DYNAMIC INTEGRATION OF BIOGEOGRAPHIC AND PHYSICAL OCEANOGRAPHIC DATA IN THE GULF OF MAINE

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

UNIVERSITY OF SOUTHERN MAINE

BY

RICHARD ANTHONY FRANKS JR.

2008
We hereby recommend that the thesis of Richard Anthony Franks Jr. entitled *Dynamic Integration of Biogeographic and Oceanographic Data within the Gulf of Maine* be accepted in partial fulfillment of the requirements for the Degree of Master of Computer Science.

Committee

Dr. Bruce MacLeod, Faculty Mentor  
Department of Computer Science

Professor David Briggs  
Department of Computer Science

Dr. Lewis Incze  
Department of Environmental Science

Accepted

Dean John R. Wright,  
School of Applied Science, Engineering and Technology
ABSTRACT    Research into the diversity of species that populate the Gulf of Maine and how changes in physical conditions may influence their distribution and abundance is urgently needed in order to manage and maintain the Gulf of Maine ecosystem. Our primary goal is to develop a general, dynamic software system that facilitates the integration of physical and biological data, each having spatial and temporal components. The culmination of our work is a computational mechanism that allows researchers to explore relationships between physical and biological data and formulate hypotheses about the interactions between the two. Data values from varying time intervals can be integrated over various temporal granularities and in a variety of ways using the system. With this tool the researcher can map biological and oceanographic data in a revealing display, all through a convenient, web-based interface. The system is based on a data mediator that supports a set of data translators and facilitates integration of multiple sources of physical and biological data into our system, including simulation model output data, other Web Feature Service (WFS) sources, shapefiles and data from other spatially enabled database management systems (DBMSs).
## Contents

1 Introduction 1

2 Background 5

2.1 Literature Review .................................................. 5

2.1.1 Ecosystem Approaches to Management (EAM) ............... 6

2.1.2 Geographic Information Systems and Spatio-Temporal Data Integration ..................................................... 9

2.1.3 Software Design Patterns ......................................... 12

2.2 Technical Landscape ............................................... 19

2.2.1 Emerging Open Source Software (OSS) ....................... 19

2.2.2 Emerging Open Standards ....................................... 24

2.2.3 Systems and Software Libraries Supporting Internet Mapping 34

3 A Case Study 39

3.1 Troubles of Commercial Fishing with Spiny Dogfish (*Squalus acanthias*) 39

3.2 Exploring the Correlation Between Spiny Dogfish and Bottom Temperature ................................................................. 40

3.2.1 Focusing Our Exploration ........................................ 40

3.2.2 Exploring the Interrelatedness of Spiny Dogfish (*Squalus acanthias*) Observations and Predicted Bottom Temperature ................................................................. 44
3.2.3  Obtaining the Integrated Biogeographic and Oceanographic Data  51

4  Overall Architecture  55
   4.1  Key Technical Challenges ...........................................  56
   4.2  Overall Architecture ..................................................  59
      4.2.1  Client-Side Architecture Components .........................  59
      4.2.2  Server-Side Architecture Components .........................  60

5  Technical Challenge I:  
   Extensibility  65
      5.1  Axes for Change ...................................................  65
      5.2  Extensibility Solution ............................................  67

6  Technical Challenge II:  
   Multiple Data Source Support  71
      6.1  Interoperability Issues ...........................................  71
      6.2  Advantages of Interoperability ..................................  73
      6.3  Multiple Data Source Support Solution ..........................  73

7  Technical Challenge III:  
   Data Integration and Delivery  81
      7.1  Data Integration ...................................................  81
         7.1.1  Conceptual Model: Point-in-Polygon ........................  81
         7.1.2  Data Integration Solution ..................................  83
      7.2  Data Delivery Solution ............................................  86

8  Technical Challenge IV:  
   User Interface  91
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The COMMAND Pattern</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>The GATEWAY Pattern</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>The FACTORY Pattern</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>The MEDIATOR Pattern</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>The SINGLETON Pattern</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>The TEMPLATE METHOD Pattern</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>Open Geospatial Consortium (OGC) Open Web Services</td>
<td>26</td>
</tr>
<tr>
<td>2.8</td>
<td>An OGC Web Mapping Service (WMS)</td>
<td>27</td>
</tr>
<tr>
<td>2.9</td>
<td>A Sample OGC WMS Request</td>
<td>28</td>
</tr>
<tr>
<td>2.10</td>
<td>A Sample OGC WMS Response</td>
<td>29</td>
</tr>
<tr>
<td>2.11</td>
<td>An OGC Web Feature Service (WFS)</td>
<td>31</td>
</tr>
<tr>
<td>2.12</td>
<td>A Sample OGC WFS Request</td>
<td>33</td>
</tr>
<tr>
<td>2.13</td>
<td>A Sample OGC WFS Response</td>
<td>33</td>
</tr>
<tr>
<td>2.14</td>
<td>OBIS-SEAMAP</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>The Spiny Dogfish (<em>Squalus acanthias</em>)</td>
<td>40</td>
</tr>
<tr>
<td>3.2</td>
<td>Adding All Spiny Dogfish (<em>Squalus acanthias</em>) Observations from the NMFS Bottom Trawl Survey</td>
<td>41</td>
</tr>
<tr>
<td>3.3</td>
<td>Downloading Spiny Dogfish (<em>Squalus acanthias</em>) Observations from the NMFS Bottom Trawl Survey</td>
<td>42</td>
</tr>
</tbody>
</table>
5.2 The LayerManagerGUI Lib Template Class ......................... 70
6.1 DataStore Singletons and Gateway Objects ....................... 75
6.2 Data Mediator and Data Translator Objects ...................... 76
6.3 Adding Data from an External Source Activity Diagram ........... 78
7.1 Database Schema for Biological Data ............................ 83
7.2 Database Schema for Physical Data ............................... 84
7.3 System Methodology for Data Integration ......................... 85
7.4 Download Activity Diagram ....................................... 87
7.5 DownloadCommand Class Diagram ................................. 88
8.1 Adding Layers to the Map Overlay Environment ................... 95
8.2 Inset of Figure 8.1 (1) ........................................... 96
8.3 Inset of Figure 8.1 (2) ........................................... 97
8.4 Map Showing Both Positive and Zero Observations ............... 98
8.5 User Interface for Data Integration ............................. 99
8.6 Features for Managing Biogeographic Layers .................. 100
8.7 Features for Managing Oceanographic Layers .................. 101
8.8 System Start Page ............................................. 102
8.9 Viewing Layer Information within the Layer Manager .......... 103
8.10 Changing Species Colors and Visibility within the Layer Manager ... 104
8.11 Adding Physical Layers to the Layer Manager ................... 105
8.12 Zooming with the System’s Web Mapping Client ............... 106
8.13 Previewing External Data in the System ......................... 107
8.14 Downloading Layers from the Layer Manager ................... 107
9.1 Sample Documentation Generated by JavaDoc Tool ............... 113
Chapter 1

Introduction

I. Introduction

This work fits into a larger body of research currently underway at the Aquatic
Systems group at the University of Southern Maine. The Census of Marine Life
(CoML) is an international undertaking, consisting of the efforts of researchers in more
the 80 nations whose main goal is to ascertain and explain the diversity, distribution,
and abundance of marine life. CoML is supported by government agencies, private
foundations and companies concerned with science, the environment, and fisheries.
CoML is a global effort to compile the first comprehensive census of ocean life, which
is scheduled to be released in October 2010. The main work that will be reported
in the census is conducted by seventeen projects around the globe covering six ocean
realms.

The Gulf of Maine Area Program (GoMA) is one of seven initial field projects of
CoML, and was selected as its ecosystem pilot study. The Gulf of Maine and Georges
Bank compose one of the planet’s most fertile and productive marine environments.
GoMA’s novel contribution to the CoML effort will be the assembly of data part-
nerships, public support, data systems and scientific knowledge into a framework for facilitating ecosystem approaches to management (EAM) within the Gulf of Maine. In order to accomplish this, GoMA will advance knowledge of biodiversity patterns and ecological processes over a variety of habitats and species, ranging from microscopic plankton to whales. This growing knowledge base will be synthesized and made available to the scientific community, public and managers as a foundation for EAM within the Gulf of Maine.

The Gulf of Maine Biogeographic Information System (GMBIS), the data component of GoMA, is charged with supporting regional needs for access to biogeographic, physical, chemical and geological data, and with providing a mechanism for improving the understanding of biological patterns, their interrelatedness to oceanographic parameters and their changes through time. GMBIS aims to function as an “aggregation server” that provides access, rapid visualization and integration facilities and data download capabilities.

Scientific accounts of real world features recognize two basic forms: objects and phenomena (Lo and Yeung 2002). Objects can be thought of as discrete and definite, such as a single point observation of where a lobster was caught. Phenomena tend to be distributed continuously over large areas, for example sea surface temperature. Spatial information consists of duplicating the real world in digital form by collecting information about geographic features and their physical location on earth (Lo and Yeung 2002). A feature is geographic when it has at least one property whose value is a simple geometry. Simple geometries are those in which coordinates are defined in two dimensions where the traditional 0, 1 and 2-dimensional geometries defined are represented by points, line strings and polygons. Spatial information can also have a temporal, or “when”, component to it.
The Gulf of Maine and Georges Bank form one of the world’s most productive areas in the global ocean (Incze 2008). Massive amounts of data are being produced through the numerous research projects and monitoring programs, including the GoMA Program and the Gulf of Maine Ocean Observing System (GoMOOS). The data providers have come together in the Gulf of Maine Ocean Data Partnership (GoMODP) in order to share their data with each other and the public. As a result, the Gulf of Maine is a rich area of scientific discovery, but the multiple, disparate sources of data introduce interoperability issues.

One such problematic issue is that of varying temporal resolution between different data sets. Currently, data provided by GoMOOS and the data provided by GMBIS are not compatible due to differences in temporal resolution. GoMOOS provides mostly physical oceanographic data, collected hourly to daily, as do most ocean observing systems. One important source of biological data that GMBIS relies upon is the resource assessment surveys that are conducted from research vessels within the Gulf of Maine once or twice a year. The integration of biogeographic and physical oceanographic data demonstrates a need for both averaged and aggregated oceanographic data.

Finally, GoMA’s aim of providing a framework for EAM relies on the ability to integrate biogeographic and oceanographic data. The current computational platform does not support this type of integration. Integrated data within the Gulf of Maine are not available for marine researchers, scientists or policy makers alike. As a result, these individuals must integrate and spatially join the data themselves using full-blown, heavyweight desktop Geographic Information Systems (GISs), which usually require a large capital investment and trained staff. Because of this, many interested parties are unable to obtain this highly desirable integrated data. Ideally, marine
researchers and decision-makers should be able to download data already integrated according to their specifications, in a manageable format. There is a great need for a system that provides them with the capability to integrate and filter data for scientific research, resource management or both.
Chapter 2

Background

2.1 Literature Review

Although historically one of the most productive fishing areas in the world, the Gulf of Maine and Georges Bank have been heavily impacted by intense human use (Incze 2008). The study of how changes in physical conditions threaten the diversity of species that populate the Gulf of Maine is urgently needed in order to maintain, manage and preserve the Gulf of Maine ecosystem. There is now broad consensus that human activities must be pursued in ways that conserve biodiversity through a more comprehensive account of the functioning of the ecosystem in more holistic ways (Incze 2008). The Gulf of Maine program lists its goals as increasing knowledge of the patterns and roles of biodiversity in the Gulf of Maine area and constructing a framework that enables EAM. The goal of our project is to develop a tool that provides access to the types of integrated data needed to facilitate EAM. This following literature review and survey of the technical landscape serve to establish a set of requirements for an EAM-based tool.
2.1.1 Ecosystem Approaches to Management (EAM)

One definition of EAM is the one that is employed by the United States’ National Oceanic and Atmospheric Administration (NOAA). According to NOAA, EAM provides a comprehensive framework for marine and coastal resource decision making and, in contrast to individual species or single issue management, considers a wider range of relevant ecological, environmental, and human factors bearing on societal choices regarding resource use (Murawski 2007). This suggests that a data integration tool that supports EAM should be capable of incorporating data from multiple species, along with relevant environmental or other data. EAM is: (1) geographically specified, (2) adaptive in its development over time as new information becomes available or as circumstances change, (3) takes into account ecosystem knowledge and uncertainties, (4) recognizes that multiple simultaneous factors may influence the outcomes of management (particularly those external to the system), and (5) strives to balance diverse societal objectives that result from resource decision making and allocation. Furthermore, the process of implementing EAM needs to be (6) incremental and (7) collaborative (Sissenwine and Murawski 2004). Some additional EAM principles (Murawski 2007) are that:

- the ecosystem approach should be undertaken at the appropriate spatial and temporal scales,
- recognizing the varying temporal scales and lag-effects that characterize ecosystem processes, objectives for ecosystem management should be set for the long term,
- the ecosystem approach should consider all forms of relevant information, including scientific, indigenous and local knowledge, innovations and practices, and
• the ecosystem approach should involve all relevant sectors of society and scientific disciplines.

What do these definitions and principles regarding EAM imply about the needs of a data integration tool that supports EAM? First, the tool must be able to filter data by various changing spatial extents. The tool should also be capable of displaying, or mapping, the data within the spatial extent of interest in an intuitive manner. Second, if EAM is adaptive in its development as new information becomes available over time, then the tool should support the incorporation of newly acquired data from previously existing or possibly new institutions. The tool should be flexible enough to incorporate new data sources, regardless of format.

Ecosystem perspectives require information about the relationships among ecosystem components as a basis for informing policy choices (Murawski and Matlock 2006). The important new focus for science supporting EAM is in data integration across traditional disciplines at appropriate geographic scales, and in understanding feedbacks and interactions among abiotic, biotic and human parts of the ecosystem (Murawski 2007). As a comprehensive framework, EAM is supported by relevant biological, oceanographic, economic and social information appropriate to the problem set being managed. Since EAM takes into account ecosystem knowledge, a data integration tool for EAM must provide an intuitive method for integrating ecosystem knowledge. It must also support the integration of such data, not only visually in a map, but also into a format that permits further analysis. This is a key requirement of the tool since an important aspect of the science supporting EAM is that it builds upon existing institutions and information collected for a variety of specific purposes, and provides a framework to combine these data in ways that add additional value (Murawski 2007). This aspect of EAM demands that an integration tool supporting it must be capable
of aggregating data from multiple institutions and data sources.

Creating an environment supportive of EAM requires participation by many scientists, managers and public stakeholders, and the open exchange of information and ideas among them[Incze 2008, Murawski 2007]. The increasing importance of the broad, cooperative stakeholder engagement aspects of EAM is becoming paramount (Murawski 2007). It also requires enhanced databases and information systems to facilitate access to information and stimulate its use and integration for research, public discourse and management (Incze 2008). This implies that the tool should act as a mechanism for incorporating broader perspectives in decision making. A logical choice of medium for the tool would be the Internet, due to its broad user base. Another implication is that the tool should require very little training, making it available to the broadest audience possible. The tool should also lack use restrictions and should be open and available to those parties interested in using such a tool. It should also be a mechanism for generating and investigating hypotheses put forth by the various stakeholders involved in EAM.

There exist two major arguments against the implementation of EAM. They are that (1) the lack of scientific readiness to support it and (2) there is insufficient information for any area currently available to answer all the ecosystem questions necessary to support EAM (Murawski 2007). Since adaptive management approaches incorporate new information as it becomes available, a tenet of EAM, then there is already enough data to drive EAM implementation. With the amount of data currently available and the fact that more data continue to be made available, what is really lacking are tools that allow for the type of data integration and exploration that is necessary for EAM.
2.1.2 Geographic Information Systems and Spatio-Temporal Data Integration

There are several reasons why we should strive for integrating GIS and spatial and temporal process models. There comes a stage in the development of science when issues mature and one needs to be more realistic with the data sets, to try out ideas, to explore from a new perspective[(Nyerges 1993),(Goodchild, Parks, and Steyaert 1993)]. In the marine realm, there is a need to explain biodiversity of species and taxonomic groups within a specific ecosystem by correlating their distribution with the spatial and temporal distribution of physical phenomena. This knowledge of biodiversity and its interrelatedness to other physical parameters can then be used in EAM. By creating groupings of physical and biological parameters that correlate with the distribution and abundance of species, distribution and habitat prediction models may also emerge. Environmental modeling can place some of the heaviest demands on GIS because of its need to handle many kinds of data in space and time in a more dynamic way(Nyerges 1993). In developing this tool, we must consider several dimensions of GIS use. These include(Nyerges 1993):

- type of task (resource inventory, assessment, management, development, etc.),
- application area (environmental, socioeconomic, etc),
- level of decision (policy, management, operations)
- spatial extent of the problem (small, medium or large study area size), and
- type of organization (public, private or not-for profit).

Environmental tasks include those for inventorying, assessing, managing and predicting the fate of environmental resources[(Nyerges 1993),(Goodchild, Parks, and Steyaert 1993)]. Environmental inventorying is an accounting of the state of a re-
source environment—what exists and what does not exist in particular areas (Nyerges 1993). This can be accomplished with a GIS at different levels of spatial and temporal resolution (Nyerges 1993). This suggests that an EAM application needs to provide the ability to vary spatial and temporal resolution when exploring resource—in our case, marine species-distribution. An assessment can be used to determine what has been lost due to environmental influences [(Nyerges 1993), (Goodchild, Parks, and Steyaert 1993)]. The relationship between what exists and what no longer exists is a difference in two maps over time (Nyerges 1993). This implies that an EAM support system must allow the user to toggle back and forth between different distribution and physical parameter maps.

Another dimension of use, decision making level, must be factored into the design and implementation of our integration tool. The largest group of GIS users is those that understand the underlying principles of GIS and its jargon [(Nyerges 1993), (Lo and Yeung 2002)]. Our application should reach the greatest number of stakeholders. To do this, we need to provide GIS principles and functionality to the user without bombarding them with the GIS jargon that goes along with it. This will allow our system to appeal to a greater decision making level, from scientists and policy managers to the general public.

Our application should also address the spatial extent of use and the types of organizations it will pertain to. Our tool should be applicable to any spatial extent, be it small, medium or large. The tool should be readily available to all stakeholders, including those from private, public and not-for-profit organizations. This implies that our tool should be open and have minimal cost.
Three primary modes of GIS use can be identified: map, query and model (Nyerges 1993). The map mode provides referential information for when a user wishes to see an overview of a spatial realm and needs to get a sense of what is there. This is key for marine researchers and policy managers [(Incze 2008), (Murawski 2007)]. In order to foster EAM, our tool needs to allow users to see, visually in a map, what is or is not in the ecosystem of interest and where they are or are not located.

A second mode, query mode, is used to address user requests for location-based information and usually involves the questions of “what” and “where” (Nyerges 1993). Our tool should not only display “what” and “where”, but also allow users to filter by “when” and “how much” in queries concerning geographic information. This allows for the accommodation of the temporal component of the data.

The third mode of GIS use, model mode, concerns providing data to a model and takes into account the interactivity of parameters, the sensitivity of those parameters and produces a result (Nyerges 1993). This is beyond the scope of our tool.

Some key analysis and manipulation functions needed to support the integration of spatial and temporal data are vector overlay operators (boolean AND, OR, XOR, NOT for point in polygon, point on line, line in polygon and polygon in polygon), spatial data selection (retrieving data by location), and joining (merging attribute information of two attributes associated with the same area and temporal resolution) (Nyerges 1993). Our tool needs to provide these functions in order to provide the user with the GIS functions needed for data integration and exploration. The tool must balance this powerful GIS functionality with simplicity of use in order to be accessible by all types of users.
2.1.3 Software Design Patterns

The COMMAND Pattern

In object-oriented software design, the COMMAND pattern is a design pattern in which objects are used to represent actions (Gamma, Helm, Johnson, and Vlissides 1994). Figure 2.1 illustrates the COMMAND pattern and all of the participants involved. In its simplest form, the COMMAND pattern employs a simple interface having a single method, the `Execute()` method (Martin 2003).

COMMAND Pattern Participants

- Command declares an interface for executing an operation,
ConcreteCommand defines a binding between a Receiver object and an action and implements Execute by invoking the corresponding operation(s) on Receiver,

Client creates a ConcreteCommand object, passing parameters to it as an object or Map, and may set its receiver,

Invoker asks the command instance to carry out the request,

Concrete Command may carry out the request itself, or elect to pass the request to a Receiver

Receiver(optional) - performs the operations associated with carrying out the request

The benefits that we can expect to gain from using the COMMAND pattern are (1) encapsulated actions and parameters (in the form of an object or Map), (2) a useful abstraction for building generic components and (3) the atomicity of transactional behavior.
The **GATEWAY** Pattern

The Gateway is an object that encapsulates access to an external system or resource (Fowler 2004). The Gateway object wraps the special API code, like that of the JDBC API for example, into a class whose interface is the method calls of a simple object. This object translates its simple method calls, used by other objects, into the appropriate specialized API. Other objects, like our PostGISDataStoreSingleton, can access the resource through this Gateway.

**GATEWAY** Pattern Participants

- External Resource usually has specialized API code or complex means of connection and interaction,
- Gateway wraps the specialized API code needed to connect to the external resource and provides simple methods for accessing the external resource,
- Gateway Clients rely on the Gateway to connect with, export to and retrieve...
The FACTORY Pattern

The FACTORY Pattern lets a class defer instantiation to subclasses (Gamma, Helm, Johnson, and Vlissides 1994). FACTORY objects are common in toolkits and frameworks where library code needs to create objects of types which may be subclassed by applications using the framework.

FACTORY Pattern Participants

- Product defines the interface of objects the factory method creates
- ConcreteProduct implements the Product interface
- Factory declares the creation method, which returns an object of type Product,
- ConcreteFactory overrides the creation method to return an instance of a ConcreteProduct.
Figure 2.4: The MEDIATOR Pattern

The MEDIATOR Pattern

We use the MEDIATOR pattern in our solution to the interoperability challenge. The MEDIATOR pattern defines an object that encapsulates the interactions between a set of objects (Gamma, Helm, Johnson, and Vlissides 1994). It keeps objects from communicating directly, promoting loose-coupling between the interacting objects. The interactions between objects can be altered by simply modifying the MEDIATOR independently of the objects that it mediates. We have added a client to the MEDIATOR pattern to represent the client-side application of the system, which is one of the “objects” communicating with the mediator.

MEDIATOR Pattern Participants

- Mediator defines an interface for communicating with Colleague objects (objects that wish to communicate with one another),
- ConcreteMediator implements cooperative behavior by coordinating communi-
• Colleagues know their Mediators and each one communicates with its mediator whenever it needs to communicate with another colleague.

The **SINGLETON** Pattern

A **SINGLETON** object is an object that is used to restrict instantiation of a class to a single object at most (Gamma, Helm, Johnson, and Vlissides 1994). This is useful when exactly one object is needed across the system or when the system will operate more efficiently when only one object exists. The **SINGLETON** pattern is implemented by creating a class with a method that creates a new instance of the class if one does not exist and returns a reference to the instance otherwise. In order to ensure that the **SINGLETON** object’s instance cannot be instantiated any other way, its constructor is made private.

There are two main differences between a **SINGLETON** and a simple static instance of a class. First, although a singleton can be implemented as a static instance, it can also be lazily constructed. This means that the **SINGLETON** will not consume memory or resources until the **SINGLETON** is needed. Second, static member classes...
cannot implement an interface. If the static member class needs to realize a contract expressed by an interface, it must be implemented as a SINGLETON.

SINGLETON Pattern Participants

• Singleton defines a static `getInstance()` method that provides access to the Singleton’s unique instance to requesting client objects. Singleton is obligated to create and maintain its unique instance.

![Diagram of Singleton Pattern](image)

Figure 2.6: The TEMPLATE METHOD Pattern

The TEMPLATE METHOD Pattern

The TEMPLATE METHOD pattern defines the algorithm skeleton, laying out the necessary operations required by the program (Gamma, Helm, Johnson, and Vlissides 1994). This algorithm skeleton is made abstract and subclasses are created to override the abstract methods, providing concrete behavior.

TEMPLATE METHOD Pattern Participants

18
AbstractClass defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm,

ConcreteClass implements the primitive operations to carry out subclass-specific steps of the algorithm.

The TEMPLATE METHOD pattern is implemented by creating a class that provides the basic steps of an algorithm using abstract methods. Subclasses are then created that change the abstract methods to realize real actions. The benefits gained by using the TEMPLATE METHOD pattern are (1) our general algorithm is saved in one place avoiding duplication in the code, (2) concrete steps may be changed by the subclasses and (3) the template method in a parent class controls the overall process by calling subclass methods as required. This allows the system to accept new components that change the behavior of the system without breaking it.

2.2 Technical Landscape

2.2.1 Emerging Open Source Software (OSS)

In recent years, interest in open source has grown dramatically (Pan and Bonk 2007). Given recently difficult monetary times, there is keen interest in higher education and corporate training related to the use of OSS. The open source model of software development, which provides an approach to software development that is different from the conventional model of software development, has attracted attention from an increasingly diverse audience. As a result, the number of OSS projects have grown in leaps and bounds within the past few years. Some successful examples of OSS are the Linux operating system, the Apache Tomcat and Apache httpd and PostgreSQL, to mention only a few.
OSS refers to software that is “of or relating to source code that is available to the public” either partially or in whole((Pan and Bonk 2007), (Mockus, Fielding, and Herbsleb 2002)). OSS products differ from commercial ones in that they are “created by a development community rather than a single vendor” and that the users are individuals working independently or affiliated with participating organizations to rewrite the source code((Pan and Bonk 2007), (Mockus, Fielding, and Herbsleb 2002)). OSS development relies on the following tenets[(Pan and Bonk 2007), (Johnson 2005)]

1. **Free Redistribution** Copies of the software can be made at no cost.
2. **Source Code** The source code must be distributed with the original work, as well as all derived works.
3. **Derived Works** Modifications are allowed; however, it is not required that the derived work be subject to the same license terms as the original work.
4. **Integrity of the Author’s Source Code** Modifications to the original work may be restricted only if the distribution of patches is allowed. Derived works may be required to carry a different name or version number from the original software.
5. **No Discrimination Against Persons or Groups** Discrimination against any person or group of persons is not allowed.
6. **No Discrimination Against Fields of Endeavor** Restrictions preventing use of the software by a certain business or area of research are not allowed.
7. **Distribution of License** Any terms should apply automatically without written authorization.
8. **License Must Not Be Specific to a Product** Rights attached to a program must not depend on that program being part of a specific software distribution.
9. **License Must Not Contaminate Other Software** Restrictions on other software distributed with the licensed software are not allowed.
There are various advantages of open source. One of the advantages of OSS is that it motivates innovation. Original source code provides a base for the receivers to begin with while frequent discussion of code improvement results in idea exchanges essential to innovation (Pan and Bonk 2007). Another advantage is that open source makes available the talent of the world ((Johnson 2005), (Wheeler 2004)). Another advantage is that open source reduces the cost of system development by capitalizing on the talents of the community of developers who are developing it. The tenets of OSS echo some of those of EAM. Both encourage community involvement and build on the knowledge base of interested parties. Since some of those involved in the efforts of CoML and EAM may have limited funding for software development, the reduced cost of OSS development is of added interest.

It is only recently that the open source software (OSS) components required to build an EAM data integration tool have been developed. These emerging open-source, open-standards tools provide a functional, open working environment for the ocean biogeographic information community (Best, Coyne, and Halpin 2004). According to Best, Coyne and Halpin, standardized, open approaches will allow researchers to better share innovations. They go on to outline several reasons to utilize these emerging OSS modules to build ocean biogeographic information systems. They cite that open source software:

- is economical,
- is customizable,
- allows for open and rapid development,
- is standards based,
- encourages community involvement.
They also assert that the greatest advantage is that open source software facilitates open development and community sharing of interoperable tools. They go on to discuss the open source tools that have arisen and that are needed to build ocean biogeographic information systems.

**DBMSs and Geodatabases**

CoML is a global effort and marine research is extremely expensive. Open source, freeware database software may be the only option for non-profits and organizations in developing countries who cannot afford expensive enterprise database systems (Best, Coyne, and Halpin 2004). Several open source DBMSs have been recently developed, most notably PostgreSQL. It is of particular interest because it has a spatial extension, PostGIS. This extension spatially-enables the PostgreSQL DBMS allowing it to serve as a geodatabase and provide spatial querying capabilities.

**Internet Mapping Servers and Clients**

In order to build a fully functional EAM data integration tool, we need a component that is capable of serving geographic information via the web. We also need to build new or rely upon pre-existing frameworks to implement a thin web mapping client to allow a user to view and navigate the served geographic information. Several open source internet mapping servers have become available. These servers provide a development environment for building spatially-enabled internet applications. They are not a full-featured GIS system, nor do they aspire to be. Instead, they excel at rendering spatial data (maps, images, and vector data) on the web. The most notable internet mapping servers are:

- Mapserver\(^1\)

\(^1\)http://mapserver.gis.umn.edu/
Another key component we need for our tool is a thin web mapping client, that is one that runs within a web browser. A thin mapping client has these advantages: (1) most computers have a web browser installed, so there is no install process for the user to complete, (2) most users are comfortable with the web, so less training is needed in its use and (3) AJAX (Asynchronous JavaScript and XML) increases the responsiveness, interactivity, functionality and usability of web applications providing the user with many of the advantages that a desktop client offers. We could build this client ourselves, from scratch, or we can capitalize on an object-oriented design tenet and reuse an existing framework or library. We utilize the latter approach. Several web mapping clients exist, for example:

- Chameleon³
- MapBuilder⁴
- OpenLayers⁵

Data Discovery and Exchange

To build an EAM data integration tool that is capable of integrating various sources of data, we need a mechanism that facilitates data discovery and exchange. This is accomplished through metadata. A metadata record is a file of information, usually presented as an XML document, which captures the basic characteristics of a data or information resource.⁶ Geospatial metadata are used to document geographic digital resources such as GIS files, geospatial databases, and earth imagery. Geospatial

---

²http://www.geoserver.org
³http://chameleon.maptools.org/
⁴http://www.mapbuilder.net/
⁵http://www.openlayers.org/
⁶http://www.fgdc.gov/metadata
metadata are critical to data discovery. Several resources for metadata discovery have emerged. The Federal Geographic Data Committee (FGDC) is tasked by Executive Order 12906 to develop procedures and assist in the implementation of a distributed discovery mechanism for national digital geospatial data.\textsuperscript{7} NASA’s Global Change Master Directory (GCMD) enables users to locate and obtain access to Earth science data sets and services relevant to global change and Earth science research.\textsuperscript{8} It holds more than 20,000 descriptions of Earth science data sets and services covering all aspects of Earth and environmental sciences.

For our tool, we also need access to biological taxonomic metadata. Taxonomic metadata creation is tedious and the metadata is constantly under going revision\textsuperscript{(Best, Coyne, and Halpin 2004)}. The Integrated Taxonomic Information System (ITIS) provides authoritative taxonomic information on plants, animals, fungi, and microbes of North America and the world.\textsuperscript{9}

Through these new metadata services, we can assimilate the available metadata into our application.

\subsection*{2.2.2 Emerging Open Standards}

Standards play a key role in enabling interoperability between systems. During the design of the system, focus was primarily placed on the implementation of OGC standards and the OBIS schema.

\textsuperscript{7}http://www.fgdc.gov/
\textsuperscript{8}http://gcmd.nasa.gov/
\textsuperscript{9}http://www.itis.gov/
Open Geospatial Consortium (OGC) Standards

The Open Geospatial Consortium (OGC)\textsuperscript{10}, a consortium comprising over 300 companies, agencies and universities, promotes the development, use and adoption of open standards for geospatial systems and geoprocessing. The results of their efforts, their OpenGIS Standards and Specifications, are technical documents that define interface and encoding specifications that facilitate interoperability between disparate geospatial data stores, services and applications. Our system uses the OGC’s Web Mapping and Web Feature Service specifications.

Web Mapping Service (WMS)

The OGC’s WMS specification\textsuperscript{11} defines an interface that allows a client to produce maps, or visual representations, of geospatial data. The WMS specification creates a map as an image from the underlying geospatial data. The specification document describes how a WMS server must respond to a specific set of well-defined, well-formed requests from a client of the WMS server.

The OGC’s WMS specification defines three WMS operations: GetCapabilities, GetMap and GetFeatureInfo. Two of these, the GetCapabilities and the GetMap operations, are required. The GetCapabilities operation returns a description of the service, the information that the service contains, and acceptable request parameters. The GetCapabilities operation returns a human-readable description of the information that the service contains, the parameters the service can handle, and the maps available via the service. The response is a well-formed XML document that follows

\textsuperscript{10}http://www.opengeospatial.org/
\textsuperscript{11}http://www.opengeospatial.org/standards/wms
Figure 2.7: Open Geospatial Consortium (OGC) Open Web Services

the encoding defined within the WMS specification for service-level metadata. The GetMap operation returns a map image, a visualization of geospatial information, whose dimensional parameters are well-defined. The GetMap operation is one that a client invokes to retrieve a rectangular image corresponding to a geographic area or a set of graphic elements that lie in a geographic area. When invoking the GetMap
operation of a WMS server, a client can specify:

1. the geographic data, or “layers” to appear within the map, and their styles
2. what geographic area, or “bounding box”, of the world to display within the map
3. the projected or geographic coordinate reference system to be used (SRS)
4. the output format of the map (png, jpeg, gif, etc.)
5. the output size (width and height) of the map in pixels
6. the background transparency and color of the map

The GetFeatureInfo, which is an optional operation, returns information about features appearing within a map as XML.

If two or more maps, or geospatial data visualizations, are produced with the same bounding box, coordinate system and output size, then these images can be layered one on top of the other, either through the WMS itself or through a web mapping client, to construct an accurate, composite map. Using image formats that support transparent backgrounds allows the lower layers of the composite map to
Map layers can be aggregated from different WMS servers, providing the foundations for a distributed network of geospatial information servers, from which, users can build their own customized composite maps. From the standpoint of solving our interoperability challenge, the OGC WMS specification provides a mechanism for utilizing various other data sets without having to maintain them ourselves. Changing from one data source to another, in most cases, simply involves changing the URL of our GetMap request to point to another WMS server.

Figure 2.9 illustrates both the XML and URL versions of a WMS GetMap request.
Styled Layer Descriptors (SLDs)

Styling of a map can be described using a user-defined XML encoding of a map’s appearance called a Styled Layer Descriptor\(^\text{12}\) (SLD). An SLD comprises a StyledLayerDescriptor XML element containing a sequence of styled-layer definitions. User-defined SLDs describe rules for symbolizers used to render the map image. These symbolizers include those for drawing lines, polygons, points, text or labels and rasters. The SLD can include NamedLayer and NamedStyle elements, which correspond to the LAYERS and STYLES of the parameters within a WMS request. A painters model is used to determine the “Z-ordering” of styling rules within the SLD. User-defined styling allows for much more complex map styling.

The OGC SLD Specification defines three methods that a client may use to take advantage of SLD symbology:

\(^\text{12}\)http://www.opengeospatial.org/standards/sld
• using the HTTP GET method, provide the URL to a remote SLD as the SLD parameter of the WMS GetMap request,
• using the HTTP GET method, provide the SLD XML document, using the proper encoding, as an SLD_BODY parameter of the WMS GetMap request, or
• using the HTTP POST method, provide an embedded SLD document as part of the WMS GetMap request

It is important to note that using the second method can encounter problems in some browsers due to excessively long URLs. According to the OGC SLD Specification, the third method is technically superior and more and more servers are providing support for the HTTP POST method for WMS requests.

Client applications can use SLDs in one of two ways. They can allow a user to switch between a number of pre-defined maps, each specified by its own pre-defined SLD or they can allow a user to interactively define how a map should be styled and construct the necessary SLD ‘on-the-fly’. This can be done using user-defined symbolology, through the use of user-defined data selection, using OGC Filter Encoding\textsuperscript{13} and applying pre-defined SLD snippets, or through a combination of the two. Pre-defined SLDs would need to be made available to the client application via URLs.

Appendix B provides an example of an OGC SLD. The example SLD was used to style the map of Figure 2.10.

\textbf{Web Feature Service (WFS)}

While the OGC’s WMS specification allows a client to produce and combine visual representations of geospatial data, the OGC’s WFS specification\textsuperscript{14} allows a client to

\footnotesize{\textsuperscript{13}\url{http://www.opengeospatial.org/standards/filter}
\textsuperscript{14}\url{http://www.opengeospatial.org/standards/wfs}}
retrieve the geospatial data itself. This geospatial data are encoded in the OGC’s Geography Markup Language\textsuperscript{15} (GML).

A WFS server’s interface is defined using XML. The WFS server utilizes GML to express features within the WFS server. The WFS server provides an interface to a datastore used to store geographic features, be it a Shapefile, a spatially enabled database, or some other format for geospatial data. The WFS interface provides a standardized interface for accessing geographic data from various datastores.

![Figure 2.11: An OGC Web Feature Service (WFS)](image)

Two types of WFS servers exist: basic and transactional. A basic WFS can be thought of as a READ-ONLY server. Basic WFS servers implement the GetCapabilities, DescribeFeatureType and GetFeature operations. The GetCapabilities operation allows the WFS to indicate which feature types it services and the operations supported on each one. The WFS’s DescribeFeatureType operation describes the structure of any feature type it can service. A client uses the WFS’s GetFeature operation to retrieve feature instances. In addition, the client is able to specify which

\textsuperscript{15}http://www.opengeospatial.org/standards/gml
feature properties to fetch and should be able to constrain the query, both spatially and non-spatially.

A transactional WFS implements the basic operations, along with a transaction operation. This operation allows a client to modify, create, update and delete geographic features within the WFS. Optionally, a transaction WFS may support a LockFeature operation, ensuring that serializable transactions are supported.

The following list outlines a typical interaction between a client and a WFS server:

1. A client application requests a capabilities document from the WFS, which contains a description of the operations that the WFS supports and a list of the feature types that it services.
2. A client application (optionally) requests the definition of one or more of the feature types that the WFS services.
3. Based on the definition of the feature type(s), the client application generates a request that conforms to the WFS specification.
4. The request is posted to a web server where the WFS is invoked to read and service the request.
5. The WFS processes the request and returns the result or an error message if it was unable to service the request.

Figure 2.12 illustrates the XML structure of a WFS feature request. Figure 2.13 is the response to the WFS request. Note that the format of both the request and response is XML conforming to the OGC specification.
Ocean Biogeographic Information System (OBIS) Schema

OBIS, the information component of CoML, is a web-based provider of global georeferenced information on marine species. In order to promote interoperability of the various distributed data sets that the OBIS Portal serves, OBIS has defined a schema that all of the distributed data contributors must implement. The OBIS schema defines a list of standard data fields, along with names, descriptions, and format notes for those fields. The full schema is provided in Appendix C.
2.2.3 Systems and Software Libraries Supporting Internet Mapping

The Ocean Biogeographic Information System (OBIS)

OBIS\textsuperscript{16} is a distributed, online information system that aims to be the primary authoritative source of data on the distribution of the marine species of the world with online quality control, mapping, and analysis tools, including correlating distributions to environmental datasets(Grassle 2005). Its goal is to be a driving force in the advancement of marine science, management, and education(Grassle 2005). OBIS intends to create a data resource of broad utility to meet the diverse needs of user groups including resource managers, navies, industries, and environmental and educational groups, with its primary objective being to make basic marine data sets and resources accessible and interoperable(Grassle 2005).

OBIS is creating a central hub for the collection, access, analysis and visualization of data on marine organisms. Currently, it lacks support for gazetteer, polygon-based spatial searching and the ability to represent more than simply the presence or absence of data(Rees and Zhang 2004). It currently lacks the ability to map environmental data and the ability to integrate biogeographic data with environmental data. These lackings call into question its support for discovering scientifically important patterns and events.

Spatial Ecological Analysis of Megavertebrate Animal Populations (SEAMAP)

SEAMAP\textsuperscript{17} is a mapping portal involved with the internet data collection of animal observations, oceanographic data and species profiles and is currently serving 215

\textsuperscript{16}http://iobis.org/
\textsuperscript{17}http://seamap.env.duke.edu/
datasets comprising 1,135,671 records, spanning the years 1935 to 2007 (Halpin and Read 2004). SEAMAP’s strategy is to attract data providers by providing features that include (1) various mapping tools, (2) biological, physical and anthropogenic data layers, (3) metadata creation and (4) download and upload facilities (Halpin and Read 2004).

SEAMAP’s online archive is searchable by species, location, time, methodology and provider. The filtered results are mapped and cross-referenced to species profiles and dataset details. SEAMAP also has the ability to map some coarse-grained environmental data, and is working towards providing the ability to import arbitrary external data. SEAMAP lacks the ability to integrate biological and environmental data, a requirement for EAM.

Figure 2.14: OBIS-SEAMAP
GeoTools Java Library

The following information about GeoTools is quoted from their project page.\(^{18}\)

GeoTools is an open source Java code library which provides standards compliant methods for the manipulation of geospatial data. The GeoTools library implements OGC specifications as they are developed, in close collaboration with the GeoAPI project\(^ {19}\), a project that is creating interfaces for Java GIS projects. GeoTools is used by a fair number of projects, both open source and commerical. The following list outlines several key features of the Geotools toolkit:

**Data Formats**

Since data are the life blood of any GIS project, geotools is committed to supporting as many data formats as possible. GeoTools uses a DataStore interface to access vector data and a GridCoverageExchange interface to access grid coverage, or raster, data formats.

Supported Vector Data Formats

- Shapefile - an ESRI shapefile
- GML - Geography Markup Language
- WFS - Features from an OGC Web Feature Server
- PostGIS - geometric objects for PostgreSQL
- Oracle Spatial – Oracle's extension for spatial data
- ArcSDE - ESRI’s middleware for spatial databases
- MySQL - support for the new geometry types
- GeoMedia - an Intergraph format
- Tiger - Topologically Integrated Geographic Encoding and Referencing developed at the US Census Bureau

Supported Grid Coverage Formats

- ArcGrid - ArcInfo ASCII Grid format and GRASS ASCII Grid format (optionally compressed)
- Image - can load images georeferenced with a world file
- GeoTIFF - a georeferenced tiff image
- WMS - OGC Web Mapping Server client

\(^{18}\)http://geotools.codehaus.org/

\(^{19}\)http://geoapi.sourceforge.net/
Features and Geometries

Features are used to represent basic geographic elements in a vector system. These are composed of both geometries and attributes. GeoTools uses the Java Topology Suite (JTS)\(^{20}\), an implementation of the OGC’s Simple Features Specification for SQL\(^{21}\), as its current geometry model. JTS also provides in memory spatial indexing support and robust topology operations.

A grid coverage provides support for the raster data format, which is a georeferenced grid of numbers that can be used to represent data ranging from satellite images, to digital elevation models, noise distribution and so on. GeoTools support for grid coverages is based on the Java Advanced Imaging (JAI)\(^{22}\) API, a library which provides support for data management, presentation, image data format access, tiling support. JAI serves as a framework for raster data processing and has various predefined operators.

Coordinate Transformation

GeoTools’ cts-coordtrans (CTS) module implements a subset of the OGC’s Coordinate Transformation Services Specification\(^{23}\) and provides an implementation for general positioning, coordinate reference systems and coordinate transformations. GeoTools supports transformations that include datum shifts, map projections (Mercator, Transverse Mercator, Lambert Conformal Conic, Albers Equal Area Conic, Stereographic, Orthographic) and others.

Styling

GeoTools provides a mechanism for specifying the symbolization of datasets. This mechanism is closely modeled on and fully supports the OpenGIS SLD Specification\(^{24}\).

Filter Encodings

Filters provide a way to specify a subset of features, using both attribute and spatial data, to operate on. GeoTools’ fully supports the OGC’s Filter Encoding Specification\(^{25}\).

\(^{20}\)http://www.vividsolutions.com/jts/jtshome.htm
\(^{21}\)http://www.opengeospatial.org/standards/sfs
\(^{22}\)http://java.sun.com/javase/technologies/desktop/media/jai/
\(^{23}\)http://www.opengeospatial.org/standards/ct
\(^{24}\)http://www.opengeospatial.org/standards/sld
\(^{25}\)http://www.opengeospatial.org/standards/filter
The use of GeoTools is a key element of our system’s design and its API has supported the solution to a number of the technical challenges to designing and implementing an EAM data integration tool. These technical challenges will be discussed further in Chapter 4.
Chapter 3

A Case Study

In this chapter, we demonstrate the scientific employment of our system with a specific example. We pose a question concerning the distribution of a species within the Gulf of Maine. We explore the interrelatedness of those biogeographic observations with oceanographic parameters. We then use the system to integrate the data of interest and download the integrated data.

3.1 Troubles of Commercial Fishing with Spiny Dogfish (Squalus acanthias)

The Spiny Dogfish (Squalus acanthias) has long been a thorn in the sides of recreational and commercial fishermen. According to commercial fishermen, the Spiny Dogfish are significantly interfering with many major fisheries (Stolpe 2006). These small sharks occur in huge numbers and are notorious for their voracious appetites, eating both more valuable species and the organisms upon which those species feed (Stolpe 2006). Their vast numbers often make it all but impossible to fish, clogging nets and damaging the intended commercial catch (Stolpe 2006).
They reportedly appear closer to shore during fall months. A marine researcher might pose the following scientific question: “Why are Spiny Dogfish more common closer to the shore during the fall months as opposed to the spring months?” We will use our system to explore the relationship between Spiny Dogfish and bottom temperature, and then use the system to integrate and download the data.

![Figure 3.1: The Spiny Dogfish (Squalus acanthias)](image)

3.2 Exploring the Correlation Between Spiny Dogfish and Bottom Temperature

3.2.1 Focusing Our Exploration

The first step in our case study is to determine the availability of biogeographic data for Spiny Dogfish. This will allow us to focus our investigation and narrow the observations of interest to the ones that will prove most valuable. It will also guide us in the selection of date ranges for oceanographic data.

We use the system to map all biogeographic observations for Spiny Dogfish. We select the National Marine Fisheries Service’s (NMFS’s) Northeast Fisheries Science Center (NEFSC) Bottom Trawl Survey, the largest survey of the Gulf of Maine, dating as far back as 1963. The NMFS Bottom Trawl Survey is designed to provide information on the abundance, biology, and distribution of the living marine resources of the
Northwest Atlantic and the environmental conditions affecting them for management purposes and to provide for broadscale ecosystems research.

We add a biogeographic layer for Spiny Dogfish to the system’s layer manager by clicking on the Add Biogeographic Layer button. Figure 3.2 shows the results of selecting all Spiny Dogfish observations from the NMFS Bottom Trawl Survey. Using the accordion menu for our Spiny Dogfish layer, we select the Layer Download option and download our layer as a comma-separated value (CSV) file (Figure 3.3).

**Biogeographical and Physical Data Integrator**

![Biogeographical and Physical Data Integrator](image)

**Figure 3.2: Adding All Spiny Dogfish (*Squalus acanthias*) Observations from the NMFS Bottom Trawl Survey**
Using a statistical software package, we graph the number of Spiny Dogfish observations from the NMFS Bottom Trawl Survey by month and day. By observation, we are referring to a tow, or cast, that resulted in a Spiny Dogfish. The result is presented in Figure 3.4. From the graph, we can see that the observations have a bimodal distribution. The majority of observations fall in the months of April and October. This suggests that we use the April and October observations as representative of the Spring and Fall surveys, respectively.
Figure 3.4: Spiny Dogfish (*Squalus acantbias*) Observations from the NMFS Bottom Trawl Survey By Month and Day from 1987 to 2005
3.2.2 Exploring the Interrelatedness of Spiny Dogfish (*Squalus acanthias*) Observations and Predicted Bottom Temperature

To distinguish observations from the two seasons, we first map data from April. We do this by clicking on the application’s Add Biogeographic Layer button. Next, we name our layer and select the NMFS Bottom Trawl Survey (see Figure 3.5).

![Biogeographical and Physical Data Integrator](image)

Figure 3.5: Adding Spiny Dogfish (*Squalus acanthias*) Observations from the NMFS Bottom Trawl Survey (1)

Next, in Figure 3.6, we specify the regions of interest. In our case, we are interested in the entire Gulf of Maine region so we check all physiographic regions. We also select the date ranges we are interested in. Since we are interested only in those Spiny Dogfish observations occurring in April, we select our date ranges by months.
and years, select from April to April for the months and from 1987 to 2005 for the years. The bottom temperature data only goes as far back as 1987. Figure 3.7 shows the results of adding all of the April Spiny Dogfish operations from the NMFS Bottom Trawl Survey.

Figure 3.6: Adding Spiny Dogfish (Squalus acanthias) Observations from the NMFS Bottom Trawl Survey (2)
Next, we add a layer for our oceanographic parameter of interest—in this case, bottom temperature. We add an oceanographic layer to the system by clicking on the application’s “Add Oceanographic Layer” button.

Figure 3.8 shows the user form for adding an oceanographic layer. We use it to name our layer, to select the type of oceanographic data (model predicted or observed), the data set (GoMOOS Circulation Model) and the measurement type (bottom temperature). We are using predicted oceanographic data from the GoMOOS Circulation Model which produces monthly averages of various oceanographic data parameters.
Figure 3.8: Adding Predicted Bottom Temperature from the GoMOOS Circulation Model (1)

Figure 3.9 shows the use of the calendar interface to specify the date range for our monthly oceanographic data. We will use an arbitrary year for our data. In this case, we will choose bottom temperature data from 2001. This will serve as a guide, as we will integrate the data and perform a more rigorous statistical analysis later. Ideally, we would want to aggregate the spring temperature data over a span of years, and the system will be expanded to add such flexibility.
Figure 3.9: Adding Predicted Bottom Temperature from the GoMOOS Circulation Model (2)

Figure 3.10 shows the resulting map. We use the accordion menu for our oceanographic layer to select the “Physical Data by Month” option. We use the drop down menu to change the month to our desired month, April 2001. An initial look at the resulting map yields an interesting observation. We can see that the Spiny Dogfish observations lie outside of the dark blue area, corresponding to a temperature of less than 3.5 degrees Celsius, along the Northern and Eastern Coastal Shelves. The Northern and Eastern Coastal Shelves are the regions immediately off the coast of Maine and south of Nova Scotia, respectively.
Next, we add a biogeographic layer for the October, or fall, observations for Spiny Dogfish from the NMFS Bottom Trawl Survey. We use the same method for adding the October observations that we used for adding the April observations. Figure 3.11 shows the result of adding the October Spiny Dogfish observations to the layer manager. Using the accordion menu for our Spiny Dogfish observations for April, we select the “Species By Common Name” and use it to toggle the visibility of the April observations.
Finally, we select the “Oceanographic Layers” tab and select the “Physical Data by Month” option of our bottom temperature layer. We use the drop down menu to change the currently mapped month to October 2001 to correspond to our Spiny Dogfish observations from October for the purposes of this demonstration. Figure 3.12 shows the resulting Spiny Dogfish observations from October mapped against the bottom temperature predicted by the GoMOOS Circulation Model for October 2001. From the resulting map, it appears that the Spiny Dogfish have moved closer to shore in the Fall following the warmer water. Next, we will use the system to integrate and download the data which will allow a researcher to perform a more rigorous analysis of the correlation.
3.2.3 Obtaining the Integrated Biogeographic and Oceanographic Data

To obtain the integrated biogeographic and oceanographic data, we click on the application’s “Integrate Data” button. This will allow us to append the predicted bottom temperature to our biogeographic observations of Spiny Dogfish. Figure 3.13 shows the first screen of the integration process. First, we specify the biogeographic layer to which we want to append oceanographic data. The integration process must be repeated for each biogeographic layer. We choose to integrate our Spiny Dogfish-October layer and click “Continue”.
Figure 3.13: Using the System to Integrate Biogeographic and Oceanographic Data (1)

Figure 3.14 shows the second screen of the integration process. Here, we specify the type of oceanographic data (model predicted or observed) and the data set to use in our integration. We want this data set to match the one we used for mapping. In this case, we should select the GoMOOS Circulation Model. We then click “Continue”.

Finally, we select the oceanographic measurement types from the chosen data set that we wish to append to our biogeographic data layer. We use the drop down menu to select each type. We can also add a time lag if we think it is appropriate. We click “Append” to add a measurement type to the list. Finally, we click “Integrate” to start the integration process. Once the integration process has been completed, we are prompted with a window that allows us to save the integrated data to our local disk (see Figure 3.16). We repeat the process for our Spiny Dogfish observations from October. A marine researcher can now use the integrated data to perform a more rigorous statistical analysis of the relationship between Spiny Dogfish and predicted bottom temperature.
Figure 3.15: Using the System to Integrate Biogeographic and Oceanographic Data

Figure 3.16: Downloading the Integrated Biogeographic and Oceanographic Data
Chapter 4

Overall Architecture

From a review of the literature and a survey of the technical landscape, we can identify several requirements that the system must meet. These criteria, also put forth by the marine researchers at the Aquatic Systems Group of the University of Southern Maine, are:

1. The ability to choose a data set, query a date range and a species (or multiple species) and display observation locations on a map and view a table of the underlying data.
2. The ability to download the mapped data in a variety of formats.
3. The ability to map “zero observations”, or observations where an attempt to catch a species was unsuccessful, along with “positive observations,” or observations where an attempt to catch a species was successful.
4. The ability to add and remove various static base layers (e.g. bathymetry, physioregions, sediment and others).
5. The ability to add WMS and WFS layers from other servers (e.g. GoMOOS) to a map.
6. The ability to serve GMBIS data as WMS and WFS layers for other web map-
ping portals to use.

7. The ability to query, map and download OBIS data.

8. The ability to perform spatial queries on biogeographic and oceanographic data.

9. The ability to interact dynamically with the map, for example, zooming and panning.

10. The ability to compare and contrast data, for example, by toggling layers of similar data sets in order to make assessments.

11. The ability to integrate biological data with environmental data in a single map or table.

In the following sections, we will discuss some of the computational issues that need to be resolved to meet these requirements and discuss the overall architecture and system components we use to satisfy these requirements.

### 4.1 Key Technical Challenges

There are four primary computational challenges that we have identified within this problem domain that must be overcome in order for the system to be successful and to ensure that the software will meet the needs of its users.

1. **Extensibility** To ensure that the systems architecture will carry into the future, it should be modular and extensible, with the ability to add or replace a capability in the current system, regardless of vendor, with minimal integration costs, and have it work seamlessly. We can discern several axes of change that the system must be capable of addressing in an extensible manner. These include handling new data sources, handling new delivery formats and integrating with different web mapping clients and GUI interface libraries.
2. **Multiple Data Source Support**  
Interoperability issues arise from incorporating data from different external sources. The system must be able to access and integrate different data sources that have a variety of different formats.

3. **Data Integration and Delivery**  
One of the primary requirements of the system is the integration, or spatial joining, of biogeographic and oceanographic data and the delivery of the result to the user in a convenient format.

4. **User Interface**  
The user interface must support spatio-temporal data exploration and hypothesis generation by marine researchers. Providing that support must balance simplicity of use with sophisticated functionality.

In the following chapters, we discuss each technical challenge in more detail and present the patterns and software architecture that we implement to solve each issue. First, we will introduce a broad overview of the entire system’s architecture.
Figure 4.1: Overall System Architecture Diagram
4.2 Overall Architecture

Figure 4.1 presents a broad overview of the overall system architecture. The system is built on the client-server model with each of the different subsystems being color-coded within each domain.

4.2.1 Client-Side Architecture Components

JavaScript Libraries

On the client-side, there are two primary components at work. The first are jQuery and JavaScript objects that handle client-specific data management (LayerManager) and the widgets of the rich graphical user interface (jQuery and jQuery plugins). jQuery\(^1\) is a concise, JavaScript Library that simplifies HTML document traversal, handles events, performs animations and allows for the easy addition of AJAX (Asynchronous JavaScript and XML) interactions to web pages.

Web Mapping Client

The second major component on the client-side is the web mapping client. We employ the TEMPLATE METHOD pattern to abstract, or decouple, the LayerManager from the web mapping client’s concrete implementation. For the web mapping client, we use OpenLayers\(^2\), a pure JavaScript library with no server-side dependencies. We utilize OpenLayers primarily because it is open source software that implements the OGC’s WMS and WFS specifications for geographic data access. The OpenLayers API allows us to build a rich web-based geographic application with a mapping client that provides the user with dynamic maps. OpenLayers provides various controls

\(^{1}\)http://jquery.com/
\(^{2}\)http://www.openlayers.org/
that allow the user to pan and zoom: an important ability in an exploratory tool for geospatial data.

4.2.2 Server-Side Architecture Components

Spatial Datastores

The first server-side component(s) are the system’s spatially enabled databases. These databases serve as datastores for the system’s application data, both alphanumeric and geographic. We employ PostGIS\(^3\) to serve as the backend spatial database for our geographic information application. PostGIS adds support for geographic objects to the PostgreSQL object-relational database, spatially enabling the PostgreSQL server. We use PostGIS for several reasons. First, it follows the OpenGIS Simple Features Specification for SQL and has been certified compliant with respect to spatial types and functions. Second, PostGIS is compatible with a large number of the other components needed to build this system. Finally, PostGIS is open source software, and while it comes with all the features of a commercial DBMS, it does not come with the substantial price tag.

Spatial Data Server

The second server-side system component is the spatial data server. A spatial data server allows users to view, and possibly edit, geographic data. In order to build a web application within the realm of GIS, the system needs to publish geospatial data in a variety of formats, such as map images and actual data. We utilize GeoServer\(^4\), a fully functional transactional WFS (WFS-T)\(^5\) and WMS server that complies with

---

\(^3\)http://postgis.refractions.net/
\(^4\)http://geoserver.org
\(^5\)http://www.opengeospatial.org/standards/wfs (see WFS Transactional)
OGC specifications, for several reasons. First, it implements OGC specifications. Second, it supports PostGIS databases as backend spatial datastores. Third, it can be used in conjunction with clients, such as OpenLayers (for web pages), to display dynamic maps. Finally, it is open source and based on GeoTools\(^6\), a Java code library which provides standards compliant methods for the manipulation of geospatial data.

**Java Technology Components**

The next three server-side components are all Java based technologies: Java objects or classes, Java Server Pages and Java Servlets. Java serves as the code base for this project. We adopted Java as our server-side programming language for several reasons. First, Java is an open source, platform-independent programming language. Since the system should have the ability to be readily adopted by others, the system needs to be platform-independent. Second, anyone can download the Java Developers Kit (JDK) and a Java integrated development environment (IDE). We want this project to be an open source, community-based development effort and the choice of Java provides a larger developer community.

**Java Objects and Classes**  
Java objects within the system can be grouped into three categories: Gateways, Downloaders and Data Translators. Gateways are objects that wrap complex or special connection APIs into a class whose interface looks like a simple Java object. The Gateway methods used by other objects are translated into the appropriate specialized API calls. Gateways wrap the connections to our spatial datastores. This decouples the system from the specific datastore implementation. For example, an Oracle Spatial Gateway can be created, replacing the PostGIS datastore backend with an Oracle Spatial backend, and the rest of the

\(^6\)http://geotools.codehaus.org/
system would continue to work seamlessly.

DownloaderCommands are COMMAND pattern objects that implement a single method, writeOutput(). The system delegates user requests for data delivery to the appropriate downloader whose output is in the format requested by the user.

Data Translators are objects that retrieve a specific type, or format, of external data. When a user requests to import external data having a particular format into the system, the applications data mediator hands the request to the data translator that understands how to retrieve data of that specific format.

It is important to note that careful thought was used in the design of the system to support ease of maintenance and extension. This careful attention to design is evident through the use of the aforementioned patterns.

JavaServer Pages JavaServer Pages (JSP)\(^7\) technology provides a way to create dynamic web content, using Java code as a server-side scripting language. Unlike other server-side technologies, JSP employs a strongly typed language for server-side scripting. It also enables development of web-based applications that are server and platform-independent. The application’s start page and modal GUI dialogs are all implemented using JSP.

\(^7\)http://java.sun.com/products/jsp/
Java Servlets  Java Servlet technology provides a mechanism for extending the functionality of a Web server and for accessing existing business systems. There are four key servlets used by the application: the SLDBuilder, the Downloader, the Data Integrator and the Data Mediator. The SLDBuilder handles all requests from the Layer Manager concerning the generation of map displays. The SLDBuilder builds SLDs, OGC documents that allow the WMS specification to be extended to allow user-defined symbolization of feature and coverage data. The Downloader servlet handles all user requests for data delivery and delegates user download requests to the appropriate DownloadCommand object. The Data Integrator servlet fields all data integration requests from the user and returns the integrated biological and physical data in a comma separated value (CSV) formatted file. The Data Mediator servlet handles all user requests for external data and delegates the user request to a Data Translator that can retrieve the requested data from the external data source.

---

8http://java.sun.com/products/servlet/
Chapter 5

Technical Challenge I: Extensibility

In the next four chapters, we discuss each of the four major technical challenges we have identified in more detail. We also discuss our solution and its implementation within the system.

5.1 Axes for Change

The Agile Development Process assumes that systems will evolve and change, and developers cannot possibly anticipate all future changes to the system (Martin 2003). A practical response to this reality is to refactor code when it is needed. At this stage of development, we can discern several key axes for change that the system must contend with. They include:

1. handling new data sources and their formats
2. handling new delivery formats
3. integrating with different web mapping client and GUI interface libraries to
allow the customization of the system interface for specific applications

The system’s design accommodates these axes of change in an extensible manner through the use of several software design patterns.

The requirements for the project have been determined in discussions with the users: marine scientists, resource managers and other public stakeholders. The long-term goals for the system are clear: the generation of knowledge from the synthesis of both existing and emerging sources of information. A successful system for the Gulf of Maine region could serve as a model for similar systems around the world.

Bricklin states that in order for a software system to be extensible, it must possess the following qualities:

- robustness and long-term stability,
- ease of modification and maintenance, and
- ease of component replacement.

A critical issue in designing a system that lasts is that of platform-independence (Bricklin 2004). The application needs to be created while keeping the possibility of new hardware, operating systems, and other “computer infrastructure” in mind. We achieve platform independence by using a platform independent code base. We insulate our system from inevitable platform changes by using one of the most platform independent languages available, Java, as much as possible.

Another major issue affecting the long-term success of software is that of data storage and interchange standards (Bricklin 2004). Users expect all information to be available all of the time and for new purposes. This is especially true for our system, which must incorporate newly available information (Murawski 2007). Standards
bodies publish best practices and although a project might be novel, there are many standards that should be applied (Bricklin 2004). In our case, we are referring to those standards that facilitate data interchange and exchange. This is a key technical challenge in designing the system and we discuss it in detail in Chapter 6.

5.2 Extensibility Solution

The purpose of object-oriented software architecture design is to deliver a system that satisfies the requirements of its users with a clean internal structure. Although the internal structure is of no concern to the user, the maintenance, porting and successful extension of the software is. Because of this, an application must have a clean internal structure to simplify the tasks of testing, porting, maintaining and extending that will recur throughout its life. The main point is that every successful piece of software will have an extended life in which it is worked on by a succession of programmers and designers, ported to new hardware, adapted to unanticipated uses, and repeatedly reorganized ((Martin 2003), (Bricklin 2004)).

To accomplish this, we employ and implement several software design patterns. A software design pattern addresses a generic problem which occurs repeatedly in the software engineering discipline and describes a generic solution to the problem that can be instantiated to concrete situations (Fowler 2004). Many such patterns have been identified and discussed in the software engineering literature ((Fowler 2004), (Gamma, Helm, Johnson, and Vlissides 1994)). The patterns implemented within this application include COMMAND, FACTORY, GATEWAY, MEDIATOR, SINGLETON and TEMPLATE METHOD. The COMMAND pattern is utilized in our solution to the data delivery challenge and is discussed in Chapter 7. The FAC-
TORY, GATEWAY, MEDIATOR and SINGLETON patterns are key patterns in our solution to the interoperability challenge and are discussed in Chapter 6. The software that we build must last into the future with a minimal maintenance effort. Design patterns help us structure the system to achieve that result.

The TEMPLATE METHOD pattern is implemented throughout the system in order to provide robustness, modularity and extensibility. We use the TEMPLATE METHOD pattern because it allows us to define the algorithm skeleton in an abstract class, laying out the necessary operations required by the program, and create subclasses to provide concrete behavior for the abstract steps of the algorithm. The TEMPLATE METHOD pattern also allows us to factor out the generic algorithm from the concrete behavior implemented by the subclasses. We employ the TEMPLATE METHOD for two key system interfaces: the interface to web mapping clients and the one to client-side user interface libraries.

The Map Template class is an abstract client-side class that implements the TEMPLATE METHOD pattern. The Map Template class defines the abstract operations that the application needs to interact with a web mapping client. These operations include:

- addLayerToMap
- removeLayerFromMap

The Map Template class is a wrapper for a web mapping client object (mapObj) and provides various instance variables for handling map basics. These instance variables apply to all web mapping clients:

- imgFormat
• mapBBox
• mapObj, a wrapper for the concrete web mapping client
• a reference to the applications wrapper object which facilitates communication between the geospatial information server and the application

The Map Template class allows new web mapping clients to be incorporated into the application rapidly. An example of a concrete instance of the abstract Map Template class is the OpenLayersMapper object which is an interface to an OpenLayers web mapping client. The Map Template class also permits new web mapping clients to override abstract operations of the MapTemplate class, allowing new functionality for interfacing with other web mapping clients.

Another example of the TEMPLATE METHOD pattern is the abstract LayerManagerGUILibTemplate class. This class defines the generic operations that the Layer Manager needs from a Javascript GUI library and allows alternate Javascript libraries to be used to generate the rich user interface of the Layer Manager. An example is
the jQueryGUILib object which uses the jQuery Javascript library to provide and generate the Layer Manager’s interface.

In this chapter, we identified several axes for change that we anticipate for the system. They include handling new data sources and formats, handling new delivery formats and integrating with different web mapping client and GUI interface libraries. The system’s design employs the TEMPLATE METHOD design pattern to accommodate integration with different web mapping client and GUI interface libraries. We now turn to the mechanisms incorporated into the system design that allow for the handling of new data sources and their formats.
Chapter 6

Technical Challenge II:
Multiple Data Source Support

The next technical challenge that we will discuss is that which arises from supporting multiple, external data sources. Interoperability issues arise from incorporating data from various distributed, disparate sources of data. The inability to aggregate and integrate data from different data sources stems from the fact that they all have a variety of different formats. These different formats include:

- varying spatial extents
- different spatial reference, or coordinate, systems
- granularity of spatial dimensions
- varying temporal resolutions

6.1 Interoperability Issues

As the geospatial web continues to grow at an astounding rate, those individuals and organizations involved with the acquisition and delivery of geospatial data through
the Internet must overcome the heterogeneous nature of the data. Marine researchers must overcome the issue of interoperability as they strive to ascertain and explain the diversity, distribution, and abundance of marine life in the oceans.

The organizations and researchers involved with CoML collect massive quantities of data. In addition, CoML researchers want to integrate their biogeographic data with the oceanographic data provided by various ocean observing systems and geological surveys. These data are stored in a variety of formats, using a myriad of geometry models, coordinate reference systems and data models. As a result, sharing geospatial data requires considerable investments of expertise, diligence and time. If the distributed data cannot be efficiently brought together in a revealing manner, then value added knowledge cannot be extracted from its integration.

Weak interoperability results in wasted resources and effort. Data providers must maintain their own data sets and are unable to share and aggregate their resources with other data providers. Developers must learn and implement multiple interfaces that have subtle details and nuances in order to interact with the multiple, disparate data sources of interest. As a result, researchers may not be able to combine different layers from different data sources to produce a desirable map.

Strong interoperability allows users to access, query and obtain information that is stored in many different sources, using many different formats. The system supports access to geospatial information from multiple organizations with minimal effort and the integration of geographic data and map images from different data servers.

One of the primary requirements of the system is to provide an “aggregation server” mechanism that delivers and relies on the combination of locally cached archival data
and dynamic access to remote data sources. Another requirement for the system is that it allow users to combine different map layers that comply with the OGC’s WMS and WFS standards and biogeographic data sources that conform to the OBIS schema.

6.2 Advantages of Interoperability

There are numerous advantages to designing a system that is highly interoperable. First, interoperability reduces the cost of building and supporting a heterogeneous data infrastructure. One system can handle multiple sources of data and can be easily extended to include additional data sources as they arise without the addition of significant overhead. Second, a system that readily integrates data from multiple sources promotes its own rapid adoption within a user community, as network effects increase the value of the system to its clients. Finally, a highly interoperable system leverages and incorporates existing investments from the community. This includes integration of existing data sources that may not conform to a standard, but from which useful knowledge can still be extracted.

6.3 Multiple Data Source Support Solution

In our solution to the technical challenge of interoperability, we implement the following design patterns:

- **GATEWAY** encapsulates access to an external system or resource (Fowler 2004)
- **FACTORY** lets a class defer instantiation to subclasses (Gamma, Helm, Johnson, and Vlissides 1994)
- **MEDIATOR** defines an object that encapsulates the interactions between a set of objects (Gamma, Helm, Johnson, and Vlissides 1994)
- **SINGLETON** restricts instantiation of a class to a single object at most (Gamma, Helm, Johnson, and Vlissides 1994)

In the following section, we explain how each pattern is employed within the system and highlight the reasons for their use.

We utilize the GeoTools API in our interoperable solution to multiple, external data source support. A DataStore is a GeoTools object that is used to access and store geospatial data in a variety of vector formats. These formats include ESRI shapefiles, GML files, spatially enabled, relational databases, WFS servers and others. In adhering to GeoTools best practices, we manage the GeoTools datastores as SINGLETON classes, restricting them to a single instance at most. We do this because DataStores are large, heavy-weight objects, many of which juggle database connections or load up spatial indices. Managing them as SINGLETON objects minimizes the overhead of creating a new DataStore object each time we need one and allows the system to operate more efficiently. The system currently provides DataStore singletons for ESRI Shapefiles, PostGIS databases and WFS servers.

Our PostGISDataStoreSingleton makes use of a DBGateway object that is used as a combination Transaction Script and GATEWAY to the non-spatial data stored in the GMBIS database. A Transaction Script is an object that serves as sort of Transaction registry (Fowler 2004). Frequently performed transactions are organized into a single procedure that makes calls, in our case, through a database GATEWAY object (Fowler 2004). Each database transaction has its own script within the Transaction Script object.

\[\text{http://docs.codehaus.org/display/GEOTDOC/04+How+to+Create+a+DataStore+or+DataAccess}\]
Figure 6.1: DataStore Singletons and Gateway Objects
In our solution to the problem of interoperability, we implement a Data Mediator as a servlet that functions as an intermediary for all client requests for external data. All client requests for external data are handled by the Data Mediator, which creates an appropriate Data Translator and then delegates the request to newly created translator. The Data Translators utilize GeoTools DataStore singletons to retrieve the data.

![Data Mediator and Data Translator Objects](image)

The abstract Data Translator of Figure 6.2 is creates concrete instances of translators for specific formats. The specific data translators that extend the Data Translator are our concrete products. We have modified the FACTORY pattern (see Chapter 2.1.3) in this manner to overcome some of its limitations. The first limitation is that,
since the Data Translator does not rely upon a private constructor, the class can be extended, unlike the FACTORY pattern. Any subclass of Data Translator must invoke the inherited constructor, but this cannot be done in the FACTORY pattern because that constructor is private. The second limitation is that, if we do extend the class, the subclass can elect to provide its own re-implementation of all factory methods with exactly the same signatures. For example, each new data translator in our system provides the concrete constructor for creating the connection to a specific datastore instance that provides data in the format that the translator understands. In the original FACTORY pattern, the subclass is forced to re-implement all of the original factory methods. Another gain is that since our factory is an abstract class, we can factor out common code from the concrete translators and store them within the factory. Our abstract Data Translator class contains core, concrete methods for interacting with the system’s cache and interacting with each of the concrete translators, like retrieving a list of layers from the external data source. Also, since the factory method of the Data Translator accepts a parameter map, an unspecified collection of attribute name attribute value pairs, instead of individual parameters, each concrete translator can require its own set of parameters. For example, the PostGIS translator needs a username and password parameter, while an ESRI shapefile translator needs only a URL. The final benefit achieved from our design is that new data translators do not change the rest of the system. As long as we use the established naming conventions for concrete translators and place them in the appropriate package, the system will automatically be able to utilize them.

Figure 6.3 is a UML 2.0 activity diagram illustrating the steps taken by the system when the user requests to add data from a supported external data source to the layer manager. The basic steps are:
1. The user requests to add data from an external data source, specifying the type of the data source [WFS, ESRI Shapefile, etc.] and queries the layers that the
external data source serves.

2. The Data Mediator creates the appropriate Data Translator for the requested format and delegates the layer list request to the created Data Translator.

3. The user specifies the layer that they would like to add to the layer manager and can optionally build an OGC style filter based on the attributes of the requested layer.

4. The user can optionally choose to preview the data that will be uploaded into the system cache of layers.

5. Once the user confirms the layer and the data to be loaded to the system cache, which is located on the server, the user can build their layer symbology while the Data Mediator coordinates with the Data Translator to retrieve the data, apply the OGC filter if provided, load the data into the system cache and make it available to the layer manager via the GeoSpatial Information Server.

6. Finally, once the user is done building their layer symbology, she can make the entire layer available for exploration within the layer manager.

In this chapter, we discussed the interoperability issues that arise from supporting multiple, external data sources. We identified some advantages from the ability to incorporate data from various distributed, disparate sources of data. We also discussed our use of mediators and translators to provide an extensible solution for supporting multiple, external data sources having a variety of formats.
Chapter 7

Technical Challenge III: Data Integration and Delivery

One of the primary requirements of the system is the integration, or spatial joining, of biogeographic and oceanographic data and the delivery of the result to the user in a convenient format. Another requirement of the software is the ability to download available biogeographic and physical data layers in various formats. To fulfill these requirements, the system’s design incorporates (1) a conceptual model within the system that facilitates the integration of biogeographic and physical data and (2) an extensible mechanism for data delivery that supports a variety of formats.

7.1 Data Integration

7.1.1 Conceptual Model: Point-in-Polygon

In order to provide a mechanism for integrating biological and physical oceanographic data, we first establish a conceptual model to facilitate the integration of spatial data. The system implements the Point-in-Polygon conceptual model.
At this point, we want to acknowledge that there are problems with Point-in-Polygon analysis. Conventionally, point-in-polygon analysis, one of the major spatial analyses in a GIS, is used to determine whether a point is located within a polygon (Cheung, Shi, and Zhou 2004). Cheung et. al. contend that, due to positional uncertainties introduced to GIS data during the data capture process, this boolean result cannot accurately describe the relationship of closeness between the point and the polygon. Another issue is determining which polygon does a boundary point lie in. It is important for us to note that our initial goal is to develop a functioning tool that supports EAM. We simplified the model to demonstrate the feasibility and usefulness of the tool.

In our data model, biological data observations are represented geographically by points and each has a latitude and longitude associated with it. The system’s oceanographic data is restricted to physical coverage data that is represented by a set of values assigned to a polygonal area. The physical data can be model predicted or observed. During data integration, biogeographic observation points that fall within a certain physical polygon are appended with the value corresponding to the physical polygon. The values of the physical polygon can vary by time, allowing us to introduce time lag into the data integration process. For example, we can append biological observations from a particular month with physical data from a certain time interval before or after the actual biological observation. This allows researchers utilizing the system to investigate event-driven time lags which may affect species distributions. Point-in-polygon makes sense because it is one of the standard GIS operations for performing spatial analysis (Cheung, Shi, and Zhou 2004). It is an operation that is normally supported within GIS desktop applications and is not available via the Internet.
7.1.2 Data Integration Solution

Figure 7.1 exhibits the internal schema used to store biogeographic data observations within the system’s PostGIS backend. The schema complies with the conventions put forth by the OBIS Schema (see Appendix C). This conformance grants the system the ability to import data from any data source that is OBIS-compliant.

Figure 7.1: Database Schema for Biological Data

Figure 7.2 reveals the internal schema for the storage of physical oceanographic data within our PostGIS datastore. We utilize a structure similar to the one used for storing biogeographic data. Each physical data observation is part of an effort and each effort belongs to a collection provided by an institution. For example, the model predicted bottom temperature provided by the system is from the Gulf of Maine Ocean Observing System’s (GoMOOS) Circulation Model. Temperature is
the measurement type, the month and year the bottom temperature was predicted are the effort, as there are multiple monthly efforts, GoMOOS is the institution and the Circulation Model is the collection. This provides a mechanism that supports an institution providing data from several different models or observation surveys, each capable of having multiple “efforts.” Each physical data observation has a corresponding polygon, stored within the Table $f_{gom\_models\_polys}$ and is used during data integration. To increase performance during data integration, we precompute the join of all physical data for a particular physical data collection, having a unique PDC_ID, into a table (see Table $gom\_physdata\_1\_all$ in 7.2).

Figure 7.3 highlights the methodology used within the PostGIS datastore during data integration. The steps are:

1. Join the biogeographic observation data with the GMBIS taxonomic informa-
Figure 7.3: System Methodology for Data Integration

2. Filter the resulting join table by species, if necessary.

3. Create a Hash on the resulting join table and, using the computed Hash, join it with the effort data.

4. Filter the resulting table using any spatial, institution and collection code filters supplied by the user when they added the layer to the layer manager.

5. Using a Generalized Search Tree (GiST), perform a spatial join of the biogeographic observations with the physical oceanographic regions. This is the Point-in-Polygon operation. The system uses a bounding box around the biological point effort to increase performance and performs a spatial join using the “within” GIS operation as our join condition. It also uses the month and year the biogeographic observation was collected as a join condition with the month and year the physical observation was collected or, in the case of models, predicted. A time lag can be introduced in the join condition during this step.

6. Join the resulting table with the Institution and Collection information for the biogeographic records.

7. Finally, deliver the result of the integration to the user in a comma-separated value (CSV) file.
7.2 Data Delivery Solution

Figure 7.4 is a UML 2.0 activity diagram illustrating the steps taken by the system when the user requests to download a specific, nonintegrated layer that they have added to their layer manager instance. The basic steps are:

1. The user requests to download a layer that they have added to their layer manager.
2. The layer manager retrieves the layers download url and forwards it to the Downloader servlet.
3. The Downloader servlet creates a DownloadCommand object that is able to generate the format that the user requested the layer be delivered in.
4. The Downloader servlet then delegates the generation of the output to the created DownloadCommand object.
5. The DownloadCommand object generates the formatted output for the layer the user requested and returns it to the Downloader servlet which then transmits the final output to the user over the internet.
We implement the COMMAND pattern in our data delivery solution. Figure 7.5 illustrates our implementation of the COMMAND pattern within the system. Within the application, the DownloadCommand object serves as our Command interface. The GML2Downloader, ESRIShapefileDownloader and CSVDownloader are all concrete commands that implement the DownloadCommand interface. The Downloader servlet acts as our invoker and initiates the actions provided by the concrete commands.
Figure 7.5: DownloadCommand Class Diagram

The system currently supports the following formats for downloading layers: GML2, ESRIShapefiles and CSV files. However, one of the benefits that we gain from the use of the COMMAND pattern is that support for new formats can be added effortlessly. A new concrete downloader, which implements the DownloadCommand interface, is created to encapsulate the steps necessary to transform and produce a layer in that particular format. Another benefit of implementing the COMMAND pattern is the ability to encapsulate complex, transaction-like behavior necessary to generate a desired output format within a single method of a command object.
The final output of data integration is a CSV file that is delivered to the user. Within the CSV file, biological data is presented first. The order of the biological fields in the CSV follows the order of the definition of the fields within the OBIS schema. After the biological data, the appended physical data columns are presented in the order that the user specified during the data integration process. The resulting file is for a single biogeographic layer from the layer manager. The integration process must be repeated for each additional biogeographic layer that the user wishes to integrate.

One of the primary requirements of the system is the integration, or spatial joining, of biogeographic and oceanographic data and the delivery of the result to the user in a convenient format. Another requirement of the software is the ability to download available biogeographic and physical data layers in various formats. To facilitate data integration, the system utilizes a Point-in-Polygon conceptual model in which biogeographic data, represented as points, can be spatially-joined with physical data, represented by polygonal regions and values associated with those regions, using a boolean “within” operation. The system’s design employs the COMMAND pattern to provide an extensible mechanism to support data delivery in a variety of formats. DownloadCommands are created to deliver data in a particular format.
Chapter 8

Technical Challenge IV:
User Interface

The overarching goal of this project was to develop a web-based tool that allows marine researchers to capitalize on cartographic cognition to generate hypotheses concerning the relationships between aquatic species distribution and physical oceanographic parameters. *Cartographic cognition* is the process by which the human brain recognizes spatial patterns and relationships (Lo and Yeung 2002). To accomplish this, the system’s user interface must support spatio-temporal data exploration by marine researchers while balancing simplicity of use with sophisticated functionality. In this section, we introduce some principles of user interface design and some key aspects of GIS GUIs that guided the design and implementation of the user interface. We then highlight some of the key features of that interface.
8.1 Behavioral Approach to Human-Computer Interaction

The behavioral approach to human-computer interaction attempts to model user interaction with the computer as closely as possible to human behavior and cognition (Egenhofer and Frank 1988). This approach encourages systems to incorporate design that allows users to interact with the system efficiently and with minimal training. This is achieved using three interrelated software design techniques: (1) what-you-see-is-what-you-get (WYSIWYG) ensures that the interface always reflects the current state of the system, (2) direct manipulation asserts that pointing devices should used to select and manipulate objects on the screen and (3) graphical user interfaces (GUIs) asserts that there should be a graphical and operational consistency across different applications and is achieved by using commonly accepted widgets, or components of a graphical user interface with which a user interacts, and metaphors, or environments for accomplishing certain tasks.

Implementation of the behavioral approach to human-computer interaction depends upon the use of GUIs for communication between the user and the application. The GUI concept was first developed by Xerox’s Palo Alto Research Center (Sheldon, Barron, and Smith 1991). In a GUI, the mouse and keyboard are used to interact with the application via various on-screen widgets.
8.2 User Interface Solution

8.2.1 Application Interface Goals

Our first goal in solving the user interface challenge was to design a GUI that could meet the functional data-processing and visualization requirements imposed by a distributed GIS application. Thus, the interface has to allow the user to conveniently enter selection criteria for the data, control and vary its display, and provide download capabilities. Another goal in our GUI design was to maintain an overall consistency of the appearance and mechanisms for interactions that have been established in the desktop GIS application realm. This allows GIS users to easily adopt and feel comfortable navigating and using our system.

GIS users have a set of unique requirements when it comes to GUIs (Lo and Yeung 2002). The application’s interface must support

1. spatial data viewing,
2. spatial querying of data, and
3. spatial analysis.

They go on to assert that GIS users have little interest in the structure of the underlying datastore. Instead, GIS users are primarily concerned with user-friendly interfaces that allow them to perform spatial tasks with minimum training.

We must also address the requirements that arise from developing a distributed, Internet-based GIS application. Internet-based GIS applications range from simple browsing of predrawn maps to interactive map generation (Lo and Yeung 2002). Another goal of our user interface solution is to allow users to break away from desktop GIS applications. To do this, we must provide a user interface that addresses the
needs for two types of GIS users: basic, non-traditional GIS users and experienced GIS users. We want our system to address the needs of basic users and the needs of marine researchers that do not currently have access to, nor the capability of investing heavy capital in, a full-blown desktop GIS system.

We want our system to allow marine researchers to do some spatial querying, analysis and spatial joining on data, supporting hypothesis generation and the download of data that can then be subjected to mathematical analysis to test the hypothesis. Our application interface solution assists a marine researcher in narrowing her focus, allowing her to download the most pertinent data.

In order for our interface solution to be viable in the GIS domain, we must capitalize on other ongoing GIS interface research. This includes the utilization of spatial query languages [(Egenhofer 1992), (Egenhofer and Frank 1988)]. We use PostGIS, a spatially enabled DBMS, to support spatial querying within our system. Previous GIS interface research also includes the map overlay metaphor, or environment, and direct manipulation method. [(Egenhofer and Richards 1993), (Lee and Chin 1995)] Our system provides the user with a map overlay environment, allowing them to easily manage the layers they have chosen to map using direct manipulation.

8.2.2 Key Interface Features

In this section we highlight some of the key features of our solution to the user interface challenge. These features focus on providing flexibilty in data combination and display.
Flexibility of Data Combination

The user is allowed great flexibility when adding biogeographic and oceanographic layers to the layer manager, part of our map overlay environment. Figure 8.1 shows an actual screenshot of the forms of the GUI that allow a user to add biogeographic and oceanographic layers.

Figure 8.1: Adding Layers to the Map Overlay Environment
Figure 8.1 highlights some key features of the forms which include:

1. **User-Defined Layer Names**  The user can provide their own unique, meaningful layer names and titles. These layer names are carried from layer creation to download and delivery.

2. **Different Types of Physical Oceanographic Data**  The system provides the user with several different types of physical oceanographic measurements. This measurement data can be either model predicted or observed oceanographic data. The user can select any of these different types of physical data and add them to the application’s layer manager.

3. **Layers from Different Data Sources and Models**  The system has several local data sources for both biogeographic and oceanographic data. The user can
select and filter data from any of these sources and add them to the application’s layer manager.

4. Layer Metadata  The user is presented with readily available layer metadata. Metadata, required of data sources by the system, is “data about the data” and is used to facilitate the understanding, characteristics, and management usage of data. This metadata includes data source descriptions, abstracts and links to full metadata records.

Figure 8.3: Inset of Figure 8.1 (2)
5a. Zero Observations  The user can elect to map observations where a biological species was not found. This means that an attempt was made to catch a certain species in a particular area, but the species was not caught. It is important when investigating species diversity to not only know where the species of interest were found, but also where they were not found. Figure 8.4 shows an example of a map where zero observations are presented along with positive observations.

5b. Filter Species by Common or Scientific Name  The user can select species by their Common Name or Scientific Name. The user also has the capability of selecting species from a convenient, auto-suggestion drop down list. The contents of the list are driven by those species found within the data source selected by the user and what the user has typed. This provides the user with the ability to combine several different species within a single layer.

![Figure 8.4: Map Showing Both Positive and Zero Observations](image)

6. Filter Species by Region of Interest  The user can filter species by regions of interest. This provides the user with the ability to perform spatial queries using a simple point-and-click interface, relieving them from the task of writing complex spatial queries in SQL.
7. **Filter By Date Range**  The user has the capability of filtering both biogeo-
graphic and oceanographic data by date range. The user can do this using a conve-
nient calendar interface. The user can also combine data from different time periods 
and sources in a single map. This allows the user to investigate time-lag events.

8. **Spatial Joining of Biogeographic and Oceanographic Data**  The user has 
the ability to perform spatial joins on biogeographic and oceanographic data. The 
user can elect to append physical data to biological observation records and download 
the resulting integrated biogeographic and oceanographic data in a CSV file. The user 
performs the spatial join using a simple point-and-click interface without having to 
install any GIS software.

![Figure 8.5: User Interface for Data Integration](image)
Flexibility of Data Display

Users can use the layer manager, a WYSIWYG interface, to manage the layers that have been mapped. The layer manager is illustrated in Figures 8.6 and 8.7.

1. Layer Metadata
2. Species by Common or Scientific Name and Accordion Menu for Switching Between Layer Metadata, Species Information and Layer Download
3. Collapsible Layers for Ease of Management
4. User-Defined Layer Titles Used as Handles for Sorting Layers
5. Layers Categorized By Tabs
6. Ability to Toggle Species Visibility
7. Ability to Change Color of Species Observations

1. User-Defined Layer Titles Used as Handles for Sorting Layers
2. Accordion Menu Similar to Biogeographic Layers
3. Ability to Toggle Month of Physical Data on-the-fly
4. Legend Graphic for Layer
5. Map Zoom Capability Applicable to All Layers
6. Map Zoom Box and Pan Capabilities Applicable to All Layers

Figure 8.7: Features for Managing Oceanographic Layers
8.2.3 System Screenshots

The system was designed as a self-contained, web application. The application uses AJAX, the asynchronous exchange of small amounts of data with the server “behind the scenes”, to increase its interactivity, speed, functionality and usability. In the next section, we present screenshots of the application that illustrate the major functionalities and components of the system. Figure 8.8 is the application’s initial start page. The application’s main GUI is divided into two components. These components are the Layer Manager, which serves as our map overlay environment, and an OpenLayers widget, which is our web mapping client.

Figure 8.8: System Start Page
Special attention was paid to metadata information and its presentation within the application. When adding layers to the Layer Manager, the user is presented with layer metadata (see Figure 8.1). Once the user has added a layer to the Layer Manager, its metadata is readily available via the Layer Manager’s interface. This layer information, shown in Figure 8.9 includes layer metadata (8.9a) and species information for biological layers (8.9b). For physical layers, a color ramp legend graphic is displayed instead of species information.

Figure 8.9: Viewing Layer Information within the Layer Manager
The user can also customize the symbology of species and elect whether on not to display a certain species. These layer attributes are easily altered through the Layer Manager interface. Figure 8.10 illustrates the GUI components that allow this type of customization and the results of changing these layer parameters.

![Figure 8.10: Changing Species Colors and Visibility within the Layer Manager](image-url)

(a) Color Picker Used to Change Species Colors

(b) Result of Color Change and Toggling Visibility
Figure 8.11 illustrates some of the physical layers from the GoMOOS Circulation Model that can been added to the Layer Manager. These include Sea Surface Salinity(8.11a) and Sea Surface Temperature(8.11b). Not pictured are Bottom Salinity and Bottom Temperature. These data are monthly averages, stored locally within the GMBIS database.

Figure 8.11: Adding Physical Layers to the Layer Manager
One of the major components of the application’s user interface is a web mapping client. Using the OpenLayers widget, the user can pan and zoom within the map (Figure 8.12). This capability provides the user with dynamic maps, not just static images.

![Image of a web mapping client](image)

Figure 8.12: Zooming with the System’s Web Mapping Client

Through the use of a mediator and various data translators, the system is capable of aggregating data from external sources. The user can specify an arbitrary external source whose format is supported by the system. The user can also apply filters to the data based on attributes of the data, preview the data and then add the data from the external source to the Layer Manager. Figure 8.13 shows an example of the system’s ability to preview filtered data from an external data source.
After specifying a filter for a layer and adding to the Layer Manager, the user can elect to download the data in a variety of formats. This is easily accomplished from within the Layer Manager’s interface, as shown in Figure 8.14.

In this chapter, we discussed some principles of the behavior approach to human-computer interaction which guided the design of the system’s user interface. We also
discussed some aspects of GIS GUIs. The system’s user interface supports spatio-temporal data exploration by marine researchers through its flexibility of data combination and display.
Chapter 9

Results

9.1 Weekly Collaborative Sessions

The system’s design and development process has been a collaboration between both computer scientists and marine biologists. The goals and technology choices utilized and implemented within the system are the product of lively, dynamic interactions between the computer scientists of the Computer Science Department and the marine biologists of the Aquatic Systems Group (ASG) at the University of Southern Maine. Initial sessions were focused primarily on discussing and crafting the specifications for the system, as proposed by the clients, and the presentation of possible, viable technologies to meet the generated requirements. The input from Lew Incze and Nicholas Wolff, marine researchers at the ASG, was absolutely essential during this process. As development of the system continued, these meetings became more technical and involved more discussion about implementation specifics and less discussion about requirements. The culmination of these weekly meetings is a novel application of available technologies to meet the needs of the research community.
This author believes that these meetings are of utmost necessity when designing a system such as this one. Without the input and guidance gained from such close, involved interactions with both the technical experts and the problem domain experts, the final product would not be what it is today.

9.2 Software Development Infrastructure

We have created a software development infrastructure to address the issues of transparency and long-term availability of individuals to work on the project. This development infrastructure consists of tools that allow the user community to determine when changes are needed and when desired functions are not being performed. These tools foster continuing development by interested parties and as a result new, better versions of the system are produced. We have also adopted the open-source development paradigm in our infrastructure. This allows for the distribution of modification costs among all involved and contributing parties.

Our system has been built on OSS. One of the major underpinnings of OSS development is the assurance that source code will be generally available; which is also one of the design requirements put forth by our clients. Not only do we want our system built on OSS, but we want the system to adopt OSS development tenets.

According to proponents of OSS, the following are a few differences that distinguish this radically different development model from that of the mainstream commercial development process (Mockus, Fielding, and Herbsleb 2002).

1. OSS systems are built by large numbers of volunteers.
2. Development work is not assigned but is rather accepted by those volunteers that choose to undertake the work.
These characteristics result in some profound effects on the development communities that undertake OSS development. The OSS development community can be characterized as a loosely coupled, geographically distributed group that rarely meets face-to-face (Mockus, Fielding, and Herbsleb 2002). As a result, an environment must be in place to allow this community to coordinate their development efforts. We have chosen to utilize TRAC, an enhanced web-based OSS system for supporting software development, to fulfill this need.

9.2.1 Wiki

In order to coordinate development efforts and ensure the continued life of the system, the OSS project must have a knowledge base that can be written collaboratively by the diverse community that is developing the OSS project.

The wiki, appropriately described by Ward Cunningham as the “simplest online database that could possibly work”, is a simple user-editable data storage medium (Tonkin 2005). The wiki seems a logical choice for supporting a collaboratively written knowledge base for an OSS system. A wiki provides a mechanism for storing knowledge that can be written and edited freely and easily by a specific community, in our case, the OSS development volunteers for the project. Wikis also tend to support wiki formatting, a means of making the articles within the knowledge base more readable. According to Tonkin, a wiki that supports collaborative writing while serving as a knowledge base will require the following:

- a page locking mechanism to prevent silent deletion of simultaneous user edits,
- a versioning system that enables tracking the history of documents,
- an efficient search function for quickly accessing appropriate data,
- effective navigation, and
• file management capabilities.

The TRAC software’s wiki provides these capabilities for us.

9.2.2 Source Code Documentation

The desire of this OSS project is to avoid the pitfall that some OSS projects fall into: the lack of user-friendly documentation. As a result, the TRAC wiki has been initially populated with the source code documentation for the JavaScript and Java classes that comprise the system. The class documentation has been created using JavaDoc\(^1\) comments for the Java classes and JSDoc\(^2\), an OSS project that follows JavaDoc commenting standards, for the JavaScript classes. JavaDoc is a tool for generating API documentation in HTML format from source code comments.

JSDoc is a tool, primarily intended for libraries of object-oriented JavaScript files, that parses inline documentation in JavaScript source files in much the same way that JavaDoc does for Java source code.

Software must be understandable to two different types of entities: compilers and people(Zokaities 2002). People need to understand the software so they can further develop, maintain and utilize the application(Zokaities 2002). Since the goal is for the project to have a continuing existence, it is imperative that the code be well documented. Human understanding of the source code is necessary for the improved development and continued utilization of the system.

\(^1\)http://java.sun.com/j2se/javadoc/
\(^2\)http://jsdoc.sourceforge.net/
Figure 9.1: Sample Documentation Generated by JavaDoc Tool

Much of the source code documentation that has already been done for the project has been consistent with the manner of documentation put forth by Robert Dunn (Dunn 1984). Each class and method has a header that provides a concise statement of the class or method’s external specifications. This prevents the code from being cluttered with inconsequential comments that may obfuscate the code itself. For methods, this includes a description of input and output data (Dunn 1984). The primary source of documentation is the automatically generated HTML API produced by the JavaDoc and JSDoc tools.

David Marin concludes that programmers tend to comment more when they are modifying code that is thoroughly commented to begin with (Marin 2005). Since source code documentation is vital to the continued existence of a project, and the OSS development paradigm involves the contributions of many developers, it is imperative to establish a precedent for well documented code within the project. As a result, programmers will comment more thoroughly when they make modifications to
an already thoroughly documented section of source code (Marin 2005).

9.2.3 Issue Tracking

One of the claims made for the OSS development paradigm is that defects are found and fixed very quickly because there are “many eyeballs” looking for the problems (Mockus, Fielding, and Herbsleb 2002). Issue tracking and debugging within any software system is an ongoing effort. Since the OSS development community is geographically distributed, our development environment must provide a centralized mechanism for tracking defects that are found and a location for the user community to suggest enhancements to the system.

The TRAC software we are utilizing includes a ticketing database that provides simple, yet effective, tracking of issues and bugs within a software project. The software development environment must also allow users to accept those tasks that they have a real passion for and wish to undertake for the project. As the central project management element of TRAC, tickets are used for project tasks, feature requests, bug reports and software support issues.\(^3\) The TRAC ticketing subsystem has been designed with the goal of making user contribution and participation as simple as possible. Figure 9.2 is a state diagram illustrating the path tickets take through TRAC’s ticketing subsystem.

A collaborator creates a new ticket which may indicate an application defect or a request for a new feature. The new ticket is either accepted by someone who is

\(^3\)http://trac.edgewall.org/wiki/TracGuide
interested in performing the work to fix the defect or add the new feature. The new
ticket can also be resolved immediately and the ticket is then marked as closed. If
the ticket is accepted, then it is marked as assigned to the interested party. The
assigned ticket can be unassigned, for one reason or another, and the ticket’s status
is changed back to new. Once the work has been completed for the assigned ticket,
it is marked as resolved and marked closed. Through quality assurance by the user
community, if it is determined that the issue is not fixed or the enhancement is not
working according to its specifications, then the ticket may be reopened.

9.2.4 Subversion (SVN)

The development environment must allow a diverse community of developers from
various locations to contribute to the software system easily and effectively. Subver-
sion (SVN) is an open-source version control system that manages files and directories.
SVN tracks changes made over time and allows users to recover older versions of files. SVN can function over networks, allowing various contributors to modify and manage the same set of files, thus fostering collaboration. The TRAC software system that we have chosen to employ has an integrated version control system browser which allows users to quickly view and navigate the source code repository.

9.3 Performance

Performance was not our primary concern during the development of the system. Our main focus was developing a system that illustrated the feasibility of using open standards to develop an open source system capable of facilitating EAM. During our development, we used spatial indexing to improve the performance of spatial queries within the system. Figure 9.3 shows the results of these optimizations. With respect to requests, request response time is dependent upon the number of observations being returned. Data integration can take anywhere from a minute for smaller queries to up to 20 minutes when integrating the entire dataset within the GMBIS database.

9.4 Areas for Improvements

We have identified several improvements or enhancements that could be added to the system. The improvements include:

- **Tools for Collaborative Mapping**
  Since EAM relies on the interaction between stakeholders from multiple disciplines, a mechanism for facilitating collaboration between the various stakeholders would be extremely useful. Collaborative mapping mechanisms could include the ability to share and save map contexts.
• **Vary Symbolization by Abundance**

EAM is concerned with the distribution of biodiversity across multiple species and taxonomic groups. It would be very beneficial to have a mechanism for visually distinguishing the species by their abundance. This means adding a “how much” component to the “where” component of the biological data. This could be achieved through bubble graphs for smaller areas of interest or color ramps for larger areas where bubble graphs tend to obfuscate the underlying data.

• **Ability to Import Physical Data from More Models**

As concrete progress is continuing to be made in the area of physical oceanographic modeling, more and more models are coming online. Through the work of the Modeling Committee of the Gulf of Maine’s Ocean Data Partnership\(^4\),

\[^4\text{http://www.gomodp.org/modeling-committee}\]
more of these data models are providing standards-based means of access to their data. Since a tenet of EAM is to incorporate new data as it becomes available, the more data that can be aggregated and integrated the better.

- **Extending the Conceptual Model**

  Extending the conceptual model is key to allowing different types of data to be aggregated and integrated within the system. This includes support for other geometry types, like linestrings and raster data. One possible method for including raster data would be to polygonize it and import it into the system.

  The system that we have developed is in its infancy and it has room for improvement. It is a new tool that applies existing standards and technologies and is a step in the right direction.
Chapter 10

Conclusions

We have designed and developed a functional, distributed, Internet-based GIS portal framework for marine researchers. The open source software system has been developed in a collaborative manner, utilizing a number of software design principles.

The portal was developed to meet the following specific goals set forth by the marine researchers at the Aquatic Systems Group of the University of Southern Maine.

1. The ability to choose a data set, query a date range and a species (or multiple species) and display observation locations on a map and view a table of the underlying data.

2. The ability to download the mapped data in a variety of formats.

3. The ability to map “zero observations”, or observations where an attempt to catch a species was unsuccessful, along with “positive observations,” or observations where an attempt to catch a species was successful.

4. The ability to add and remove various static base layers (e.g. bathymetry, physioregions, sediment and others).

5. The ability to add WMS and WFS layers from other servers (e.g. GoMOOS)
6. The ability to serve GMBIS data as WMS and WFS layers for other web mapping portals to use.

7. The ability to query, map and download OBIS data.

8. The ability to perform spatial queries on biogeographic and oceanographic data.

9. The ability to interact dynamically with the map, for example, zooming and panning.

10. The ability to compare and contrast data, for example, by toggling layers of similar data sets in order to make assessments.

11. The ability to integrate biological data with environmental data in a single map or table.

During the course of the project, we identified several key technical challenges to overcome to develop a GIS portal for EAM that meets the needs of clients and end users. We provided solutions to the following issues:

1. System Extensibility
2. Support for Multiple Data Sources
3. Data Integration and Delivery
4. User Interface Design

The server-side components of the GIS portal, namely GeoServer, the PostGIS datastores and the various components built using Java technology, were high performance, reliable and performed without issue. On the client-side, the AJAX components and widgets developed using jQuery provide the user with a rich client-side interface and allows them great flexibility in selecting, displaying, navigating, exploring and downloading the available data. OpenLayers provided basic controls to allow
the user to interact with the map widget, allowing for dynamic map viewing and navigation. Utilizing OGC and OBIS standards, we have developed a distributed, Internet-based portal that is highly interoperable, allowing users to map and download other external data sources through the GIS portal. It is important to note that the current system is compatible only with the Mozilla Firefox\(^1\) web browser and work is currently underway to make the system cross-browser compatible.

Finally, we have also created a software development infrastructure. This infrastructure is based on a TRAC project site\(^2\) that includes a Wiki, software design documentation, JSDoc and JavaDoc documentation, an issue tracking system and a source code browser. The site provides a support mechanism to allow the system to build on the OSS development model, allowing various interested parties to contribute to the project. This also ensures that the system will carry into the future.

\(^1\)http://www.mozilla.com/firefox

\(^2\)The TRAC site is available at http://130.111.123.60:81/projects/gmbis_dataintegrator
Appendix A

Glossary of Terms and Abbreviations

AJAX *Asynchronous JavaScript and XML*

AJAX refers to web development techniques used for creating interactive web applications through the asynchronous exchange of small amounts of data with the server “behind the scenes.” The result is that entire web pages do not have to be reloaded each time there is a need to fetch data from the server and the web page’s interactivity, speed, functionality and usability are increased.
**API** *Application Programming Interface*

An API is a source code interface that an operating system or library provides to support requests for services to be made of it by computer programs.

---

**CoML** *Census of Marine Life*

CoML\(^1\) is a global network of researchers in more than 80 nations engaged in a 10-year scientific initiative to assess and explain the diversity, distribution, and abundance of life in the oceans.

---

**coverage**

A single theme or layer of data (e.g. vegetation, roads, temperature) in a geographic database.

---

**CSV** *Comma Separated Value*

CSV file format is a file type that stores tabular data in a delimited text file, which uses a comma to separate values.

---

**DBMS** *Database Management System*

A DBMS is a collection of software tools, procedures and rules for the creation, management and use of databases.

\(^{1}\)http://www.CoML.org/
**EAM Ecosystem Approaches to Management**

*From NOAA:* EAM provides a comprehensive framework for marine and coastal resource decision making and takes into consideration a wider range of relevant ecological, environmental, and human factors bearing on societal choices regarding resource use.

---

**ecosystem**

An ecosystem is defined as a geographically specific collection of animals, plants, and supporting environmental processes.

---

**ESRI Environmental Systems Research Institute**

ESRI was founded in 1969 as a privately held consulting firm and its business involves the development and support of GIS software. Their Shapefile format is a popular and widely used geospatial vector data format for GIS software.

---

**FGDC Federal Geographic Data Committee**

The FGDC\(^2\) is an interagency committee that promotes the coordinated development, use, sharing, and dissemination of geospatial data on a national basis within the United States.

\(^2\)[http://www fgdc gov/](http://www.fgdc.gov/)
**GCMD** *Global Change Master Directory*

The GCMD\(^3\) is a directory that enables users to locate and obtain access to Earth science data sets and services relevant to global change and Earth science research.

---

**GIS** *Geographic Information System*

GIS refers to any system for capturing, storing, analyzing and managing data and associated attributes which are spatially referenced to Earth.

---

**GMBIS** *Gulf of Maine Biogeographic Information System*

GMBIS\(^4\), the data component of GoMA, supports the regional needs for access to biological and biogeographic data from the Gulf of Maine area for research, public policy (management) and education.

---

**GML** *Geography Markup Language*

OpenGIS GML\(^5\) is an XML schema defined by the OGC and used to express geographic features.

---

\(^3\)http://gcmd.nasa.gov/

\(^4\)http://research.usm.maine.edu/gulfofmaine-census/data-mapping/

\(^5\)http://www.opengeospatial.org/standards/gml
GoMA  *Gulf of Maine Area Program*

GoMA\(^6\) is one of seven initial field projects of CoML, and was selected as the ecosystem pilot study for CoML. Its goal is to gain enough knowledge to enable EAM in a large marine environment.

---

GoMOOS  *Gulf of Maine Ocean Observing System*

GoMOOS\(^7\) is a national pilot program designed to distribute hourly oceanographic data from the Gulf of Maine.

---

GUI  *Graphical User Interface*

A GUI is used to facilitate interaction and communication between a human user and a computer through the use of windows, pointers, icons and widgets.

---

HTML  *Hyper Text Markup Language*

HTML provides a means to describe the structure of text-based information in a document by denoting certain text as links, headings, paragraphs, lists, and so on. HTML can supplement text with interactive forms, embedded images, and other objects and is the predominant format for documents on the World Wide Web.

---

\(^6\)http://www.usm.maine.edu/gulfofmaine-census/
\(^7\)http://www.gomoos.org/
HTTP  *Hypertext Transfer Protocol*

HTTP is a communications protocol for the transfer of information on intranets and the World Wide Web. Its original purpose was to provide a way to publish and retrieve hypertext pages over the Internet.

---

IDE  *Integrated Development Environment*

An IDE is a software application that provides comprehensive facilities to computer programmers for software development. Notable Java IDEs are Eclipse⁸ and Netbeans⁹.

---

JDK  *Java Development Kit*

The JDK is a Sun Microsystems product aimed at Java developers and provides a Java loader, compiler and various other components that facilitate the development of Java-based software.

---

JSP  *Java Server Page*

JSP¹⁰ is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request.

---

⁸http://www.eclipse.org/
⁹http://www.netbeans.org/
¹⁰http://java.sun.com/products/jsp/
layer

A subset of a geographic database containing data associated with a particular theme.

metadata

Data about data or descriptions of the data in a data file, including date of collection, sources, map projection, scale, quality, format and custodian.

NOAA National Oceanic and Atmospheric Administration

NOAA\textsuperscript{11} is a federal agency of the United States that is focused on the condition of the oceans and the atmosphere.

OBIS\textsuperscript{12} Ocean Biogeographic Information System

OBIS is an on-line, open-access, globally-distributed network of systematic, ecological, and environmental information systems.

\textsuperscript{11}http://www.noaa.gov/
\textsuperscript{12}http://www.iobis.org/
OGC *Open Geospatial Consortium, Inc.*

The OGC\(^{13}\) is a non-profit, international, voluntary consensus standards organization that is involved in the development of standards for geospatial and location based services.

---

OSS *Open Source Software*

OSS is computer software for which the human-readable source code is made available under a copyright license (or arrangement such as the public domain) that meets the Open Source Definition, permitting users to use, change, and improve the software, and to redistribute it in modified or unmodified form and is often developed in a public, collaborative manner.

---

PHP *Hypertext Preprocessor*

PHP\(^{14}\) is a computer scripting language, originally designed for producing dynamic web pages, that is mainly used in server-side scripting.

---

raster

A data structure that organizes data values as entries in a matrix.

---

\(^{13}\)http://www.opengeospatial.org/

\(^{14}\)http://www.php.net/
SQL *Structured Query Language*

SQL is a database computer language designed for the retrieval and management of data in relational DBMSs.

---

SVN *Subversion*

SVN\(^{15}\) is a version control system that is used to maintain current and historical versions of files such as source code, web pages, and documentation.

---

vector

A coordinate-based spatial data structure in which the data are represented, using computer graphics, as points, lines and polygons.

---

WFS *Web Feature Service*

The OpenGIS WFS Implementation Specification\(^{16}\) allows a client to retrieve and update geospatial data encoded in GML from multiple Web Feature Services. The specification defines interfaces for data access and manipulation operations on geographic features, using HTTP as the distributed computing platform. Via these interfaces, a Web user or service can combine, use and manage geodata – the feature information behind a map image – from different sources.

---

\(^{15}\)http://subversion.tigris.org/

\(^{16}\)http://www.opengeospatial.org/standards/wfs
**WMS** *Web Map Service*

The OpenGIS WMS Implementation Specification\(^\text{17}\) provides an interface in support of the creation and display of map-like views of information that come simultaneously from multiple remote and heterogeneous sources.

---

**WYSIWYG** *what-you-see-is-what-you-get*

WYSIWYG is used in computing to describe a system in which content during editing appears very similar to the final product. It can also be used to mean that the GUI represents the current state of the system.

---

**XML** *Extensible Markup Language*

XML is a general-purpose specification for creating custom markup languages that allows its users to define their own elements.

---

\(^{17}\)http://www.opengeospatial.org/standards/wms
Appendix B

OGC Styled Layer Descriptor (SLD) Example

The following pages are an example of an OGC Styled Layer Descriptor.\(^1\) The following SLD was used to style the map of Figure 2.10.

\(^1\)http://www.opengeospatial.org/standards/sld
<NamedLayer>
  <Name>GoM Land</Name>
  <UserStyle>
    <Title>Style for GoM Land Masses</Title>
    <Abstract>
    </Abstract>
    <FeatureTypeStyle>
      <Name>Countries</Name>
      <Rule>
        <Name>usaStyle</Name>
        <Title>USA</Title>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>country
            </ogc:PropertyName>
            <ogc:Literal>USA</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#99FF99</CssParameter>
          </Fill>
          <Stroke>
            <CssParameter name="stroke">#999999</CssParameter>
          </Stroke>
        </PolygonSymbolizer>
      </Rule>
      <Rule>
        <Name>canadaStyle</Name>
        <Title>Canada</Title>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>country
            </ogc:PropertyName>
            <ogc:Literal>Canada</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#CCFFCC</CssParameter>
          </Fill>
          <Stroke>
            <CssParameter name="stroke">#CCFFCC</CssParameter>
          </Stroke>
        </PolygonSymbolizer>
      </Rule>
    </FeatureTypeStyle>
  </UserStyle>
</NamedLayer>
Appendix C

The OBIS Schema

The OBIS Schema\(^1\) is the content standard used by OBIS, the Ocean Biogeographic Information System. It represents an extension of the Darwin Core Version 2 and is designed for marine biodiversity data, specifically to record the capture or observation of a particular species or other taxonomic group at a particular location. The following pages outline the OBIS Schema-Version 1.1, which was published in July 2005.

\(^1\)http://www.iobis.org/tech/provider/schemadef1/
<table>
<thead>
<tr>
<th>Name</th>
<th>Required Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Last Modified</td>
<td>Optional for OBIS (Required for GBIF/Darwin Core servers) Date Time The date and time the record was last modified. Format: ISO 8601</td>
</tr>
<tr>
<td>Institution Code</td>
<td>Required Text A &quot;standard&quot; code identifier that identifies the institution to which the collection belongs, if there is one.</td>
</tr>
<tr>
<td>Collection Code</td>
<td>Required Text A unique alphanumeric value which identifies the collection within the institution (e.g. FishBase).</td>
</tr>
<tr>
<td>Catalog Number</td>
<td>Required Text / Numeric A unique alphanumeric value which identifies an individual record within the collection, i.e. the key.</td>
</tr>
<tr>
<td>Record URL</td>
<td>Optional Text Gives the web address of the page where more information on this particular record (not on the whole dataset) can be found.</td>
</tr>
<tr>
<td>Scientific Name</td>
<td>Required Text The full name of lowest level taxon the Cataloged Item can be identified as a member of; includes genus, specific epithet, and subspecific epithet (zool.) or infraspecific rank abbreviation, and infraspecific epithet (bot.).</td>
</tr>
<tr>
<td>Basis of record</td>
<td>Highly Recommended Text An abbreviation indicating whether the record represents an observation (O) (this can include a visual observation, a survey catch, a commercial landing record, etc), a collected living organism, such as a tree in a botanical garden (L), a specimen in a collection/museum (S), a collected germplasm/seed (G), a photo (P), or derived from literature, where original basis unknown (D).</td>
</tr>
<tr>
<td>Source</td>
<td>Optional Text OBIS does not encourage the use of this field - it is a legacy field. Indicates who gave the record to the data provider.</td>
</tr>
<tr>
<td>Citation</td>
<td>Highly Recommended Text Indicates how this record should be attributed if used. (e.g. &quot;Jones, T. 2005. Electronic atlas of eel distributions version 3. <a href="http://www.eels.com">www.eels.com</a>&quot;).</td>
</tr>
<tr>
<td>Kingdom</td>
<td>Highly Recommended Text The kingdom to which the organism belongs</td>
</tr>
<tr>
<td>Phylum</td>
<td>Optional Text The phylum (or division) to which the organism belongs</td>
</tr>
<tr>
<td>Class</td>
<td>Optional Text The class name of the organism.</td>
</tr>
<tr>
<td>Order</td>
<td>Optional Text The order name of the organism</td>
</tr>
<tr>
<td>Family</td>
<td>Optional Text The family name of the organism</td>
</tr>
<tr>
<td>Genus</td>
<td>Highly Recommended-when known Text The genus name of the organism.</td>
</tr>
<tr>
<td>Subgenus</td>
<td>Optional Text The subgenus name of the organism</td>
</tr>
<tr>
<td>Species</td>
<td>Highly Recommended-when known Text The specific epithet of the organism</td>
</tr>
<tr>
<td>Subspecies</td>
<td>Optional Text The sub-specific epithet of the organism</td>
</tr>
<tr>
<td>Scientific Name Author</td>
<td>Optional Text The author of a scientific name. Example: (Hastings, 1986)</td>
</tr>
<tr>
<td>Identified By</td>
<td>Optional Text The name(s) of the person(s) who applied the Scientific Name to the Cataloged Item.</td>
</tr>
<tr>
<td>Year Identified</td>
<td>Optional Numeric The year portion of the date when the Collection Item was identified; four digits [1999..9999], e.g., 1900, 2002.</td>
</tr>
<tr>
<td>Month Identified</td>
<td>Optional Numeric The month portion of the date when the Collection Item was identified; as two digits [01..12].</td>
</tr>
<tr>
<td>Day Identified</td>
<td>Optional Numeric The day portion of the date when the Collection Item was identified; as two digits [01..31].</td>
</tr>
<tr>
<td>Type Status</td>
<td>Optional Text Indicates the kind of nomenclatural type that a specimen represents, for example holotype, syntype, paratype, lectotype, paralectotype, neotype, schizotype, allotype, hapantotype.</td>
</tr>
<tr>
<td>Collector Number</td>
<td>Optional Text An identifying &quot;number&quot; (really a string) applied to specimens (in some disciplines) at the time of collection.</td>
</tr>
<tr>
<td>Field Number</td>
<td>Optional Text A &quot;number&quot; (really a string) created at collection time to identify all material that resulted from a collecting event, e.g. station or sample numbers</td>
</tr>
<tr>
<td>Collector</td>
<td>Optional Text The name(s) of the collector(s), people or organisation(s) responsible for collecting the specimen, taking the observation, fishing the catch or doing whatever is the underlying basis of the record.</td>
</tr>
<tr>
<td>Year Collected</td>
<td>Highly Recommended Numeric The year (expressed as an integer) the sample/observation/record event occurred. The full year should be expressed (e.g. 1972 must be expressed as &quot;1972&quot; not &quot;72&quot;). Where the event covers a range of years in a single data point, indicates the mid-point of that range.</td>
</tr>
<tr>
<td>Start Year Collected</td>
<td>Optional Numeric For samples/observations/record events that were taken over time this gives the start year of the collecting event. The full year should be expressed (e.g. 1972 must be expressed as &quot;1972&quot; not &quot;72&quot;). Must always be a four digit integer</td>
</tr>
<tr>
<td>End Year Collected</td>
<td>Optional Numeric For samples/observations/record events that were taken over time this gives the end year of the collecting event. The full year should be expressed (e.g. 1972 must be expressed as &quot;1972&quot; not &quot;72&quot;). Must always be a four digit integer</td>
</tr>
<tr>
<td>Month Collected</td>
<td>Highly Recommended Numeric The month of year the sample/observation/record event occurred in the field. Where the event covers a range of values for month, indicates the mid-point of that range. Leave blank if even spans multiple months.</td>
</tr>
<tr>
<td>Start Month Collected</td>
<td>Optional Numeric For samples/observations/record events that were taken over time this gives the start month of the collecting event. Possible values range from 01..12 inclusive</td>
</tr>
<tr>
<td>End Month Collected</td>
<td>Optional Numeric For samples/observations/record events that were taken over time this gives the end month of the collecting event. Possible values range from 01..12 inclusive</td>
</tr>
<tr>
<td>Day Collected</td>
<td>Highly Recommended Numeric The day of the month the sample/observation/record event occurred in the field. Possible value ranges from 01..31 inclusive. Where the event covers a range of values for day, indicates the mid-point of that range. Leave blank if event spans multiple months.</td>
</tr>
<tr>
<td>Start Day Collected</td>
<td>Optional Numeric For samples/observations/record events that were taken over time this gives the start day of the collecting event. Possible value ranges from 01..31 inclusive</td>
</tr>
<tr>
<td>End Day Collected</td>
<td>Optional Numeric For samples/observations/record events that were taken over time this gives the end day of the collecting event. Possible value ranges from 01..31 inclusive</td>
</tr>
<tr>
<td>Name</td>
<td>Required</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Julian Day</td>
<td>Optional</td>
</tr>
<tr>
<td>Start Julian Day</td>
<td>Optional</td>
</tr>
<tr>
<td>End Julian Day</td>
<td>Optional</td>
</tr>
<tr>
<td>Time of Day</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Start Time of Day</td>
<td>Optional</td>
</tr>
<tr>
<td>End Time of Day</td>
<td>Optional</td>
</tr>
<tr>
<td>Time Zone</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Continent Ocean</td>
<td>Optional</td>
</tr>
<tr>
<td>Country</td>
<td>Optional</td>
</tr>
<tr>
<td>State Province</td>
<td>Optional</td>
</tr>
<tr>
<td>County</td>
<td>Optional</td>
</tr>
<tr>
<td>Locality</td>
<td>Optional</td>
</tr>
<tr>
<td>Longitude</td>
<td>Required</td>
</tr>
<tr>
<td>Start Longitude</td>
<td>Optional</td>
</tr>
<tr>
<td>End Longitude</td>
<td>Optional</td>
</tr>
<tr>
<td>Latitude</td>
<td>Required</td>
</tr>
<tr>
<td>Start Latitude</td>
<td>Optional</td>
</tr>
<tr>
<td>End Latitude</td>
<td>Optional</td>
</tr>
<tr>
<td>Coordinate Precision</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Start/End Coordinate Precision</td>
<td>Optional</td>
</tr>
<tr>
<td>Name</td>
<td>Required</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Bounding Box</td>
<td>Optional</td>
</tr>
<tr>
<td>Minimum Elevation</td>
<td>Optional</td>
</tr>
<tr>
<td>Maximum Elevation</td>
<td>Optional</td>
</tr>
<tr>
<td>Minimum Depth</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Maximum Depth</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Depth Range</td>
<td>Optional-not preferred</td>
</tr>
<tr>
<td>Temperature</td>
<td>Optional</td>
</tr>
<tr>
<td>Sex</td>
<td>Optional</td>
</tr>
<tr>
<td>Life Stage</td>
<td>Optional</td>
</tr>
<tr>
<td>Preparation Type</td>
<td>Optional</td>
</tr>
<tr>
<td>Individual Count</td>
<td>Optional</td>
</tr>
<tr>
<td>Sample Size</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Observed Individual Count</td>
<td>Highly Recommended</td>
</tr>
<tr>
<td>Observed Weight</td>
<td>Optional</td>
</tr>
<tr>
<td>Previous Catalog Number</td>
<td>Optional</td>
</tr>
<tr>
<td>Relationship Type</td>
<td>Optional</td>
</tr>
<tr>
<td>Related Catalog Item</td>
<td>Optional</td>
</tr>
<tr>
<td>Notes</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**OBIS EXPERIMENTAL FIELDS:** The following are not part of the current OBIS Schema, but are under consideration for future versions. They represent format

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GML Feature</td>
<td>Optional</td>
<td>Text</td>
<td>Geographic Markup Language(GML) description of the feature for representing complex shapes such as lines and polygons, per Open GIS Consortium (OGC) standards - <a href="http://www.opengis.net/gml/01-029/GML2.html">http://www.opengis.net/gml/01-029/GML2.html</a>.</td>
</tr>
</tbody>
</table>
Bibliography


Boston: Addison-Wesley.


