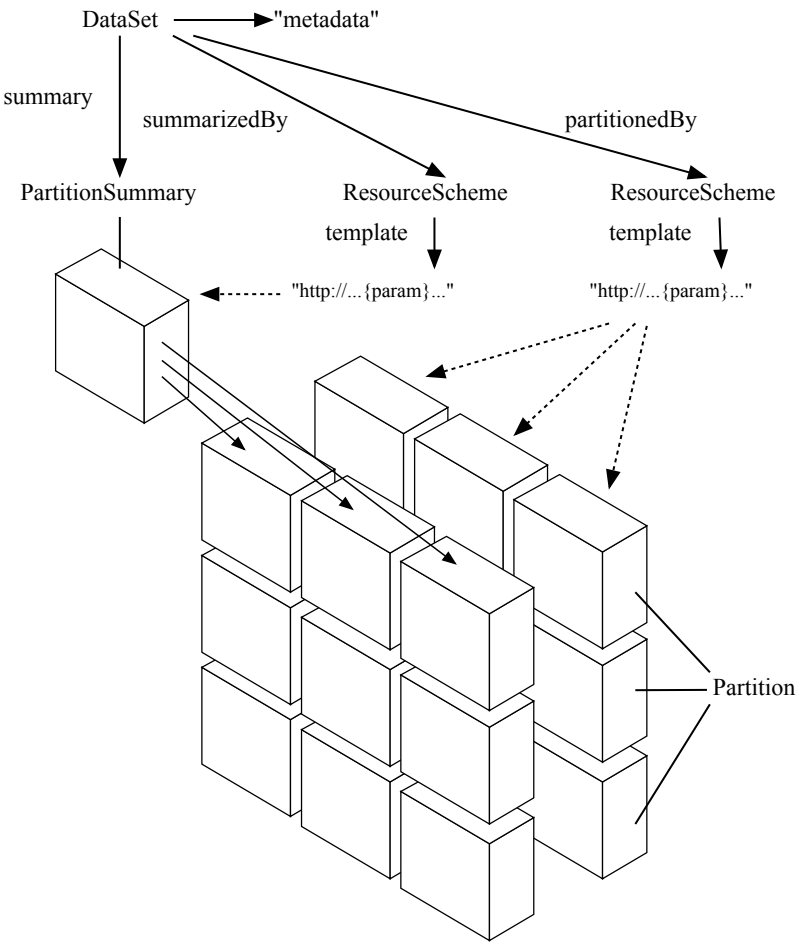


There are two general properties for resource schemes, `summarizedBy` and `partitionedBy`, whose relationships are shown in context in *Figure B.2,*

Resource Schemes and Partitions. The `summarizedBy` property is intended to result in `PartitionSummary` instances and is used to retrieve summarizations of sets of partitions.

Similarly, the `partitionedBy` property is an alternative way to understand how to directly access a data set partition resource and results in a `Partition` typed resource. In the case where an application just needs data set partitions, they can enumerate directly over basic facet ranges and use the URI template provided by this resource scheme to construct URIs for the various data partition resources. As such, they can avoid retrieving and traversing a summary of partitions resource for that same basic facet range.

Figure B.2 Resource Schemes and Partitions



Instances of the `ResourceScheme` class describe a URI template that can be used to construct the URI of a resource. The substitution variables are described as facets that an application can map to known quantities. While not necessarily completely automatic, these classes give more information to tools which can result in hints to users for how they might access data.

Appendix C

Scripting Details

An application can locate columns by navigating from the document into the annotation graph and examining the column properties. Once a desired column has been located, the column's subject URI can be used to navigate back to the particular element in the document. This process (see *Figure C.1, Finding Air Temperature*) demonstrates how an application uses this script to determine the index of a particular column of data.

Figure C.1 Finding Air Temperature

```
var columns = document.data.getValues(table.data.id,"pan:column");
var column = null;  // A variable to hold the subject URI.

for (var i=0; !column && i<columns.length; i++) {
  // Find the column labeled with the air temperature property
  if (document.data.getValues(columns[i],"pan:property")
        .indexOf("http://mesonet.info/airTemperature")>=0) {
     column = columns[i];
  }
}

// Find the index by finding the column element by subject URI.
var index = document.getElementsBySubject(column)[0].cellIndex;
```

As the API currently lacks more complex query capabilities, finding joint values such as “air temperature whose unit is Celsius” is more complicated. The PAN ontology attaches the unit to the `valueSpace` property's subject and so the iteration must navigate an additional set of properties in the annotation graph (see *Figure C.2, Finding Air Temperature and Unit*), and is only slightly more complicated.

In both examples, the table column can be located in each row by using the `cellIndex` property. This property provides the array index of the table

Figure C.2 Finding Air Temperature and Unit

```
var columns = document.data.getValues(table.data.id,"pan:column");
var column = null; // A variable to hold the subject URI.

for (var i=0; !column && i<columns.length; i++) {
  // Find the column labeled with the air temperature property
  if (document.data.getValues(columns[i],"pan:property")
        .indexOf("http://mesonet.info/airTemperature")>=0) {

     // Find the subject URI (blank node) of the value space
     var valueSpace = document.data.getValues(columns[i],"pan:valueSpace")[0];
     if (document.data.getValues(valueSpace,"pan:unit")
           .indexOf("http://qudt.org/vocab/unit#DegreeCelsius")>=0) {
        column = columns[i];
     }
  }
}

// Find the index by finding the column element by subject URI.
var index = document.getElementsBySubject(column)[0].cellIndex;
```

cell within each `rows` property array value. An application can now simply enumerate the table rows, skipping the column definitions, and pick out the specific column of data by accessing the value (see *Figure C.3, Enumerating a Column*).

Figure C.3 Enumerating a Column

```
for (var i=1; i<table.rows.length; i++) {
   var cell = table.rows[i].cells[index];
   ...
}
```

The Map / Reduce library requires that the particular columns of interest be identified before the process starts. An array of column descriptions (e.g., property URI, unit description, etc.) are constructed by the user (see *Figure C.4, Using Map / Reduce to Calculate Average Temperature*). Afterwards, the user must only supply the map and reduce functions and then apply the instance to particular region and time period.

Each of these Map / Reduce functions are passed an array of data. In the case of the map function, an array of rows of data is passed to the function. It is expected to return a value that will be collected in the result. The process is

similar for the reduce function except that it receives the an array of all the return values from all the invocations of the map function.

Figure C.4 Using Map / Reduce to Calculate Average Temperature

```
// Calculate Average Temperature

var mr = new MapReduce();

// The date/time as of now.
var endDateTime = new Date();
// The date/time as of one hour ago.
var startDateTime = new Date(endDateTime.getTime()-60*60*1000);

mr.init("http://www.mesonet.info/");
mr.columns.push({ uri: "http://mesonet.info/airTemperature",
                  unit: "http://qudt.org/vocab/unit#DegreeFahrenheit" });

// Calculates the average per quadrangle data partition
mr.mapper = function(data) {
   var total = 0;
   var count = 0;
   for (var i=0; i<data.length; i++) {
      total += data[i][0];
      count++;
   }
   return total / count;
}

// Calculates the average over all quadrangles
mr.reducer = function(data) {
  var total = 0;
  for (var i=0; i<data.length; i++) {
     total += data[i];
  }
  return total / data.length;
}
mr.apply(
   [38,-123,37,-122],         // geospatial region
   startDateTime,endDateTime, // time period
   2.5                        // quadrangle size
);
```

The process for producing a sparse grid of average values for a specific property (e.g., air temperature) is shown in *Figure C.5, Example Barnes Interpolation Script* and how that class uses the Map / Reduce library is shown in *Figure C.6, Example Grid Average Script*. The grid average map process finds values and places them into specific cells of the grid. Afterwards, the reduction constructs a full grid and calculates a mean value for cells that have observed values.

Once the grid average Map / Reduce process finishes, the interpolation can be run on the grid. The interpolation process needs the location of each grid cell and that is precomputed in *Figure C.5, Example Barnes Interpolation Script*. A simple invocation to that Barnes Interpolation implementation (listed in *Figure F.1, Barnes JavaScript Implementation*) then populates the whole grid with interpolated values.

Figure C.5 Example Barnes Interpolation Script

```
// Input Parameters
var duration = 60*60*1000;
var endDateTime = new Date();
var startDateTime = new Date(endDateTime.getTime()-duration); // one hour ago
var region = [40,-125,35,-120];

// Initialize grid average map/reduce
// Note: Uses MapReduce class internally.
var gridavg = new GridAverage("http://www.mesonet.info/data/");
gridavg.onComplete = function(grid) {

   // Compute actual position of each grid cell
   for (var i=0; i<grid.length; i++) {
      for (var j=0; j<grid[i].length; j++) {
         grid[i][j].position = [ app.region[0]-j*app.resolution,
                                 app.region[1]+i*app.resolution];
      }
   }

   // Interpolate over the grid
   var interpolator = new BarnesInterpolation();
   interpolator.interpolate(grid);

   // display grid in browser ...
}

// Perform grid averaging map/reduce for quadrangles of size 2.5 degrees
gridavg.apply(region,startDateTime,endDateTime,2.5)
```

The triple harvester invocation uses the Map / Reduce library again (see *Figure C.7, Harvester Implementation*). A set of mappings of roles to property URIs are given in the initialization of the object instance. These property URIs are used to map PAN ontology triples to the target vocabulary. In this particular case, the resulting triples will only use properties in the `http://mesonet.info` namespace.

Figure C.6 Example Grid Average Script

```
// Input Parameters
var duration = 60*60*1000;
var endDateTime = new Date();
var startDateTime = new Date(endDateTime.getTime()-duration); // one hour ago
var region = [40,-125,35,-120];
var resolution = 2.5;

var dimLat = Math.ceil( Math.abs(region[0]-region[2]) / resolution);
var dimLon = Math.ceil(Math.abs(region[1]-region[3]) / resolution);

// Initialize MapReduce engine
var mr = new MapReduce();
mr.parallelMax = 4;
mr.init("http://www.mesonet.info/data/");

// Define columns from the mesonet data we need
mr.columns.push({ uri: "http://mesonet.info/id" });
mr.columns.push({ uri: "http://mesonet.info/lat" });
mr.columns.push({ uri: "http://mesonet.info/long" });
mr.columns.push({ uri: "http://mesonet.info/airTemperature",
                  unit: "http://qudt.org/vocab/unit#DegreeCelsius" });

// Setup the map, reduce, and completion tasks

// mapper produces grid cell entries by computing mean values
mr.mapper = function(data) {
   var cells = {};

   // Collect values for each observation into cells
   for (var i=0; i<data.length; i++) {
      if (data[i][3]===null || isNaN(data[i][3])) {
         continue;
      }
      var latitude = data[i][1];
      var longitude = data[i][2];
      var latOffset = Math.abs(mr.context.quadrangle[0] - latitude);
      var longOffset = Math.abs(mr.context.quadrangle[1] - longitude);
      var position = [ Math.floor(latOffset / resolution),
                       Math.floor(longOffset / resolution) ];
      if (dimLat==position[0]) {
         position[0]--;
      }
      if (dimLon==position[1]) {
         position[1]--;
      }
      var key = position[0]+","+position[1];

      var cell = cells[key];
      if (!cell) {
         cell = {
            position: position,
            values: []
         };
         cells[key] = cell;
      }

      cell.values.push(data[i][3]);
   }
```

```
   // Compute mean for values within a certain standard deviation
   for (var key in cells) {
      var cell = cells[key];

      cell.mean = cell.values.reduce(
         function(previous,current) { return previous+current}
      ) / cell.values.length;

      var sdev = Math.sqrt(
         cell.values.map(
            function(current) {
               var diff = current - cell.mean;
               return diff*diff;
            }
         ).reduce(
             function(previous,current) { return previous+current; }
         ) / cell.values.length);

      var tolerance = sdev<1 ? 3 : 3*sdev;
      cell.accepted = [];
      for (var i=0; i<cell.values.length; i++) {
         if (Math.abs(cell.mean - cell.values[i])<tolerance) {
            cell.accepted.push(cell.values[i]);
         }
      }
      cell.finalMean = cell.accepted.length==0 ?
         cell.mean :
         cell.accepted.reduce(
            function(previous,current) { return previous+current}
         ) / cell.accepted.length;
   }
   return cells;
}

// Reducer combines cell entries into a single grid
mr.reducer = function(data) {
   var grid = [];
   for (var i=0; i<dimLon; i++) {
      var row = [];
      grid.push(row);
      for (var j=0; j<dimLat; j++) {
         row[j] = { value: Number.NaN, means: [] };
      }
   }
   for (var i=0; i<data.length; i++) {
      for (var key in data[i]) {
         var cell = data[i][key];
         try {
            var gridCell = grid[cell.position[1]][cell.position[0]];
            gridCell.means.push(cell.finalMean);
            gridCell.value = gridCell.means.reduce(
               function(previous,current) { return previous+current}
            ) / gridCell.means.length;
         } catch (ex) {
            console.log("Cannot process grid cell "+key+" at position "+i);
            throw ex;
         }
      }
   }
   return grid;
}

// Perform map/reduce for quadrangles of size 2.5 degrees
mr.apply(region,startDateTime,endDateTime,2.5)
```

Figure C.7 Harvester Implementation

```
var mr = new MapReduce();
mr.init("http://www.mesonet.info/data/");
mr.columns.all = true;
harvester = new CombineHarvester(mr,
   {
      id: "http://mesonet.info/id",
      rowType: "http://mesonet.info/Report",
      groupType: "http://mesonet.info/Station",
      collectionType: "http://mesonet.info/Weather",
      columnType: "http://mesonet.info/Column",
      groups: "http://mesonet.info/stations",
      items: "http://mesonet.info/reports",
      title: "http://mesonet.info/title",
      datatype: "http://mesonet.info/datatype",
      quantity: "http://mesonet.info/quantity",
      multiple: "http://mesonet.info/multiple",
      unit: "http://mesonet.info/unit",
      symbol: "http://mesonet.info/symbol",
      value: "http://mesonet.info/value",
      valueSpace: "http://mesonet.info/valueSpace"
   },
   {
      "w" : "http://mesonet.info/" ,
      "unit" : "http://qudt.org/vocab/unit#",
      "quantity" : "http://qudt.org/vocab/quantity#"
   }
);
mr.mapper = harvester.getMapper();
mr.reducer = harvester.getReducer();
mr.apply(region,startDateTime,endDateTime,size);
```

Appendix D

Annotation Triples

Note that all the classes and properties in the ontology used are in the `http://pantabular.org/` namespace but their property values may take on structures from other ontologies (e.g., `http://schema.org/GeoShape` or QUDT units). All ontology references in this appendix should be assumed to belong to the PAN ontology unless prefixed and, when necessary for clarity or syntax, the `pan:` prefix will be used for the PAN ontology.

Figure D.1 Partition Summary Triples

```
@prefix pan: <http://pantabular.org/> .
<http://www.mesonet.info/data/q/36/2013-11-27T20:30:00Z> a pan:PartitionSummary;
 pan:range _:1;
 pan:count "24421";
_:1 a pan:FacetPartition;
 pan:start "2013-11-27T20:30:00Z";
 pan:end "2013-11-27T21:00:00Z";
 pan:length "PT30M" .
```

Figure D.2 Labeled Table Triples

```
@prefix pan: <http://pantabular.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:4 a pan:LabeledTable;
 pan:title "Weather Report Count by Quadrangle of size 36° by 36°";
 pan:entry _:5 .
_:5 a pan:Entry;
 pan:valueType xsd:int;
 pan:columnLabel <http://www.w3.org/2003/01/geo/wgs84_pos#lat>;
 pan:rowLabel <http://www.w3.org/2003/01/geo/wgs84_pos#long> .
```

Figure D.6 Partition Link Triples

```
@prefix pan: <http://pantabular.org/> .

<> pan:next <http://www.mesonet.info/data/q/5/n/840/2013-09-04T05:00:00Z>
<http://www.mesonet.info/data/q/5/n/840/2013-09-04T05:00:00Z> a pan:Partition;
  pan:range _:1 .
_:1 a pan:FacetPartition;
  pan:length "PT30M";
  pan:start "2013-09-04T05:00:00Z" .
```

Figure D.3 Labeled Table Row Triples

```
@prefix pan: <http://pantabular.org/> .
_:4 pan:label _:7, _:8, _:9, ..., _:17, ... .
_:7 a pan:ColumnLabel;
 pan:value "-180" .
_:8 a pan:ColumnLabel;
 pan:value "-144" .
_:9 a pan:ColumnLabel;
 pan:value "-108" .
...
_:17 a pan:RowLabel;
 pan:value "90" .
```

Figure D.4 Data Partition Triples

```
@prefix pan: <http://pantabular.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix schema: <http://schema.org/> .

<http://www.mesonet.info/data/q/36/n/28/2013-11-27T20:30:00Z> a pan:Partition;
 pan:count "31";
 pan:range _:1, _:2 .

_:1 a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#receivedTime>;
 pan:valueType xsd:dateTime;
 pan:start "2013-11-27T20:30:00Z";
 pan:end "2013-11-27T21:00:00Z";
 pan:length "PT30M" .

_:2 a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#latitude>,
           <http://www.mesonet.info/data/#longitude>;
 pan:shape _:3 .

_:3 a schema:GeoShape;
 schema:box "18 -108 -18 -108 -18 -72 18 -72" .
```

Figure D.5 Data Partition Table Triples

```
@prefix pan: <http://pantabular.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:21 a pan:Table;
 pan:column _:27, ..., _:43, _:45, ... .

_:27 a pan:Column;
 pan:title "Station";
 pan:property <http://mesonet.info/id>;
 pan:valueSpace _:28 .
_:28 a pan:ValueDescription;
 pan:datatype xsd:string;
 pan:quantity dc:identifier .

_:43 a pan:Column;
 pan:title "Temperature";
 pan:property <http://mesonet.info/airTemperature>;
 pan:valueSpace _:44 .
_:44 a pan:ValueDescription;
 pan:symbol "C";
 pan:datatype xsd:double;
 pan:quantity <http://qudt.org/vocab/quantity#ThermodynamicTemperature>;
 pan:unit <http://qudt.org/vocab/unit#DegreeCelsius> .

_:45 a pan:Column;
 pan:title "Humidity";
 pan:property <http://mesonet.info/airHumidity>;
 pan:valueSpace _:46 .
_:46 a pan:ValueDescription;
 pan:symbol "%";
 pan:datatype xsd:int;
 pan:quantity <http://qudt.org/vocab/quantity#AbsoluteHumidity>;
 pan:unit <http://qudt.org/vocab/unit#Percent> .
```

Figure D.7 Summary Link Triples

```
@prefix pan: <http://pantabular.org/> .
<http://www.mesonet.info/data/q/36/n/6/2013-09-04T04:30:00Z> a pan:Partition;
   pan:count "165" .
<http://www.mesonet.info/data/q/36/n/7/2013-09-04T04:30:00Z> a pan:Partition;
   pan:count "16" .
```

Figure D.8 Summary Row Triples from mesonet.info

```
@prefix pan: <http://pantabular.org/>
<_:data> pan:label <_:1> .
<_:1> a pan:RowLabel ; pan:value "40" .
<n/757/2013-10-29T21:00:00Z> a pan:Partition;
   pan:count "0" .
<n/758/2013-10-29T21:00:00Z> a pan:Partition;
   pan:count "0" .
...
<n/768/2013-10-29T21:00:00Z> a pan:Partition;
   pan:count "217" .
<n/769/2013-10-29T21:00:00Z> a pan:Partition;
   pan:count "80" .
<n/770/2013-10-29T21:00:00Z> a pan:Partition;
   pan:count "22" .
```

Figure D.9 Previous & Next Link Annotation Triples

```
@prefix pan: <http://pantabular.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.mesonet.info/data/q/36/2013-11-21T00:30:00Z>
 a pan:PartitionSummary;
 pan:range _:1;
 pan:count "12141";
 pan:previous <http://www.mesonet.info/data/q/36/2013-11-21T00:00:00Z>;
 pan:next <http://www.mesonet.info/data/q/36/2013-11-21T01:00:00Z>;
 pan:item _:4 .

_:1 a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#receivedTime>;
 pan:valueType xsd:dateTime;
 pan:start "2013-11-21T00:30:00Z";
 pan:end "2013-11-21T01:00:00Z";
 pan:length "PT30M" .

<http://www.mesonet.info/data/q/36/2013-11-21T00:00:00Z>
 a pan:PartitionSummary;
 pan:range _:2 .

_:2 a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#receivedTime>;
 pan:valueType xsd:dateTime;
 pan:length "PT30M";
 pan:start "2013-11-21T00:00:00Z" .

<http://www.mesonet.info/data/q/36/2013-11-21T01:00:00Z>
 a pan:PartitionSummary;
 pan:range _:3 .

_:3 a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#receivedTime>;
 pan:length "PT30M";
 pan:start "2013-11-21T01:00:00Z" .
```

Figure D.10 LabeledTable Annotation Triples

```
@prefix pan: <http://pantabular.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix schema: <http://schema.org/> .

_:4 a pan:LabeledTable;
 pan:title """Weather Report Count
by
Quadrangle of size 36° by 36°""";
 pan:entry _:5;
 pan:label _:7, _:8, _:9, _:10, _:11, _:12, _:13, _:14, _:15,
   _:16, _:17, _:18, _:19, _:20, _:21 .

_:5 a pan:Entry;
 pan:description "Weather Report Count";
 pan:valueType xsd:int;
 pan:summaryOf _:6;
 pan:linkType pan:Partition;
 pan:columnLabel <http://www.w3.org/2003/01/geo/wgs84_pos#lat>;
 pan:rowLabel <http://www.w3.org/2003/01/geo/wgs84_pos#long> .

_:6 a schema:GeoShape;
 schema:description "Quadrangle";
 schema:box "0 0 36 0 36 36 0 36" .

_:7 a pan:ColumnLabel;
 pan:value "-180" .
_:8 a pan:ColumnLabel;
 pan:value "-144" .
_:9 a pan:ColumnLabel;
 pan:value "-108" .
_:10 a pan:ColumnLabel;
 pan:value "-72" .
_:11 a pan:ColumnLabel;
 pan:value "-36" .
_:12 a pan:ColumnLabel;
 pan:value "0" .

...

_:17 a pan:RowLabel;
 pan:value "90" .

<http://www.mesonet.info/data/q/36/n/6/2013-11-21T00:30:00Z>
 a pan:Partition;
 pan:count "55" .
<http://www.mesonet.info/data/q/36/n/7/2013-11-21T00:30:00Z>
 a pan:Partition;
 pan:count "12" .
<http://www.mesonet.info/data/q/36/n/8/2013-11-21T00:30:00Z>
 a pan:Partition;
 pan:count "0" .
<http://www.mesonet.info/data/q/36/n/9/2013-11-21T00:30:00Z>
 a pan:Partition;
 pan:count "4" .
<http://www.mesonet.info/data/q/36/n/10/2013-11-21T00:30:00Z>
 a pan:Partition;
 pan:count "72" .
<http://www.mesonet.info/data/q/36/n/1/2013-11-21T00:30:00Z>
 a pan:Partition;
 pan:count "257" .

...
```

Figure D.11 Nearby Link Annotation Triples

```
@prefix pan: <http://pantabular.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix schema: <http://schema.org/> .

<http://www.mesonet.info/data/q/36/n/16/2013-11-21T01:00:00Z>
 a pan:Partition;
 pan:count "50";
 pan:range _:timePeriod, _:1;
 pan:previous <http://www.mesonet.info/data/q/36/n/16/2013-11-21T00:30:00Z>;
 pan:nearby <http://www.mesonet.info/data/q/36/n/5/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/6/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/7/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/15/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/17/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/25/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/26/2013-11-21T01:00:00Z>,
            <http://www.mesonet.info/data/q/36/n/27/2013-11-21T01:00:00Z>;
 pan:next <http://www.mesonet.info/data/q/36/n/16/2013-11-21T01:30:00Z>;
 pan:item _:21 .
<http://www.mesonet.info/data/#reports>
   pan:partition <http://www.mesonet.info/data/q/36/n/16/2013-11-21T01:00:00Z> .
_:timePeriod a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#receivedTime>;
 pan:valueType xsd:dateTime;
 pan:start "2013-11-21T01:00:00Z";
 pan:end "2013-11-21T01:30:00Z";
 pan:length "PT30M" .
_:1 a pan:FacetPartition;
 pan:facet <http://www.mesonet.info/data/#latitude>,
           <http://www.mesonet.info/data/#longitude>;
 pan:shape _:2 .
_:2 a schema:GeoShape;
 schema:box "54 180 18 180 18 -144 54 -144" .

<http://www.mesonet.info/data/q/36/n/5/2013-11-21T01:00:00Z> a pan:Partition;
 pan:range _:4, _:timePeriod .
_:4 a pan:FacetPartiton;
 pan:facet <http://www.mesonet.info/data/#latitude>,
           <http://www.mesonet.info/data/#longitude>;
 pan:shape _:5 .
_:5 a schema:GeoShape;
 schema:box "90 144 54 144 54 -180 90 -180" .
```

Figure D.12 Weather Report Triples

```
@prefix w: <http://mesonet.info/> .
@prefix unit: <http://qudt.org/vocab/unit#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:1 a w:Report;
 w:id "AD6QC"^^xsd:string;
 w:lat 37.983;
 w:long -122.59467;
 w:receivedAt "2013-11-11T22:09:48.000Z"^^xsd:dateTime;
 w:windDirection "0"^^xsd:int;
 w:windSpeed 0;
 w:airTemperature (_:2 _:3);
 w:airHumidity "80"^^xsd:int;
 w:airPressure "102010"^^xsd:int;
 w:rainLastHour 0;
 w:rainLastDay 0;
 w:rainSinceMidnight 0 .
_:2 w:value 61;
 w:unit unit:DegreeFahrenheit .
_:3 w:value 16.1;
 w:unit unit:DegreeCelsius .

...
```

Figure D.13 Navigation Triples

```
@prefix w: <http://mesonet.info/> .

_:2681 a w:Weather;
 w:reports (_:1 ... _:2676);
 w:stations (_:3218 ... _:4072) .
_:3218 a w:Station;
 w:id "AD6QC";
 w:reports _:3219 (_:1 _:6 _:11) .

...

_:4072 a w:Station;
 w:id "psychon";
 w:reports ( _:2676);
```

Figure D.14 Column Definition Triples

```
@prefix w: <http://mesonet.info/> .
@prefix unit: <http://qudt.org/vocab/unit#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dc: <http://purl.org/dc/terms/> .
@prefix quantity: <http://qudt.org/vocab/quantity#> .

w:id a w:Column;
 w:title "Station";
 w:valueSpace _:4394 .
_:4394 w:datatype xsd:string;
 w:quantity dc:identifier .
w:lat a w:Column;
 w:title "Latitude";
 w:valueSpace _:4395 .
_:4395 w:datatype xsd:double;
 w:quantity <http://www.w3.org/2003/01/geo/wgs84_pos#lat> .
w:long a w:Column;
 w:title "Longitude";
 w:valueSpace _:4396 .
_:4396 w:datatype xsd:double;
 w:quantity <http://www.w3.org/2003/01/geo/wgs84_pos#long> .

...

_:4401 a w:Column;
 w:title "Temperature";
 w:valueSpace _:4402 .
_:4402 w:datatype xsd:double;
 w:quantity quantity:ThermodynamicTemperature;
 w:symbol "F";
 w:unit unit:DegreeFahrenheit .
_:4403 a w:Column;
 w:title "Temperature";
 w:valueSpace _:4404 .
_:4404 w:datatype xsd:double;
 w:quantity quantity:ThermodynamicTemperature;
 w:symbol "C";
 w:unit unit:DegreeCelsius .
w:airTemperature w:multiple (_:4401 _:4403) .
w:airHumidity a w:Column;
 w:title "Humidity";
 w:valueSpace _:4407 .
_:4407 w:datatype xsd:int;
 w:quantity quantity:AbsoluteHumidity;
 w:symbol "%";
 w:unit unit:Percent .
```

Appendix E

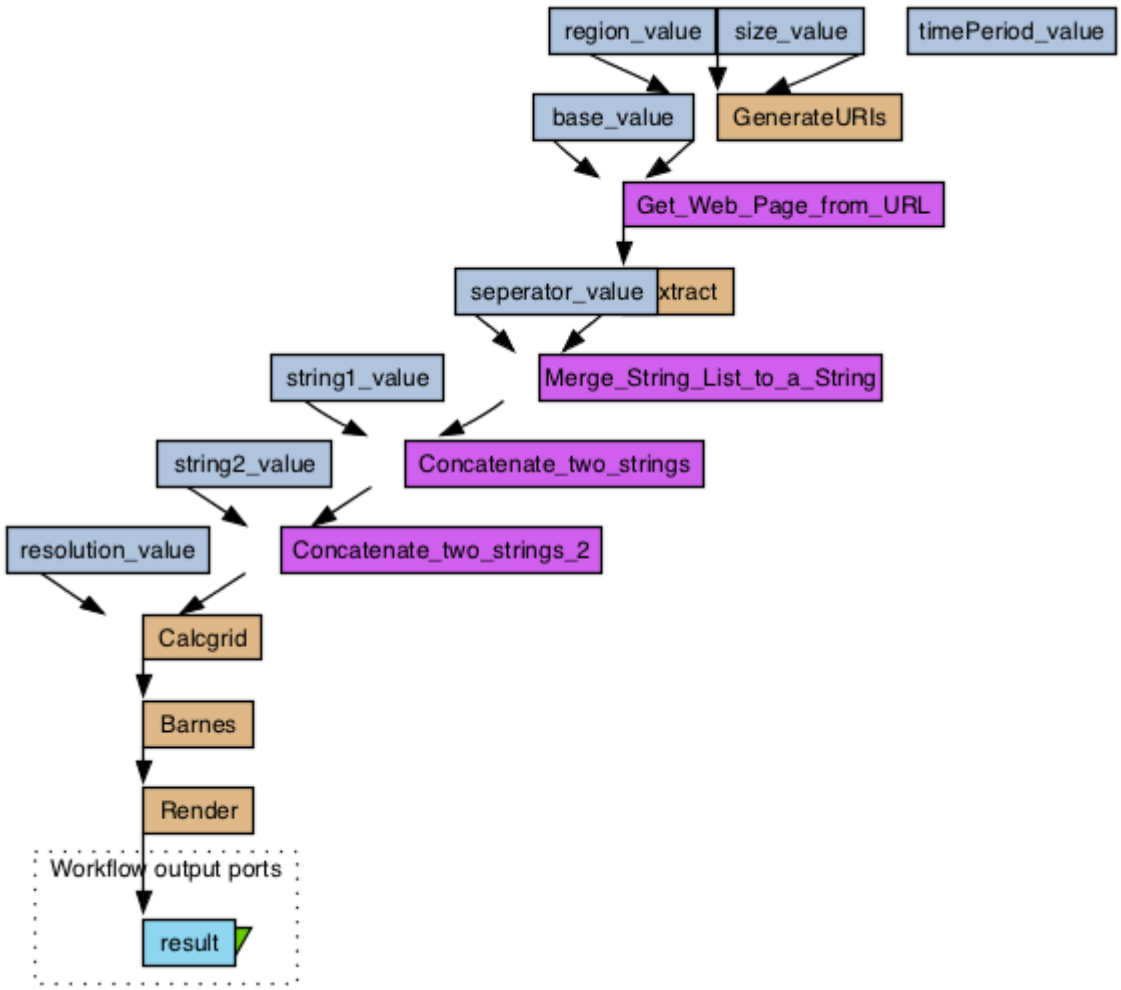
eScience Workflows

The workflow for Taverna that implements this process is shown in *Figure E.1, Taverna Interpolation Workflow*. Each of the steps, with the exception of fetching the Web resources, are implemented in BeanShell and colored in tan. The iteration of the sequence generated by the "Generate URIs" step is implicit and not shown in the diagram. Taverna knows from annotations that "Generate URIs" produces a list, applies the "Get_Web_Page_from_URL" and "Extract" steps to the list, and then merges the result back into a single string.

Since Taverna only passes strings between steps, the result is now a text string that contains a sequence of elements. A document element is wrapped around this string by concatenating a start and end tag and the resulting XML text is input to "Calcgrid" step, where the sparse grid is computed via a BeanShell script.

Finally, the Barnes Interpolation is computed via a BeanShell script using a supporting Java library (see *Figure F.2, Barnes Java Implementation*) written to support this workflow to produce the interpolated grid. The full set of values are provided as an XML document serialized to text and passed to the last step to render as an SVG document. Taverna has no ability to pass each document produced as a non-serialized DOM object even though all steps run within the same virtual machine environment.

Figure E.1 Taverna Interpolation Workflow

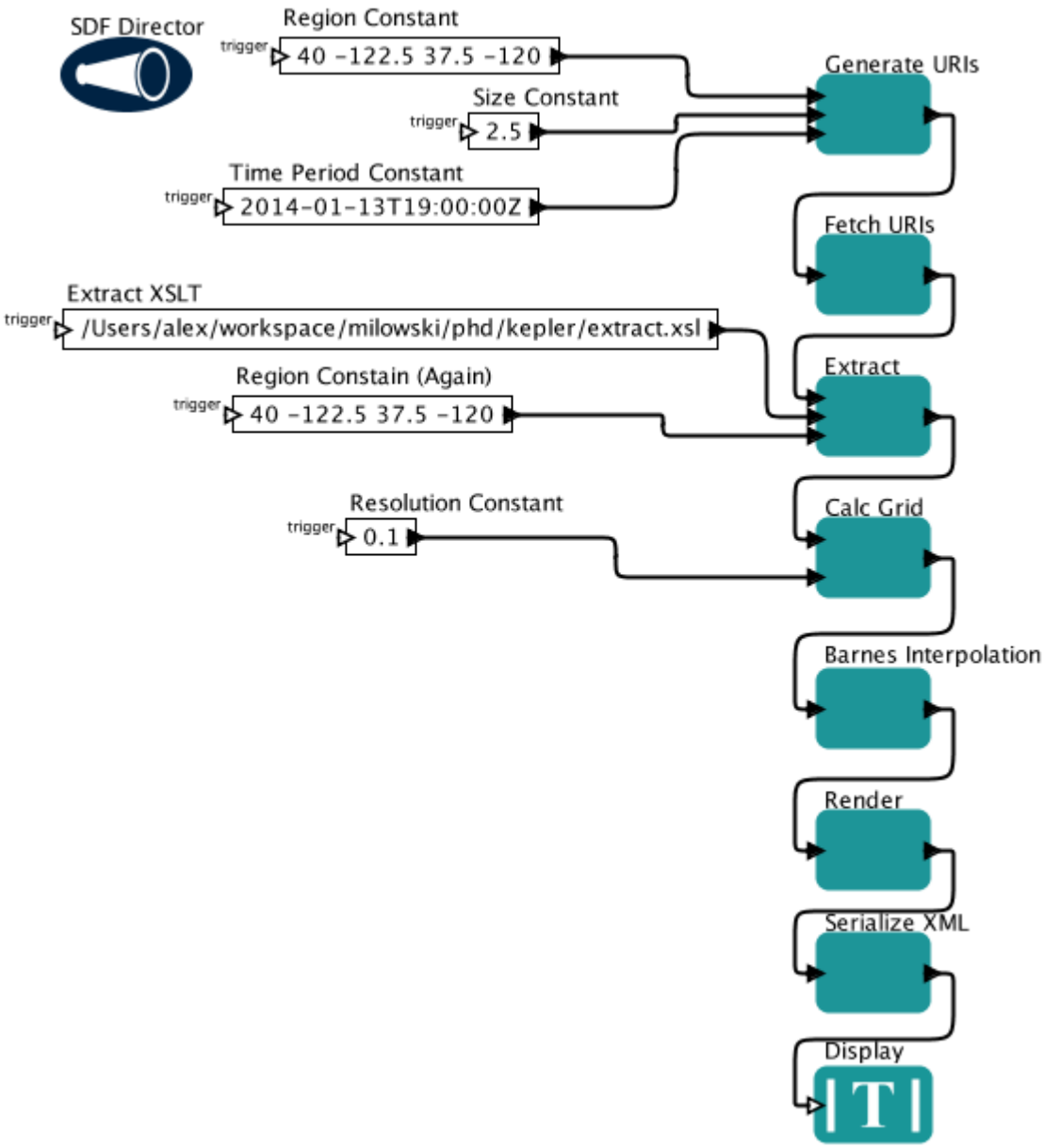


The workflow diagram, generated by Taverna via graphviz, has some rendering issues and arrows are misplaced.

The Kepler workflow is shown in *Figure E.2, Kepler Interpolation Workflow*, and is divided into a similar number of steps, but the steps are allow to produce non-string output. As such, a list of URIs is passed to the "fetch" step and the "extract" step processes a set of retrieved Web Resources into a single document. This workflow will use an increasing amount of memory as it does not support iteration and processes all the data in each step.

The output of the "Calc Grid" step is an in-memory document object that holds an XML representation of the observed values as a sparse grid. No serialization of this document is incurred when the "Barnes Interpolation" step receives the data. The interpolation process (see *Figure F.3, Barnes Python Implementation*) produces a grid of values (an array of arrays of numbers) that is passed directly to be rendered into SVG.

Figure E.2 Kepler Interpolation Workflow



Both workflows use the same XSLT 2.0 transform to extract the data from the Web pages via their RDFa annotations. In *Figure E.3, Extraction XSLT - Find the Table*, the table of data is found in the document by traversing the annotations. Once the partition and table are found, certain facts are found out about the partition being processed.

Afterwards, in *Figure E.4, Extraction XSLT - Processing the Table*, the table is processed to extract the column of data. In this particular case, the set of columns and the property label for temperature are fixed. Those particular choices could easily be parametrized to make the extraction more generic.

Figure E.3 Extraction XSLT - Find the Table

```
<xsl:template match="/">
   <xsl:apply-templates select="(//*[rdfa:is-type(.,'http://pantabular.org/Partition')])[1]"/>
</xsl:template>

<xsl:template match="*[rdfa:is-type(.,'http://pantabular.org/Partition')]">
   <xsl:variable name="shape" select="
      for $e in rdfa:element-with-property(.,'http://pantabular.org/range')
         return rdfa:element-with-property($e,'http://pantabular.org/shape')
      "/>
   <xsl:variable name="timePeriod" select="
      for $e in rdfa:element-with-property(.,'http://pantabular.org/range')
         return if (rdfa:element-with-property($e,'http://pantabular.org/start')) then $e else ()
      "/>
   <xsl:variable name="table" select="rdfa:element-with-property(.,'http://pantabular.org/item')"/>
   <xsl:variable name="box" select="
      tokenize(rdfa:element-with-property($shape,'http://schema.org/box'),'\s+')
      "/>
   <xsl:variable name="quad" select="($box[1], $box[2], $box[3], $box[6])"/>
   <xsl:variable name="start" select="
      rdfa:element-property-value(
         rdfa:element-with-property($timePeriod,'http://pantabular.org/start'))
      "/>
   <xsl:variable name="end" select="
      rdfa:element-property-value(
         rdfa:element-with-property($timePeriod,'http://pantabular.org/end'))
      "/>
   <data quad="{$quad}" start="{$start}" end="{$end}">
      <xsl:apply-templates select="$table"/>
   </data>
</xsl:template>
```

Finally, in both workflows, an SVG document is generated. The output is fixed to 500 x 500 pixel image that is divided into rectangles based on the input region. Each rectangle is color coded on a scale of -10° to 37° C by first mapping to 350 to 750 nanometer wavelengths of light and then translating that value into an RGB value. This produces the characteristic image of deep

Figure E.4 Extraction XSLT - Processing the Table

```
<xsl:template match="h:table">
   <xsl:variable name="indices" select="
      for $c in rdfa:element-with-property(.,'http://pantabular.org/column')
         return
            if (rdfa:element-property-value(
                   rdfa:element-with-property($c,'http://pantabular.org/property')) =
                'http://mesonet.info/id')
            then concat('id=',count($c/preceding-sibling::*)+1,' ')
            else if (rdfa:element-property-value(
                       rdfa:element-with-property($c,'http://pantabular.org/property')) =
                  'http://mesonet.info/lat')
            then concat('lat=',count($c/preceding-sibling::*)+1,' ')
            else if (rdfa:element-property-value(
                       rdfa:element-with-property($c,'http://pantabular.org/property')) =
                   'http://mesonet.info/long')
            then concat('lon=',count($c/preceding-sibling::*)+1,' ')
            else if (rdfa:element-property-value(
                       rdfa:element-with-property($c,'http://pantabular.org/property')) =
                    'http://mesonet.info/airTemperature')
            then if (rdfa:element-property-value(
                       rdfa:element-with-property(
                         rdfa:element-with-property($c,'http://pantabular.org/valueSpace'),
                          'http://pantabular.org/unit'))='http://qudt.org/vocab/unit#DegreeCelsius')
               then concat('T=',count($c/preceding-sibling::*)+1,' ')
               else ()
            else ()
      "/>
   <xsl:variable name="sindices" select="concat(string-join($indices,' '),' ')"/>
   <xsl:variable name="idIndex"
                 select="number(substring-before(substring-after($sindices,'id='),' '))"/>
   <xsl:variable name="latIndex"
                 select="number(substring-before(substring-after($sindices,'lat='),' '))"/>
   <xsl:variable name="lonIndex"
                 select="number(substring-before(substring-after($sindices,'lon='),' '))"/>
   <xsl:variable name="TIndex"
        select="number(substring-before(substring-after(string-join($sindices,' '),'T='),' '))"/>
   <xsl:apply-templates select="h:tbody">
      <xsl:with-param name="idIndex" select="$idIndex" tunnel="yes"/>
      <xsl:with-param name="latIndex" select="$latIndex" tunnel="yes"/>
      <xsl:with-param name="lonIndex" select="$lonIndex" tunnel="yes"/>
      <xsl:with-param name="TIndex" select="$TIndex" tunnel="yes"/>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="h:tbody">
   <xsl:param name="idIndex" tunnel="yes"/>
   <xsl:param name="latIndex" tunnel="yes"/>
   <xsl:param name="lonIndex" tunnel="yes"/>
   <xsl:param name="TIndex" tunnel="yes"/>
   <observations id="{h:tr[1]/h:td[$idIndex]}"
                 lat="{h:tr[1]/h:td[$latIndex]}" lon="{h:tr[1]/h:td[$lonIndex]}">
      <xsl:value-of select="string-join(h:tr/h:td[$TIndex],' ')"/>
   </observations>
</xsl:template>
```

blue/purple hues for cold areas and reds hues for hot areas. The rendering process is quickly executed in both workflows via Python or BeanShell.

Appendix F

Barnes Interpolation Implementation

The Barnes Interpolation algorithm was implemented in JavaScript (see *Figure F.1, Barnes JavaScript Implementation*), Java (see *Figure F.2, Barnes Java Implementation*), and Python (see *Figure F.3, Barnes Python Implementation*). The algorithm is the same in all three implementations except for slight differences in the choices of inputs. Both the Java and Python implementations take a list of observed values while the JavaScript implementation requires a full grid containing empty cells associated with NaN (not a number) values.

The process is divided into three sections:

1. Generate a grid, the same size as the output, whose values are the distanced from the grid entry (0,0). This will be used to retrieve the distance between any two grid cells. Distance calculations use the Haversine formula for pairs of latitude and longitude coordinates with radius of the earth of 6378.1 km.

2. A complete grid is populated with initial values where each weight is computed relative to each observed value.

3. A preset number of additional computation passes are computed for the interpolated value. If the grid cell for each observed value is within a preset difference (error tolerance), the process will terminate early.

Figure F.1 Barnes JavaScript Implementation

```
var clat = grid[0][0].position[0];
var clon = grid[0][0].position[1];
var observed = [];
var distances = [];
for (var i=0; i<grid.length; i++) {
   for (var j=0; j<grid[i].length; j++) {
      distances.push([]);
      if (!isNaN(grid[i][j].value)) {
         observed.push({ position: [ i, j], value: grid[i][j].value });
      }
      distances[i].push(this.D(clat,clon,grid[i][j].position[0],grid[i][j].position[1]));
   }
}

// pass 0: initial weighted sum
for (var i=0; i<grid.length; i++) {
   for (var j=0; j<grid[i].length; j++) {
      var osum = 0;
      var wsum = 0;
      for (var o=0; o<observed.length; o++) {
         var oi = observed[o].position[0];
         var oj = observed[o].position[1]
         var x = i<oi ? oi-i : i-oi;
         var y = j<oj ? oj-j : j-oj;
         var d = distances[x][y];
         var w = Math.exp(-1*Math.pow(d/this.R,2)/this.C[0]);
         osum += w*observed[o].value;
         wsum += w;
      }
      grid[i][j].e = osum/wsum;
   }
}
// pass 1 through N
var converged = false;
for (var pass=1; !converged && pass<this.limit; pass++) {
   var cindex = pass>=this.C.length ? this.C.length-1 : pass;
   // compute subsequent estimated values from errors from past value
   for (var i=0; i<grid.length; i++) {
      for (var j=0; j<grid[i].length; j++) {
         var osum = 0;
         var wsum = 0;
         for (var o=0; o<observed.length; o++) {
            var oi = observed[o].position[0];
            var oj = observed[o].position[1]
            var x = i<oi ? oi-i : i-oi;
            var y = j<oj ? oj-j : j-oj;
            var d = distances[x][y];
            var w = Math.exp(-1*Math.pow(d/this.R,2)/this.C[cindex]);
            osum += w*(observed[o].value - grid[i][j].e);
            wsum += w;
         }
         grid[i][j].e = grid[i][j].e + osum/wsum;
      }
   }
   // check for convergence of observed values within the tolerance
   var nearenough = true;
   for (var o=0; o<observed.length; o++) {
      if (Math.abs(observed[o].value-grid[observed[o].position[0]][observed[o].position[1]].value) >
          this.tolerance) {
         nearenough = false;
      }
   }
   converged = nearenough;
}
```

Figure F.2 Barnes Java Implementation

```
double [][] grid = new double[gridDimX][gridDimY];
double [][] distances = new double[gridDimX][gridDimY];
for (int x=0; x<distances.length; x++) {
   for (int y=0; y<distances[x].length; y++) {
      double gx = gridPosX + x*gridSizeX;
      double gy = gridPosY + y*gridSizeY;
      double d = this.D(gridPosX,gridPosY,gx,gy);
      distances[x][y] = d;
   }
}

// pass 0: initial weighted sum
for (int i=0; i<grid.length; i++) {
   for (int j=0; j<grid[i].length; j++) {
      double osum = 0;
      double wsum = 0;
      for (int o=0; o<observations.length; o++) {
         int [] pos = observations[o].getPosition();
         int x = i<pos[0] ? pos[0]-i : i-pos[0];
         int y = j<pos[1] ? pos[1]-j : j-pos[1];
         double d = distances[x][y];
         double w = Math.exp(-1*Math.pow(d/this.R,2)/this.C[0]);
         osum += w*observations[o].getValue();
         wsum += w;
      }
      grid[i][j] = osum/wsum;
   }
}

// pass 1 through N
boolean converged = false;
for (int pass=1; !converged && pass<this.passLimit; pass++) {
   int cindex = pass<this.C.length ? pass : this.C.length-1;
   // compute subsequent estimated values from errors from past value
   for (int i=0; i<grid.length; i++) {
      for (int j=0; j<grid[i].length; j++) {
         double osum = 0;
         double wsum = 0;
         for (int o=0; o<observations.length; o++) {
            int [] pos = observations[o].getPosition();
            int x = i<pos[0] ? pos[0]-i : i-pos[0];
            int y = j<pos[1] ? pos[1]-j : j-pos[1];
            double d = distances[x][y];
            double w = Math.exp(-1*Math.pow(d/this.R,2)/this.C[cindex]);
            osum += w*(observations[o].getValue() - grid[i][j]);
            wsum += w;
         }
         grid[i][j] = grid[i][j] + osum/wsum;
      }
   }
   // check for convergence of observed values within the tolerance
   boolean nearenough = true;
   for (int o=0; o<observations.length; o++) {
      int [] pos = observations[o].getPosition();
      if (Math.abs(observations[o].getValue()-grid[pos[0]][pos[1]])>this.tolerance) {
         nearenough = false;
      }
   }
   converged = nearenough;
}
```

Figure F.3 Barnes Python Implementation

```
distances = []
for x in range(0,gridDimX):
   distances.append([])
   for y in range(0,gridDimY):
      gx = gridPosX + x*gridSizeX
      gy = gridPosY + y*gridSizeY
      d = self.D(gridPosX,gridPosY,gx,gy)
      distances[x].append(d)

# pass 0: initial weighted sum and create grid
grid = [];
for i in range(0,gridDimX):
   grid.append([])
   for j in range(0,gridDimY):
      osum = 0.0
      wsum = 0.0
      for o in range(0,len(observations)):
         pos = observations[o]["pos"]
         x = pos[0]-i if i<pos[0] else i-pos[0]
         y = pos[1]-j if j<pos[1] else j-pos[1]
         d = distances[x][y]
         w = math.exp(-1*math.pow(d/self.R,2)/self.C[0])
         osum += w*observations[o]["value"]
         wsum += w
      grid[i].append(osum/wsum)

# pass 1 through N
for p in range(1,self.passLimit):

   cindex = p if p<len(self.C) else len(self.C)-1

   # compute subsequent estimated values from errors from past value
   for i in range(0,gridDimX):
      for j in range(0,gridDimY):
         osum = 0.0
         wsum = 0.0
         for o in range(0,len(observations)):
            pos = observations[o]["pos"]
            x = pos[0]-i if i<pos[0] else i-pos[0]
            y = pos[1]-j if j<pos[1] else j-pos[1]
            d = distances[x][y]
            w = math.exp(-1*math.pow(d/self.R,2)/self.C[cindex])
            osum += w*(observations[o]["value"] - grid[i][j])
            wsum += w
         grid[i][j] = grid[i][j] + osum/wsum

   # check for convergence of observed values within the tolerance
   nearenough = True;
   for o in range(0,len(observations)):
      pos = observations[o]["pos"]
      if math.fabs(observations[o]["value"]-grid[pos[0]][pos[1]])>self.tolerance:
         nearenough = False
         break
   if nearenough:
      break;
```

Bibliography

[1] "*Size of Wikipedia*", 2013-06, Wikipedia ; see also http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

[2] "*Deep Web*", 2012-05, Wikipedia ; see also http://en.wikipedia.org/wiki/Deep_web

[3] "*Architecture of the World Wide Web, Volume One*", 2004-12, W3C Ian Jacobs and Norman Walsh ; see also http://www.w3.org/TR/webarch/

[4] "*XML in the Browser: the Next Decade*", R. Alexander Miłowski, Balisage: The Markup Conference, August 11 - 14, 2009, Montreal, Canada, vol 3, 2009, doi:10.4242/BalisageVol3.Milowski01 ; see also http://www.balisage.net/Proceedings/vol3/epub/Milowski01/BalisageVol3-Milowski01.epub

[5] "*Document Object Model (DOM) Level 2 Core Specification*", 2000-11-13, Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne ; see also http://www.w3.org/TR/DOM-Level-2-Core/

[6] "*WikiProject RNA*" ; see also http://en.wikipedia.org/wiki/Wikipedia:RNA

[7] "*The RNA WikiProject: Community annotation of RNA families*", doi:10.1261/rna.1200508 ; see also http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2590952/

[8] "*The application/json Media Type for JavaScript Object Notation (JSON)*", D. Crockford, 2006-07 ; see also http://www.ietf.org/rfc/rfc4627.txt

[9] "*Esri Profile of the Content Standard for Digital Geospatial Metadata*", 2003-03, ESRI ; see also http://www.esri.com/metadata/esriprof80.html

[10] "*ESRI Shapefile Technical Description*", 1998-07 ESRI ; see also http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[11] "*International Virtual Observatory Alliance*" ; see also http://www.ivoa.net/

[12] "*List of File Formats*" ; see also http://en.wikipedia.org/wiki/List_of_file_formats

[13] "*FITS*", P. Grøsbøl, R.H. Harten, E.W. Greisen, and D.C. Wells, A&AS, vol. 73, pp. 359 ; see also http://fits.gsfc.nasa.gov/

[14] "*Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*", 2011-06-07, W3C, Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie ; see also http://www.w3.org/TR/CSS2/

[15] "*XSL Transformations (XSLT) Version 2.0*", 2007-01-23, W3C, Michael Kay ; see also http://www.w3.org/TR/xslt20/

[16] "*The Semantic Web*", Tim Berners-Lee, James Hendler, and Ora Lassila, Scientific American Magazine, 2001-05-17 ; see also http://www.scientificamerican.com/article.cfm?id=the-semantic-web

[17] "*The VizieR System for Accessing Astronomical Data*", F. Ochsenbein, ASP Conference, vol. 145, pp. 387, 1998

[18] "*The VizieR database of astronomical catalogues*", F. Ochsenbein, P. Bauer, and J. Marcout, Astronomy & Astrophysics Supplement Series, vol. 143, pp. 23-32, 2000-04-01

[19] "*Extensible Markup Language (XML) 1.0 (First Edition)*", 1998-02-10, W3C, Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen ; see also http://www.w3.org/TR/1998/REC-xml-19980210

[20] "*Using XML for Accessing Resource in Astronomy*", F. Ochsenbein, M. Albrecht, A. Brighton, P. Fernique, D. Guillaume, R. Hanisch, E. Shaya, and A. Wicenec, Astronomical Data Analysis Software and Systems IX; ASP Conference Series, vol. 216, 2000

[21] "*ESO/CDS Data-mining Tool Development Project*", Patrcio F. Ortiz, François Ochsenbein, Andreas Wicenec, and Miguel Albrecht, Astronomical Data Analysis Software and Systems VIII; ASP Conference Series, vol. 172, 1999

[22] "*UCD in the IVOA context*", Sébastien Derriere, Norman Gray, Jonathan McDowell, Robert Mann, François Ochsenbein, Pedro Osuna, Andrea Preite Martinez, Guy Rixon, and Roy Williams, Astronomical Data Analysis Software and Systems XIII; ASP Conference Series, vol. 314, 2004

[23] "*VOTable Format Description, Version 1.2*", 2009-11-30, IVOA, Francois Ochsenbein and Roy Williams ; see also http://www.ivoa.net/documents/VOTable/20091130/

[24] "*AstroGrid opens for science*", Nic Walton, Astronomy & Geophysics, Oxford Journals , 2005, vol. 46, no. 3, pp. 3.23-3.26, doi:10.1111/j.1468-4004.2005.46323.x ; see also http://astrogeo.oxfordjournals.org/content/46/3/3.23.full

[25] "*Simple Cone Search, Version 1.03*", 2008-02-22, IVOA, Roy Williams, Robert Hanisch, Alex Szalay, and Raymond Plante ; see also http://www.ivoa.net/Documents/latest/ConeSearch.html

[26] "*The USNO-B Catalog*", David G. Monet et. al., The Astronomical Journal, vol. 125, pp. 984-993, 2003-02 ; see also http://www.usno.navy.mil/USNO/astrometry/optical-IR-prod/icas/2003.AJ.125.984M.pdf

[27] "*An IVOA Standard for Unified Content Descriptors, Version 1.1*", 2005-08-12, IVOA, Sébastien Derriere, Norman Gray, Robert Mann, Andrea Preite Martinez, Jonathan McDowell, Thomas Mc Glynn, François Ochsenbein, Pedro Osuna, Guy Rixon, and Roy Williams ; see also http://www.ivoa.net/documents/REC/UCD/UCD-20050812.html

[28] "*The UCD1+ controlled vocabulary, Version 1.23*" 2007-04-02, IVOA, Andrea Preite Martinez, Sebastien Derriere, Nausicaa Delmotte, Norman Gray, Robert Mann, Jonathan McDowell, Thomas Mc Glynn, François Ochsenbein, Pedro Osuna, Guy Rixon, and Roy Williams ; see also http://www.ivoa.net/documents/REC/UCD/UCDlist-20070402.html

[29] "*Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey*", Chris J. Lintott, Kevin Schawinski, Anže Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg, Monthly Notices of the Royal Astronomical Society, vol. Volume 389, no. 3, pp. 1179-1189, 2008-09, DOI:10.1111/j.1365-2966.2008.13689.x

[30] "*Galaxy Zoo 1 : Data Release of Morphological Classifications for nearly 900,000 galaxies*", Chris J. Lintott, Kevin Schawinski, Anže Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, Jan Vandenberg, Monthly Notices of the Royal Astronomical Society vol. 410, no. 1, pp. 166-178, 2010, MNRAS (2011) 410 (1): 166-178. doi: 10.1111/j.1365-2966.2010.17432.x

[31] "*The Sixth Data Release of the Sloan Digital Sky Survey*", Jennifer K. Adelman-McCarthy, Marcel A. Agüeros, Sahar S. Allam, et al. The Astrophysical Journal Supplement Series, vol. 175 (2008-04-01), pp. 297-313, doi:10.1086/524984

[32] "*The Seventh Data Release of the Sloan Digital Sky Survey*", bazajian, Kevork N, Adelman-McCarthy, Jennifer K., Agüeros, Marcel A., Allam, Sahar S., Allende Prieto, Carlos, An, Deokkeun, Anderson, Kurt S. J., and et al. The Astrophysical Journal Supplement, vol. 182, no. 2, pp. 543-558 2009-06 doi:10.1088/0067-0049/182/2/543 ; see also http://iopscience.iop.org/0067-0049/182/2/543/pdf/apjs_182_2_543.pdf

[33] "*The Zooniverse*", K. D. Borne, L. Fortson, P. Gay, C. Lintott, M. J. Raddick, and J. Wallin, American Geophysical Union Fall Meeting Abstract, 2009-12, pp. C7+

[34] "*OGC® KML, version 2.2.0*", Tim Wilson, Open Geospatial Consortium, 2008-04-14, OGC 07-147r2

[35] "*OpenGIS® Web Map Server Implementation Specification*", Jeff de la Beaujardiere, Open Geospatial Consortium Inc., 2006-03-15, OGC 06-042, edition 1.3.0 ; see also http://portal.opengeospatial.org/files/?artifact_id=14416

[36] "*OpenGIS® Geography Markup Language (GML) Encoding Standard*", Clemens Portele, Open Geospatial Consortium Inc., 2007-08-27, OGC 07-036, edition 3.2.1 ; see also http://portal.opengeospatial.org/files/?artifact_id=20509

[37] "*XML Schema Part 1: Structures Second Edition*", W3C, 2004-10-28, edition 1.0, Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn ; see also http://www.w3.org/TR/xmlschema-1/

[38] "*Observations and Measurements - XML Implementation*", The Open Goespatial Consortium, 2011-03-22, OGC 10-025r1, Simon Cox ; see also http://www.opengis.net/doc/IS/OMXML/2.0

[39] "*OGC Reference Model*", The Open Goespatial Consortium, 2011-12-19, OGC 08-062r7 ; see also http://www.opengis.net/doc/orm/2.1

[40] "*Keyhole, Inc.*" ; see also http://en.wikipedia.org/wiki/Keyhole,_Inc

[41] "*OGC Approves KML as an Open Standard*" ; see also http://www.opengeospatial.org/node/857

[42] "*The GeoJSON Format Specification*", edition 1.0, 2008-06-16, Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt ; see also http://geojson.org/geojson-spec.html

[43] "*USGS GeoJSON Summary Format*" ; see also http://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php

[44] "*NODC Standard Format Marine Mammals of Coastal Alaska Data (1975-1981): Marine Mammal Specimens (F025) (NODC Accession 0014150)*", 1975-04-22, US National Oceanographic Data Center ; see also http://www.nodc.noaa.gov/cgi-bin/OAS/prd/accession/0014150

[45] "*Marine mammal specimen and other data from the SURVEYOR and other platforms as part of the Outer Continental Shelf Environmental Assessment Program (OCSEAP) from 20 March 1977 to 02 November 1977 (NODC Accession 7900319)*", 1979-11-08, Alaska State Department of Fish and Game - Anchorage ; see also http://www.nodc.noaa.gov/cgi-bin/OAS/prd/accession/7900319

[46] "*The Self-Describing Web*", W3C, 2009-02-09, Noah Mendelsohn ; see also http://www.w3.org/2001/tag/doc/selfDescribingDocuments.html

[47] "*iNaturalist.org: A Community for Naturalists*", Ken-ichi Ueda and Scott Loarie ; see also http://www.inaturalist.org/

[48] "*The Atom Syndication Format*", IETF, 2005-12, M. Nottingham and R. Sayre ; see also https://tools.ietf.org/html/rfc4287

[49] "*The future of applications: W3C TAG perspectives*", Henry S. Thompson, School of Informatics, University of Edinburgh, 2011-03-28, W3C Technical Architecture Group ; see also http://www.w3.org/2001/tag/doc/IAB_Prague_2011_slides.html

[50] "*Information Management: A Proposal*", Tim Berners-Lee, CERN, 1989-03 ; see also http://www.w3.org/History/1989/proposal.html

[51] "*The birth of the web*" ; see also http://home.web.cern.ch/about/birth-web

[52] "*Technical Details*", Tim Berners-Lee ; see also http://info.cern.ch/hypertext/WWW/Technical.html

[53] "*IANA Media Types Registry*", Internet Assigned Numbers Authority ; see also http://www.iana.org/assignments/media-types

[54] "*RDF Semantics*", W3C, 2004-02-10, Patrick Hayes and Brian McBride ; see also http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[55] "*REST APIs must be hypertext-driven*", Roy T. Fielding, 2008-10-20 ; see also http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[56] "*Hypermedia as the Engine of Application State (HATEOAS)*" ; see also http://en.wikipedia.org/wiki/HATEOAS

[57] "*RDF 1.1 Primer*", W3C, 2014-02-25, Frank Manola, Eric Miller, and Brian McBride ; see also http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/

[58] "*Microformats: The Next (Small) Thing on the Semantic Web?*", Rohit Khare, IEEE Internet Computing, vol. 10, no. 1, pp. 68-75 2006-01, doi:10.1109/MIC.2006.13 ; see also http://www.computer.org/csdl/mags/ic/2006/01/w1068-abs.html

[59] "*RDFa in XHTML: Syntax and Processing; A collection of attributes and processing rules for extending XHTML to support RDF*", W3C, 2008-10-14, Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton ; see also http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/

[60] "*HTML Microdata*", W3C, 2013-10-29, Ian Hickson ; see also http://www.w3.org/TR/2013/NOTE-microdata-20131029/

[61] "*XHTML and RDF*", W3C, 2004-02-14, Mark Birbeck ; see also http://www.w3.org/MarkUp/2004/02/xhtml-rdf.html

[62] "*RDFa Core 1.1*", W3C, 2012-06-07, Ben Adida, Mark Birbeck, Shane McCarron, and Ivan Herman ; see also http://www.w3.org/TR/rdfa-core/

[63] "*RDFa Lite 1.1*", W3C, 2012-06-07, Manu Sporny ; see also http://www.w3.org/TR/rdfa-lite/

[64] "*Turtle: Terse RDF Triple Language*", W3C, 2013-02-13, David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers ; see also http://www.w3.org/TR/turtle/

[65] "*Linked Data*", 2006-07-27, Tim Berners-Lee ; see also http://www.w3.org/DesignIssues/LinkedData.html

[66] "*Linked Data Platform 1.0*", W3C, 2013-07-13, Steve Speicher, John Arwe, and Ashok Malhotra ; see also http://www.w3.org/TR/ldp/

[67] "*XHTML™ 1.1 - Module-based XHTML - Second Edition*" W3C, 2010-11-23, Shane McCarron, Masayasu Ishikawa, and Murray Altheim ; see also http://www.w3.org/TR/xhtml11/

[68] "*HTML 4.01 Specification*", W3C, 1999-12-24, Dave Raggett, Arnaud Le Hors, and Ian Jacobs ; see also http://www.w3.org/TR/REC-html40/

[69] "*HTML5*", W3C, 2013-09-06, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer, and Ian Hickson ; see also http://www.w3.org/TR/html/

[70] "*QUDT - Quantities, Units, Dimensions and Data Types in OWL and XML*", NASA, 2011-09-11, Ralph Hodgson and Paul J. Keller ; see also http://www.qudt.org/

[71] "*The use of Metadata in URIs*", W3C, 2006-12-04, Noah Mendelsohn and Stuart Williams ; see also http://www.w3.org/2001/tag/doc/metaDataInURI-31-20061204.html

[72] "*Cool URIs don't change*", W3C ; see also http://www.w3.org/Provider/Style/URI.html

[73] "*RFC 5013: The Dublin Core Metadata Element Set*", IETF, 2007-08, J. Kunze and T. Baker ; see also http://www.ietf.org/rfc/rfc5013.txt

[74] "*Describing Linked Datasets with the VoID Vocabulary*", W3C, 2011-03-03, Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao ; see also http://www.w3.org/TR/void/

[75] "*The RDF Data Cube Vocabulary*", W3C, 2014-01-16, Richard Cyganiak, Dave Reynolds, and Jeni Tennison ; see also http://www.w3.org/TR/vocab-data-cube/

[76] "*Basic Geo (WGS84 lat/long) Vocabulary*", W3C, 2003-01, Dan Brickley ; see also http://www.w3.org/2003/01/geo/

[77] "*Citizen Weather Observation Program*" ; see also http://www.wxqa.com/

[78] "*Automatic Packet Reporting System-Internet Service*", Bob Bruninga ; see also http://www.aprs-is.net/

[79] "*Automatic Position Reporting System; APRS Protocol Reference, Protocol Version 1.0*", 2000-08-29, Ian Wade ; see also http://www.aprs.org/doc/APRS101.PDF

[80] "*MarkLogic*" ; see also http://developer.marklogic.com/inside-marklogic

[81] "*XProc: An XML Pipeline Language*", W3C, 2010-05-11, Norman Walsh, Alex Miłowski, and Henry S. Thompson ; see also http://www.w3.org/TR/xproc/

[82] "*XQuery 3.0: An XML Query Language*", W3C, 2013-10-22, Jonathan Robie, Don Chamberlin, Michael Dyck, and John Snelson ; see also http://www.w3.org/TR/xquery-30/

[83] "*D3 Data-Driven Documents*", Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer, IEEE Transactions on Visualization and Computer Graphics, vol. 17 no. 12, December 2011 pp. 2301-2309 doi:10.1109/TVCG.2011.185

[84] "*Leaflet*", Vladimir Agafonkin ; see also http://leafletjs.com

[85] "*OpenStreetMap: User-Generated Street Maps*", M. Haklay and P. Weber, Pervasive Computing, vol. 7, no. 4, Oct.-Dec. 2008, pp. 12-18, doi:10.1109/MPRV.2008.80

[86] "*Document Object Model (DOM) Level 2 HTML Specification; version 1.0*", W3C, 2003-01-09, Johnny Stenback, Philippe Le Hégaret, and Arnaud Le Hors ; see also http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/

[87] "*RDFa API*", W3C, 2012-07-5, Nathan Rixham, Mark Birbeck, and Ivan Herman ; see also http://www.w3.org/TR/rdfa-api/

[88] "*Green Turtle*", R. Alexander Miłowski ; see also https://code.google.com/p/green-turtle/

[89] "*Cross-Origin Resource Sharing*", W3C, 2013-01-29, Anne van Kesteren ; see also http://www.w3.org/TR/cors/

[90] "*Local Knowledge for In Situ Services*", R. Alexander Miłowski and Henry S. Thompson, XML Prague 2013, 2013-02-09 ; see also http://archive.xmlprague.cz/2013/files/xmlprague-2013-proceedings.epub

[91] "*MapReduce: Simplified Data Processing on Large Clusters*", Jeffrey Dean and Sanjay Ghemawat, OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004-12 ; see also http://research.google.com/archive/mapreduce-osdi04.pdf

[92] "*ECMAScript Language Specification, 5th Edition*", ECMA International, 2011-06 ; see also http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[93] "*A Technique for Maximizing Details in Numerical Weather Map Analysis*", Stanley L. Barnes, Journal of Applied Meteorology, American Meteorological Society, vol. 3, no. 4, pp. 396-409, 1964-08-01 ; see also http://dx.doi.org/10.1175/1520-0450(1964)003<0396:ATFMDI>2.0.CO;2

[94] "*Barnes Analysis for Surface Interpolation*", Martin Davis ; see also http://lin-ear-th-inking.blogspot.com/2012/02/barnes-analysis-for-surface.html

[95] "*Scientific workflow systems - can one size fit all?*", V. Curcin and M. Ghanem, Biomedical Engineering Conference, Cairo International, 18-20 Dec. 2008 doi:10.1109/CIBEC.2008.4786077 ; see also http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4786077&isnumber=4786031

[96] "*The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud*", Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble, Nucleic Acids Research, 2013-05-02, doi:10.1093/nar/gkt328 ; see also http://nar.oxfordjournals.org/content/early/2013/05/02/nar.gkt328

[97] "*Kepler: an extensible system for design and execution of scientific workflows*", I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, 16th International Conference on Scientific and Statistical Database Management, 2004 pp. 423-424

[98] "*XSL Transformations (XSLT) Version 1.0*", James Clark, W3C, 1999-11-16 ; see also http://www.w3.org/TR/xslt

[99] "*SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*", Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon, 2007-04-27, W3C ; see also http://www.w3.org/TR/soap12-part1/

[100] "*Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation*", Roy Thomas Fielding, 2000, University of California, Irvine ; see also http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[101] "*XSL Transformations (XSLT) Version 2.0*", W3C, 2007-01-23, Michael Kay ; see also http://www.w3.org/TR/xslt20/

[102] "*BeanShell*", Pat Niemeyer ; see also http://beanshell.org/

[103] "*JPython*", Jim Hugunin, Barry Warsaw, Samuele Pedroni, Brian Zimmer, and Frank Wierzbicki. ; see also http://jpython.org/

[104] "*JSR 274: The BeanShell Scripting Language*", Patrick Niemeyer ; see also https://jcp.org/en/jsr/detail?id=274

[105] "*IVOA Data Access Layer Interface; Version 1.0*", IVOA, 2013-09-19, Patrick Dowler, Markus Demleitner, Mark Taylor, and Doug Tody ; see also http://www.ivoa.net/documents/DALI/20130919/PR-DALI-1.0-20130919.html

[106] "*The OAuth 2.0 Authorization Framework; RFC 6749*", IETF, 10-2012, D. Hardt ; see also http://tools.ietf.org/html/rfc6749

[107] "*The Pompeii Bibliography and Mapping Resource*", Eric Poehler, 2013 ; see also https://www.academia.edu/2493873/Pompeii_Bibliography_and_Mapping_Project

Colophon

This dissertation was encoded in XML using Docbook 5 schemas as a single book. Within the Docbook XML, MathML was used for any mathematics and SVG was primarily used for diagrams.

Authoring was performed using the Oxygen XML Editor from Syncro Soft. Original diagrams were created in the iDraw software from Indeeo. Transformations were performed via the Calabash XProc implementation written by Norman Walsh. The print output was produced by Prince XML.

The print production was performed by an XProc pipeline that transforms the document in several stages. The Docbook markup is first transformed into HTML that preserves some of the structure through a combination of HTML elements and RDFa annotations. A subsequent transformation readies that document for print. Finally, the Prince formatter is applied to produce PDF output via a variety of CSS stylesheets.