Metadata Subsetting Framework

Cherri M. Pancake and Daehyun Yoon

Department of Computer Science
Oregon State University
Corvallis, OR 97331

pancake@cs.orst.edu
yoon@cs.orst.edu

February 13, 2003

Technical Report

Abstract

The amount of research data online is growing exponentially, scattered across a multitude of locations and stored in various formats on a wide variety of platforms. The value of metadata which assists us in determining the relevance, location, and accessibility of data and related research, has become correspondingly more important. Although there are a number of international standards that govern the format and structure of metadata in specific disciplines, the complexity of these standards makes it difficult for data owners and clearinghouses to create and manage metadata effectively. The Metadata Subsetting Framework is a system that allows metadata clearinghouses to define a subset of a metadata standard, containing the fields most relevant to the specific domain. The system automatically generates an interface specifically customized for the subset, with capabilities for validating user inputs against constraints expressed in the subset. Implemented as a collection of Java components, the system produces a complete in-memory representation of XML Schema documents expressing the structural and data constraints of a subsetted metadata standard. A knowledgeable user builds the subset by manipulating these constraints via a web-based interface. Subsetting makes the metadata entry process easier and less error-prone, thereby improving the quality of metadata.

Table of Contents

1. Introduction 1
1.1. Why the Quality of Metadata Is Critical 1
1.2. Motivation 2
1.3. Organization of This Report 3

2. Metadata Subsetting Framework 4
2.1. Architecture Overview 4
2.2. Schema Model Builder 6
2.2.1. XML Schema Language 6
2.2.2. Parsing Schema Documents 7
2.3. Subset Generator 10
2.3.1. Subsetting a Metadata Standard 11
2.3.2. Building a Subset 14
2.4. XSO Serializer 15
2.5. HTML Generator 18
2.6. Metadata Framework Servlets 18

3. Usability Issues for the Metadata Subsetting Framework 23
3.1. Types of Errors 23
3.2. Execution-Evaluation Framework 23
3.3. Characteristics of Web-based Applications 25
3.4. Designing for the Web Applications 26

4. Selected Design and Implementation Details 34
4.1. Building XML Schema Object Group 34
4.1.1. Modeling XML Schema Components 34
4.1.2. Syntax Analyzer 36
4.1.3. Symbol Resolver 39
4.2. Generating HTML Pages for Metadata Entry 41
4.3. Subsetting Schema Objects 43

5. Conclusions and Future Work 45
5.1. Contribution of This Work 45
5.2. Related Work 45
5.3. Challenges Faced 46
5.4. Future Work 48

Appendix A 50
Appendix B 81
Appendix C 82

Bibliography 84

1. Introduction

Scientific data and information are virtually unlimited in their quantity. With the advancement of technology, the collection of data has been made simpler and the accuracy with which data can be compiled has increased dramatically. Although the amount of data that exists today is enormous, it is likely to continue to grow as technology continues to improve. As the quantity of scientific data continues to grow exponentially, there needs to be a description of the content, quality, condition, and other characteristics of data in a standardized format using a common set of terms. One of the important issues in this era of "information explosion" is the topic known as information discovery [57]. This is the study of the location and identification of information that is relevant, identical, similar, or related to the requested data of query. As outlined by Sheth, in the future, the primary issue will not be the efficient processing of data already known to be relevant, but the determination of which data is relevant and where it is located [43, 44]. Metadata helps us to identify the relevance of data by allowing us to quickly discover what data we have (identification), what it means (definition), where it is (location), how it got there (source), and how we could get it (access) [46].

International standards dictating the format and structure of metadata are available within various disciplines. The Content Standard for Digital Geospatial Metadata from the Federal Geographic Data Committee (FGDC), for example, was developed in response to a Presidential decree in 1994 [14]. It provides a common set of terminology and definitions, and was intended to facilitate discovery of electronic resources and interoperable online metadata, in support of a broad range of purposes and business models. Another example, the Ecological Metadata Language from the U.S. Long Term Ecological Research Network, was developed to serve the needs of the ecological community [12]. It was based on prior work by the Ecological Society of America and associated efforts [26]. Standards such as these provide a consistent approach and format for the description of data characteristics in particular disciplines, potentially enabling users to access the information required to decide if data will fulfill specific needs.

1.1. Why the Quality of Metadata Is Critical

Tremendous value may be found in sharing and cooperatively synthesizing data. The importance of data sharing and standardization has been recognized by professionals, researchers, and government officials in recent years [5, 36]. In addition to assisting users locate and access the data they are looking for—which may be spread across many physical locations in various formats on a wide variety of platforms—metadata also provides us several key benefits.

- Metadata helps organizations maintain their investment in data. As an organization's personnel changes over time, information about data may be lost and data may lose its value. Later workers may have little understanding of the existence, content, and uses for digital data, and find that they can't trust the results. Even if reports exist describing when, where, and how the data was created, it still may be difficult to

locate and use data sources if each has its own format. Standardized metadata descriptions of data content can encourage the appropriate archiving and use of data.

- Metadata allows data to transcend people and time. Data may be used many years from now. Without metadata, intimate details about the data (such as quality control and the meanings of attribute values) may be lost or forgotten, and it may not be possible to reuse the data. Data collected when humans first landed on the Moon is still being used today, and it is reasonable to assume that today's scientific data will be used in the future in order to study climate change, ecosystems, and other natural processes. Metadata standards increase the value of such data by facilitating data sharing through time and space.
- Metadata provides information for data catalogs and clearinghouses. Few organizations can afford to create all the data they need. Also, data created by one organization may be even more useful to others. By making metadata available through data catalogs and clearinghouses, organizations can access data, find partners with whom to share data collections and maintenance efforts, and locate customers for data.
- Metadata provides information to aid data transfer. Metadata can also help the organization receiving the data by providing information about where the data came from, how it was collected, how old and how accurate it is, what coordinate system was used, and what the attribute and code definitions are.

1.2. Motivation

Although the value of metadata is tremendous, data owners are consistently discouraged by the difficulty of creating high-quality metadata [37]. Scientific researchers, for example, are often unfamiliar with the standard itself, and are overwhelmed by the number of fields in typical metadata standard [48]. For instance, there are over 300 fields in the FGDC's Content Standard for Digital Geospatial Metadata. Not all the fields have to be filled in. Many can be left blank if they are not applicable to the data the metadata is describing. Nonetheless, the complexity and comprehensiveness of these standards makes it difficult for data owners to properly create metadata entries because they must figure out which fields are the "right" ones for storing key information, and which other fields are also required by the standard. It is also difficult to determine whether or not metadata entry is being performed correctly.

From the viewpoint of organizations responsible for collecting and managing metadata, some fields in a standard may not be needed for a specific field of study. For example, the Oregon Coastal Geospatial Clearinghouse has users fill in only 28 of the 300+ CSDGM fields [11, 50]. Thus, while organizations may not invent their own metadata standard, they typically select just a subset of the standard. This eliminates the need to create yet another standard and reduces the difficulty of metadata creation, while remaining compliant to the standard.

Another issue of data clearinghouses managing scientific data is the fact that they are typically managed by scientists, due to budget constraints [18]. These domain experts are not usually experts in computing, and may not be familiar with the programming languages and expertise required to build and manage a tool for integrating metadata entries.

To address these problems, we present a solution built around the technology of XML Schema Languages. We begin with the observation that several recent metadata standards, including Ecological Metadata Language, are written in schema languages to express constraints on structure and data. Other specifications can be converted to schema documents relatively easily.

Given a metadata standard expressed in XML Schema Language, we will show that a set of HTML form pages reflecting the structure of schema documents can be generated, eliminating the needs to manually write tedious HTML tags. Moreover, a generalized tool can perform validation of input documents against the constraints expressed in the schema documents, ensuring the integrity of input data with respect to specification. Finally, the data persistency issue is resolved by taking advantage of the availability of several tools that map user inputs expressed in XML syntax to database systems. This framework will allow the rapid development of a metadata entry portal site, tailored for a specific scientific domain. Site managers only need to supply customized constraints for their metadata specification—which forms a subset of an existing metadata standard—without worrying about writing HTML and other specialized languages.

1.3. Organization of This Report

This section provided an introduction to scientific metadata and the challenge that data owners face when creating a metadata entry. It also addressed the motivation for developing a framework where scientists and engineers with little or no experience in computer programming can rapidly develop a portal site for collecting metadata entries.

Section 2 describes the Metadata Subsetting Framework. After an overview of the architecture and its sub-components, there is an in-depth discussion of how Schema Languages can be used to address the issues outlined previously.

Section 3 discusses how our solution can assist users to avoid errors in the metadata entry process. The characteristics of errors in human-computer interaction are discussed, followed by an analysis of how to prevent, detect, and recover from errors in web-based applications.

Section 4 discusses selected implementation details of the Metadata Subsetting Framework. Topics covered here include modeling XML Schema components, generating HTML pages, and implementing the subsetting tool.

Section 5 concludes the report with a discussion of the contributions of this project, and how it differs from other tools currently available. We outline the lessons learned in designing and implementing the system, and propose some potential future enhancements.

2. Metadata Subsetting Framework

In the previous section, we established the motivation for an XML Schema-based metadata subsetting framework for scientific communities. The goal is to allow them to rapidly develop a subset of an existing metadata standard in order to meet the needs of a particular clearinghouse or organization.

Our design offers two advantages. First, the Metadata Subsetting Framework was designed with the objective of improving metadata quality by assisting users to perform the metadata entry process correctly, by making it easier for data clearinghouses to construct subsets of a metadata standard, and by allowing them to quickly build a metadata portal site with little or no special expertise. Second, the architecture is modular and extensible, for the benefit of other framework developers. Its high-level API provides a foundation on which new tools can be easily built. The framework is object-oriented and platform independent in terms of both operating system and database management systems, and it is modular so that changes in one area have minimal effects on others.

2.1. Architecture Overview

As shown in Figure 2.1, the architecture of the Metadata Subsetting Framework consists of the following components.

- Schema Model Builder: parses input schema documents into a collection of object representations in memory.
- Subset Generator: allows users to manipulate a set of object models representing schema documents.
- XSO Serializer: serializes the object models that have been processed by the previous two components.
- HTML Generator: produces a set of HTML pages containing form fields for data entry.
- Metadata Framework Servlets: group of servlets responsible for managing the entire framework

When a subset designer opens a web session and requests a metadata standard to a subset, the HTTP server receives the request and delegates the request to the Metadata Framework Servlet.

Phase I (Subset Definition): The servlet fetches the relevant schema documents from the file system and invokes the Schema Model Builder which, in turn, parses the input schema documents and produces a set of object models. The models reside in the server's memory, associated with the user's session until the session expires. The Subset Generator then dynamically produces HTML forms that are displayed in the browser window. The user builds a subset by selecting the desired fields and editing the constraints for each.

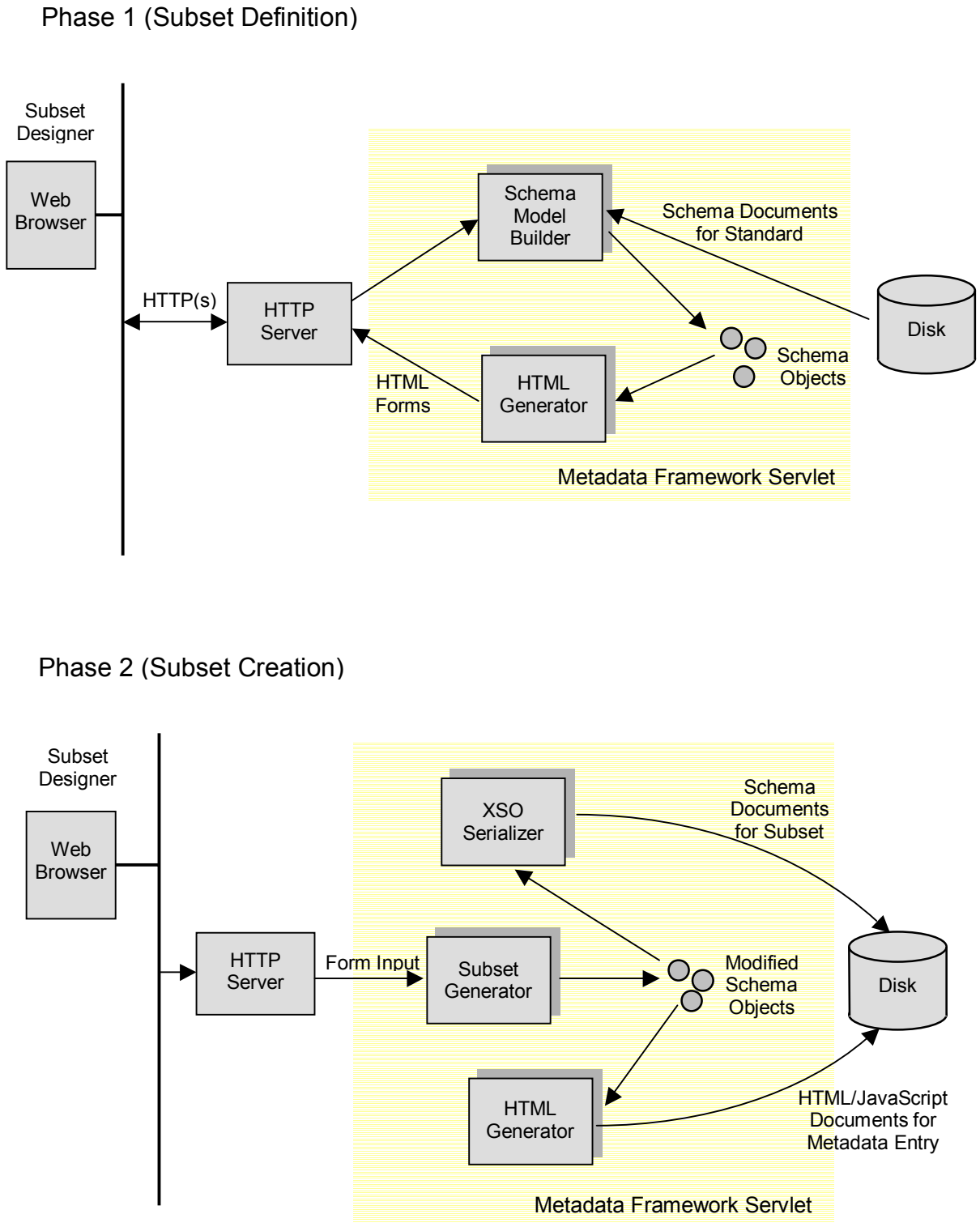


Figure 2.1. Schema Metadata Framework architecture

Phase II (Subset Creation): When the form input is submitted, the Subset Generator parses it and modifies the schema objects in memory accordingly. The XSO Serializer then serializes the objects into a set of schema documents, meanwhile the HTML Generator produces a set of HTML form documents and JavaScript pages for front-end validation (checking the integrity of form inputs). The framework registers these documents with the subset, and stores them on disk. The following sections describe each component in detail.

2.2. Schema Model Builder

The Schema Model Builder (SMB) parses schema documents and produces object representations. This involves two steps, which we will refer to as *XML-parsing* and *Schema-parsing*. First, the schema documents, written in XML, are XML-parsed into an internal representation tree using the Document Object Model (DOM), a standard API for XML processing. The SMB then traverses the internal tree, resolving each reference expressed as XML element's attribute. The result is an in-memory representation of XML schema documents. This representation reveals the structure that instance documents conform to. We will refer to this in-memory representation as XML Schema Object Group (XSOG).

2.2.1. XML Schema Language

The XML Schema Definition Language, developed by The World Wide Web Consortium, is a language for describing and constraining the content of XML documents. It is "a formalization of the constraints, expressed as rules or a model of structure, that apply to a class of XML documents, describing the constraints on structures as well as data for instance documents" [47]. Schema Language was born out of the frustration developers were experiencing with DTD (Document Type Definition). Some shortcomings of DTDs are:

- They are written in a non-XML syntax.
- They have no support for namespaces (scoping).
- They only offer extremely limited data typing. DTDs can only express the type of attributes in terms of explicit enumerations and a few coarse string formats; there is no facility for describing numbers, dates, currency values, and so forth. Furthermore, DTDs have no ability to express the type of character data in elements.
- They have a complex and fragile extension mechanism based on little more than string substitution.

In contrast, the XML Schema Language offers a variety of new features designed to address the shortcomings of DTD.

- Richer data types, such as booleans, numbers, dates and times, URIs (Universal Resource Identifiers), integers, decimal numbers, real numbers, intervals of time, etc.
- Ability to extend the predefined types by creating aggregate types
- User-defined types can be created (such as a "sequence" datatype with related "nucleicAcidSequence" and "aminoAcidSequence" elements)
- Attribute grouping, so that common attributes that apply to all elements in a schema can be assigned as a group
- Refinable inheritance mechanism (probably the most significant new feature in XML Schemas)
- Namespace support to allow the co-existence of multiple schemas without name conflicts

Schemas are being used for many applications. The purposes of those applications can be classified as validation, documentation, query, binding, or guided editing.

Validation is a process of ensuring that the documents' contents conform to an expected set of rules. It can be further categorized as structural validation, data validation, or business logic validation. Structural validation makes certain that XML element and attribute structures meet specified requirements. Data validation, on the other hand, ensures that the contents in the structures conform to rules about what type of information should be present. Validation of business logic is usually the domain of procedural code, not schema-based validation. It may check the relationship between data in one field and that in another. Most of the constraints in the realm of business logic cannot be expressed in XML Schema Language.

Schemas provide a rich set of *documentation* facilities via both human readable (xs:documentation) and machine-processable (xs:appInfo) information. Some widely used applications include the Dublin Core, a set of elements widely used to qualify web pages [10], and the Schema Adjunct Framework, a proposal to complement schemas with the information needed to generate applications [42].

A query language that uses the structure of XML intelligently can express *queries* across a wide range of data, whether physically stored in XML or viewed as XML via middleware. XQuery is a language designed to provide a query facility via concise and easily understood syntax [55]. It is also flexible enough to be used for a broad spectrum of XML information sources, including both databases and documents.

Other applications such as *data binding* or *guided editing* can also benefit from the Schema Language, but they are outside the scope of this report. More information will be found in the XML Schema Specification [53].

2.2.2. Parsing Schema Documents

XML-parsing. In order to produce an object representation that models the structure of schema documents, the documents must first be parsed into an object model representing the XML document. The Document Object Model (DOM) [9], developed by the World Wide Web Consortium, transforms input XML documents into a complete in-memory representation using a tree format, as shown in Figure 2.2.

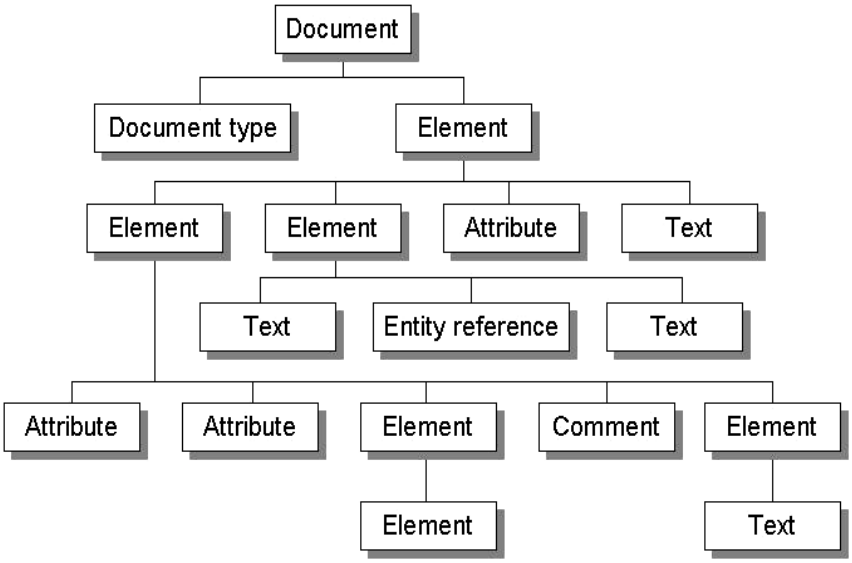


Figure 2.2. An Example of a DOM Tree

This object model only reflects the syntactic structure of the input schema documents, and does not tell us the semantic relationship of each element (i.e., the structure of *instance documents* that conform to given schema documents). Thus, an additional step must be performed to generate the representation that fully reveals the data model of each element in input schema documents. Figure 2.3 illustrates these procedures. In the figure, the input schema document contains one global definition for an element, and one global definition for a complex type. The global element `book` is defined with a reference to the globally defined complex type, `bookType`. BookType consists of four child elements (title, author, genre, and price), all of which are grouped as a sequence.

Schema-parsing: As this document is parsed in, the resulting object model reflects only the syntactic parent-child relationship of each XML elements such as "the XML element `xs:complexType` has a child element `xs:sequence`, which contains four child elements." However, it does not answer such questions as "What is the root element?" and "What are the child elements of the element book?". Only after each node in the tree is traversed and every reference resolved can the transformed model answer such questions.

The resulting model, after XML-parsing and Schema-parsing, will be referred to as the XML Schema Object Group (XSOG), and is shown in Figure 2.4. An XSOG consists of a set of schema objects, each representing an individual schema document, and a global object which serves as a mapping table between the target namespace of each schema and the schema object corresponding to that namespace. Each schema object holds references to the objects corresponding to the global element and type definitions defined within the schema. Among the global element definitions, the schema object also marks up the one that will be used as the root element for instance documents. If the schema does not contain a single root element definition, the object for that schema will not be constructed. Rather,

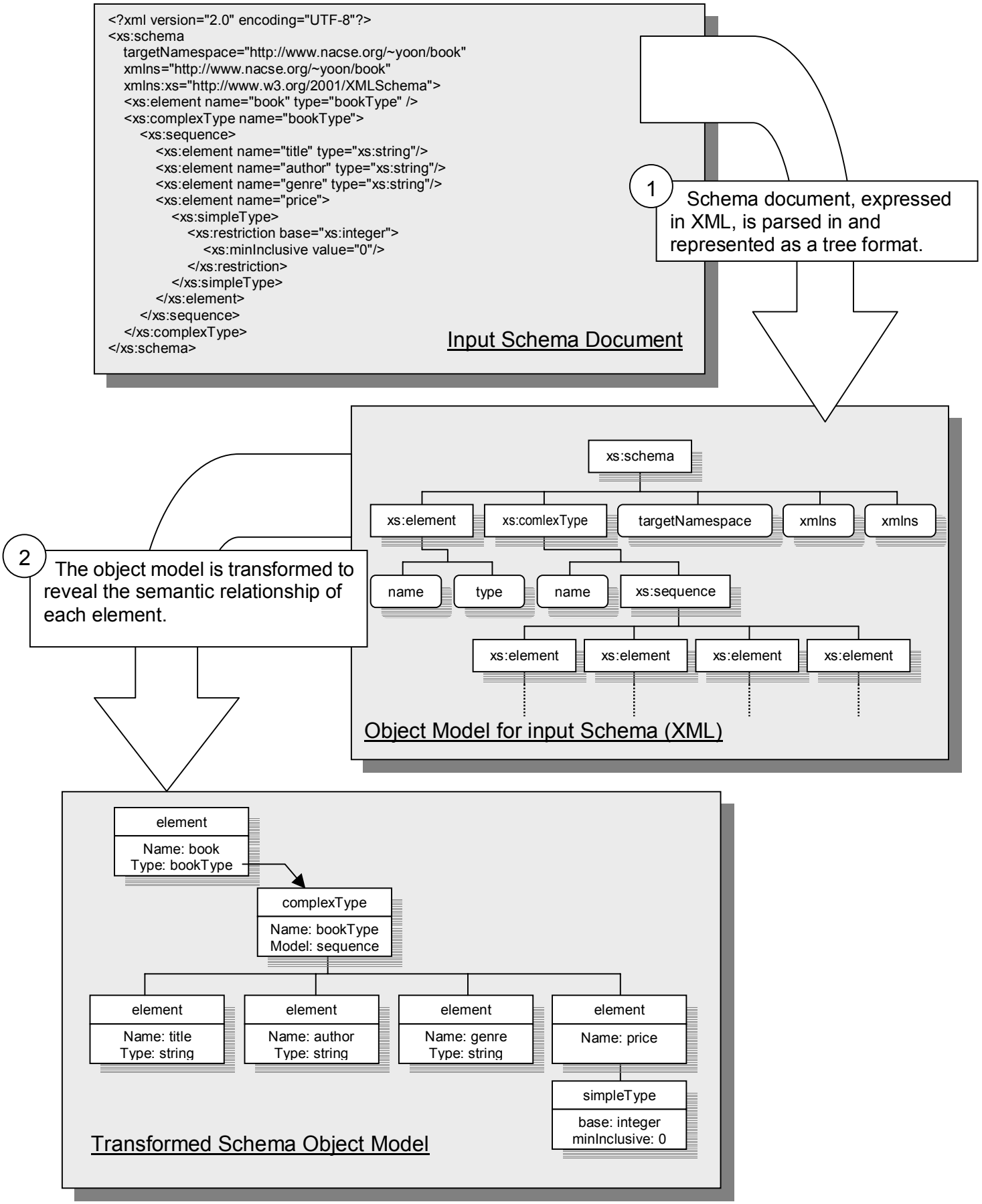


Figure 2.3. Two-Step Parsing of Schema Documents

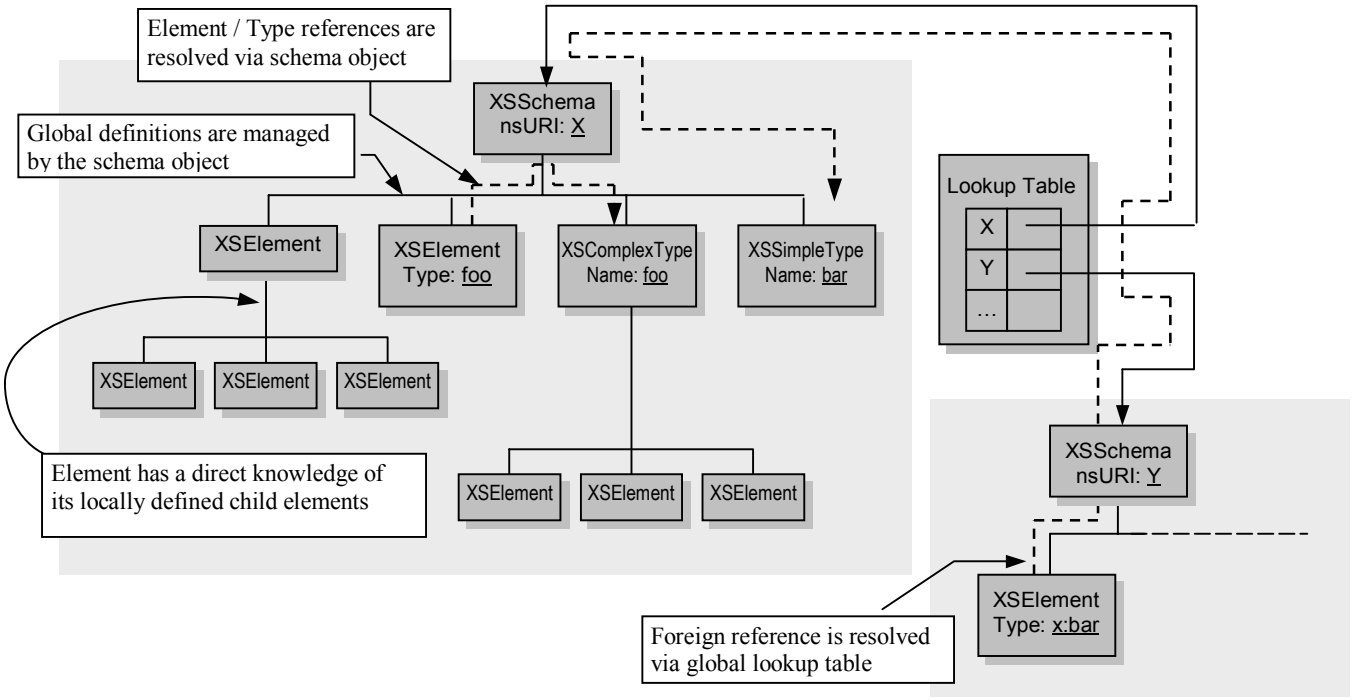


Figure 2.4. XML Schema Object Group

each global definition within that schema will be used independently by other schema documents.

An element definition can maintain its type information locally, or by reference using the `type` attribute. The former will directly contain the child elements within its hierarchy, while in the latter case the type information will be resolved during the Schema-parsing stage. Element and type references that are not defined locally are resolved via their namespace prefixes. The location of the foreign schema is resolved in two steps. First, the schema object converts the namespace prefix to the corresponding URI. The global mapping table in turn looks up the URI and returns the location of the schema that corresponds to that URI. The returned foreign schema then finds and returns the requested element or type definition. Note that if the foreign schema has not been Schema-parsed, returning global definition cannot completely expose the structural information that we're interested in. Section 4 examines how we worked out this issue with a recursive-decent parsing technique.

2.3. Subset Generator

This tool allows an organization to quickly build a subset of a metadata standard, containing only the fields that are relevant for the particular domain the organization is interested in. The schema documents for the standard are transformed to object models, and the Subset Generator uses these objects to build an interface allowing the subset designer to define selections and constraints. The modified objects are then used to generate a set of HTML forms and JavaScript interfaces controlling metadata entry process and related structural/data validations.

2.3.1. Subsetting a Metadata Standard

To further understand why metadata subsetting is desired, and how it will be actually performed, we will first examine the structure of the most widely used metadata standard, the FGDC's Content Standard for Digital Geospatial Metadata. CSDGM is an extensive standard comprised of ten subsections.

1) Identification Information — basic information about the data set.
2) Data Quality Information — general assessment of the quality of the data set.
3) Spatial Data Information — mechanism used to represent spatial information in the data set.
4) Spatial Reference Information — description of the reference frame for, and the means to encode, coordinates in the data set.
5) Entity and Attribute Information — details about the information content of the data set, including entity types, their attributes, and the domains from which attribute values may be assigned.
6) Distribution Information — information about the distributor of and options for obtaining the data set.
7) Metadata Reference Information — information on the currentness of the metadata information, and the responsible party.
8) Citation Information — the recommended reference to be used for the data set.
9) Time Period Information — information about the date and time of an event.
10) Contact Information — identity of, and means to communicate with, person(s) and organization(s) associated with the data set.

As shown in Figure 2.5, each section contains a number of nested subsections. The ten sections include over three hundred individual data elements, some of which can be repeated an unlimited number of times. Each field can be classified as one of three types: mandatory, mandatory if applicable, and optional. *Mandatory* fields must be filled in for the metadata to be compliant with the FGDC standard. Examples of mandatory fields include title, originator, abstract, purpose of the metadata, progress status, etc. *Mandatory if applicable* elements must be provided if the data set exhibits the characteristics defined by the element. For example, if the data has vertical characteristics such as elevation or bathymetry, then the vertical positional accuracy information in section 2.4.2 must be reported. *Optional* fields are those elements supplied at the discretion of the data owners.

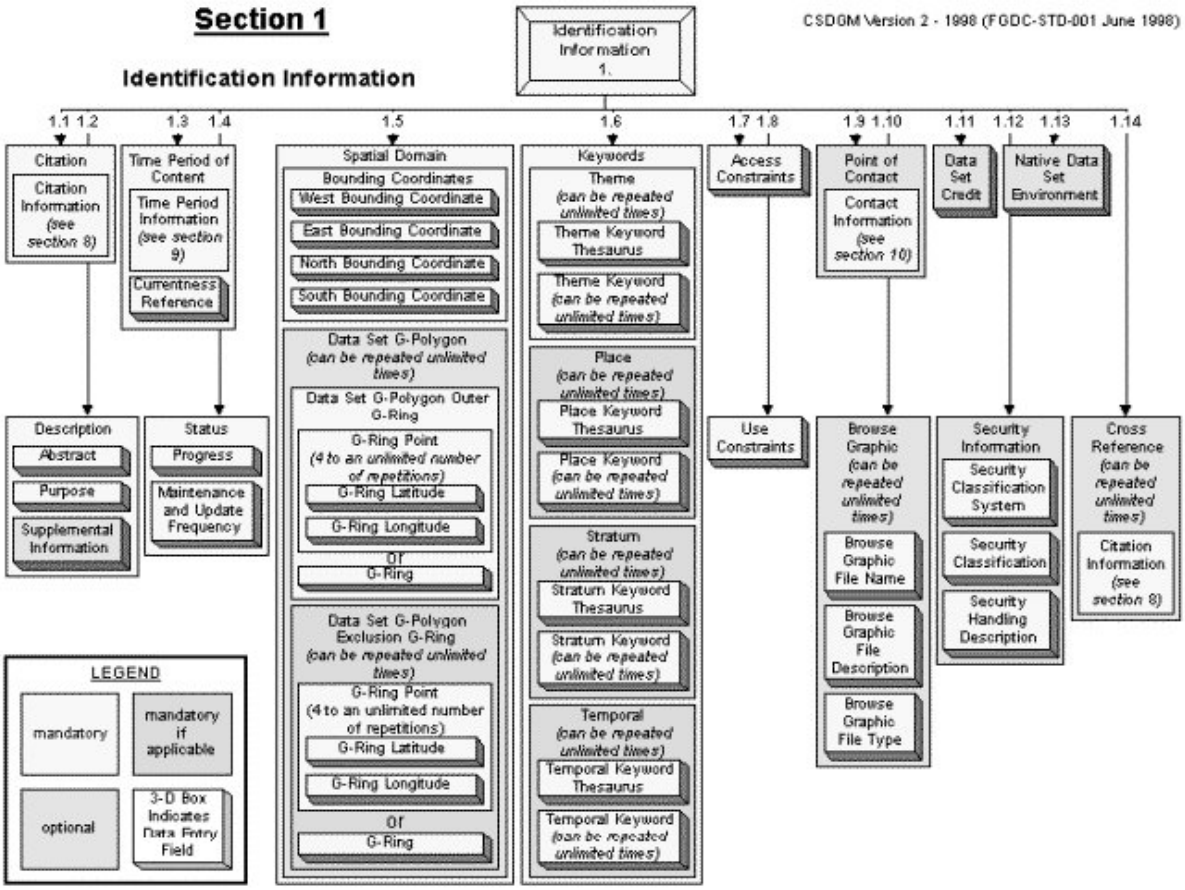


Figure 2.5. Fields included in FGDC's "Identification Information"

Pancake reports that since the FGDC Content Standard is so complex and comprehensive, data owners are often faced with variety of difficulties in metadata entry process [37]. She points out that the majority of scientific researchers, although some may have knowledge of metadata and perhaps even experience with clearinghouses, are unfamiliar with many basic concepts of the standard. Lack of familiarity with the standard imposes a number of challenges for data owners and is a major reason why creating and managing metadata is not a standard practice among scientific communities. Even the mere mention of the term "metadata" is cause for frustration [24].

To enter metadata effectively, users must figure out which fields to use for storing key information. Data owners may be aware that they want to record the altitude of the location where the data was collected. They may not know, however, that the field "vertical positional accuracy" is located in section 2.4.2—or even more likely, may not know that the field exists at all.

Even if users manage to locate the fields for information they are ready to provide, the process of metadata entry is not complete. They must successfully identify all required fields in order to make their entry FGDC-compliant. The process is complicated by the fact that CSDGM introduces a number of compound fields which are optional in one situation, but required in others. For instance, sections 2.5.1.1 through 2.5.1.6 in Source Information are initially classified as mandatory if applicable; all fields become mandatory, however, as soon as any one of them is filled in.

Unfamiliarity with the standard means that users are also uncertain whether they are performing the entry process correctly. They may not understand terms like "G-Polygon Outer G-Ring" (1.5.2.1) or "SDTS Point and Vector Object Type" (3.3.3.1). They may not know that only one of "Format Version Number" (6.4.2.1.2) or "Format Version Data" (6.4.2.1.3) may be used, but not both. They may not know that "Unknown" or "Not Applicable" are valid inputs for "Attribute Accuracy Report" (2.1.1).

An organization also faces the challenge of verifying and maintaining the integrity of metadata entries. Users may include fields that the organization may not want or need, or omit to fill in others that the organization requires.

The Oregon Coast Geospatial Clearinghouse (OCGC) attempts to remedy this situation by introducing the concept known as "Abstracted Metadata", which contains a limited number of data elements, but still remains FGDC-compliant [11, 50]. These elements are carefully chosen to provide the specific information necessary for cataloging data sets pertaining to the coastal areas in Oregon (see Appendix B).

The benefits of abstracted metadata are apparent. First, the user's effectiveness in locating the fields for key information immediately increases because the size of the search area that the user must navigate becomes much smaller.

A reduced set of fields that is closely tied to the specifics of a particular disciplinary domain also means that it is the user's domain familiarity that affects the quality of the metadata most, not his/her familiarity with the metadata standard. Scientific researchers typically possess a significant degree of knowledge within a disciplinary domain, and thus will be comfortable with these fields.

From the clearinghouses' point of view, abstracted metadata is easier to maintain and validate. Not only are the number of fields that should be verified for compliance to the standard and relevance to the organization's interest reduced, but also the data domain of individual fields become more specific. For example, the bounding coordinates for the spatial domain represented in Oregon may be restricted to 116° 45'W to 124° 30'W and 42°N to 46° 15'N [16]. Supplemental location information (1.2.3) could be specified to be the name of one of the counties adjacent to the Oregon coast.

It is important to note that in order for this abstracted metadata to be compliant with the original standard, certain restrictions must be enforced in constructing a subset. The most obvious rule is that no new field must be introduced to the existing standard. It turns out that this is not much of a concern, since current metadata standards are designed to be very comprehensive. Other restrictions include:

- All mandatory fields in the original standard must remain mandatory, and cannot be demoted to optional.
- On the other hand, optional fields can be promoted to be mandatory if they are applicable to the specific domain the subset is built for.

- If a field can hold any free text, the valid input for this field can be reduced to a set of controlled vocabulary.
- If a field only allows a set of controlled vocabulary, this set can be reduced but not extended.
- If a field expects numerical data, the range and precision of the numerical input can be reduced but not extended.

In general, the constraints for each field in the original standard can be made stricter, but not relaxed.

2.3.2. Building a Subset

As described in the previous section, subsetting is essentially modification of the structural and data constraints of the metadata standard. The Subset Generator was designed to perform this operation with the following requirements.

- Subset designers must be able to choose the elements and element groups that are relevant to the subset.
- Subset designers must be able to modify structural and data constraints of each element and element group, such as its optionality, cardinality, minimum and maximum values of input data, etc.
- The tool must present an interface that accurately reflects the structure of the standard. Elements within an element group must be properly nested, mandatory elements must be visually distinguished from optional ones, etc.

On the user's request, the framework servlet fetches the appropriate schema documents and performs the two-step parsing, described in section 2.2, to produce XSOG. The Subset Generator then dynamically generates the HTML pages that are used as interfaces for subsetting and serves them to the user.

Figure 2.6 shows the main interface of the Subset Generator. In the initial screen, which will be referred to as the "selection window," the tree-like representation reflects the structure of the metadata standard. Compound elements (those with child elements) are displayed with a toggle icon to expand or collapse their contents. A square block before the name of the element uses color-coding to indicate whether the element is mandatory or optional.

Optional elements are associated with a checkbox. Subset designers can include an optional field by checking the box. This activates a pair of radio buttons next to the checkbox, allowing the subset designer to promote the selected field from optional to mandatory in the subset. If an element is promoted, all its child elements are also promoted to mandatory by default, but the subset designer has the freedom to demote them if desired.

Each element's label is associated with a hyperlink, which invokes a pop-up window (Figure 2.7) which will be referred to as the "attribute window." The attribute window is dynamically generated from the XSOG stored in server's memory. It serves as an interface for modifying the constraints associated with each element. Modifications in the attribute windows are not finalized until the subset designer presses the submit button. At this point,

the integrity of the modified constraints on individual elements can be checked with respect to other elements, since the entire page is submitted to the server. Therefore, each constraint modification is stored in session variables until either the selection window is submitted, or the user's session expires.

Upon submission, the framework servlet retrieves form variables from the selection window, as well as the attribute constraints stored in session variables, and delegates them to the Subset Generator. The Subset Generator then modifies the XSOG by deleting the object instances that were not chosen by the user, and modifying the attributes of individual object instances. Integrity checking is performed at this stage.

This step could also have been performed on the client-side using a script language such as JavaScript, or when an individual attribute window is submitted. Client-side validation is not viable, however, because it requires that the entire structure of the standard as well as all constraints of each data element be downloaded to the client, which involves considerable bandwidth as well as the client's computing resources. If we let the Subset Generator modify the objects when an attribute window is submitted, it would have to keep the original state of the modified objects in memory and perform an "undo" action if the user changes his/her mind later, which would be wasteful for both processing resource and memory resource.

The modified XSOG is used by XSO Serializer and HTML Generator, described in the following sections.

2.4. XSO Serializer

Modified schema objects must be serialized for future use. Subset designers may use the modified XSOG as a template for further modification of the standard, while system developers may compare the serialized schema objects with the original input schema documents to verify whether the system executes intended operations correctly. Note that the schema documents that are not used alone—that is, schema documents without the root element for instance documents specified—are never unmarshalled as an independent unit. Rather, individual tag definitions within these documents are used by other documents. Thus these "template" schema documents (sections 8, 9, and 10 in CSDGM's case) are never serialized alone. Rather, individual components of these templates may be copied onto other schema documents.

Filenames for the serialized schema documents are determined by their respective `targetNamespace` attributes. The `targetNamespace` of each schema object is tokenized, separated by the "`/`" character, with the last token used as the filename. For instance, if the `targetNamespace` of a schema object is "http://www.nacse.org/library/book", the resulting filename for the serialized object will be "book.xsd". If there is a collision due to identical tokens in two namespaces, the token is concatenated until a unique name is achieved. For example, the namespaces "http://www.nacse.org/library/book" and "http://www.nacse.org/bookstore/book" will result in "library-book.xsd" and "bookstore-book.xsd", respectively.

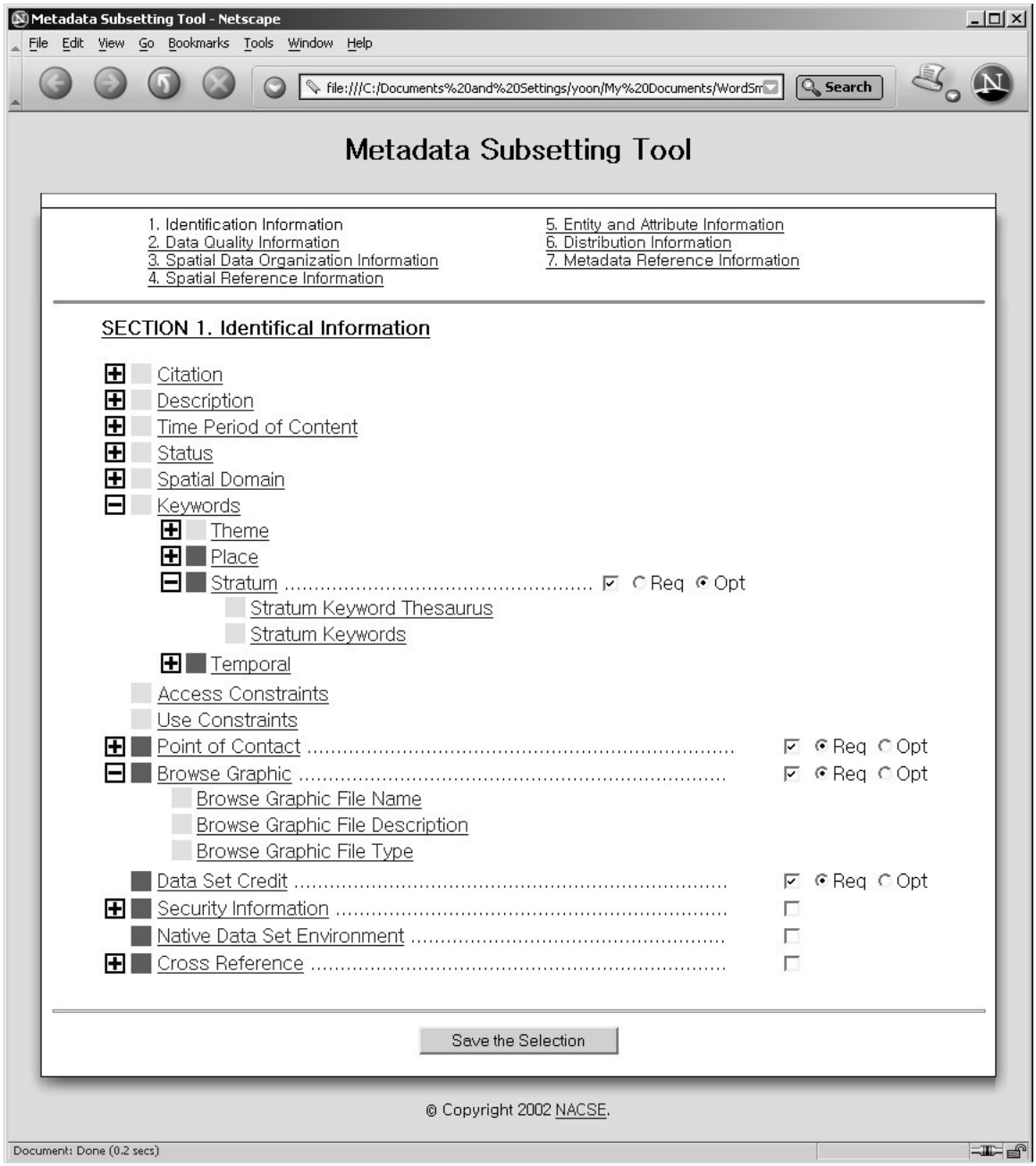


Figure 2.6. Subset Generator

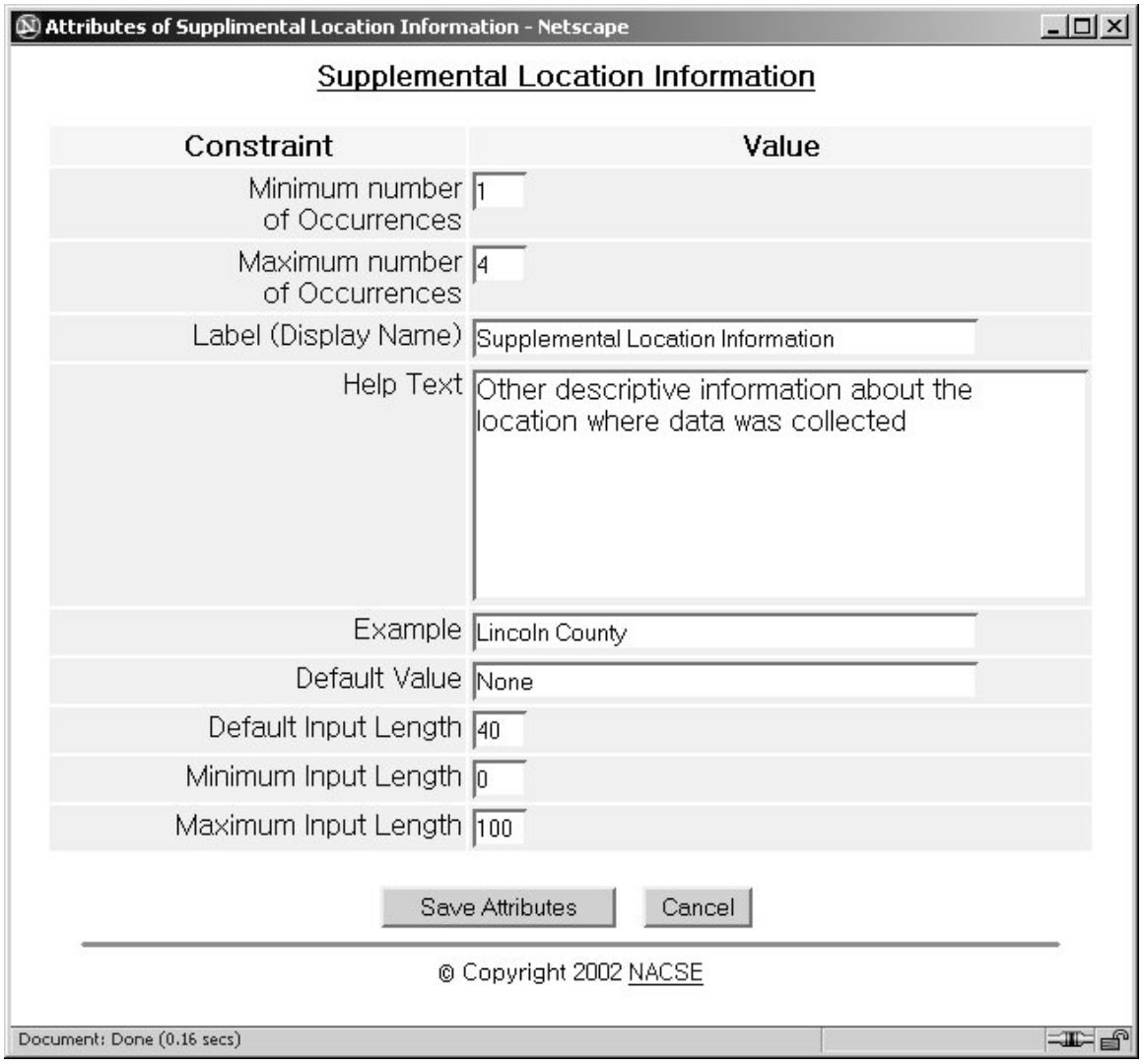


Figure 2.7. Subset Generator - Attribute Window

2.5. HTML Generator

Subsetted metadata has little or no use without a metadata entry interface specifically customized for the subset. Traditionally, these entry tools are constructed manually, and often involve writing HTML pages and some sort of CGI program that processes input data. Unfortunately, scientific data clearinghouses are usually managed by scientists, who are not familiar with the computer programming languages required to build and manage these sites. The HTML Generator solves this issue by automatically producing a metadata entry tool specifically for the subset. Since the subset is constructed from schema documents, this process only involves transforming the structural and data constraints of the schema documents to HTML pages.

Given a schema document, we first observe that, as shown in Figure 2.8, metadata entry only takes place on the element declarations that have simple content model (i.e., elements declared either with one of the primitive types, such as `xs:string` or `xs:date`, or with simpleType definitions constructed by extension from a primitive type or another simpleType). Therefore, we can map these elements with one of the HTML form input methods, such as <`text`>, <`textarea`>, etc.

On the other hand, element declarations with complex content model (i.e., declared with complexType definition) are used in order to group their child elements together. This "grouping" can be represented with one of the HTML tags (<`hr`>, <`table`>, <`div`>, etc.) as a separator. We chose the <`table`> tag because of its clear visibility as well as its flexibility, such as the ability to contain sub-tables within a table. Figure 2.9 shows output automatically generated from a schema document, containing a description of a book.

Elements sharing the same local name must be distinguished. XML does this via its namespace mechanism. Unfortunately, HTML does not have a facility that restricts the scoping of each input field's name, so it is impossible to differentiate two input fields with the same name. To solve this issue, we devised a scheme that superimposes naming conventions on the structure of XML documents. As shown in Figure 2.10, starting from the root element, the name of each element having a complex content model (i.e., with child elements) is passed along for constructing a unique field name via concatenation. Each token is separated by a colon (:) character because XML prohibits its use in element names except for indicating namespace [53].

The input metadata must be validated with respect to the constraints imposed on the subset. This validation process can be further subdivided into client-side and server-side validation. Simple validations, such as verifying whether a mandatory field is omitted, or if non-numerical input is found in a numerical field, are performed on the client-side using JavaScript. More complex validations, such as verifying whether an input data is within the restricted range, are performed on the server-side with Java servlets, which are part of the Metadata Framework Servlets (MFS). MFS is described in the next section.

2.6. Metadata Framework Servlets

The Metadata Framework Servlets (MFS) is a group of Java servlets responsible for all behind-the-scenes work, and is the core of the entire framework. The tasks that the MFS

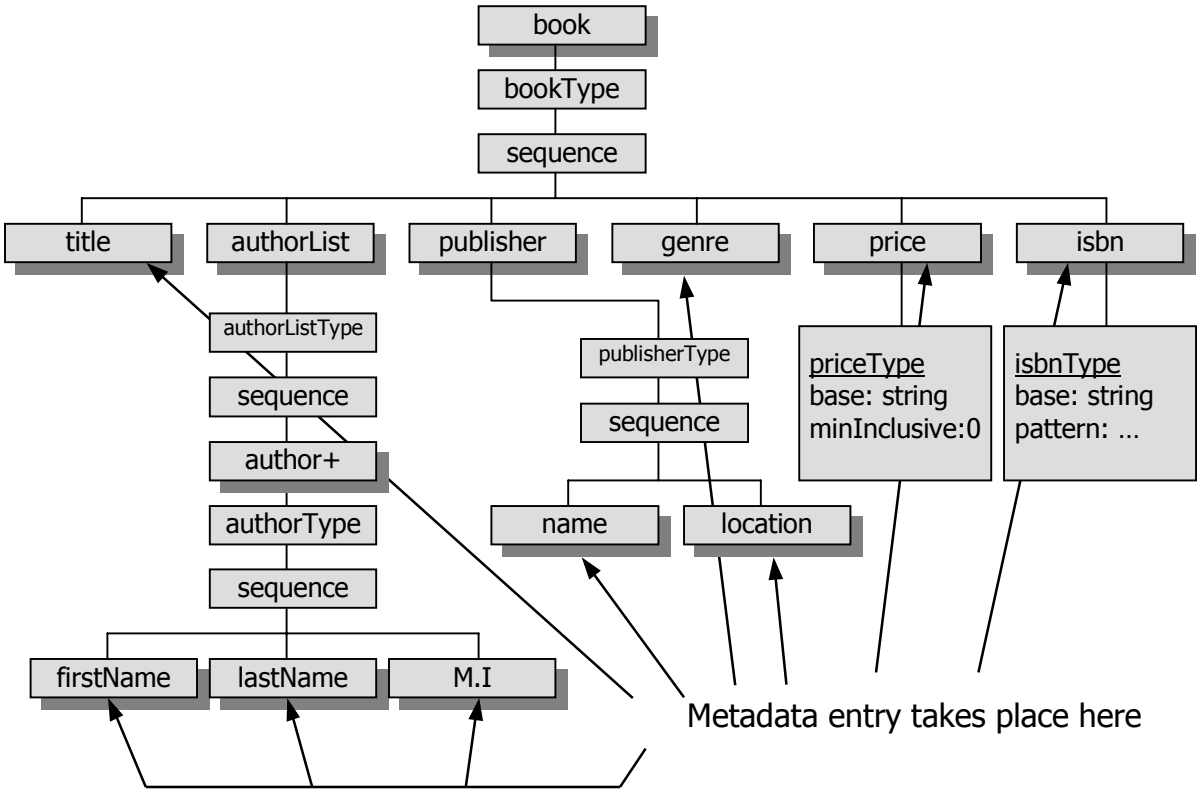


Figure 2.8. Metadata Entry Locations within a Schema Document

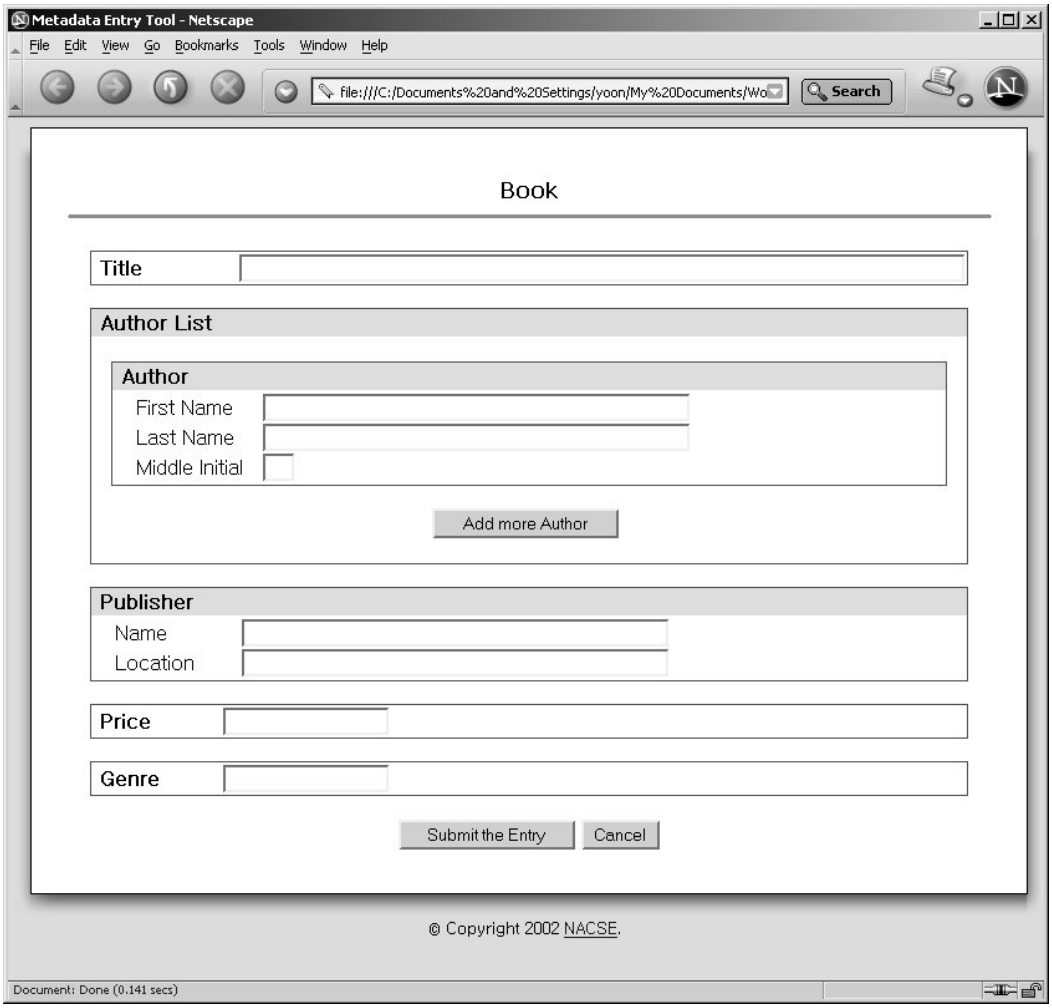


Figure 2.9. Example of Generated HTML Output

deals with can be divided into two categories: metadata subsetting-related and metadata entry-related. For the first, the MFS is responsible for housekeeping all the components in the entire framework—retrieving schema documents from the file system, invoking each component, transferring returned objects from each component to the subsequent one, managing session objects, checking constraints validation submitted from the client, etc.

For the metadata entry process, the MFS is responsible for additional tasks. First, it registers generated HTML and JavaScript pages with respect to the subsetted metadata standard. This involves generating a URL for the subsetted metadata so that the HTTP server can serve these pages. The interface for the metadata entry process typically includes a number of HTML pages. For instance, the FGDC standard consists of ten subsections, seven of which are displayed individually. As the data owner completes one section and moves to another, his/her input data must be stored in the session objects as well; this process is also managed by the MFS.

When the user completes the entry process and submits the forms, all input data must be validated. As in the case of the subsetting process, simple validation is handled by associated JavaScripts. More complex validation logic, on the other hand, requires several steps. First, the MFS unmarshalls the schema documents corresponding to the subset. The resulting XSOG will be used for validation and persistency. The servlet constructs a mapping table (Figure 2.11). This indexes each element object in the XSOG with the element's concatenated name, as described in Section 2.5. Then, all field names within a form page are retrieved and stored in memory. A standard Java servlet function, `getParameterNames`(), is used for this purpose, returning a collection of string objects, each containing the name of a request parameter. Every input field is then processed in the following way.

- The value of the input field is retrieved, using the standard Java servlet function `getParameterValue`(). The parameter for this function call is provided from the collection.
- Constraints for the field are obtained from the corresponding object, via the mapping table.

The retrieved value is then evaluated against the constraints. If the value violates a constraint, a flag is set for that field, indicating an error condition; otherwise the value is saved in the memory. These steps are repeated for all input fields. If no error was found, the saved values must be stored in the file system or database. The persistency mechanism is not covered in this project, although several alternative designs were evaluated. One possibility is to have the servlet to build XML instance documents with the schema objects and input values, and use one of the XML-to-database connectivity tools [52].

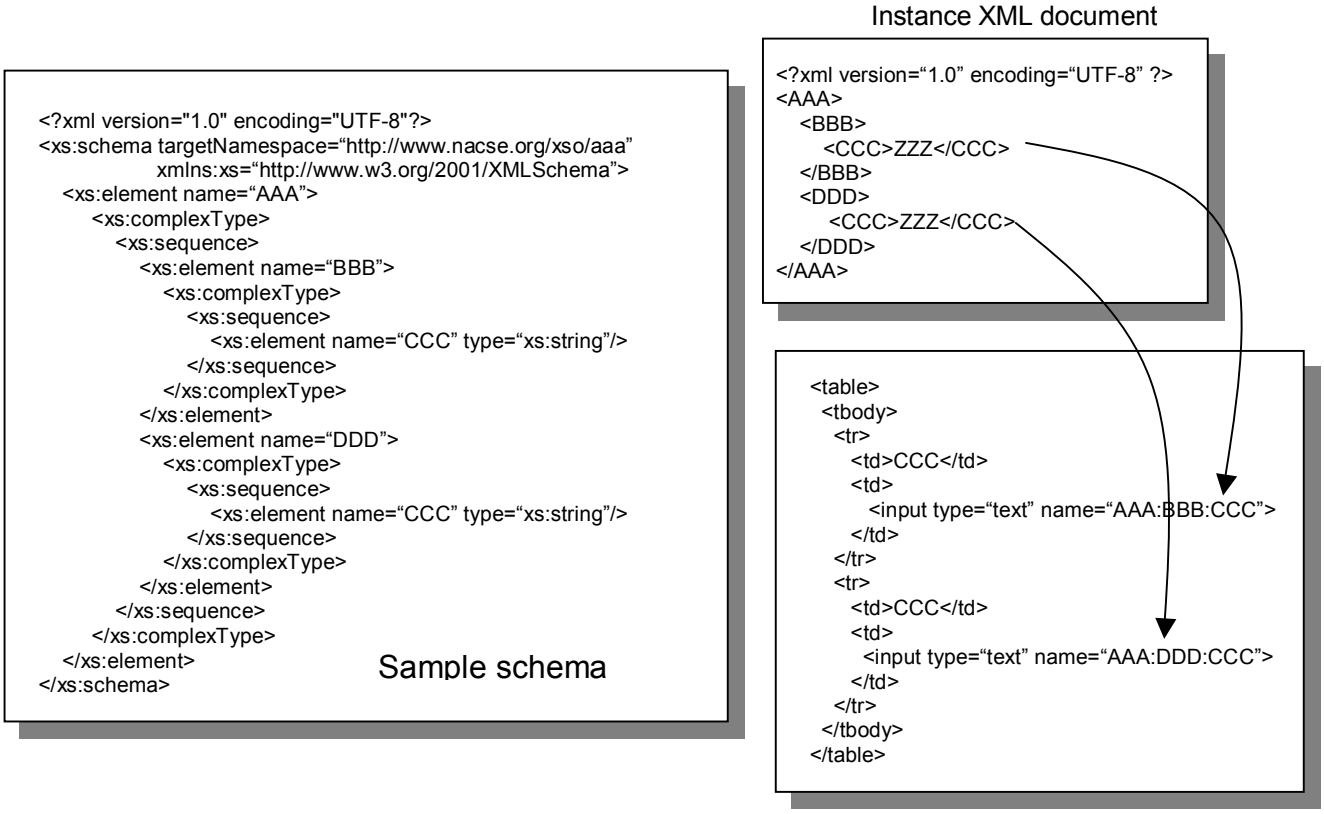


Figure 2.10. Unique Name Is Assigned to Each Field

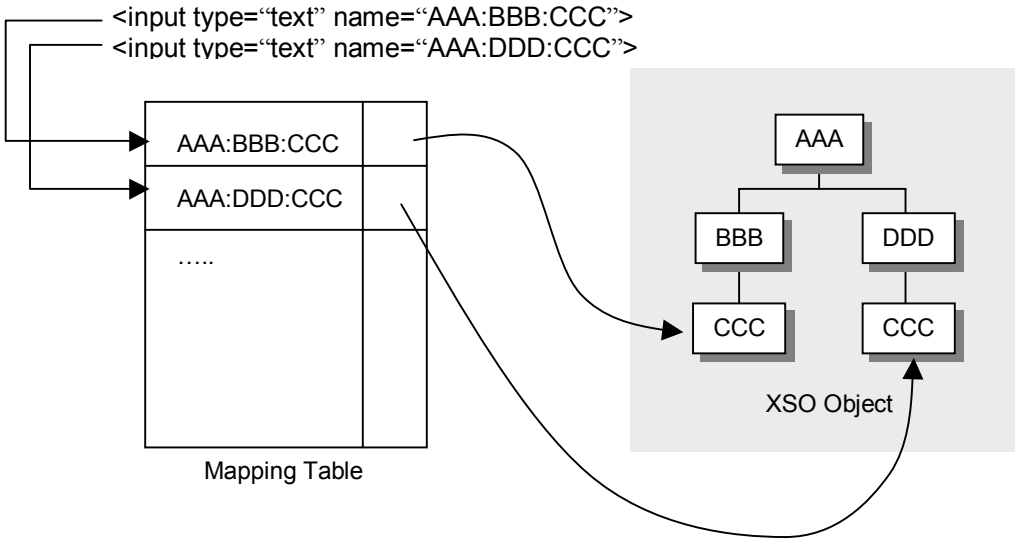


Figure 2.11. Mapping Table

3. Usability Issues for the Metadata Subsetting Framework

In the previous sections, we examined a number of difficulties data owners face in the metadata entry process, and proposed a solution based on the idea of subsetting metadata standards. In this section, we discuss the types and sources of human errors that occur during the metadata entry process, and formulate some design goals that may be applied to prevent and recover from these errors.

Problematic interactions with computer software are usually referred to as errors, although this term is misleading in the sense that it implies the user is always to blame [41]. Lewis and Norman point out that in most user interaction, errors arise without any intent and so should be analyzed as misunderstandings or confusions [23]. Regardless of terminology, such problems are inevitable and designing for errors is an important piece of software design.

3.1. Types of Errors

Norman categorized various types of errors into two main classes, slips and mistakes [33]. *Mistakes* occur through conscious deliberation. If an inappropriate intention is established and pursued, a mistake is committed. Examples of mistakes include typing "`del`" instead of "`rm`" on a Unix platform, or dragging a shortcut icon into the Recycle Bin in order to uninstall a program on Windows platforms. Many mistakes can be prevented with a good interface design, but it is generally hard to detect mistakes and recover from them because they stem from cognitive breakdowns, which are often influenced by a number of external factors.

Slips are unintentional, accidental errors. They are caused by a misbalancing between mind and body, and are the most frequent types of errors [39]. Examples of slips include typographic errors, clicking the "close" button by accident when attempting to maximize a window, etc. In general, slips are often unavoidable, although it is easier to detect and prevent them from occurring than mistakes. Lewis and Norman also discuss techniques for helping users detect or recover from errors [23, 3]. A general technique is to provide a forcing function that prevents the user from continuing down an error path. Examples include:

- Gags (e.g., locking the keyboard)
- Warning (e.g., an alert dialog)
- Do nothing (e.g., ignoring the request)
- Auto-correct (e.g., DWIM, or Do What I Mean)
- Let's talk about it (e.g., initiating a dialog – "Can't find that filename")
- Teach me (e.g., letting the user add words to a spelling dictionary)

3.2. Execution-Evaluation Framework

In order to understand the situation that errors arise from, Norman [34] suggested an analysis of human-computer interaction called the Execution-Evaluation Framework. It is a conceptual model of the manner in which humans interact with computer systems. According to him, the distance between the user's goals and the means of achieving them through the system is referred to as the "gulf of execution," and the discrepancies between the system's behavior and the user's goals can be expressed in terms of "gulfs of evaluation." Rosson and Carroll [41] illustrate the concept of Execution-Evaluation Framework with the following example:

> "The user begins with a task goal, such as the desire to investigate an irregularity in last month's budget. To pursue this goal, the real-world objective is translated into an appropriate system goal—a computer-based task such as an Excel spreadsheet. The system goal is elaborated as an action plan that specifies the steps needed to achieve the system goal: point at an Excel icon, double-click to open it, point at the first cell containing a sum, and so on. Finally, the plan is executed: The mouse is grasped and moved until the cursor is over the icon, a double-click is performed to launch Excel, and the pointer is moved to the bottom of the first column."

In the Execution-Evaluation Framework, the above example can be decomposed to the following seven steps [41].

1. Forming a task goal: At this stage the user sees a need to achieve something, possibly a problem that calls for a solution. For example, "I'd like to investigate the last month's budget. I should check the column sums."

2. System goal: The user translates the task goal into a software-oriented goal, known as a system goal. For example, "I need to open the Excel file for the budget to check the equations."

3. Planning an action sequence: The intention is elaborated into a sequence of actions to be taken. For example, "Point at the Excel icon, double-click to open it, point to the cell at the bottom of first column, click to highlight, read equation."

4. Execution: The user executes the action sequence. For example, "Grasp mouse, move cursor to icon, click twice rapidly, move pointer to new position, click once, etc."

5. Perception: The user observes the state of the system. "Pointer over icon, icon highlighted, rectangle with text appears, pointer at the bottom of column, highlighted symbols appear in box above column, etc."

6. Interpretation of the state of the system: The user draws conclusions based on his/her perceptions as to what has happened to the system as the result of his/her actions. For example, "I opened an Excel file and selected the cell that should contain a sum equation."

7. Evaluation of the outcome. “The user compares the interpretation with respect to the original intention. For example, “I see the equation and it looks OK, so I will move on.”

Since mistakes and slips are unavoidable in user-system interaction, the system should be designed to anticipate and cope with them. Given this conceptual model of how users interact with systems, the following sections discuss how to assist those users to avoid committing mistakes and slips in web-based applications. Section 3.3 discusses some characteristics and limitations of web-based applications, while Section 3.4 explains the techniques and guidelines we used in designing our system to overcome such limitations.

3.3. Characteristics of Web-based Applications

Recently, a new use for the web has become more and more popular, namely for application delivery [17]. Initially, this consisted largely of using the web as a front-end interface for legacy databases, but the range of applications is broadening [40]. Web-based applications are defined as interactive applications accessed through a web-based interfaces, which distinguishes them from conventional web sites characterized by low interactivity. Compared to graphical user interfaces, the web can only accommodate very limited interaction styles. The HTML standard provides several commonly-used elements: text entry fields, buttons, menus, scrolled lists, radio buttons, and checkboxes. Ehrencrona also defines some characteristics of the web interface that affect user interaction [13]:

- Atomicity of the page: The page is the fundamental information unit and can generally only be viewed as a whole. Pages have distinct addresses (URLs).
- Spatial metaphor: Conceptually, loading web pages is movement through the page space. A page therefore has a location.
- Free movement between pages: Users can leave a page at any time and move to any other page (e.g., by bookmarks or manually entering an address).
- Interaction only at page loads: Web page content is provided by the server. It rests unchanged until it is reloaded from the server. Communication with the server can only occur at page loads.
- Pages only in scrollable region: Web page content is presented in a scrollable, rectangular region, separate from the browser.

These characteristics are being partially invalidated in some web situations. JavaScript makes limited interaction on the client-side possible; in some applications, users cannot bookmark any pages; interaction between page loads is possible with active content, such as Java applets or ActiveX controls. Nonetheless, in most cases a web application will only be able to communicate with the user through the set of interaction objects defined by the HTML standard.

The intrinsic characteristics of web interfaces make it difficult to build a highly flexible and interactive interface. Besides the limited set of interaction objects provided, interactions between the user and the server can only take place when the page is loaded. The server's response to the user action must also travel over the network, typically causing slow response time and thereby limiting the interaction methods possible. Despite these drawbacks, web applications offer many important advantages, both for users and for developers. First, users do not have to download and execute application code (other than the browser itself, of course). This eliminates the need for installing applications, a source of time loss and potential cost for users. Upgrades of the application are instantaneously available. Further, web applications are cross-platform by their very nature; web content can be viewed on any computer architecture as long as a browser has been ported. This saves application programmers an enormous amount of work. Web-based applications are easier to build and maintain than those relying on specialized or proprietary windowing toolkits. Consequently, developers can spend more time on the logic of their applications, thus improving the quality of the applications.

3.4. Designing for the Web Applications

This section analyzes some of the potential usability problems we have catalogued in web applications, using the conceptual model and terminologies discussed in previous sections. As we saw in Section 3.2, human-computer interaction involves a series of stages: goal, intention, actions, execution, perception, interpretation, and evaluation of the outcome. It must be noted, however, that this is not a provable psychological theory, but rather an approximate model of action [35]. Furthermore, it is normally difficult to classify which stage of the Execution-Evaluation Framework individual problems belong to, since transitions from one stage to another are not always discrete. In addition, some errors can be attributed to many different factors from multiple stages. The premise of this section is that by examining the situations in which human errors arise, developers can establish better design strategies and apply them to build systems that are intuitive and pleasant for users. Designing an interface to prevent errors is important, but it is also critical that the system notify the user of the occurrence of errors in unambiguous ways, so that the discrepancy between the user's intention and the state of the system can be rectified as early as possible. Undetected or unnotified errors can cause further mistakes, which will make it difficult for the system to recover by restoring its previous state. Since not all errors can be prevented, system developers must incorporate error resistance strategies into the user interface life cycle.

3.4.1. Assisting Users' Navigation

One of the biggest differences between traditional GUIs and web interfaces is that a web interface does not impose any structure on the application. Conventional software user interfaces control where and when the user can go. For instance, menu options can be grayed out if they do not apply in a particular context, or users may have to answer a question in a

modal dialog box prior to navigating to a location. On the Web, the user fundamentally controls his or her own navigation. As discussed in Section 3.3, the user can leave a page at any time and move to any other page by using a bookmark or manually entering a URL.

Nielsen points out that a major usability problem with hypertext is the user's risk of disorientation while navigating the information space, a condition known as the "lost in hyperspace" syndrome [7, 30]. It is largely attributed to the fact that the hypertext system allows users to freely move from one location to another at any time, without providing information on *modality*, or the current state in the context of the application. The situation is aggravated in the case of web applications, which are generally more complex than traditional web sites in that they can undergo more state changes over time. Server states also enter in. For instance, a web application that introduces a login procedure for security reasons adds the state "logged in" and "not logged in." With no explicit mechanism for communicating the current state to the user, disorientation is relatively easy.

The disorientation issue is pervasive in metadata entry tools. Metadata standards are typically subdivided into sections, each of which may include many fields to be filled in. Thus, it may be essential to divide the interface into several pages, probably one page per subsection. Users freely navigate among these pages. They may perform data entry for some fields in a page, jump to another page and enter some inputs there, then return to the original page to continue. In this scenario, user disorientation is increasingly likely to occur as the number of pages grows. Unless a user moves to a subsequent page only after the current page is completed, he or she must keep track of the pages already visited, and what has been done in each page. Questions such as "which subsections have been completed?", "which need further editing?", and "which have not been visited?" will recur throughout the metadata entry process. Furthermore, if the user decides to save the information already entered, or if the session was idle for too long and expires, which page should be dislplayed later when the user wants to resume the entry process? The first incomplete page, or the last page visted? Not only must these questions be answered, their answers must be communicated clearly to users.

Several papers advocate the idea of displaying web locations as the nodes in a hierarchy [21, 25, 32]. The main advantage a hierarchy offers is that it is relatively easy to indicate the whole structure, as well as the user's current location. Most simply, the location can be indicated by a "bread-crumb trail," i.e., listing the path from the root (main page) to the current node (current page). Alternatively, the hierarchy of all pages can be represented graphically in a "sitemap." This approach has been promoted in many usability guidelines, including Nielsen's famous Alertbox series [31]. Several best-practices on using sitemaps have been recommended, including:

- The sitemap must be accessible from all pages. *Users should not have to search out a navigation aid.*
- Sitemaps should not be too complicated, so that users can see the overview of the site's areas at a single glance.
- The use of animation, expanding views, etc., should be avoided in sitemaps, because they unnecessarily complicate the presentation.
- The sitemap should always contain an indicator of the user's current location. It is, after all, a map.

Many studies discuss the effect of selectively using colors to manipulate the user's attention [15, 27, 38]. Thus, the effectiveness of a sitemap can be maximized with color-coding each node (page) based on some category. For example, we might assign different background colors for each page based on how far the editing process has progressed—not begun, in-progress, and completed.

3.4.2. Errors Due to the Characteristics of Web Elements

As shown in Figure 3.1, web interfaces provide a limited number of interaction elements (labels, buttons, text box, scrolled lists, checkbox, etc.). Although users are generally familiar with the standard web elements, care must be taken to assist the user to apply them correctly (i.e., as they are intended by the developer). In this section we discuss several usability issues related to web interaction elements.

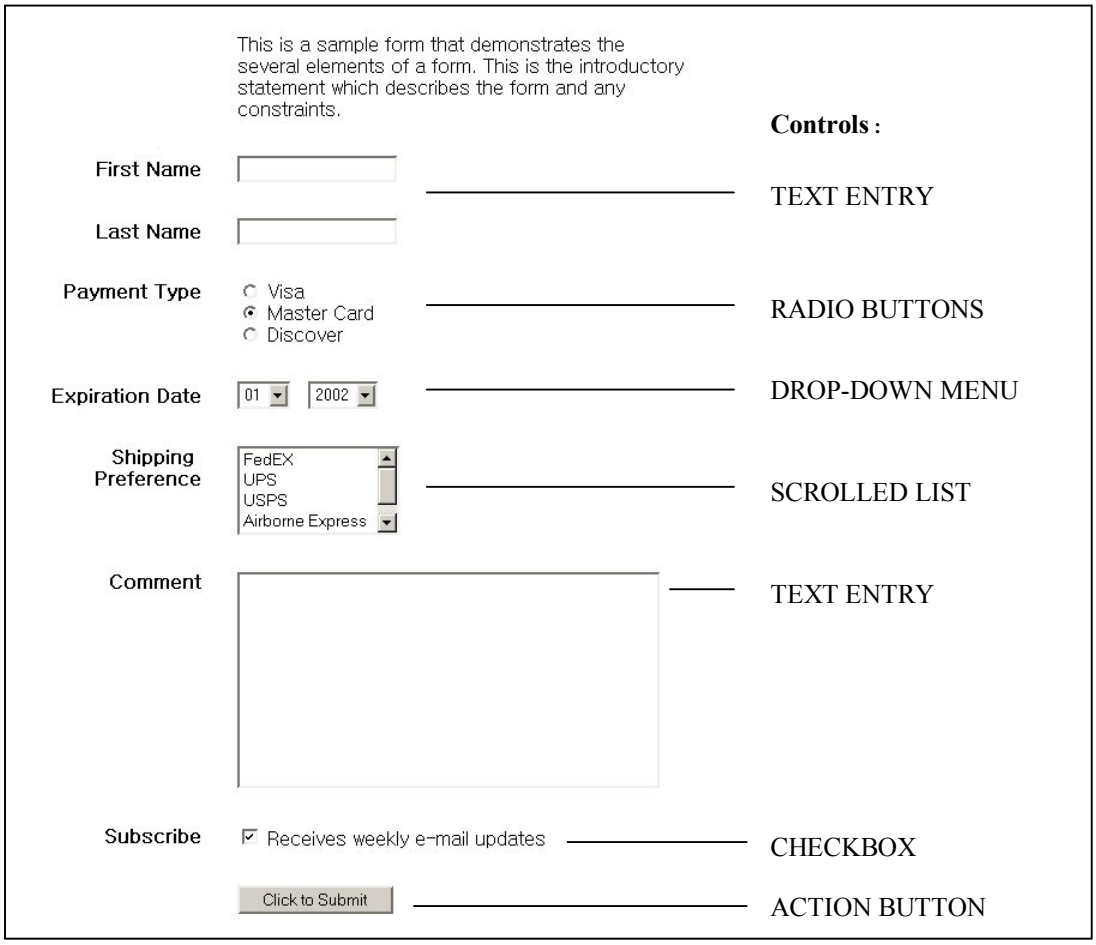


Figure 3.1. Interaction Elements in Web-based Applications

The most widely used mechanism for getting user input is the text entry field. Textual inputs are actually handled with two web elements: *text fields* and *text areas*. Text fields are created with the <text> tag and allow users to type in a single line of text. Text areas are created with the <textarea> tag and are intended for multi-line inputs, such as comment fields. The appearance of the two elements is distinctive, and they must be chosen carefully to reflect the type of content expected. For instance, a "Title" would probably be less than 20 words in length, while an "Abstract" usually requires more than a few lines. Careless use of these web elements will increase the likelihood that users place an entry in the wrong field, especially when the label of the field is ambiguous, or the user is unfamiliar with the terminology used. This can be considered as both a mistake (in action plan) and a slip. In any case, this type of error is generally hard to detect.

Users may also fail to fill in mandatory fields. This is most likely to occur if there are many fields in one page and there is no visual distinction between mandatory and optional fields. However, limiting the number of fields in one page, e.g., by splitting them across two or more pages may not be a viable option because it may conflict with the user's mental model of the metadata standard, or for technical reasons. On the other hand, using a different background color to distinguish mandatory fields from optional ones is a widely advised technique and is fairly easy to implement [15, 27, 38]. It also simplifies notifying the user of errors; the technique is discussed later in this section.

Slips can also occur in textual inputs. Spelling errors are the most common. They include transpositions (*teh* instead of *the*), use of homonyms (*their* and *there*), repetition (... *that that* ...), etc. Typographic errors involving homonyms are difficult to detect, as semantic analysis is required. The others are relatively easy to detect, using available spell-check tools (such as *ispell* on Unix platforms [19]). The misuse of special characters, such as periods, commas, underscores, etc., are likely to occur as well. Detecting such errors is relatively easy in numerical contexts, but difficult in textual ones.

The allowable input values can be limited with various forms of lists and checkboxes. Although the appearance of these elements is distinctive, both are used to allow the user to select one or more values from a list of choices. A *drop-down list* limits the user to a single value from the choice. It initially shows the default value and becomes populated when the user clicks a selection; if the user does not take any action, the default value will be selected automatically. Therefore, the default value should either be the most likely choice, or alternatively, a description of the contents (preferably beginning with a verb—for example, "Select a State"). Drop-down lists pose other usability issues as well. They may not allow the user to see all choices in one glance, if there are too many values to fit for the height of the page; this makes it difficult for the user to locate the most appropriate value. Visual inconsistency among different browsers and platforms is an another issue. For example, on Windows platform browsers automatically create a scrolling bar in the list, while under Unix it "spans" side by side, as shown in Figures 3.2 and 3.3. This behavior, which we refer to as a *cascading drop-down list*, can cause serious problems, since the user has to be extremely careful in maneuvering the pointer.

A *list box* or *scrolled list* is used when the user may select any subset from a list of predetermined values. Developers often make the mistake of assuming that users will know they can select more than one item, generally with the Shift and Control keys. This is not a good assumption, especially with inexperienced users. List boxes and scrolled lists also present visual inconsistencies similar to that of drop-down lists. The scrolled list has another

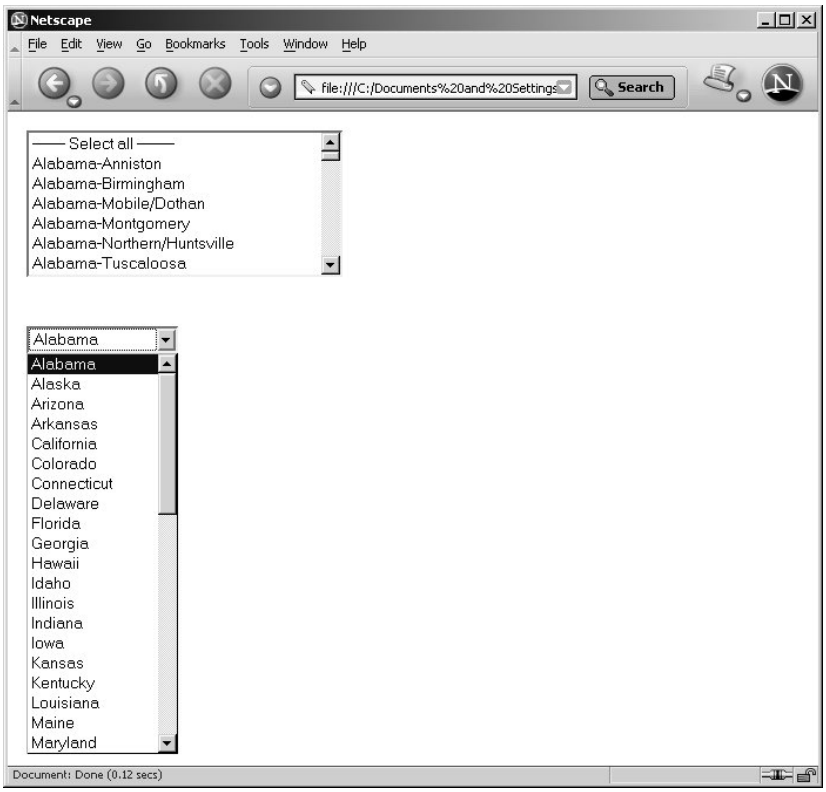


Figure 3.2. Scrolling Drop-Down List (Windows)

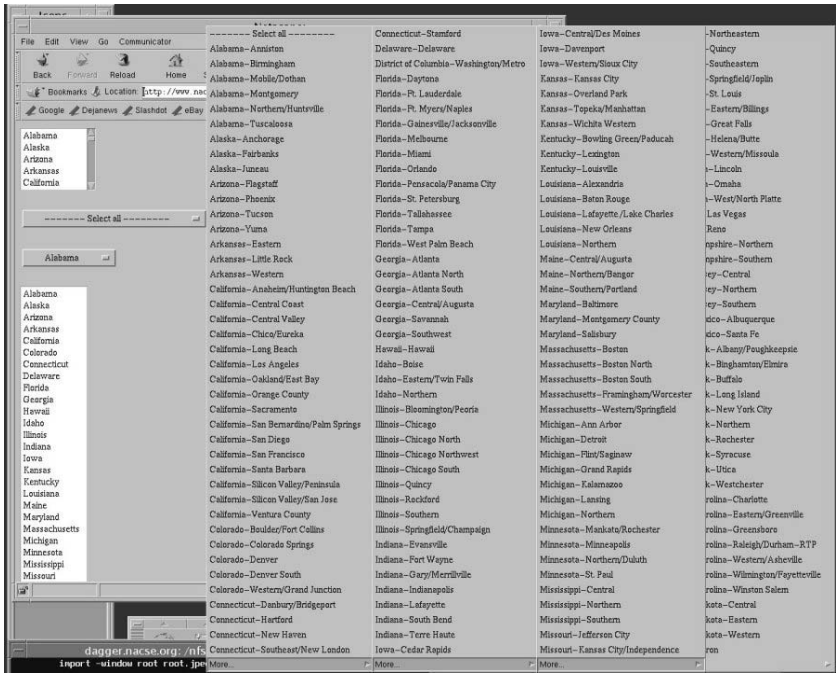


Figure 3.3. Cascading Drop-Down List (Unix Platform)

usability issue as well. If the height of the display areas is sized inadequately, only a few items will be visible at a time, and users will have difficulties locating the most appropriate value (they effectively have to memorize items as they scroll down). There is a second reason for taking care with the height of lists. If it is not tall enough to accommodate a good proportion of the list, the *thumb* (the handle of the scroll bar) becomes tiny, making it extremely difficult for users to control it (Figure 3.4). Weinschenk and Yeo recommend using filters when there are more than 40 items in the list, either dividing the list alphabetically or categorizing items into classes and splitting them into multiple lists [49]. This technique can be applied to drop-down lists and checkboxes as well.

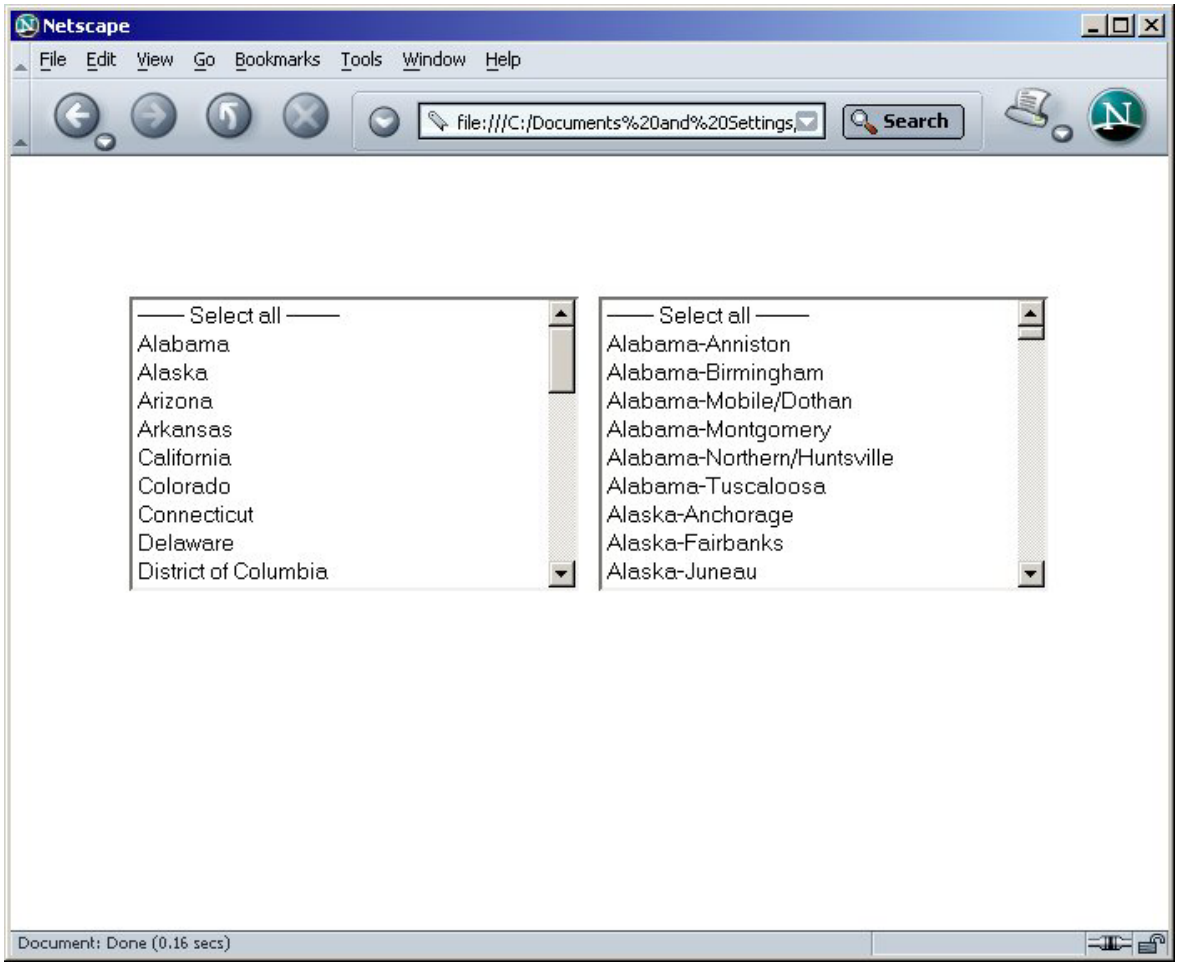


Figure 3.4. Scrollbar Thumbs Can Become Too Small

Checkboxes are similar to scrolling lists in that they provide a list of elements from which the user can select any number of choices. Most web users are familiar with this idiom, although it is safer to include a label reminding the user (e.g., "Check ALL that

apply"). Although checkboxes are generally easier to use than scrolling lists, they do not scale well when there are a large number of choices. It is also difficult to reflect when certain combinations of items are not allowed. This is best illustrated with an example. Consider an online computer store, where buyers can customize their orders. An internal CD-ROM and an internal floppy drive may not be chosen for the same notebook computer. One technique to prevent users from selecting mutually exclusive items is to deactivate them (which can be implemented using extension languages such as JavaScript). A less usable alternative is to let errors happen, then advise users of the problem afterward.

3.4.4. Error Detection and Recovery

Error prevention techniques will never eliminate all occurrences of human error. Thus it is essential that *error correction* mechanisms—sometimes called error management—be included as part of the user interface life cycle. Lenman and Robert identify three sequential tasks for correcting an error: *error detection* (the user needs to know that an error occurred), *error explanation* (the user must understand the nature of the error), and *error recovery* (the user has to counteract the effects of the error) [22].

Error detection and error explanation are the first steps in error correction. Users must be informed in a clear and obvious manner that an error (or errors) occurred in processing input data, why they occurred, and where they occurred. There are a number of challenges in implementing this guideline in our application, however, due to the fundamental nature of metadata processing. First, data entry fields are most likely divided into multiple sections, each presented in an individual page. It is not clear how the location of erroneous inputs should be displayed if they occurred in more than one section. Furthermore, due to the large number of fields users may not be able to see all errors from one page at a glance.

Notification of spelling errors in text input introduces even more serious problems. HTML does not support highlighting within text fields. This means that even if it is possible to inform the user *which field* the spelling errors occurred in, the misspelled *words* cannot be highlighted.

We are currently evaluating a number of alternative techniques to resolve these issues. We may aggregate all erroneous input data and display them in a single page. The downside of this approach is that it interferes with the user's mental model of the metadata standard and he or she cannot see the entire context where the error occurred. Another possible solution, although it does not ultimately alleviate the situation, is to warn the user of errors as early as possible with pop-up windows, thus minimizing the number of erroneous inputs submitted. For instance, checking the omission in mandatory fields can be very easily implemented with JavaScript, but since JavaScript only applies to the *current* page, it only detects omissions in the page where the submit button was pressed. If, on the other hand, JavaScript validation is performed when the user attempts to leave the current page, he or she loses the freedom to temporarily interrupt completion of one section in order to review or edit another.

A distinction among error recovery methods in interactive systems is forward versus backward error recovery [8]. Backward error recovery is an attempt to restore the system state after an error has occurred, whereas in forward recovery the user has to execute some

tasks to recover from the fault. Backward recovery is most commonly implemented as the *undo* command, although it can be loosely categorized into three kinds of commands (*undo*, *cancel,* and *stop*) [56]. The role of the undo command in our web-based metadata application is somewhat limited, but applies in the following circumstances:

- The system failed to process the input data, but was not able to identify the cause of error(s). In an ideal world this should never happen, but it is practically unavoidable in real-world software development.
- The system generated a message informing the user what the error was, but the user does not understand what it means, either because the message was ambiguous or because the user is unfamiliar with the terminology used.

In traditional GUI applications where the system must manage the entire lifecycle of an application, implementing undo function is extremely complex. In web-based applications, however, the undo function is facilitated with the browser's back button, although its functionality is to some extent limited. The most serious drawback of relying on the back button is that the locations of the errors are no longer available (although that applies only to the second case described above).

In practice, forward error recovery is often the only way to recover from human errors. The user hopefully can identify both the cause of the problem and the correct solution, but unfortunately this is often not the case. Users may have to hypothesize about how and why the errors occurred and how to recover, or look for an additional source to analyze them. The system can assist users to rectify errors by notifying them of the location of erroneous input and the reason why it could not be accepted, as discussed above, and by suggesting possible solutions. This form of assistance is employed with the spell-checking service, which provides a list of suggested corrections. Another way to facilitate assistance is to provide sample inputs, or templates. The OCGC abstracted metadata template practices this method, providing sample metadata entries that have been submitted by its participating organizations.

In this section, we examined usability issues relevant for designing web-based applications, in particular for our metadata entry tool, which involves significant number of user-system interactions. We reviewed potential causes for user errors and suggested possible solutions. These have been implemented in the initial system. However, ongoing efforts will be required to gradually improve usability, by listening to user feedback and studying the characteristics of user errors logged by the applications.

4. Selected Design and Implementation Details

This section goes into detail about selected implementation issues. Section 4.1 discusses some technical challenges in modeling schema documents, and presents information on how each component of the Schema Model Builder was implemented. Section 4.2 discusses the HTML Generator. The interfaces produced by the HTML Generator are distinct, depending on the purpose and intended audience: subsetting interface for subset designers or metadata entry interface for the data owners.

4.1. Building XML Schema Object Group

As discussed in previous sections, the Schema Model Builder (SMB) produces as output a collection of object representations for input documents written in XML Schema Language. Figure 4.1 shows the architecture of the SMB. Its software components include a DOM Parser and a Schema Parser. The DOM Parser produces an abstract data structure that represents XML documents as trees of nodes. It reads an XML document from a stream and builds a DOM `Document` object that represents a complete well-formed XML document. We used the Xerces DOM Parser from the Apache Xerces project to initially parse the schema documents into the tree structure. The output of a parse intended for use with the DOM interface is an `org.w3c.dom.Document` object. This object acts as a handle to the tree representing the input XML data. In terms of the element hierarchy, it is equivalent to one level above the root element in the XML document. Elements in the XML document can be accessed via the instance of the `Document` object. The root element is retrieved with the `getDocumentElement()` method, which returns an instance of the `org.w3c.dom.Element` object. The client program, in this case the Schema Parser, calls the methods of `Document` and the other DOM interfaces to traverse the tree in memory and to manipulate tree nodes.

The Schema Parser consists of two sub-components: a Syntax Analyzer and a Symbol Resolver. The first traverses each node in the DOM `Document` object and instantiates appropriate XSO objects based on the name and type of each node, while setting the parent-child relationship for each XSO object. The Symbol Resolver finalizes the transformation from DOM objects to XSO objects by resolving element/type references in the XSO objects.

4.1.1. Modeling XML Schema Components

Each schema element is represented with an individual class definition in XSO, located in the package `org.nacse.xso.models`. The list of classes and corresponding schema elements can be found in Appendix C. Similar to DOM objects, an XSO object represents the input schema document as a tree of nodes. Unlike the DOM object, however, the XSO object represents the same document with the `XSSchema` interface as a handle to the tree. Element definitions in the schema document are represented as nodes in the tree, and they can be accessed as the child elements of the `XSSchema` interface. `XSSchema`

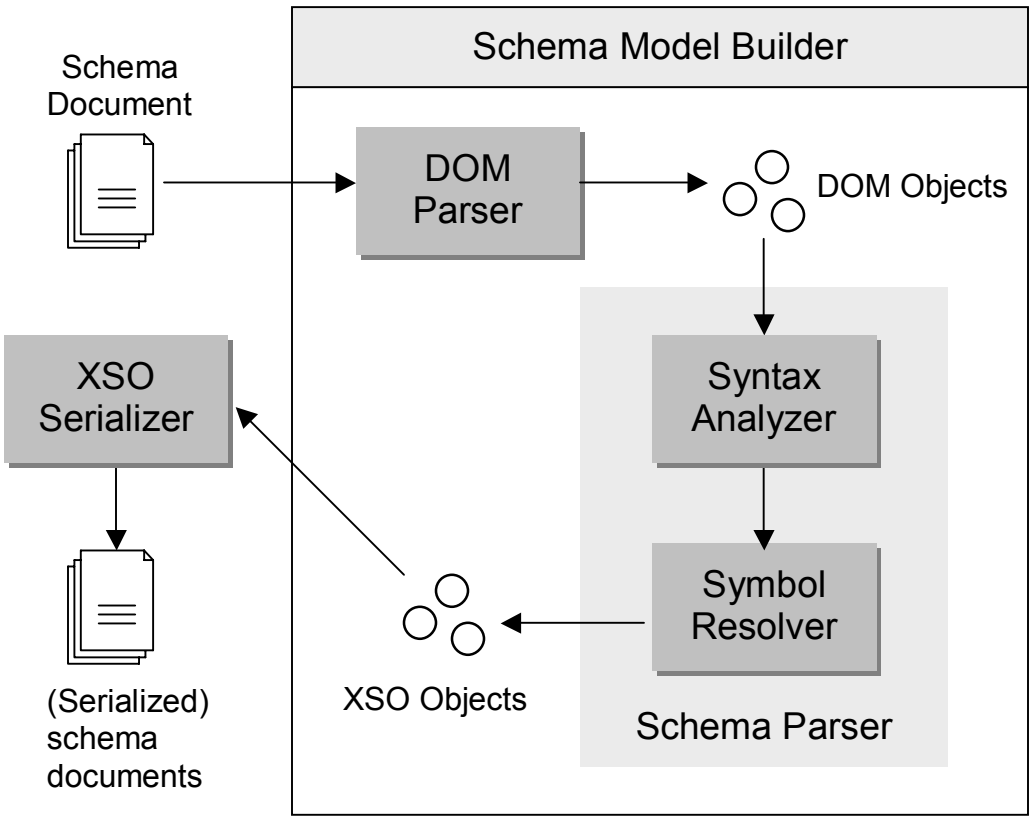


Figure 4.1. Structure of the SMB

manages, in addition to other information, the schema document's `targetNamespace`, and prefixes and corresponding URIs for other schema documents.

All XSO classes extend one abstract class: `XSNode`. This class contains a Java `ArrayList` that is used to hold the references to child elements. Each schema element has a different set of allowed child elements. For instance, an `xs:list` element may include `xs:annotation` and `xs:simpleType` elements, while `xs:all` may include `xs:annotation` and `xs:element`. Thus, `XSNode` defines an abstract method `addChild()`, and each class that extends `XSNode` overrides it for its specific allowable child elements. Public methods in the `XSNode` interface are shown in Table 4.1.

The type of class that extends `XSNode` can be identified with the `getNodeType`() method, which returns an integer value. The integers that identify types of classes are defined in a helper class, `SchemaSymbols`. In addition to these integer definitions, the `SchemaSymbols` class is also used as a "dictionary" of the predefined symbols for primitive data types in the W3C XML Schema Language definition. The dictionary is used to distinguish the tokens for primitive data type from the ones for user-defined data types.

Method	Description
getNodeType()	Returns an integer describing the type of the node. The integer values are defined in class SchemaSymbols.
getContainer()	Returns the XSSchema object that is the root of the hierarchy this node belongs to.
addChild()	Adds the node referenced by the parameter to the ArrayList that stores the child nodes.
getChildList()	Returns the ArrayList storing the child nodes.
getChild()	Returns the child node with the name and type passed as the parameters.
getPrefix()	Returns the prefix for the W3C XML Schema elements, specified in this schema document.
getParent()	Returns the parent node of this node.

Table 4.1. Public Methods In the XSNode Interface

In a DOM object, the attributes of an element are represented as child nodes of that element, as shown in Figure 2.2. In an XSO object, they are represented as fields in the corresponding class definition, associated with accessor (get) and mutator (set) methods. Most attributes expect a string for their value, and thus are defined with the Java String data type. Numerical attributes, such as maxOccurs and minOccurs, are still represented with the String data type, because strings are more convenient in serialization, and they may contain a string value in certain circumstances (for example, maxOccurs can contain "unbounded" as well as nonnegative integers).

4.1.2. Syntax Analyzer

The Syntax Analyzer is responsible for transforming DOM objects representing input schema documents to intermediate XSO objects. It traverses each node in a DOM object, processes the information stored in the node, and instantiates appropriate XSO objects. The process of transforming a DOM node to a corresponding XSO node is illustrated with the example shown in Figure 4.3. As the Syntax Analyzer visits a DOM Element node, it first retrieves the name of the node with the getNodeName() method. If it determines that the name is xs:element, it instantiate an XSElement node. Then the child nodes of the currently visited node are retrieved. The Syntax Analyzer visits Attribute nodes first, setting the fields in the last instantiated XSElement node. In the example, the name of the instantiated XSElement node is set to "book". The analyzer then starts visiting the remaining Element nodes, repeating the process for each. Before moving on to the next node, it adds each instantiated XSO node to the array in the node that is "one step above". In the example, the XSSequence node is added to the array in the XSComplexType node, then the XSComplexType node is added to the array in the XSElement node. This technique, which starts with the top-most element and processes

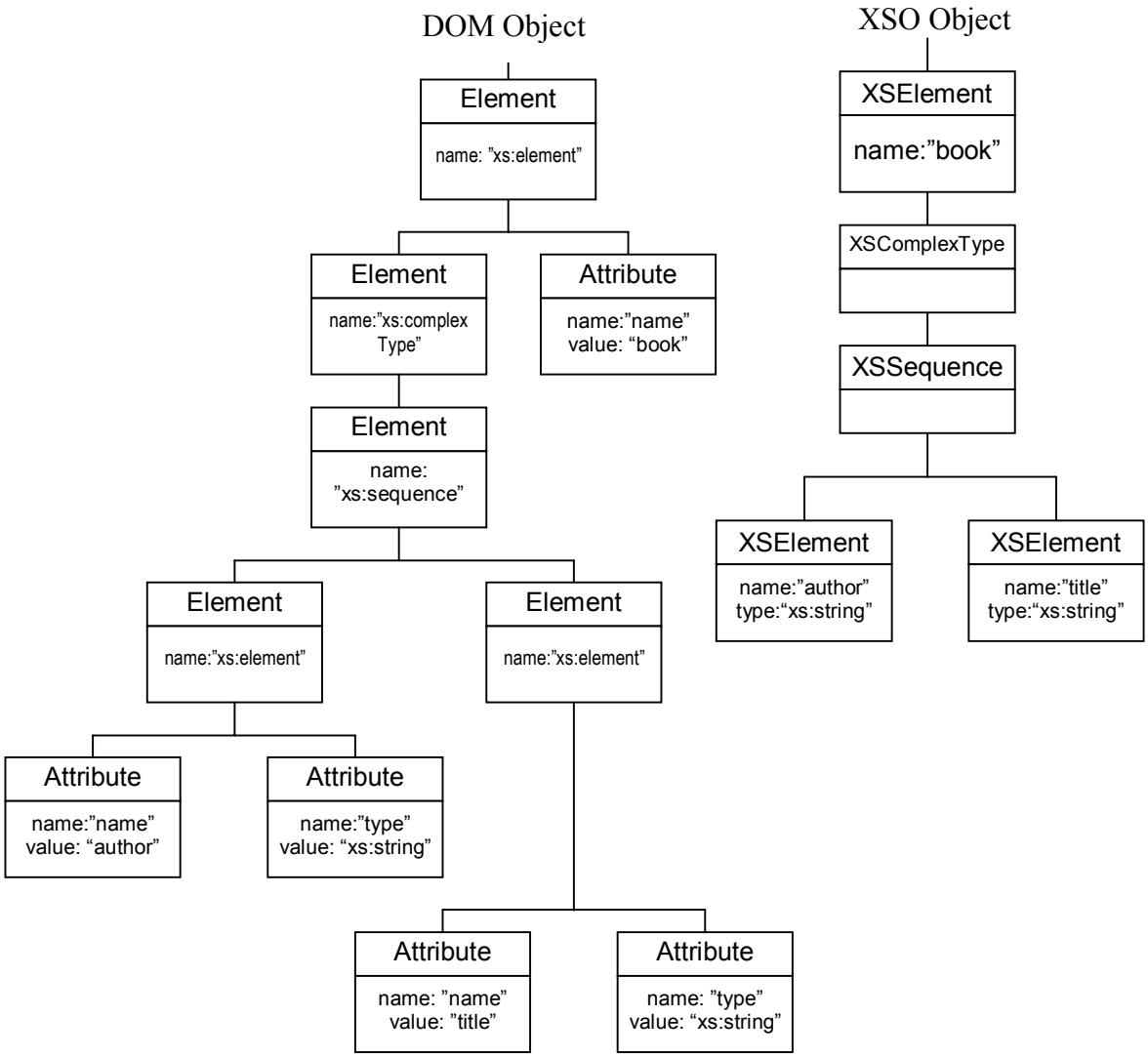


Figure 4.2. Differences between DOM and XSO objects

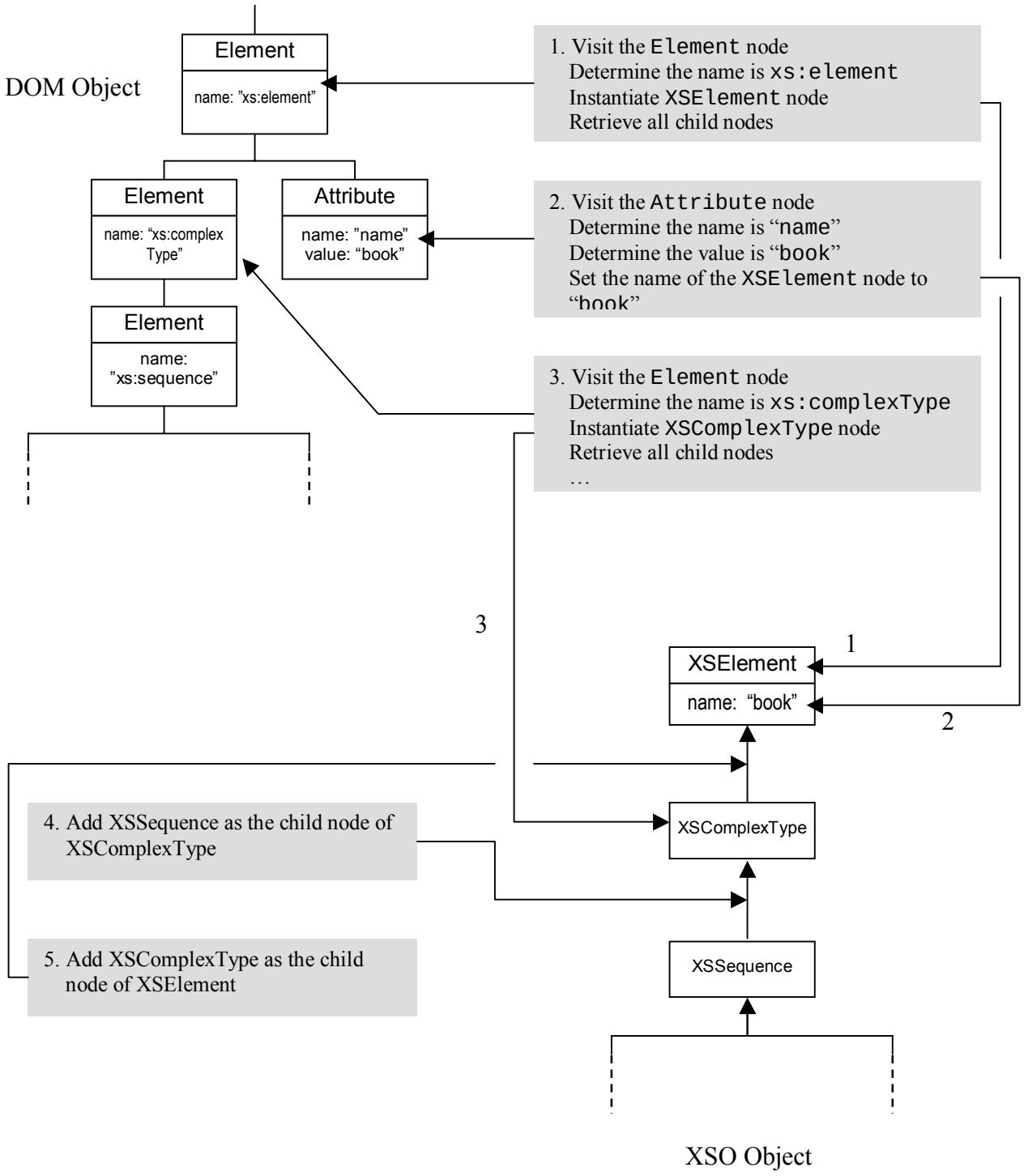


Figure 4.3. Transforming a DOM Object to an XSO Object

each child element recursively as it is encountered, is a type of *recursive-descent parsing* [1].

4.1.3. Symbol Resolver

The transformed XSO objects are finalized when the external references are resolved. External references in XML Schema Language can appear in the following forms:

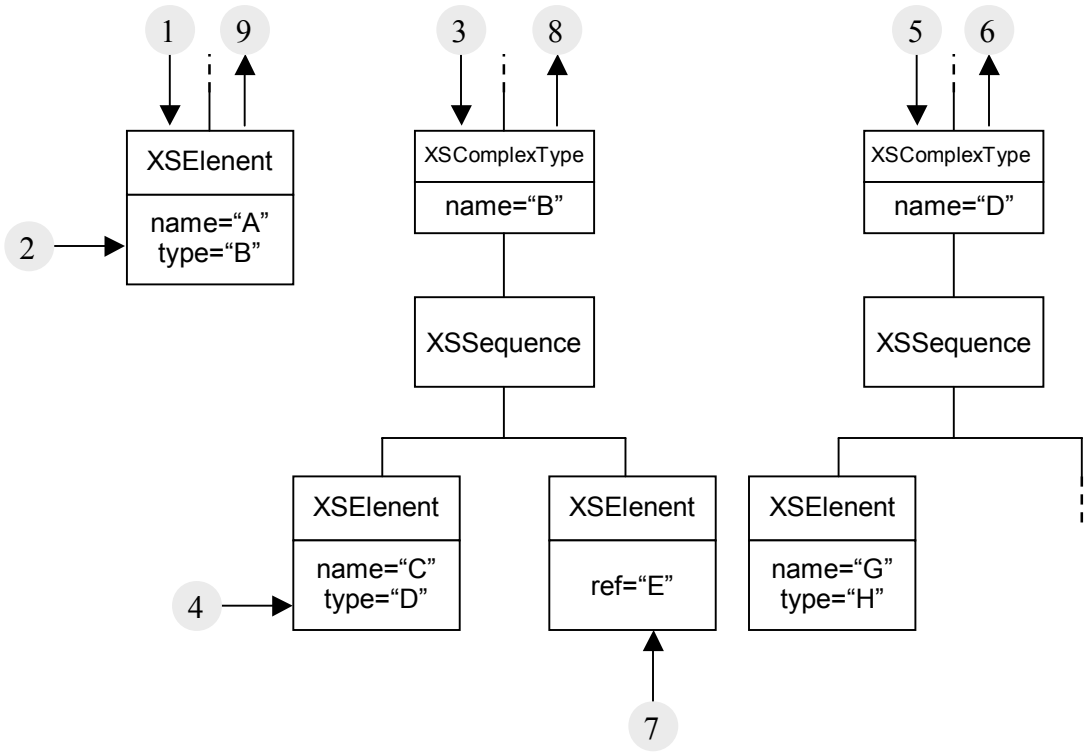
- Element references: <xs:element ref="foo"/>
- Attribute references: <xs:attribute ref="foo"/>
- Type references in the element definition: <xs:element name="foo" type="bar"/>
- Type references in an extension: <xs:extension base="foo">
- Type references in a restriction: <xs:restriction base="foo">

There are other occasions where external references are used, such as `xs:group`, `xs:union`, `xs:list`, `xs:attributeGroup`, etc. However, they are used very rarely, and the current version of the Symbol Resolver does not support the use of external references in these tags.

Symbol resolution is performed in two steps. First, the Symbol Resolver probes existence of the entity the symbol references. This involves resolving the location of the global element/type definitions within the schema the symbol references. The identifiers used in the references can be locally defined, or located in another schema document. (Determining the location of the `XSSchema` object that corresponds to the symbol's prefix is illustrated in Figure 2.4.) The `XSSchema` object then returns the `XSNode` object that matches the name and the type of the symbol we are looking for. If the referenced symbol is located in another schema, the Symbol Resolver requests the `XSSchema` object within which the symbol was used to translate the prefix of the symbol to a corresponding URI. The URI is then passed to the global mapping table to locate the `XSSchema` object whose `targetNamespace` matches it. Finally, the returned `XSSchema` object verifies whether it contains the definition that the symbol is referencing. If the entity is not found, an exception will be thrown.

The second step is to import the node that has been resolved, as well as the child nodes that branch out from it. Copying out the whole branch is necessary for the cases when the referenced entities are to be altered in some way. This includes adding data to the nodes (as in the case of getting user inputs from the form interface) and modifying constraints of the referenced entities (as in metadata subsetting). A top-down parsing technique is also employed here. If the node currently being resolved makes external reference to another node, the current resolution is set aside and the process recurs. This is illustrated in Figure 4.4.

This algorithm has one serious problem. Since XML Schema Language does not prohibit a child element definition from referencing its parent, there is a danger of entering an infinite cycle. The situations where this can occur include the following scenarios:



1. Resolving A begins
2. Reference B encountered
3. Resolving B begins
4. Reference D encountered
5. Resolving D begins
   …
6. Resolving D ends
7. Reference E encountered
   …
8. Resolving B ends
9. Resolving A ends

Figure 4.4. Recursively Resolving External Symbols

- A (local) element definition refers to the complexType within which the element is defined.

```
<xs:complexType name="XXX">
    <xs:sequence>
        <xs:element name="EEE" type="XXX"
         minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

- An element definition contains a nested element that refers to itself.

```
<xs:element name="AAA">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="AAA" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Two complexType definitions are referenced by each other's nested element definitions.

```
<xs:complexType name="XXX">
    <xs:all>
        <xs:element name="AAA" type="YYY"/>
    </xs:all>
</xs:complexType>

<xs:complexType name="YYY">
    <xs:all>
        <xs:element name="BBB" type="XXX"/>
    </xs:all>
</xs:complexType>
```

To resolve this issue, the Symbol Resolver manages a reference counter for all global identifiers in the schema document. Upon entering a process for resolving a symbol A, the reference counter for A is incremented, and it is decremented when the process for A finishes. When the symbol A is referenced, the Symbol Resolver looks up the reference counter for A. If the value is not 0, it means that the process for A has not terminated yet, and the reference is not further resolved.

4.2. Generating HTML Pages for Metadata Entry

In Section 2.6, we showed that it is relatively easy to generate HTML form pages that reflect the structural and data constraints expressed in schema documents. Structural constraints let us determine the location where each input field should appear in a form page,

and what type of input field is needed. Data constraints tell us what kind of validations are required for each input field.

The HTML Generator uses standard HTML tables to collect and arrange input fields that are logically related, using rows and columns of cells. Compound elements (an element with child elements) are used to generate containers for their child elements, while elements with simple content (those having a primitive data type, or `simpleType`) are transformed into input fields where actual data entries take place.

The type of input field is determined by the data type of the corresponding element. The default input type is a text field. The types of element that correspond to this include `xs:string`, `xs:integer`, `xs:date`, etc. The `XSElement` object also contains a field `defaultSize` which can be manipulated by the subset designer. This stores the expected length of the input data. If it is greater than a certain value (the default value is 40), the object is transformed to a `textarea` field.

Enumerations are transformed to a selection, which includes checkboxes, radio-buttons, drop-down lists, and scrolling-lists. Again, the subset designer chooses which input type should be used. If multiple values can be selected, checkboxes and scrolling-lists are available; otherwise, radio-buttons and drop-down lists are used. Radio-buttons are also used with the boolean type (`xs:boolean`) to represent an on/off switch.

HTML does not have a namespace facility, so all input fields must have unique identifiers to prevent collisions. As described in Section 2.5, the HTML Generator superimposes the scope of the XML elements in naming the form fields. The method `buildElement`() takes as parameters the `String` that holds the list of all element names in the hierarchy, concatenated with ":", and the current `XSElement` that is currently processed, and attaches the name of the current `XSElement` at the end. If the current `XSElement` has any child `XSElement`s, the String object is passed along to them; otherwise the input field is generated there.

The HTML Generator also generates JavaScript documents for validating input fields on the client-side. Validations are roughly classified into the following categories:

- Omission in mandatory fields:

```
if (form.FIELD.value == "") {
    alert("The field " + FIELD.name + " must be filled in.");
    form.FIELD.className = 'invalid';
    form.FIELD.focus();
    form.FIELD.select();
    return false;
}
```

- Text input in numerical fields:

```
function isAlpha(str) {
      var regexp = /^([A-Za-z])+$/;
      return regexp.test(str);
}

if (isAlpha(form.NUMERICAL.value) == true) {
```

```
        alert('Fill out the required field!');
        form.NUMERICAL.className = 'invalid';
        form.NUMERICAL.focus();
        form.NUMERICAL.select();
        return false;
    }
```

Other types of validations are also possible. However, there are disadvantages in using too much client-side validation. It takes more time for the client to download the code necessary for validation. Client-side validation also uses processing and memory resources on client's machine. Finally, it is not feasible to employ complex and complicated validation logic. For those reasons, the current version of the HTML Generator only supports the above-mentioned categories.

4.3. Subsetting Schema Objects

The interface of the Subset Generator was presented in Section 2.3.2. As discussed in that section, it dynamically generates an interface for the subsetting process, applying a mechanism similar to that of the HTML Generator. If the element the subset designer wants to manipulate has a complex content model, the following attribute information will be presented to him or her in a pop-up window.

- maxOccurs: maximum number of times the element can be repeated.
- minOccurs: minimum number of occurrences for the field.
- Label: string label for the field that will be displayed.
- Help: detailed information about the field
- Example: example user input

If the element has a simple content type, the following attribute information will be presented:

- All attribute information for the complex content model
- A default Value

Elements with simple content type can be further categorized, based on the type of input. Table 4.2 summarizes the attributes for each type.

Most of these attributes are directly expressed in the W3C XML Schema Language definition, and the modified values for these attributes can be saved as standard XML Schema tags. Some attributes, however, including Label, Help, and Example, are not defined in Schema Language. These are stored as child elements of <xs:appInfo>, with the namespace prefix "xso", as shown in Figure 4.6.

Attribute	What it does	Integer	Float	Text
enumeration	The set of allowed input values	X	X	X
maxExclusive	An exclusive maximum value	X	X	
maxInclusive	An inclusive maximum value	X	X	
minExclusive	An exclusive minimum value	X	X	
minInclusive	An inclusive minimum value	X	X	
pattern	A regular expression over the lexical space	X	X	X
maxLength	Maximum length (number of characters)			X
minLength	Maximum length (number of characters)			X
length	Fixed length (number of characters)			X
totalDigits	Maximum number of decimal digits	X	X	
fractionDigits	Maximum number of fractional digits		X	

Table 4.2. Attribute Information for Data Types

```
<xs:appInfo>
     <xso:label>Label Text</xso:label>
     <xso:help>Help Text</xso:help>
    <xso:example>Example Text</xso:example>
</xs:appInfo>
```

Figure 4.6. Non-standard Attributes Are Stored in `xs:appInfo`

5. Conclusions and Future Work

5.1. Contribution of This Work

As the amount of research data that exists online grows exponentially, the value of cataloguing descriptions of the data to assist us in quickly determining its relevance, location, and accessibility becomes more and more important. Although there are a number of international standards that govern the format and structure of metadata in several disciplines, the complexity of these standards makes it difficult for data owners to create a high-quality metadata that is useful and functional. Reducing the number of fields from a relevant standard helps both the data owners and the organizations that manage and distribute the metadata. Moreover, even after organizations compile the most relevant fields from the standard, these organizations, often managed by scientists, have little or no resources and expertise required to build and manage a tool for integrating metadata entries.

The Metadata Subsetting Framework allows data clearinghouses to create a subset of a metadata standard, containing the fields that are most relevant to a specific disciplinary domain. The subset then enables data owners to create metadata more easily, because the entry process is less time-consuming and less error-prone. Reducing the number of fields for which users have to figure out the meaning and applicability means that the metadata created is likely to be of higher quality. Even more importantly, it encourages data owners to create more metadata. Given the concerns and frustration data owners have been expressing about the complexity and comprehensiveness of metadata standards, metadata subsetting is a practical and timely development.

The ability to automatically generate interfaces for metadata entry helps data clearinghouses to quickly build and deploy a portal site customized for their specific discipline. Scientists managing these sites do not have to worry about writing tedious HTML pages, and can focus on more important issues. The validation mechanism provided by the framework helps ensure the integrity of input metadata, making it easier to create standard-compliant metadata.

5.2. Related Work

There are a number of tools designed to assist the metadata entry process, some of them commercial products. These are generally designed for a specific metadata standard and a specific computing platform. The Spatial Metadata Management System (SMMS) from Intergraph, shown in Figure 5.1, is the most widely used commercial system for creating FGDC-compliant metadata [45]. Internally, it uses Microsoft Access tables to manage the metadata. SMMS covers all FGDC metadata, plus the NBII biological data profile extension [2]. It can automatically collect geospatial metadata elements from GIS data layers. SMMS works only on Microsoft Windows platforms, and does not support subsetting. This means that users must grapple with all 300+ fields in performing metadata entry.

MetaMaker, shown in Figure 5.2, is a metadata creation tool for the FGDC CSDGM standard and is written in Microsoft Access [29]. The metadata fields are organized into seven major sections, each of which is completed through forms with text entry boxes. It only works on Microsoft Windows platform, and does not support metadata subsetting. The project is currently defunct, and no support is available.

Figure 5.3 depicts Morpho, a data management application designed for the Ecological Metadata Language [28]. It includes a data query tool, a metadata creation tool, and data management facilities. Written as a Java application, it works on Windows, Macintosh, and UNIX platforms. However, it is an academic research project rather than a commercial product, so future support is uncertain. It does not support metadata subsetting.

There are a variety of tools available for processing XML in programming languages. Xerces, from the Apache Software Foundation, is an XML parser written in Java and C++ [51]. A Perl wrapper is provided for the C++ version, allowing access to a fully validating DOM XML parser from Perl scripts, while a COM wrapper allows compatibility with the Microsoft MSXML parser. Xerces allows validation of instance documents against XML schema documents.

XSV (XML Schema Validator) from the University of Edinburgh is an open source work that attempts a conformant schema-aware processor [54]. It is implemented in Python. A forms-based interface on the web allows online use; a command-line utility is also available.

Castor is an open source data binding framework for Java and XML [4]. From XML schema documents, it generates Java source code that can be used for marshalling and unmarshalling XML instance documents. It also provides the Castor JDO package, which provides a mechanism for binding a Java object model to a relational database model, usually referred to as object-to-relational mapping.

Cocoon provides an XML publishing framework [6]. It is a Java servlet for automatically creating HTML through the processing of statically or dynamically generated XML files. Cocoon is also able to perform more sophisticated formatting, such as XSL:FO rendering of PDF, WML formatting for WAP-enabled devices, or direct XML serving to XML and XSL-aware clients.

5.3. Challenges Faced

Because of the complexity of metadata standards (in terms of both the number of fields and individual fields' cardinality requirements and dependency on other fields) and the variety of metadata standards in various disciplines, it was difficult to analyze and digest the standards in conceptualizing the requirements and design goals for this project. After the decision was made to use XML Schema Language for the project, the standards that were not written in XML Schema Language had to be converted. For example, the FGDC standard had been written in DTD format, so it was converted manually to Schema Language.

It was soon discovered that none of the tools discussed in Section 5.2 had the capability of *unmarshalling* schema documents, or producing an in-memory representation for schema documents. Several XML parsers are available for *validating* XML instance documents against schema documents, but none of them could perform the Schema-parsing

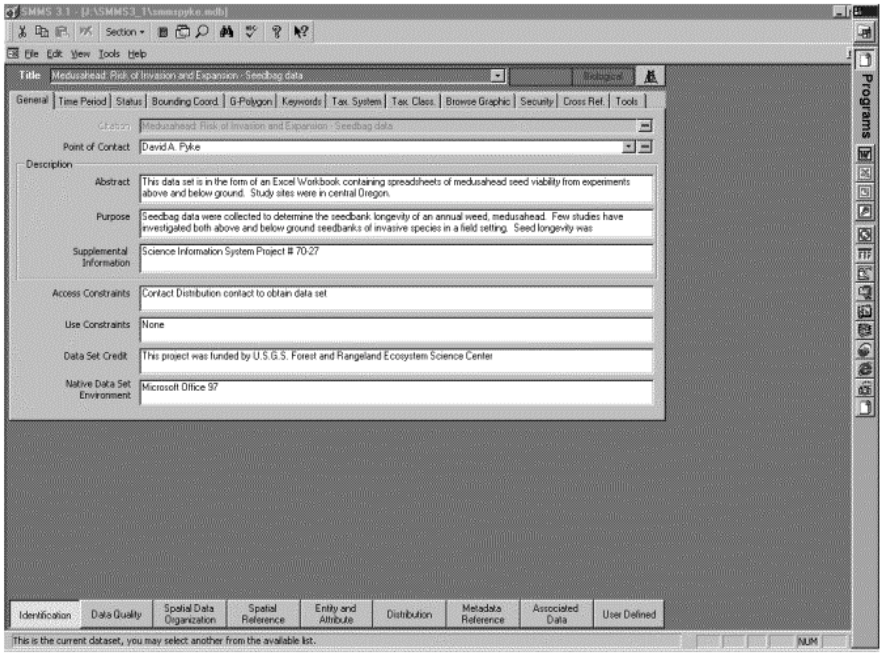


Figure 5.1. Spatial Metadata Management System

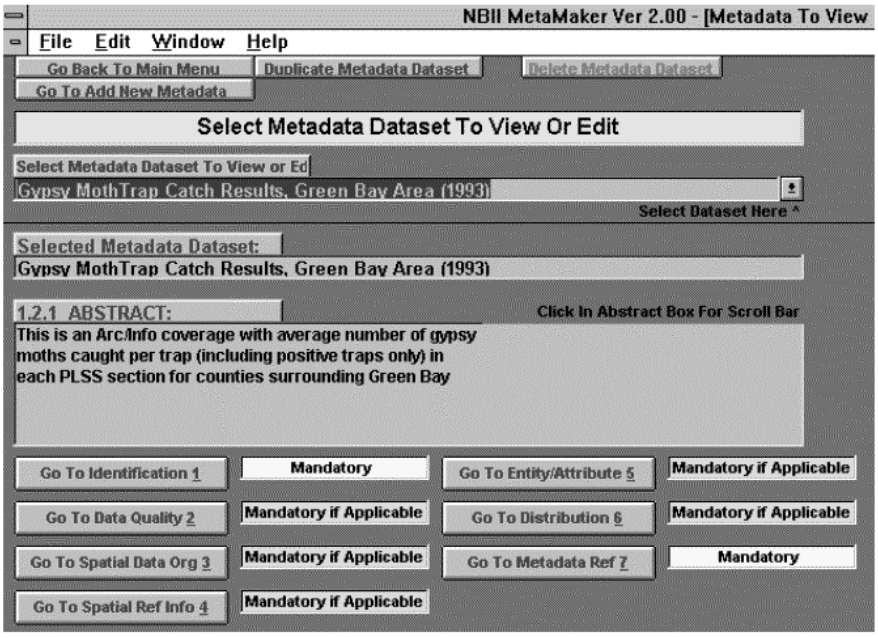


Figure 5.2. MetaMaker

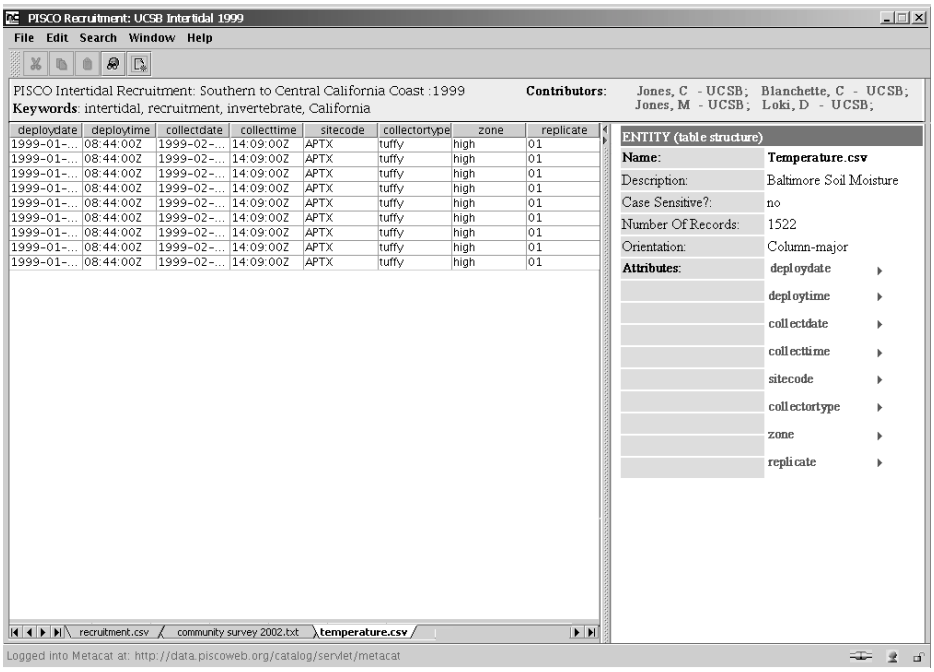


Figure 5.3. Morpho

operation discussed in Section 2.2.2. Some Java-XML data binding tools, such as Castor, can generate Java source code reflecting the structure of schema documents. However, these files must be compiled and linked with the framework before they can be used, which was not feasible in our framework where several subsetted schema documents may be created dynamically. Thus the Schema Model Builder had to be written from the ground up. This occupied the largest portion of development effort in terms of time and resources.

Another challenge was that the complexity of XML Schema Language made it difficult to create an interface that reflects the structure imposed in input schema documents. The depth of nested child elements can be sometimes more than is practical for displaying in nested tables. Creating an interface and validation mechanism that is efficient and serves multiple platforms was also challenging. Several programming and modeling languages, such as JavaScript, CSS, DOM, and HTML, were incorporated for this purpose, each of which involved a significant learning curve.

5.4. Future Work

A number of features would enhance the Metadata Subsetting Framework's capabilities. Their implementation has either not begun, or is still in a very early stage.

Integrating the project with the J2EE framework needs to be completed. Currently several essential features, such as managing session objects, are not functional. Some of usability techniques discussed in Section 3 have not yet been incorporated, and further research is required to enhance the usability of the framework.

Data persistency also needs to be integrated. Several products, both commercial and open-source, are available for maintaining XML data in permanent databases. The functionality of automatically creating Data Description Language for schema documents could be implemented relatively easily.

Further, number of standard tags in XML Schema Language are either not supported or only partially supported. Most of theme are related to XPath-based identity and uniqueness checks, including `xs:key`, `xs:keyref`, `xs:selector`, etc. The coverage of standard tags in future versions will hopefully be more complete.

Finally, supporting validations for semantic constraints and business logic should be incorporated into the framework. Complicated logic, such as "The value entered in field A must be greater than the value entered in field B, and less than the value entered in field C" or "The value in field A must match one of the values entered in B, C, or D," cannot be expressed using XML Schema alone. Such features could be implemented with procedural programming languages such as Java, or with XPath/XSLT-based validation techniques [20].

Appendix A: XML Schema Representation of the FGDC Content Standard for Geospatial Metadata

Section 1: Identification Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schemas/csdgm/idinfo"
  xmlns="http://www.nacse.org/xso/schemas/csdgm/identinfo"
  xmlns:citeinfo="http://www.nacse.org/xso/schemas/csdgm/citeinfo"
  xmlns:cntinfo="http://www.nacse.org/xso/schemas/csdgm/cntinfo"
  xmlns:timeinfo="http://www.nacse.org/xso/schemas/csdgm/timeinfo"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="citeinfo" schemaLocation="./citeinfo.xsd"/>
  <xs:import namespace="cntinfo" schemaLocation="./cntinfo.xsd"/>
  <xs:import namespace="timeinfo" schemaLocation="./timeinfo.xsd"/>

  <xs:element name="idinfo" type="idinfoType"/>

  <xs:complexType name="idinfoType">
    <xs:sequence>
      <xs:element ref="citeinfo:citeinfo"/>
      <xs:element name="descript" type="descriptType"/>
      <xs:element name="timeperd" type="timeperdType"/>
      <xs:element name="status" type="statusType"/>
      <xs:element name="spdom" type="spdomType"/>
      <xs:element name="keywords" type="keywordsType"/>
      <xs:element name="accconst" type="noneOrFreeType"/>
      <xs:element name="useconst" type="noneOrFreeType"/>
      <xs:element ref="cntinfo:cntinfo" minOccurs="0"/>
      <xs:element name="browse" type="browseType" minOccurs="0"
       maxOccurs="unbounded"/>
      <xs:element name="datacred" type="xs:string" minOccurs="0"/>
      <xs:element name="secinfo" type="secinfoType" minOccurs="0"/>
      <xs:element name="native" type="xs:string" minOccurs="0"/>
      <xs:element ref="citeinfo:citeinfo" minOccurs="0"
       maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="noneOrFreeType">
    <xs:union>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="None"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:string"/>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>

  <xs:complexType name="descriptType">
    <xs:sequence>
```

```
    <xs:element name="abstract" type="xs:string"/>
    <xs:element name="purpose" type="xs:string"/>
    <xs:element name="supplinf" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="timeperdType">
  <xs:sequence>
    <xs:element ref="timeinfo:timeinfo"/>
    <xs:element name="current">
      <xs:simpleType>
        <xs:union>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="ground condition"/>
              <xs:enumeration value="publication date"/>
            </xs:restriction>
          </xs:simpleType>
          <xs:simpleType>
            <xs:restriction base="xs:string"/>
          </xs:simpleType>
        </xs:union>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="statusType">
  <xs:sequence>
    <xs:element name="progress">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Complete"/>
          <xs:enumeration value="In work"/>
          <xs:enumeration value="Planned"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="update">
      <xs:simpleType>
        <xs:union>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Continually"/>
              <xs:enumeration value="Daily"/>
              <xs:enumeration value="Weekly"/>
              <xs:enumeration value="Monthly"/>
              <xs:enumeration value="Annually"/>
              <xs:enumeration value="Unknown"/>
              <xs:enumeration value="As needed"/>
              <xs:enumeration value="Irregular"/>
              <xs:enumeration value="None planned"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:union>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="spdomType">
```

```
  <xs:sequence>
    <xs:element name="bounding">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="westbc" type="eaboundingType"/>
          <xs:element name="eastbc" type="eaboundingType"/>
          <xs:element name="northbc" type="nsboundingType"/>
          <xs:element name="southbc" type="nsboundingType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="dsgpoly" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="dsgpolyo" type="gRingType"/>
          <xs:element name="dsgpolyx" type="gRingType" minOccurs="0"
           maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="gRingType">
  <xs:choice>
    <xs:element name="grngpoin" minOccurs="4" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="gringlat">
            <xs:simpleType>
              <xs:restriction base="xs:double">
                <xs:minInclusive value="-90.0"/>
                <xs:maxInclusive value="90.0"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="gringlon">
            <xs:simpleType>
              <xs:restriction base="xs:double">
                <xs:minInclusive value="-180.0"/>
                <xs:maxInclusive value="180.0"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="gring">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="gring"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>

  </xs:choice>
</xs:complexType>

<xs:simpleType name="eaboundingType">
  <xs:restriction base="xs:double">
```

```
    <xs:minInclusive value="-180.0"/>
    <xs:maxInclusive value="180.0"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nsboundingType">
  <xs:restriction base="xs:double">
    <xs:minInclusive value="-90.0"/>
    <xs:maxInclusive value="90.0"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="keywordsType">
  <xs:sequence>
    <xs:element name="theme" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="themekt" type="noneOrFreeType"/>
          <xs:element name="themekey" type="xs:string"
           maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="place" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="placekt" type="noneOrFreeType"/>
          <xs:element name="placekey" type="xs:string"
           maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="stratum" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="stratkt" type="noneOrFreeType"/>
          <xs:element name="stratkey" type="xs:string"
           maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="temporal" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="tempkt" type="noneOrFreeType"/>
          <xs:element name="tempkey" type="xs:string"
           maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="browseType">
  <xs:sequence>
    <xs:element name="browsen" type="xs:string"/>
    <xs:element name="browsed" type="xs:string"/>
    <xs:element name="browset">
      <xs:simpleType>
        <xs:union>
          <xs:simpleType>
```

```
              <xs:restriction base="xs:string">
                <xs:enumeration value="CGM"/>
                <xs:enumeration value="EPS"/>
                <xs:enumeration value="EMF"/>
                <xs:enumeration value="GIF"/>
                <xs:enumeration value="JPEG"/>
                <xs:enumeration value="PBM"/>
                <xs:enumeration value="PS"/>
                <xs:enumeration value="TIFF"/>
                <xs:enumeration value="WMF"/>
                <xs:enumeration value="XWD"/>
              </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
              <xs:restriction base="xs:string"/>
            </xs:simpleType>
          </xs:union>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="secinfoType">
    <xs:sequence>
      <xs:element name="secsys" type="xs:string"/>
      <xs:element name="secclass">
        <xs:simpleType>
          <xs:union>
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="Top secret"/>
                <xs:enumeration value="Secret"/>
                <xs:enumeration value="Confidential"/>
                <xs:enumeration value="Restricted"/>
                <xs:enumeration value="Unclassified"/>
                <xs:enumeration value="Sensitive"/>
              </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
              <xs:restriction base="xs:string"/>
            </xs:simpleType>
          </xs:union>
        </xs:simpleType>
      </xs:element>
      <xs:element name="sechandl" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Section 2: Data Quality Information

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
  targetNamespace="http://www.nacse.org/xso/schema/csdgm/dataqual"
  xmlns="http://www.nacse.org/xso/schema/csdgm/dataqual"
  xmlns:cntinfo="http://www.nacse.org/xso/schema/csdgm/cntinfo"
```

```
xmlns:citeinfo="http://www.nacse.org/xso/schema/csdgm/citeinfo">

<xs:element name="attracc" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="attraccr" type="xs:string"/>
      <xs:element name="qattracc" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="attraccv" type="xs:string"/>
            <xs:element name="attracce" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>


<xs:element name="logic" type="xs:string"/>

<xs:element name="complete" type="xs:string"/>

<xs:element name="posacc" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="horizpa" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="horizpar" type="xs:string"/>
            <xs:element name="qhorizpa" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="horizpav" type="xs:string"/>
                  <xs:element name="horizpae" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="vertacc" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="vertpa" type="xs:string"/>
            <xs:element name="vertpar" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="vertpav" type="xs:string"/>
                  <xs:element name="vertpae" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="lineage">
```

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="srcinfo" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="srccite" type="citeinfo:citeinfoType"/>
          <xs:element name="srcscale" type="xs:integer"
           minOccurs="0"/>
          <xs:element name="typesrc">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="paper"/>
                <xs:enumeration value="stable-base material"/>
                <xs:enumeration value="microfiche"/>
                <xs:enumeration value="microfilm"/>
                <xs:enumeration value="audiocassette"/>
                <xs:enumeration value="chart"/>
                <xs:enumeration value="filmstrip"/>
                <xs:enumeration value="transparency"/>
                <xs:enumeration value="videocassette"/>
                <xs:enumeration value="videodisc"/>
                <xs:enumeration value="videotape"/>
                <xs:enumeration value="physical model"/>
                <xs:enumeration value="computer program"/>
                <xs:enumeration value="disc"/>
                <xs:enumeration value="cartridge tape"/>
                <xs:enumeration value="magnetic tape"/>
                <xs:enumeration value="online"/>
                <xs:enumeration value="CD-ROM"/>
                <xs:enumeration value="electronic bulletin board"/>
                <xs:enumeration value="electronic mail system"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="srctime">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="srccurr">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="ground condition"/>
                      <xs:enumeration value="publication date"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="srccitea" type="xs:string"/>
          <xs:element name="srccontr" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="procstep" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="procdesc" type="xs:string"/>
          <xs:element name="srcused" type="xs:string" minOccurs="0"
           maxOccurs="unbounded"/>
          <xs:element name="procdate" type="xs:date"/>
          <xs:element name="proctime" type="xs:time"/>
```

```
              <xs:element name="srcprod" type="xs:string" minOccurs="0"
               maxOccurs="unbounded"/>
              <xs:element name="proccoat" type="cntinfo:cntinfoType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="cloud" type="xs:integer"/>
</xs:schema>
```

Section 3: Spatial Data Organization Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schema/csdgm/spdoinfo"
  xmlns="http://www.nacse.org/xso/schema/csdgm/spdoinfo">

  <xs:element name="spdoinfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="indspref" type="xs:string" minOccurs="0">
        <xs:element name="direct" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Point"/>
              <xs:enumeration value="Vector"/>
              <xs:enumeration value="Rastor"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="ptvctinf" minOccurs="0">
          <xs:complexType>
            <xs:choice>
              <xs:element name="sdtsterm">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="sdtstype">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="Point"/>
                          <xs:enumeration value="Entity point"/>
                          <xs:enumeration value="Label point"/>
                          <xs:enumeration value="Area point"/>
                          <xs:enumeration value="Node, planar graph"/>
                          <xs:enumeration value="Node, network"/>
                          <xs:enumeration value="String"/>
                          <xs:enumeration value="Link"/>
                          <xs:enumeration value="Complete chain"/>
                          <xs:enumeration value="Area chain"/>
                          <xs:enumeration value="Network chain, planar
                           graph"/>
                          <xs:enumeration value="Network chain,
                           nonplanar graph"/>
                          <xs:enumeration value="Circular arc, three
                           point center"/>
```

```
                    <xs:enumeration value="Elliptical arc"/>
                    <xs:enumeration value="Uniform B-spline"/>
                    <xs:enumeration value="Piecewise Bezier"/>
                    <xs:enumeration value="Ring with mixed
                     composition"/>
                    <xs:enumeration value="Ring composed of
                     arcs"/>
                    <xs:enumeration value="G-polygon"/>
                    <xs:enumeration value="GT-polygon composed of
                     rings"/>
                    <xs:enumeration value="GT-polygon composed of
                     chains"/>
                    <xs:enumeration value="Universe polygon
                     composed of rings"/>
                    <xs:enumeration value="Universe polygon
                     composed of chains"/>
                    <xs:enumeration value="Void polygon composed
                     of rings"/>
                    <xs:enumeration value="Void polygon composed
                     of chains"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="ptvctcnt" type="xs:integer"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="vpfterm">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="vpflevel" type="xs:integer"/>
              <xs:element name="vpfinfo">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="vpftype">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="Node"/>
                          <xs:enumeration value="Edge"/>
                          <xs:enumeration value="Face"/>
                          <xs:enumeration value="Text"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            <xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="rastinfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="rasttype" type="xs:string"/>
        <xs:element name="rowcount" type="xs:integer"/>
        <xs:element name="colcount" type="xs:integer"/>
        <xs:element name="vrtcount" type="xs:integer"/>
      </xs:sequence>
```

```
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:comlexType>
  </xs:element>
</xs:schema>
```

Section 4: Spatial Reference Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schema/csdgm/spref"
  xmlns="http://www.nacse.org/xso/schema/csdgm/spref">

  <xs:element name="spref">
    <xs:complexType>
      <xs:choice>
        <xs:element name="horizsys" type="horizsysType" minOccurs="0"/>
        <xs:element name="vertdef" type="vertdefType" minOccurs="0">
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="detailedType">
    <xs:sequence>
      <xs:choice>
      <xs:element name="geograph">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="latres" type="xs:float"/>
            <xs:element name="longres" type="xs:float"/>
            <xs:element name="geogunit">
              <xs:simpleType>
                <xs:extension base="xs:string">
                  <xs:enumeration value="Decimal degrees"/>
                  <xs:enumeration value="Decimal minutes"/>
                  <xs:enumeration value="Decimal seconds"/>
                  <xs:enumeration value="Degrees and decimal minutes"/>
                  <xs:enumeration value="Degrees, minutes, and decimal
                   seconds"/>
                  <xs:enumeration value="Radians"/>
                  <xs:enumeration value="Grads"/>
                </xs:extension>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="planar" type="planarType"/>
      <xs:element name="local">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="localdes" type="xs:string"/>
            <xs:element name="localgeo" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

```
    </xs:choice>
  <xs:element name="geodetic" minOccurs="0">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="horizdn" minOccurs="0">
          <xs:simpleType>
            <xs:extension base="xs:string">
              <xs:enumeration value="North American Datum of 1927"/>
              <xs:enumeration value="North American Datum of 1983"/>
            </xs:extension>
          </xs:simpleType>
        </xs:element>
        <xs:element name="ellips">
          <xs:simpleType>
            <xs:extension base="xs:string">
              <xs:enumeration value="Clarke 1866"/>
              <xs:enumeration value="Geodetic Reference System 80"/>
            </xs:extension>
          </xs:simpleType>
        </xs:element>
        <xs:element name="semiaxis" type="xs:float"/>
        <xs:element name="denflat" type="xs:float"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="planarType">
  <xs:sequence>
  <xs:choice>
    <xs:element name="mapproj" type="mapprojType"/>
    <xs:element name="gridsys" type="gridsysType"/>
    <xs:element name="localp">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="localpd" type="xs:string"/>
          <xs:element name="localpgi" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
  <xs:element name="planci" type="planciType">
  </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="mapprojType">
  <xs:sequence>
  <xs:element name="mapprojn">
    <xs:simpleType>
      <xs:extension base="xs:string">
        <xs:enumeration value="Albers Conical Equal Area"/>
        <xs:enumeration value="Azimuthal Equidistant"/>
        <xs:enumeration value="Equidistant Conic"/>
        <xs:enumeration value="Equirectangular"/>
        <xs:enumeration value="General Vertical Near-sided
         Projection"/>
        <xs:enumeration value="Gnomonic"/>
        <xs:enumeration value="Lambert Azimuthal Equal Area"/>
        <xs:enumeration value="Lambert Conformal Conic"/>
```

```
        <xs:enumeration value="Mercator"/>
        <xs:enumeration value="Modified Stereographic for Alaska"/>
        <xs:enumeration value="Miller Cylindrical"/>
        <xs:enumeration value="Oblique Mercator"/>
        <xs:enumeration value="Orthographic"/>
        <xs:enumeration value="Polar Stereographic"/>
        <xs:enumeration value="Polyconic"/>
        <xs:enumeration value="Robinson"/>
        <xs:enumeration value="Sinusoidal"/>
        <xs:enumeration value="Space Oblique Mercator"/>
        <xs:enumeration value="Stereographic"/>
        <xs:enumeration value="Transverse Mercator"/>
        <xs:enumeration value="van der Grinten"/>
      </xs:extension>
    </xs:simpleType>
  </xs:element>
  <xs:choice>
    <xs:element name="albers">
      <xs:sequence>
        <xs:element ref="stdparll" minOccurs="1" maxOccurs="2"/>
        <xs:element ref="longcm"/>
        <xs:element ref="latprjo"/>
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="azimequi">
      <xs:sequence>
        <xs:element ref="longcm"/>
        <xs:element ref="latprjo"/>
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="equicon">
      <xs:sequence>
        <xs:element ref="stdparll" minOccurs="1" maxOccurs="2"/>
        <xs:element ref="longcm"/>
        <xs:element ref="latprjo"/>
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="equirect">
      <xs:sequence>
        <xs:element ref="stdparll"/>
        <xs:element ref="longcm"/>
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="gvnsp">
      <xs:sequence>
        <xs:element ref="heightpt"/>
        <xs:element ref="longpc"/>
        <xs:element ref="latprjc"/>
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="gnomonic">
```

```
  <xs:sequence>
    <xs:element ref="longpc"/>
    <xs:element ref="latprjc"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="lamberta">
  <xs:sequence>
    <xs:element ref="longpc"/>
    <xs:element ref="latprjc"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="lambertc">
  <xs:sequence>
    <xs:element ref="stdparll" minOccurs="1" maxOccurs="2"/>
    <xs:element ref="longcm"/>
    <xs:element ref="latprjo"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="mercator">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="stdparll"/>
      <xs:element ref="sfctrlin"/>
    </xs:choice>
    <xs:element ref="longcm"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="modsak">
  <xs:sequence>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="miller">
  <xs:sequence>
    <xs:element ref="longcm"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="obqmerc">
  <xs:sequence>
    <xs:element ref="stdparll"/>
    <xs:choice>
      <xs:element ref="obqlazim"/>
      <xs:element ref="obqlpt"/>
    </xs:choice>
    <xs:element ref="latprjo"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="orthogr">
```

```
  <xs:sequence>
    <xs:element ref="longpc"/>
    <xs:element ref="latpc"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="polarst">
  <xs:sequence>
    <xs:element ref="svlong"/>
    <xs:choice>
      <xs:element ref="stdparll"/>
      <xs:element ref="sfprjorg"/>
    </xs:choice>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="polycon">
  <xs:sequence>
    <xs:element ref="longcm"/>
    <xs:element ref="latprjo"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="robinson">
  <xs:sequence>
    <xs:element ref="latprjc"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="sinusoid">
  <xs:sequence>
    <xs:element ref="latprjc"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="spaceobq">
  <xs:sequence>
    <xs:element ref="landsat"/>
    <xs:element ref="pathnum"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="stereo">
  <xs:sequence>
    <xs:element ref="longpc"/>
    <xs:element ref="latpc"/>
    <xs:element ref="feast"/>
    <xs:element ref="fnorth"/>
  </xs:sequence>
</xs:element>
<xs:element name="transmer">
  <xs:sequence>
    <xs:element ref="sfctrmer"/>
    <xs:element ref="longcm"/>
    <xs:element ref="latprjo"/>
```

```
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="vdgrin">
      <xs:sequence>
        <xs:element ref="longcm"/>
        <xs:element ref="feast"/>
        <xs:element ref="fnorth"/>
      </xs:sequence>
    </xs:element>
  </xs:choice>
  </xs:sequence>
</xs:complexType>


<xs:element name="stdparll">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minInclusive value="-90.0"/>
      <xs:maxInclusive value="90.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="longcm">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minInclusive value="-180.0"/>
      <xs:maxInclusive value="180.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="latprj0">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minInclusive value="-90.0"/>
      <xs:maxInclusive value="90.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="feast" type="xs:float"/>

<xs:element name="fnorth" type="xs:float"/>

<xs:element name="sfequat">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minExclusive value="0.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="heightpt" type="xs:float"/>

<xs:element name="longpc">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minInclusive value="-180.0"/>
```

```
      <xs:maxInclusive value="180.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="latprjc">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minInclusive value="-90.0"/>
      <xs:maxInclusive value="90.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="sfctrlin">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minExclusive value="0.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="obqlazim">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="azimangl">
        <xs:simpleType>
          <xs:extension base="xs:float">
            <xs:minInclusive value="0.0"/>
            <xs:maxExclusive value="360.0"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
      <xs:element name="azimptl">
        <xs:simpleType>
          <xs:extension base="xs:float">
            <xs:minInclusive value="-180.0"/>
            <xs:maxExclusive value="180.0"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="obqlpt">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="obqllat">
        <xs:simpleType>
          <xs:extension base="xs:float">
            <xs:minInclusive value="-90.0"/>
            <xs:maxInclusive value="9-.0"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
      <xs:element name="obqllong">
        <xs:simpleType>
          <xs:extension base="xs:float">
            <xs:minInclusive value="-180.0"/>
            <xs:maxExclusive value="180.0"/>
```

```
          </xs:extension>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="svlong">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minInclusive value="-180.0"/>
      <xs:maxExclusive value="180.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="sfprjorg">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minExclusive value="0.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>

<xs:element name="landsat" type="xs:integer"/>

<xs:element name="sfctrmer">
  <xs:simpleType>
    <xs:extension base="xs:float">
      <xs:minExclusive value="0.0"/>
    </xs:extension>
  </xs:simpleType>
</xs:element>


<xs:complexType name="gridsysType">
  <xs:sequence>
  <xs:element name="gridsysn">
    <xs:simpleType>
      <xs:extension base="xs:string">
        <xs:evaluation value="Universal Transverse Mercator"/>
        <xs:evaluation value="Universal Polar Stereographic"/>
        <xs:evaluation value="State Plane Coordinate System 1927"/>
        <xs:evaluation value="State Plane Coordinate System 1983"/>
        <xs:evaluation value="ARC Coordinate System"/>
        <xs:evaluation value="other grid system"/>
      </xs:extension>
    </xs:simpleType>
  </xs:element>
  <xs:choice>
    <xs:element name="utm">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="utmzone">
            <xs:simpleType>
              <xs:union>
                <xs:simpleType>
                  <xs:extension base="xs:integer">
                    <xs:minInclusive value="1"/>
                    <xs:maxInclusive value="60"/>
                  </xs:extension>
```

```
                </xs:simpleType>
                <xs:simpleType>
                  <xs:extension base="xs:integer">
                    <xs:minInclusive value="-60"/>
                    <xs:maxInclusive value="-1"/>
                  </xs:extension>
                </xs:simpleType>
              </xs:union>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ups">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="upszone">
            <xs:simpleType>
              <xs:extension base="xs:string">
                <xs:enumeration value="A"/>
                <xs:enumeration value="B"/>
                <xs:enumeration value="Y"/>
                <xs:enumeration value="Z"/>
              </xs:extension>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="spcs">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="spcszone" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="arcsys">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="arczone">
            <xs:simpleType>
              <xs:extension base="xs:integer">
                <xs:minInclusive value="1"/>
                <xs:maxInclusive value="18"/>
              </xs:extension>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="othergrd" type="xs:string"/>
  </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="planciType">
  <xs:sequence>
  <xs:element name="horizdn" minOccurs="0">
    <xs:simpleType>
      <xs:extension base="xs:string">
        <xs:enumeration value="North American Datum of 1927"/>
```

```
        <xs:enumeration value="North American Datum of 1983"/>
      </xs:extension>
    </xs:simpleType>
  </xs:element>
  <xs:element name="ellips">
    <xs:simpleType>
      <xs:extension base="xs:string">
        <xs:enumeration value="Clarke 1866"/>
        <xs:enumeration value="Geodetic Reference System 80"/>
      </xs:extension>
    </xs:simpleType>
  </xs:element>
  <xs:element name="semiaxis">
    <xs:simpleType>
      <xs:extension base="xs:float">
        <xs:minExclusive value="0.0"/>
      </xs:extension>
    </xs:simpleType>
  </xs:element>
  <xs:element name="denflat">
    <xs:simpleType>
      <xs:extension base="xs:float">
        <xs:minExclusive value="0.0"/>
      </xs:extension>
    </xs:simpleType>
  </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="vertdefType">
  <xs:sequence>
    <xs:element name="altsys" type="altsysType" minOccurs="0"/>
    <xs:element name="depthsys" type="depthsysType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="altsysType">
  <xs:sequence>
    <xs:element name="altdatum">
      <xs:simpleType>
        <xs:extension base="xs:string">
          <xs:enumeration value="National Geodetic Vertical Datum of
           1929"/>
          <xs:enumeration value="North American Vertical Datum of
           1988"/>
        </xs:extension>
      </xs:simpleType>
    </xs:element>
    <xs:element name="altres" type="xs:float"/>
    <xs:element name="altunits">
      <xs:simpleType>
        <xs:extension base="xs:string">
          <xs:enumeration value="meters"/>
          <xs:enumeration value="feet"/>
        </xs:extension>
      </xs:simpleType>
    </xs:element>
    <xs:element name="altenc">
      <xs:simpleType>
        <xs:extension base="xs:string">
          <xs:enumeration value="Explicit elevation coordinate
```

```
           included with horizontal coordinates"/>
          <xs:enumeration value="Implicit coordinate"/>
          <xs:enumeration value="Attribute values"/>
        </xs:extension>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="depthsysType">
  <xs:sequence>
    <xs:element name="depthdn">
      <xs:simpleType>
        <xs:extension base="xs:string">
          <xs:enumeration value="Local surface"/>
          <xs:enumeration value="Chart datum; datum for sounding
           reduction"/>
          <xs:enumeration value="Lowest astronomical tide"/>
          <xs:enumeration value="Highest astronomical tide"/>
          <xs:enumeration value="Mean low water"/>
          <xs:enumeration value="Mean high water"/>
          <xs:enumeration value="Mean sea level"/>
          <xs:enumeration value="Land survey datum"/>
          <xs:enumeration value="Mean low water springs"/>
          <xs:enumeration value="Mean high water springs"/>
          <xs:enumeration value="Mean low water neap"/>
          <xs:enumeration value="Mean high water neap"/>
          <xs:enumeration value="Mean lower low water"/>
          <xs:enumeration value="Mean lower low water springs"/>
          <xs:enumeration value="Mean higher low water"/>
          <xs:enumeration value="Mean lower high water"/>
          <xs:enumeration value="Spring tide"/>
          <xs:enumeration value="Tropical lower low water"/>
          <xs:enumeration value="Neap tide"/>
          <xs:enumeration value="High water"/>
          <xs:enumeration value="Higher high water"/>
          <xs:enumeration value="Low water"/>
          <xs:enumeration value="Low-water datum"/>
          <xs:enumeration value="Lowest low water"/>
          <xs:enumeration value="Lower low water"/>
          <xs:enumeration value="Lowest normal low water"/>
          <xs:enumeration value="Mean tide level"/>
          <xs:enumeration value="Indian spring low water"/>
          <xs:enumeration value="High-water full and charge"/>
          <xs:enumeration value="Low-water full and charge"/>
          <xs:enumeration value="Columbia River datum"/>
          <xs:enumeration value="Gulf Coast low water datum"/>
          <xs:enumeration value="Equatorial springs low water"/>
          <xs:enumeration value="Approximate lowest astronomical
           tide"/>
          <xs:enumeration value="No correction"/>
        </xs:extension>
      </xs:simpleType>
    </xs:element>
    <xs:element name="depthres" type="xs:float"/>
    <xs:element name="depthdu">
      <xs:simpleType>
        <xs:extension base="xs:string">
          <xs:enumeration value="meters"/>
          <xs:enumeration value="feet"/>
        </xs:extension>
```

```
        </xs:simpleType>
      </xs:element>
      <xs:element name="depthem">
        <xs:simpleType>
          <xs:extension base="xs:string">
            <xs:enumeration value="Explicit depth coordinate included
             with horizontal coordinates"/>
            <xs:enumeration value="Implicit coordinate"/>
            <xs:enumeration value="Attribute values"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Section 5: Entity and Attribute Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schema/csdgm/eainfo"
  xmlns="http://www.nacse.org/xso/schema/csdgm/eainfo">

  <xs:element name="eainfo">
    <xs:complexType>
      <xs:choice>
        <xs:element name="detailed" type="detailedType"
         maxOccurs="unbounded"/>
        <xs:element name="overview" type="overviewType"
         maxOccurs="unbounded"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="detailedType">
    <xs:sequence>
      <xs:element name="enttype">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="enttypl" type="xs:string"/>
            <xs:element name="enttypd" type="xs:string"/>
            <xs:element name="enttypds" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="attr" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="attrlabl" type="xs:string"/>
            <xs:element name="attrdef" type="xs:string"/>
            <xs:element name="attrdefs" type="xs:string"/>
            <xs:element name="attrdomv" type="xs:string" minOccurs="1"
             maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="edomv" type="xs:string"/>
                  <xs:element name="edomvd" type="xs:string"/>
```

```
              <xs:element name="edomvds" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="rdom" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="rdommin" type="xs:string"/>
              <xs:element name="rdommax" type="xs:string"/>
              <xs:element name="attrunit" type="xs:string"/>
              <xs:element name="attrmres" type="xs:float"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="codesetd" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="codesetn" type="xs:string"/>
              <xs:element name="codesets" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="udom" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="begdatea" type="xs:date"
               minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="enddatea" type="xs:date"
               minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="attrvai">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="attrva" type="xs:float">
                    <xs:element name="attrvae" type="xs:string">
                  </xs:sequence>
                </xs:complexType>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="attrmfrq" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Unknown"/>
              <xs:enumeration value="All needed"/>
              <xs:enumeration value="Irregular"/>
              <xs:enumeration value="None planned"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="overviewType">
  <xs:sequence>
    <xs:element name="eaover" type="xs:string"/>
    <xs:element name="eadetcit" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
</xs:schema>
```

Section 6: Distribution Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://www.nacse.org/xso/schemas/csdgm/distinfo"
    xmlns="http://www.nacse.org/xso/schemas/csdgm/distinfo"
    xmlns:cnt="http://www.nacse.org/xso/schemas/csdgm/cntinfo"
    xmlns:time="http://www.nacse.org/xso/schemas/csdgm/timeinfo"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="distinfo">
        <xs:complexType>
      <xs:sequence>
        <xs:element ref="cnt:cntinfo"/>
        <xs:element name="resdesc" type="xs:string" minOccurs="0"/>
        <xs:element name="distliab" type="xs:string"/>
        <xs:element ref="stdorder" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="custom" minOccurs="0"/>
        <xs:element name="techpreq" type="xs:string" minOccurs="0"/>
        <xs:element ref="availabl" minOccurs="0"/>
      </xs:sequence>
        </xs:complexType>
    </xs:element>

  <xs:element name="stdorder">
        <xs:complexType>
      <xs:choice>
        <xs:element name="nondig" type="xs:string"/>
        <xs:element name="digform" maxOccurs="unbounded">
          <xs:complexType>
            <xs:element ref="diginfo"/>
            <xs:element ref="digtopt" maxOccurs="unbounded"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
        </xs:complexType>
    </xs:element>

  <xs:element name="diginfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="formname">
          <xs:simpleType>
            <xs:extension base="xs:string">
              <xs:enumeration value="ARCE"/>
              <xs:enumeration value="ARCG"/>
              <xs:enumeration value="ASCII"/>
              <xs:enumeration value="BIL"/>
              <xs:enumeration value="BIP"/>
              <xs:enumeration value="BSQ"/>
              <xs:enumeration value="CDF"/>
              <xs:enumeration value="CFF"/>
              <xs:enumeration value="COORD"/>
              <xs:enumeration value="DEM"/>
              <xs:enumeration value="DFAD"/>
```

```
            <xs:enumeration value="DGN"/>
            <xs:enumeration value="DIGEST"/>
            <xs:enumeration value="DLG"/>
            <xs:enumeration value="DTED"/>
            <xs:enumeration value="DWG"/>
            <xs:enumeration value="DX90"/>
            <xs:enumeration value="DXF"/>
            <xs:enumeration value="ERDAS"/>
            <xs:enumeration value="GRASS"/>
            <xs:enumeration value="HDF"/>
            <xs:enumeration value="IGDS"/>
            <xs:enumeration value="IGES"/>
            <xs:enumeration value="MOSS"/>
            <xs:enumeration value="netCDF"/>
            <xs:enumeration value="NITF"/>
            <xs:enumeration value="RPF"/>
            <xs:enumeration value="RVC"/>
            <xs:enumeration value="RVF"/>
            <xs:enumeration value="SDTS"/>
            <xs:enumeration value="SIF"/>
            <xs:enumeration value="SLF"/>
            <xs:enumeration value="TIFF"/>
            <xs:enumeration value="TGRLN"/>
            <xs:enumeration value="VPF"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
      <xs:group ref="formatGroup" minOccurs="0"/>
      <xs:element name="formcont" type="xs:string" minOccurs="0"/>
      <xs:element name="filedec" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  </xs:element>

  <xs:element name="digtopt">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="onlineopt"/>
      <xs:element ref="offoptn"/>
    </xs:choice>
  </xs:complexType>
  </xs:element>

  <xs:element name="onlineopt">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="computer" maxOccurs="unbounded">
        <xs:complexType>
          <xs:choice>
            <xs:element name="networka" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="networkr" type="xs:string">
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="dialinst">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="lowbps">
                    <xs:simpleType>
```

```
                    <xs:extension base="xs:integer">
                      <xs:minInclusive value="110"/>
                    </xs:extension>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="highbps" minOccurs="0">
                  <xs:simpleType>
                    <xs:extension base="xs:integer">
                      <xs:minInclusive value="110"/>
                    </xs:extension>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="numdata">
                  <xs:simpleType>
                    <xs:extension base="xs:integer">
                      <xs:minInclusive value="7"/>
                      <xs:maxInclusive value="8"/>
                    </xs:extension>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="numstop">
                  <xs:simpleType>
                    <xs:extension base="xs:integer">
                      <xs:minInclusive value="1"/>
                      <xs:maxInclusive value="2"/>
                    </xs:extension>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="parity">
                  <xs:simpleType>
                    <xs:extension base="xs:string">
                      <xs:enumeration value="None"/>
                      <xs:enumeration value="Odd"/>
                      <xs:enumeration value="Even"/>
                      <xs:enumeration value="Mark"/>
                      <xs:enumeration value="Space"/>
                    </xs:extension>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="compress" minOccurs="0">
                  <xs:simpleType>
                    <xs:extension base="xs:string">
                      <xs:enumeration value="V.32"/>
                      <xs:enumeration value="V.32bis"/>
                      <xs:enumeration value="V.42"/>
                      <xs:enumeration value="V.42bis"/>
                    </xs:extension>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="dialtel" type="xs:string"
                 maxOccurs="unbounded">
                <xs:element name="dialfile" type="xs:string"
                 maxOccurs="unbounded">
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
        <xs:element name="accinstr" type="xs:string" minOccurs="0"/>
        <xs:element name="oncomp" type="xs:string" minOccurs="0"/>
      </xs:complexType>
    </xs:element>
```

```
    <xs:sequence>
  </xs:complexType>
  </xs:element>

<xs:element name="offoptn">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="offmedia">
        <xs:simpleType>
          <xs:extension base="xs:string">
            <xs:enumeration value="CD-ROM"/>
            <xs:enumeration value="3-1/2 inch floppy disk"/>
            <xs:enumeration value="5-1/4 inch floppy disk"/>
            <xs:enumeration value="9-track tape"/>
            <xs:enumeration value="4 mm cartridge tape"/>
            <xs:enumeration value="8 mm cartridge tape"/>
            <xs:enumeration value="1/4-inch cartridge tape"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
      <xs:element name="reccap">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="recden" maxOccurs="unbounded">
              <xs:simpleType>
                <xs:extension base="xs:float">
                  <xs:minExclusive value="0.0"/>
                </xs:extension>
              </xs:simpleType>
            </xs:element>
            <xs:element name="recdenu" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="recfmt" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:extension base="xs:string">
            <xs:enumeration value="cpio"/>
            <xs:enumeration value="tar"/>
            <xs:enumeration value="High Sierra"/>
            <xs:enumeration value="ISO 9660"/>
            <xs:enumeration value="ISO 9660 with Rock Ridge
             extension"/>
            <xs:enumeration value="ISO 9660 with Apple HFS
             extension"/>
          </xs:extension>
        </xs:simpleType>
      </xs:element>
      <xs:element name="compat" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="formatGroup">
  <xs:sequence>
    <xs:choice>
      <xs:element name="formvern" type="xs:string"/>
      <xs:element name="formverd" type="xs:date"/>
    </xs:choice>
    <xs:element name="formspec" type="xs:string"/>
  </xs:sequence>
```

```
  </xs:group>

  <xs:element name="availabl">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="time:timeinfo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Section 7: Metadata Reference Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schema/csdgm/metainfo"
  xmlns="http://www.nacse.org/xso/schema/csdgm/metainfo">

  <xs:element name="metainfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="metd" type="xs:date"/>
        <xs:element name="metrd" type="xs:date"/>
        <xs:element name="metfrd" type="xs:date"/>
        <xs:element name="metc">
          <xs:complexType>
            <xs:sequence>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="metstdn" type="xs:string"/>
        <xs:element name="metstdv" type="xs:string"/>
        <xs:element name="mettc" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="local time"/>
              <xs:enumeration value="local time with time differential
               factor"/>
              <xs:enumeration value="universal time"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="metac" type="xs:string"/>
        <xs:element name="metuc" type="xs:string"/>
        <xs:element name="metsi">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="metscs" type="xs:string"/>
              <xs:element name="metsc">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="Top secret"/>
                    <xs:enumeration value="Secret"/>
                    <xs:enumeration value="Confidential"/>
                    <xs:enumeration value="Restricted"/>
                    <xs:enumeration value="Unclassified"/>
```

```
                    <xs:enumeration value="Sensitive"/>
                    <xs:enumeration value="None"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="metshd" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="metextns" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="onlink" type="xs:string" minOccurs="0"
               maxOccurs="unbounded"/>
              <xs:element name="metprof" type="xs:string"
               minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Section 8: Citation Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schemas/csdgm/citeinfo"
  xmlns="http://www.nacse.org/xso/schemas/csdgm/citeinfo"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="citeinfo" type="citeinfoType" />

  <xs:complexType name="citeinfoType">
    <xs:sequence>
      <xs:element name="origin" type="xs:string"/>
      <xs:element name="pubdate" type="xs:date"/>
      <xs:element name="pubtime" type="xs:time" minOccurs="0"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="edition" type="xs:string" minOccurs="0"/>
      <xs:element name="geoform" type="geoformType"/>
      <xs:element name="serinfo" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="sername" type="xs:string"/>
            <xs:element name="issue" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="pubinfo" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="pubplace" type="xs:string"/>
            <xs:element name="publish" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
```

```
      </xs:element>
      <xs:element name="othercit" type="xs:string" minOccurs="0"/>
      <xs:element name="onlink" type="xs:string" minOccurs="0"/>
      <xs:element name="lworkcit" type="citeinfoType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="geoformType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="atlas"/>
      <xs:enumeration value="audio"/>
      <xs:enumeration value="diagram"/>
      <xs:enumeration value="document"/>
      <xs:enumeration value="globe"/>
      <xs:enumeration value="map"/>
      <xs:enumeration value="model"/>
      <xs:enumeration value="multimedia presentation"/>
      <xs:enumeration value="profile"/>
      <xs:enumeration value="raster digital data"/>
      <xs:enumeration value="remote-sensing image"/>
      <xs:enumeration value="section"/>
      <xs:enumeration value="spreadsheet"/>
      <xs:enumeration value="tabular digital data"/>
      <xs:enumeration value="vector digital data"/>
      <xs:enumeration value="video"/>
      <xs:enumeration value="view"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Section 9: Time Period Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schemas/csdgm/timeinfo"
  xmlns="http://www.nacse.org/xso/schemas/csdgm/timeinfo"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="timeinfo">
    <xs:complexType>
      <xs:choice>
        <xs:element name="sngdate" type="sngdateType"
maxOccurs="unbounded"/>
        <xs:element name="rngdates" type="rngdatesType"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="sngdateType">
    <xs:sequence>
      <xs:element name="caldate" type="xs:date"/>
      <xs:element name="time" type="xs:time" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="rngdatesType">
```

```
    <xs:sequence>
      <xs:element name="begdate" type="xs:date"/>
      <xs:element name="begtime" type="xs:time" minOccurs="0"/>
      <xs:element name="enddate" type="xs:date"/>
      <xs:element name="endtime" type="xs:time" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Section 10: Contact Information

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.nacse.org/xso/schemas/csdgm/cntinfo"
  xmlns="http://www.nacse.org/xso/schemas/csdgm/cntinfo"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="cntinfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="contact">
          <xs:complexType>
            <xs:choice>
              <xs:element name="cntperp">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="cntper" type="xs:string"/>
                    <xs:element name="cntorg" type="xs:string"
                     minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="cntorgp">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="cntorg" type="xs:string"/>
                    <xs:element name="cntper" type="xs:string"
                     minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:element name="cntpos" type="xs:string" minOccurs="0"/>
        <xs:element name="cntaddr">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="addrtype">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="mailing"/>
                    <xs:enumeration value="physical"/>
                    <xs:enumeration value="mailing and physical"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
```

```
            <xs:element name="address" type="xs:string" minOccurs="0"
             maxOccurs="unbounded"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="state" type="xs:string"/>
            <xs:element name="postal" type="xs:string"/>
            <xs:element name="country" type="xs:string"
             minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="cntvoice" type="xs:string"
       maxOccurs="unbounded"/>
      <xs:element name="cnttdd" type="xs:string" minOccurs="0"
       maxOccurs="unbounded"/>
      <xs:element name="cntfax" type="xs:string" minOccurs="0"
       maxOccurs="unbounded"/>
      <xs:element name="cntemail" type="xs:string" minOccurs="0"
       maxOccurs="unbounded"/>
      <xs:element name="hours" type="xs:string" minOccurs="0"/>
      <xs:element name="cntinst" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Appendix B: The Oregon Coast Geospatial Clearinghouse Metadata

1. Data Layer (1.1) – the name given to the data set by the originator
2. Data Title (1.1) – the actual descriptive title of the data set
3. Data Originator (1.1) – who created the data set
4. Publisher (1.1) – who published the data set
5. Publication Place (1.1) – where the data set was published or released
6. Publication Date (1.1) – when the data set was made available
7. Abstract (1.2.1) – a brief description of the data set
8. Purpose (1.2.2) – the reasons behind the development of the data set
9. Beginning Date of Content (1.3) – when the data collection began
10. Ending Date of Content (1.3) – when the data collection stopped
11. Progress (1.4.1) – completeness of the data set
12. Update Frequency (1.4.2) – how often the data set is maintained
13. Access Constraints (1.7) – issues concerned with the access of the data set
14. Use Constraints (1.8) – restrictions or disclaimers for the use of the data set
15. West Bounding Coordinate (1.5.1.1)
16. East Bounding Coordinate (1.5.1.2)
17. North Bounding Coordinate (1.5.1.3)
18. South Bounding Coordinate (1.5.1.4)
19. Location (1.2.3) – the specific location of the data set
20. Scale (2.5.1.2)
21. Map Projection (4.1.2.1) – the map projection of the data set
22. Spatial Reference Method (3.2) – Point, Vector (line, polygon), or Raster (grid)
23. Contact Person or Organization (1.9) – contact for the data set
24. Contact Street Address (1.9)
25. Contact City, State, Zip Code (1.9)
26. Contact Telephone Number (1.9)
27. Contact E-mail Address (1.9)
28. Metadata Date (7.1) – the date the metadata was created

Appendix C: List of XSO Classes

Class Name	Tag Name	Tag Definition
XSAll	xs:all	Compositor describing an unordered group of elements.
XSAnnotation	xs:annotation	Informative data for human or electronic agents.
XSAny	xs:any	Wildcard to replace any element.
XSAnyAttribute	xs:anyAttribute	Wildcard to replace any element.
XSAppInfo	xs:appInfo	Information for application
XSAttribute	xs:attribute	Global attribute definition / Reference to global attribute definition.
XSAttributeGroup	xs:attributeGroup	Global attribute group declaration / Reference to global attribute group.
XSChoice	xs:choice	Compositor to define group of mutually exclusive elements or compositors.
XSComplexContent	xs:complexContent	Definition of a complex content by derivation of a complex type.
XSComplexType	xs:complexType	Global definition of a complex type Reference to global complex type definition.
XSDocumentation	xs:documentation	Human-targeted documentation.
XSElement	xs:element	Global element definition / Reference to global definition.
XSEnumeration	xs:enumeration	Facet to restrict a datatype to finite set of values.
XSExtension	xs:extension	Extension of a simple/complex content model.
XSField	xs:field	Definition of the field to use for a uniqueness constraint.
XSFractionDigits	xs:fractionDigits	Facet to define the number of fractional digits of a numerical datatype.
XSGroup	xs:group	Global elements group declaration / Reference to global elements group.
XSImport	xs:import	Import of a W3C XML Schema for another namespace.
XSInclude	xs:include	Inclusion of a W3C XML Schema for the same target namespace.
XSKey	xs:key	Definition of a key
XSKeyRef	xs:keyref	Definition of a key reference.
XSLength	xs:length	Facet to define the length of a value.
XSList	xs:list	Derivation by list.
XSMaxExclusive	xs:maxExclusive	Facet to define a maximum (exclusive) value.
XSMaxInclusive	xs:maxInclusive	Facet to define a maximum (inclusive) value.
XSMaxLength	xs:maxLength	Facet to define a maximum length.

XSMinExclusive	xs:minExclusive	Facet to define a minimum (exclusive) value.
XSMinInclusive	xs:minInclusive	Facet to define a minimum (inclusive) value.
XSMinLength	xs:minLength	Facet to define a minimum length.
XSNotation	xs:notation	Declaration of a notation.
XSPattern	xs:pattern	Facet to define a regular expression pattern constraint.
XSRedefine	xs:redefine	Inclusion of a W3C XML Schema for the same namespace with possible override.
XSRestriction	xs:restriction	Derivation of a simple/complex datatype by restriction.
XSSchema	xs:schema	Document element of a W3C XML Schema.
XSSelector	xs:selector	Definition of the path selecting an element for a uniqueness constraint.
XSSequence	xs:sequence	Compositor to define an ordered group of elements.
XSSimpleContent	xs:simpleContent	Simple content model declaration.
XSSimpleType	xs:simpleType	Global simple type declaration / Reference to simple type definition.
XSTotalDigits	xs:totalDigits	Facet to define the total number of digits of a numeric datatype.
XSUnion	xs:union	Derivation of simple datatypes by union.
XSUnique	xs:unique	Derivation of a uniqueness constraint.
XSWhiteSpace	xs:whiteSpace	Facet to define whitespace behavior.

Bibliography

1. Aho, A., Sethi, R., and Ullman, J., *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986, pp. 181-182.

2. Biological Data Profile of the Content Standard for Digital Geospatial Metadata, FGDC-STD-001. 1-1999. Online: http://www.fgdc.gov/standards/documents/standards/biodata/biodatap.pdf

3. Carroll, J., *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT Press, Cambridge, MA, 1990.

4. Castor Project. Online: http://www.castor.org/

5. Chen, J. and Zhilin, L., Advances and Applications of GIS in China. *Journal of the American Society for Photogrammetry and Remote Sensing,* Vol. 68, No. 4, 2002.

6. Cocoon Project. Online: http://xml.apache.org/cocoon/

7. Conclin, J., Hypertext: An Introduction and Survey. *IEEE Computer*, Vol. 20, No. 9, 1987, pp. 17-41.

8. Dix, A., Finlay, J., Abowd, G. and Beale, R., *Human-Computer Interaction,* Prentice Hall, Cambridge, England, 1993.

9. Document Object Model. Online: http://www.w3.org/DOM/

10. The Dublin Core Initiative. Online: http://www.dublincore.org/

11. Duvall, G., *Metadata for the Oregon Coast Geospatial Clearinghouse: Concept, Implementation, and Maintenance*. Technical Report, Department of Geosciences, Oregon State University, Corvallis, OR, 2000.

12. Ecological Metadata Language. Online: http://knb.ecoinformatics.org/software/eml/

13. Ehrencrona, A., Interface Guidelines for Web Applications, 1999. Online: http://www.d.kth.se/~d95-aeh/pdf/GuidelinesForWebApps.pdf

14. Executive Order 12906, *Coordinating Geographic Data Acquisition and Access: The National Spatial Data Infrastructure.* Edition of the Federal Register. Vol. 59, No. 71, 1994, pp. 17671-17674.

15. Garner, K., 20 Rules for Arranging Text on a Screen. In *Training and Development Yearbook,* eds. Frantzreb. R., Prentice Hall, Englewood Cliffs, NJ, 1991, pp. 13-17.

16. The Geography of Oregon. Online: www.netstate.com/states/geography/or_geography.htm

17. Girgensohn, A., Lee. A. and Schlueter. K., Developing Collaborative Applications Using the World Wide Web "Shell". *Hypertext '96 Proceedings*. ACM Press, New York, NY, 1996, pp. 246-255.

18. Gu, J. and Pancake, C., *A Unified Architecture for Web-based Interfaces to Scientific Databases*. Technical Report 00-60-05, Department of Computer Science, Oregon State University, Corvallis, OR, 2000.

19. Ispell. Online: http://www.gnu.org/software/ispell/ispell.html

20. Jelliffe, R., *Using XSL as a Validation Language*. Online: http://www.ascc.net/xml/en/utf-8/XSLvalidation.html

21. Kleinberg, D., interviewed in Builder.com. Online: http://www.builder.com/Graphics/UserInterface/ss02.html

22. Lenman, S. and Robert, J., A Framework for Error Recovery. *Proceedings of the International Ergonomics Association (IEA'94),* International Ergonomics Association, 1994, pp. 374-376.

23. Lewis, C. and Norman, D., Designing for Error. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, eds. Norman, D. and Draper, S. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

24. Lienkaemper. G., USGS Forest and Rangeland Ecosystem Science Center, personal interview, 2001.

25. Lynch, P. and Horton, S., *Yale Style Manual*. Online: http://info.med.yale.edu/caim/manual/

26. Michener, W., Brunt, W., Helly, J., Kirchner, T., and Stafford, S., Ecological Applications. *Nongeospatial Metadata for the Ecological Sciences,* Vol. 7, No. 1, 1997, pp. 330-342.

27. Milheim, W., Screen Design for Computer-Based Training and Interactive Video: Practical Suggestions and Overall Guidelines. *Performance and Instruction,* Vol. 31, No. 5, 1992, pp. 13-21.

28. Morpho. Online: http://knb.ecoinformatics.org/software/morpho/

29. NBII MetaMaker. Online: http://www.umesc.usgs.gov/metamaker/nbiimker.html

30. Nielsen. J., The Art of Navigating through Hypertext. *Communications of the ACM*, Vol. 33, No. 3, (March 1990), pp. 296-310.

31. Nielsen, J., *Site Map Usability*. Online: http://www.nngroup.com/reports/sitemaps/

32. Nielsen, J., *Why Yahoo is Good.* Online: http://www.useit.com/alertbox/981101.html

33. Norman D., Categorization of Action Slips. *Psychological Review*, Vol. 88, 1981, pp. 1-15.

34. Norman, D., Cognitive Engineering. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, eds. Norman, D. and Draper, S. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

35. Norman, D. *The Design of Everyday Things*. Currency Book, New York, NY, 1988.

36. Onsrud, H. and Rushton, G., *Sharing Geographic Information*. CUPR Press, New Brunswick, New Jersey, 1995.

37. Pancake, C., personal communication, based on field studies with users of metadata and clearinghouses, 2002.

38. Pastoor, S., Legibility and Subjective Preference for Color Combinations in Text. *Human Factors*, Vol. 32, 1990, pp. 157-171.

39. Preece, J., *Human-Computer Interaction*. Addison-Wesley, Reading, MA, 1994, pp.159-161.

40. Rice, J., Farquhar, A., Piernot, P. and Gruber, T., Using the Web instead of a Window System. *Proceedings of CHI'96*. ACM Press, New York, NY, 1996, pp.103-117.

41. Rosson, M. and Carroll, J., *Usability Engineering*. Morgan Kaufmann, San Francisco, CA, 2002.

42. Schema Adjunct Framework Specification. Online: http://www.extensibility.com/saf/spec/

43. Sheth, A., Semantic Issues in Multidatabase Systems. *Special Issue of SIGMOD Record,* Vol. 20, No. 4, 1991, pp. 5-80.

44. Sheth, A. and Klas, W., *Multimedia Data Management: Using Metadata to Integrate and Apply Digital Media*. McGraw-Hill, New York, NY, 1998.

45. The Spatial Metadata Management System. Online: http://www.intergraph.com/gis/smms/

46. Tannenbaum, A., *Metadata Solutions.* Addison-Wesley, Reading, MA, 2001, pp. 21-24.

47. Vlist, E., *XML Metadata*. O'Reilly & Associates, Inc. Sebastopol, CA, 2002.

48. Walsh, K., Pancake, C., Wright., D., Haerer, S. and Hanus, F., Humane Interfaces to Improve the Usability of Data Clearinghouses. *Proceedings of the GIScience Second International Conference,* Boulder, CO, 2002, pp. 333-345.

49. Weinschenk S. and Yeo. S., *Guidelines for Enterprise-Wide GUI design*, John Wiley & Sons, New York, NY, 1995.

50. Wright, D., Duvall, G. and Ward, B., Metadata Matters: A National Geospatial Clearinghouse for the Oregon Coast. *Proceedings of The Coastal Society 17th International Conference*, Portland, Oregon, http://dusk.geo.orst.edu/tcs/, 2000.

51. Xerces2 Java Parser. Online: http://xml.apache.org/xerces2-j/index.html

52. XML database product. Online: http://www.rpbourret.com/xml/XMLDatabaseProds.htm

53. XML Schema Specification. Online: http://www.w3.org/XMLSchema

54. XML Schema Validator. Online: http://www.w3.org/2001/03/webdata/xsv/

55. XQuery Specification. Online: http://www.w3.org/TR/xquery/

56. Yang, Y., Anatomy of the Design of an Undo Support Facility. *International Journal of Man-Machine Studies*, Vol. 1, 1992, pp. 81-95.

57. Zisman, A. and Kramer, J., An Information Discovery Process for Interoperable Heterogeneous Databases. *Proceedings of the Americas Conference on Information Systems,* 1999, pp. 617-619.