A General Purpose Registry/Repository Information Model

by

Len Gallagher
Lgallagher@nist.gov
Lisa Carnahan
lisa.carnahan@nist.gov

Information Technology Laboratory
National Institute of Standards and Technology

Draft October 16, 2000

- Abstract -

We propose a UML model and an XML services interface for a general-purpose registry/repository. In general, a *registry* is a facility that stores relevant descriptive information about registered objects, and a *registered object* is something important that an author or producer wants to have visible to the world so that it can be discovered and used by a client or customer. A registry can operate independently, or it can be paired with a repository that stores the registered objects. A *repository* is simply a storage facility for registered objects, with an access method allowing one to retrieve individual objects by object reference. In electronic commerce, a *registry/repository* is an integrated software system that supports access to registry metadata in order to locate and retrieve registered objects useful toward solving some problem. A registry/repository implementation supports a *registry services* interface that can be used by abstract agents to assist a human or some other software process to register new objects, provide appropriate metadata for those objects, browse or query registry content, filter out irrelevant references, and retrieve the content of selected items.

We anticipate a large number of registry/repository implementations in electronic commerce, each focusing on registering objects of interest to some sponsoring group. Each implementation will provide one or more classifications of the registered objects according to classification schemes important to its sponsoring group and will identify specific dependencies or associations among the registered objects to help an agent determine those objects of specific interest. If each such implementation had a different agent interface, then an agent might be overwhelmed with the hundreds, or even thousands, of specialized interfaces available. NIST is developing a general-purpose registry/repository information model capable of supporting a wide range of implementations, each catering to the specific requirements of some sponsoring group, yet having a well-defined, standard registry services interface accessible to virtually any electronic agent. We are also developing a prototype implementation for the OASIS specialization.

Table of Contents

Table of Figures

1. Introduction

We propose a UML class model and an XML services interface for a general-purpose registry/repository that can be used to satisfy registry and repository requirements of many different organizations. Currently, we see different consortia and standardization groups all attempting to define and/or register their specific metadata for the purpose of improving electronic commerce among their members. The DublinCore group [4] is trying to standardize on metadata for library resources, IMS [8,9,10] and LTSC [7] are defining metadata for learning objects, cXML [3] is defining DTD's based on ontologies, OASIS [16] is defining a registry/repository for XML document types, ebXML [5] is defining business trading partner agreements, XBRL [21] is developing XML support for financial reports, and eCO [2], OBI [17], BizTalk [1], UDDI [20], and RosettaNet [19] are all defining frameworks and schemas for representing and registering various business processes.

What is needed and not yet available is a simple information model that allows each such organization to register the objects of their choosing, classify those objects according to various classification schemes, capture different associations and dependencies amongst the registered objects, and record additional metadata for registered objects as appropriate. In addition, the information model should provide an interface for easy access to this collection of information about the registered objects. Then the members of each sponsoring group can exchange specialized information with one another, and the sponsoring groups can exchange general information with one another. The major advantage of relying on a general-purpose information model to represent multiple specialized metamodels for registry content is that the services interface to that model can remain stable for each specialization.

The registry/repository information model we define herein has the following desirable properties:

- The model is simple enough to be easily implemented with existing off-the-shelf data management technologies;
- The model is flexible enough to support the required specializations of many different kinds of sponsoring groups;
- The static logical structures of the model are able to support an identical core registry services interface across all specializations;
- The model defines separable registry and repository structures, with registry entries able to point to external registered objects by URL and to registered objects or registry entries in other conforming registry/repositories by URN;
- The model assumes local autonomy for each registry/repository implementation, but allows one implementation to send messages to other implementations as part of a distributed federation;

- The registry services interface allows registry-to-registry interoperability among conforming implementations, even if the registries have implemented different specializations of the information model.

Figure 1 presents a high level representation of a registry/repository implementation as a software component consisting of two separable sub-components, with the main component having a single Registry Services interface. In general, a *registered object* will be an object instance in the Repository Objects component and the metadata describing that object, i.e. a *registry entry*, will be an object instance in the Registry Objects component. However, the registry and repository objects need not be in a one-to-one correspondence. A registry entry may point to a registered object in some other conforming registry/repository or it may point to an external object not in any repository. The latter situation is supported by the LTSC LOM model [7], where the primary registered object, i.e. a learning object, is retained by the submitting organization since a fee or license may be required to obtain it. It's also possible for a registered object to have more than one registry entry pointing to it in the same implementation; this could happen if an object is registered with its metadata as one registry entry and then later that object is replaced with a second object. The second object will have its metadata as a second registry entry, and both registry entries will now point to the second registered object. The first registry entry will remain in existence even though it will now label the object it initially pointed to as having been *replaced* by the second registered object. Our approach allows withdrawal and replacement of registered objects in the repository while maintaining referential integrity among registry entries in the registry.

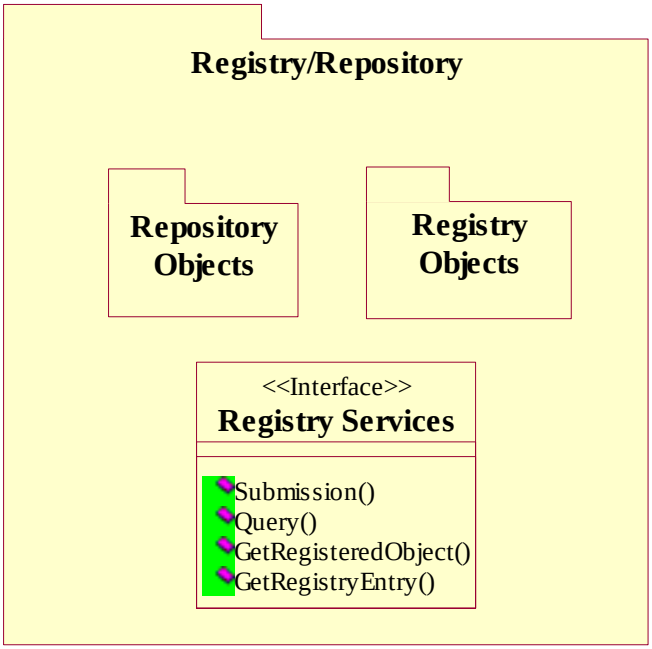


Figure 1 - Registry/Repository Component

The Registry Services interface presented in Figure 1 provides a single interface to the integrated registry/repository system. If the structure of a registered object in the repository is known to the registry, then the services interface may include operations on the registered object itself. Later

we define two special registered objects, i.e. classification scheme and registry package, that fall into this category. However, in many cases the specific structure of a registered object will be unknown to the registry so the only access to that object will be by direct object reference. For example, if a GIF graphic or some application program is a registered object, then we cannot normally expect registry services to be able to query the graphic or program directly. So the main purpose of the registry services interface is to provide a mechanism for creating and maintaining registry objects, and for querying the registry objects to obtain a set of references to registered objects whose corresponding registry entries satisfy the query. We define a specific collection of Registry Services in section 5, after the Registry Objects have been defined in sections 3 and 4.

The remainder of this paper is structured as follows. Section 2 identifies the registry and repository objects of interest. Section 3 provides a UML class diagram for representing registry and repository objects, with special emphasis on the subordinate classes for associations and classifications. Section 4 provides the details of registry administration facilities together with a UML class diagram and workflow discussion for submissions, requests, contacts, organizations, and impacts on registry entries. Section 5 gives an overview of the registry services needed for registry implementation and defines XML elements for creating and manipulating registry entries. Section 6 defines conformance alternatives for registry/repository implementations. Section 7 provides some conclusions and looks forward to the next steps in providing federations of cooperating registry/repository implementations.

2. Registry and Repository Objects

If we open the software packages identified in Figure 1, and focus just on the different independent object types that need to be considered in a registry/repository implementation, then the result is the UML Class diagram shown in Figure 2. The registered objects package consists of just a single super-class, RegisteredObject, while the registry objects package produces all of the other classes represented in the diagram, namely RegistryEntry, Organization, Contact, and Submission. Since a submission is a collection of registry service requests, we provide a Request class as dependent on the Submission class.

The RegisteredObject class identifies all registered objects that are managed by the local registry/repository implementation. We know that for some sponsoring groups all submittal requests to register an object will be accompanied by the object itself and that for some other sponsoring groups the actual object will be kept by the submitting organization. Our model accommodates either approach. If the object to be registered is submitted to the registry/repository for storage and safe-keeping, then it will become a persistent instance of this class and will be assigned a permanent object identifier.

The RegistryEntry class identifies the metadata for a registered object. Each instance includes a URL that can be used to locate the corresponding registered object. It also includes additional attributes to define its persistence and mutability, its administrative and payment status, and its submitting and responsible organizations. We provide the details of the RegistryEntry class, together with all of its dependent classes, in section 3. These dependent classes will maintain the classifications, associations, and other metadata associated with the registered object identified by a registry entry. Each registry entry instance has a non-public, local identifier created by the registration authority. This identifier is used to maintain relationships of this item with other

items in the same registry. In addition, each registry entry has both a common name and a global name. The common name is intended for use by humans and is not necessarily unique except in some local human context. The global name, called the assignedURN, is used as an alias for the local object identifier since that identifier may not be visible to users. It is intended for use both by humans and by software systems and is unique within all registries that claim conformance to the registry/repository specification. The assignedURN can be used by registery services to Get the metadata for a registered object, or to Get the object itself.

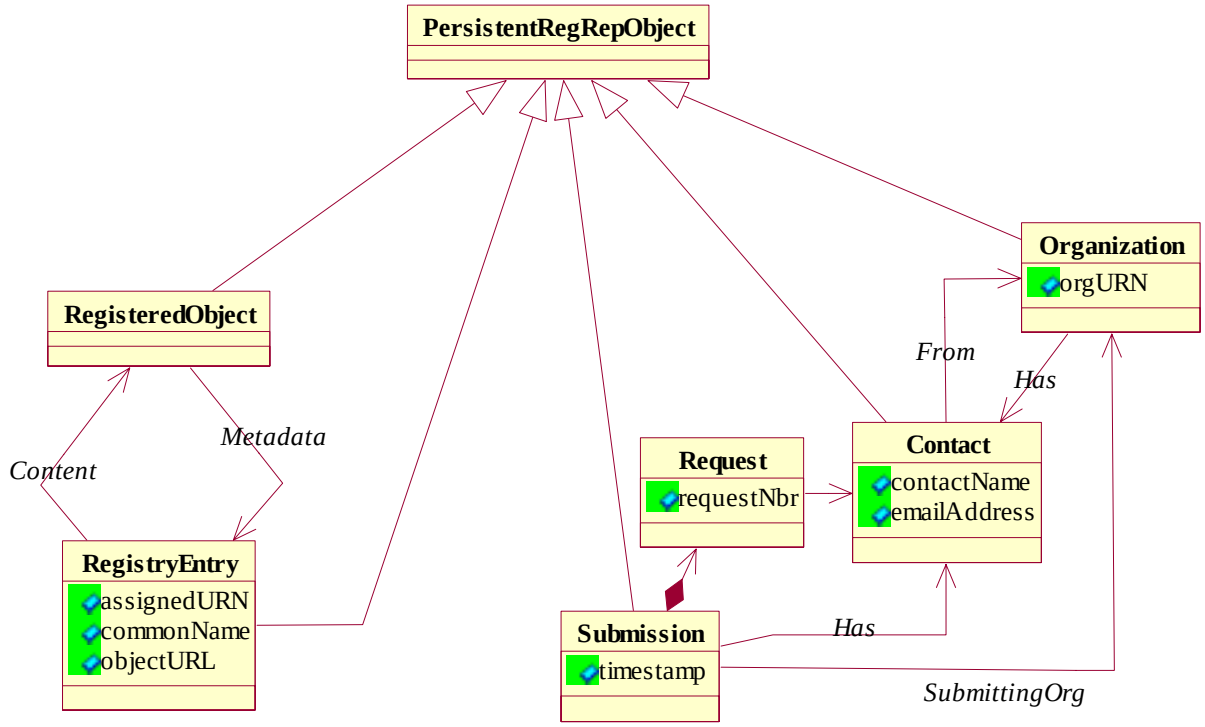


Figure 2 - Persistent Registry/Repository Objects

In the figure, we represent the persistent objects of a registry/repository implementation by the PersistentRegRepObject class. This class has no attributes. Its sole purpose is to provide a collection point of object identifiers for all persistent objects managed by the RegRep implementation. The five classes identified as subtypes of this class are the only classes that require unique and persistent object identifiers. Other classes will be defined in Sections 3 and 4 that provide additional detail and maintain the required associations among instances of these primary classes.

We assume that every instance of PersistentRegRepObject has a unique identifier locally assigned by that implementation. For now, we assume that this identifier is never visible to the public or as part of the registry services interface. Later, if the specification is extended to formalize distributed cooperation, then this identifier could be the basis of a globally unique identifier for registry/repository objects understood by all conforming implementations.

A registry/repository has administrative obligations, so it must be concerned with more than just the metadata for a specific registered item. It is also concerned with organizations and contacts within those organizations that submit or maintain submissions made to a registry. It is important that a registry be able to identify who has the privileges necessary to add or modify registry content and to maintain a log of all changes made. We believe that the registry will be able to accomplish all of these expectations with just the inclusion of the new persistent registry object types identified in the figure.

The Organization class identifies all organizations that have some relationship to a registered object. This includes a submitting organization (SO), a responsible organization (RO), and a registration authority (RA). An organization wishing to become a submitting organization must first make an application to a recognized registration authority via a registry services request for that purpose. The end result is that a submitting organization and responsible contacts within that organization will be known to the registration authority before follow-on submissions from that organization will be accepted. Each organization will be assigned a globally unique orgURN that can be used as an alias for its local object identifier.

The Contact class identifies each person, role, or other entity within an organization that has some relationship to a registered object. A contact will consist of a contactName, i.e. an arbitrary string, used to identify the contact within an organization, and some specific contact information such as telephone number and email. Each contact instance has a non-public, local identifier created by the registration authority that is used to maintain relationships among organizations, contacts, and registry entries. Each contact includes a mandatory reference to some organization. It is not necessary to maintain a global, unique name for contacts because the global name for the organization together with the common name for the contact will be sufficient to identify the contact.

A submission is a collection of requests, in the form of a message, sent from a submitting organization to a registry. For administrative accountability the registry logs each submission, timestamps it with the date and time received, and stores this information as a persistent instance of the Submission class. Each request received as part of a submission is logged as an instance of the Request class for potential subsequent scrutiny. Any registry entities that are created, deleted, or modified by a request should be traceable back to the submission instance and to its submitting organization. For accountability purposes, a registry should be able to track all modifications to the metadata for any registered object. Thus there will be a many-to-many relationship among requests and registry entries to identify the type of impact each request has on one or more entries. The specific associations that must be maintained by a registry implementation are not all visible in Figure 2; instead, they are presented in more detail in Section 4 below. The Impact class shown in Figure 5 captures the many-to-many relationship between requests and registry entries.

A general user of a registry/repository may not be interested in the administrative requirements of the implementation, so may have no interest in the Submission, Request, and Contact classes. The main interest of most users will be in the details of the RegisteredObject and RegistryEntry classes, so their detailed representations are presented separately in Section 3 below.

3. Registered Object and Registry Entry

A general-purpose information model needs to be comprehensive enough to capture the major requirements of multiple registering organizations, but must also be simple enough so that every such organization feels comfortable using it. The types of objects that can be registered are determined by standardization organizations that sponsor registry/repository specifications. For example, the OASIS registry/repository [16] is interested in registration of arbitrary XML and SGML objects, CommerceNet [2], RosettaNet [19], UDDI [20], and ebXML [5] are interested in the registration of electronic business objects such as company profiles, business processes, and trading partner agreements, XBRL [21] needs to register financial reports, Dublin Core [4] is interested in the registration of library resource objects, and LTSC [7] and IMS [8] are interested in the registration of learning objects such as courses, training materials, and educational programs. All of these organizations are interested in classifying their registered objects according to well-established classification schemes, in specifying associations and dependencies among the registered objects, and in being able to identify collections of registered objects at the same time. The registry/repository information model proposed herein hopes to be broad and inclusive enough to accommodate the common base requirements of each of these registration efforts. It singles out classifications, associations, and registry packages as special objects that should be supported by every conformant registry/repository. The more specific requirements of each registry sponsor can be supported by specialization of the various roles defined as generic features of the model, or if necessary, as extensions to the base model.

Our registry/repository information model is strongly influenced by the ISO Metadata Registry specification [12], especially its notions of classification schemes, context names, and some associations. We believe that our information model is a friendly colleague of the ISO Metadata Registry, i.e. ISO 11179 Part 3, but it does not claim conformance to that specification. Instead, it focuses on registered object and registry entry rather than on data element and metadata about data elements, allows associations among any registered objects rather than just between data elements and concepts, and de-emphasizes the programming language aspects of data elements. We were careful to ensure that the structures of the proposed model satisfy the pre-conditions of the OMG MOF [18], so it would be possible to use the Corba IDL interface facilities defined therein for registry/repository access. However, due to the complexity of the MOF interface, we define a simple XML interface that can be used until MOF facilities become more ubiquitous. The XML interface is discussed in Section 5, Registry Services.

3.1 UML Class Diagram

If we focus only on the RegisteredObject and RegistryEntry classes of Figure 2, then the details of those classes can be represented in Figure 3 below. We see that registered objects are characterized by one of two possible subtypes; either the object's structure is known or unknown to the registry implementation. If the structure is unknown, then as far as the registry is concerned its content is simply a binary large object, i.e. a BLOB. It will be up to the submitting organization to use the registry services interface to create a registry entry for that object, with appropriate values for the required attributes of that class, and to submit the instances of the other metadata classes dependent on RegistryEntry. If the structure of the registered object is known to the registry, then the registry services interface will provide the ability to create and modify those objects. For example, to create a new classification scheme a submitting

organization need only submit an XML document that validates to the ClassificationScheme DTD defined in section 3.4 below. The registry will parse the XML document and populate the ClassificationScheme class and its dependent subclasses as appropriate. Similarly, to create a new registry package, a submitting organization need only submit an XML document that validates to the RegistryPackage DTD defined by this model.

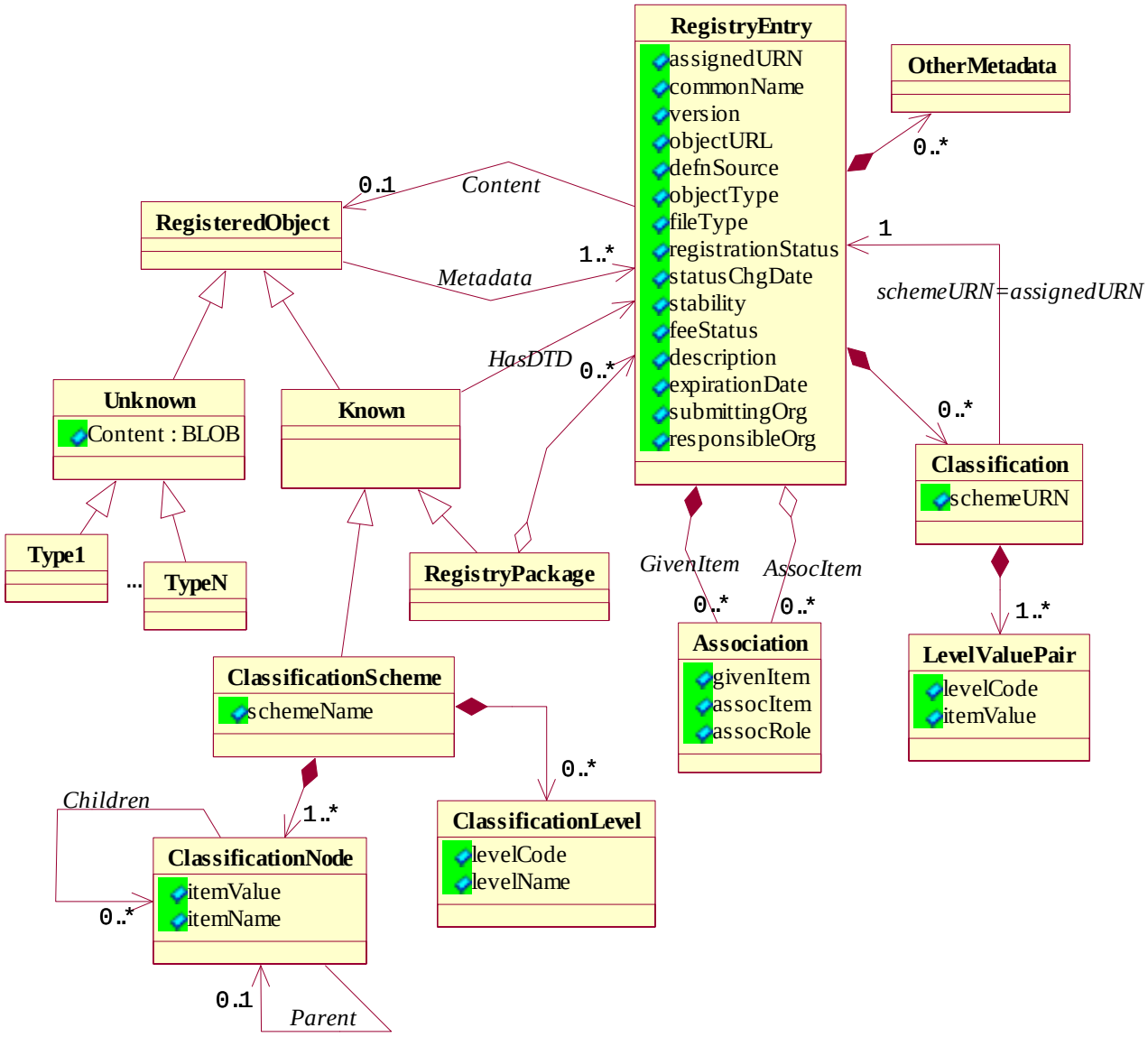


Figure 3 - Registry/Repository Class Diagram

Initially, classification schemes and registry packages are the only registered objects whose content is managed by registry services. This does not prohibit the existence of a rule saying that unknown registered objects must still validate to a registered XML DTD if they are defined as XML documents subject to the constraints of that DTD. In addition, we expect that many sponsoring groups will provide validation routines as part of their model specialization. For example, if a sponsoring group wants to register business processes, it may provide a schema template for defining a new business process, register the template, and then require that newly

submitted business processes validate to the template before they will be accepted by the registry. If the template is not a commonly known schema definition type, then the sponsoring group will provide the validation routines as an extension of the information model.

The purpose of the Association class is to record in the registry any real-world associations that may exist among the unknown registered objects. This allows new associations to be entered and retrieved through a standard interface without the need for accessing the internals of each such object. The purpose of the Classification class is to allow the submitting organization to classify its registered objects according to any previously registered classification scheme. It also allows third parties to classify registered objects according to classification schemes they are more comfortable with. The RegistryEntry, Association, and Classification classes are the most important in this model; they are discussed in sections 3.2 through 3.4 below. The OtherMetadata class is an abstract placeholder for simple convenient metadata features of the model whose details are presented in section 3.6 below.

As an example of registry/repository usage, consider a business trade organization that desires to register objects of three business types: company profiles, business processes, and trading partner agreements. The first thing they may do is register several different XML CompanyProfileDTD's for defining company profiles in various industry segments. They'll also register a collection of business processes. Then, any company that wants to participate in trading partner agreements would submit an XML CompanyProfile document that validates to one of the CompanyProfile DTD's. The company would need to tell the world which DTD its profile validates to by submitting an association instance that links the given profile to the associated DTD with the role *definedBy*. The company could also classify itself according to an arbitrary number of different classification schemes by submitting classification instances that link its profile entry to a registered classification scheme and give a classification value defined by that scheme. Next the company might list the business processes it supports by adding association instances to the registry with its profile entry as the given item and the business objects it supports as the associated items under an association role of *supports*. We're assuming that the association roles of *definedBy* and *supports* are defined by the sponsoring organization as part of its specialization of the registry/repository information model. The advantage is that a browsing agent can retrieve this information through the standardized registry services interface; the agent does not have to learn how to parse a multitude of different schemas.

3.2 Registry entry - the metadata for a registered object

A RegistryEntry instance contains information that identifies, names, and describes each registered object, gives its administrative and payment status, defines its persistence and mutability, classifies it according to pre-defined classification categories, declares its object and file representation types, and identifies the submitting and responsible organizations. A registry entry is a root entity in a registry in the sense that many other metadata items in the registry are directly dependent on it. For example, deletion of a registry entry instance results in deletion of all other metadata items that depend on it.

Often different types of objects are registered in the same registry. It is very important to maintain this type distinction for it may be the basis of required associations among different types of registered objects. For example, the Oasis registry registers both XML data type

definitions (DTDs) and XML documents that validate to some data type definition. It is an Oasis requirement that each registered XML *instance* document be associated with the registered *definition* document that it validates to. To support such requirements the object type and file type of a registered object are maintained by the corresponding registry entry. The sponsoring group provides a list of valid values and the explicit meaning of each value. The sponsor is identified by a special attribute, defnSource, meaning that the specialized semantics of certain abstract attributes are defined by that source. The advantage is that each different sponsoring body can define its unique type requirements, but access for declaring, querying, or modifying the types is identical across all registries.

The following table identifies all of the attributes of the RegistryEntry class. It also indicates who supplies the value for that attribute and who provides the list of valid values with specified semantics. The term SO indicates a submitting organization, RA indicates a registration authority, W3C indicates the World Wide Web consortium, and defnSource identifies the sponsor of some specialization of this model. We define the *owner* of a registry entry to be the submitting organization that submitted it.

RegistryEntry class

Attribute	Datatype	Supplies Value	Defines Meaning	Examples
assignedURN	URN	RA	This Model	us:gov:nist:xml:dtd:profile37
commonName	String	SO	SO	My Personal Profile
version	String	SO	SO	3.01.05, 2nd draft, DIS
objectURL	URL	SO	W3C	ftp://xsun.sdct.itl.nist.gov/regrep/RegRepArticle.pdf
defnSource	CodeText	RA	This Model	OASIS, ebXML, UDDI, eCo, BizTalk, cXML, etc.
objectType	CodeText	SO	defnSource	CompanyProfile, BusinessProcess, registry package, classification scheme, TradeAgreement, xml/defn, xml/instance
fileType	CodeText	SO	defnSource	MimeType, text/xml/schema, text/xml
registrationStatus	CodeText	RA	defnSource	Submitted, Registered, Superseded, Replaced, Deprecated, Withdrawn, etc.
statusChgDate	Date	RA	This Model	2000-10-18
stability	CodeText	SO	defnSource	Static, Dynamic, Compatible, etc.
feeStatus	CodeText	SO	defnSource	Free, Password, Payment, etc.
description	String	SO	SO	This is an IEEE purchase order template for electronic products
expirationDate	Date	RA	RA	2 years from registration date
submittingOrg	Organization	RA	This Model	REF to ebXML
responsibleOrg	Organization	SO	This Model	REF to Oasis

3.3 Associations

The Association class represents binary associations among registered objects. Since we cannot represent these associations directly among unknown registered objects without extending the information model, we represent them implicitly as associations among registry entries. If a TypeX object has an association with a TypeY object, then that association is represented by an association between a registry entry for the TypeX object with a registry entry for the TypeY object. The name of the original association is represented by the *assocRole* attribute of the registry association.

An Association instance maintains a pairwise relationship among registry entries. One item in the pair is called the *given item* and the other item in the pair is called the *associated item*. Both the given item and the associated item are registry entries in the same registry as the association instance.

NOTE: I would like to change the Oasis model to require that both the given item and the associated item in an association instance reference a registry entry in the same registry as the association instance! This puts a small burden on a definer to create a local registry entry for something registered in an external registry, but it makes the model much cleaner!

Each *assocRole* attribute identifies the role played by the given item in relationship to the associated item. For example, the Dublin Core specification [4] identifies the following association roles for relationships among resources:

Is Version Of	Has Version
Is Replaced By	Replaces
Is Required By	Requires
Is Part Of	Has Part
Is Referenced By	References
Is Format Of	Has Format

Alternatively, the Oasis specification currently has the following roles for associations from a given item to an associated item.

Uses	-- stronger than references - requires use of the associated item
Supersedes	-- captures both supersedes and replaces - registryStatus distinguishes
Contains	-- special usage to identify elements of a registry package
RelatedTo	-- the weakest form of references

Unlike the Dublin Core specification, the Oasis specification assumes that the association instance is owned by the given item, so the inverse association is only implicit. For example, in Oasis it is not possible to specify that a given item is UsedBy an associated item; instead, it must be declared the other way around and then queried to determine the implicit relationship. The reason for doing this is that the owner of the associated item has complete control over the metadata directly maintained for that item and may not want to maintain superfluous associations.

Like Oasis, our information model assumes that association instances are owned by the given item. This means that an association cannot be inserted or modified in the registry unless the submitting organization of the given item requests the change. Any group sponsoring a specialization of this information model can declare whatever association roles are most appropriate for its unique situation. However, a sponsor will want to choose these roles very carefully. In general, the roles chosen will depend upon the data types of the registered objects and the real-world relationships expected among those objects.

We recommend consideration of the following association roles before deciding on the official ones for a given specialization.

Potential Association Roles

assocRole	Role Name	Meaning
validatesTo	Validates To	The given item validates to the specification provided by the associated item. Examples: an XML document validates to an XML schema; a business process validates to a process template; a classification scheme validates to the Classification Scheme DTD defined herein; a program validates to its UML class specification. Note: Validates To is a special case of Requires - decide if the distinction is appropriate.
requires	Requires	The given item requires the presence of the associated item. The expectation is that the associated item must be retrieved before the given item can be processed or used. Examples: an XML element requires the presence of some other XML element or entity that it references; a trading partner agreement requires two or more company profiles and one or more business processes; a software program requires the installation of some other program before it will execute properly. NOTE: make sure the distinction between Requires and References is clear; in some cases Requires → References.
references	References	The given item references the associated item. If the given item is retrieved, the default action is that the associated items are NOT retrieved along with it. However, a retrieve request may have variants that allow recursive retrievals. Examples: a DTD references other registered XML elements or entities; a registry package references its package elements; a specification references other specifications.
supersededBy	Is Superseded By	The given item is superseded by the associated item. Only the SO of a given item can say that it is superseded by some other registered object. The registered object of the given item is still registered, its registry entry still points to it, and its registrationStatus becomes *superseded*. Example: one version of a registered object is superseded by a newer version and the old version remains available.
replacedBy	Is Replaced By	The given item is replaced by the associated item. Only the SO of the given item can say that it is replaced by another registered object. The given registered object is no longer registered, but its registry entry remains in the registry, its registrationStatus becomes *replaced*, and the objectURL of that registry entry now points to the other registered object. Example: a new, upward compatible version of a registered object replaces the existing version. All pointers to the old version now point to the new version.
relatedTo	Is Related To	The given item is related to the associated item. This is a very loose association that has no dependency implications. Example: a GIF graphic is related to the classification scheme that it visualizes, and *vice versa*; a

		company catalog is related to a company purchase order DTD, and *vice versa*. But each one-way relationship is created and maintained by the SO of the given item.
hasFormat	Has Format	The given item has its format determined by the associated item. Example: an XML document has its format defined by an XML stylesheet.
formatOf	Is Format Of	The given item determines the format of the associated item. Example: an XML stylesheet is one of many possible formats for an XML DTD; an XML document that validates to the DTD could use the stylesheet for visual presentation..

The advantage of the proposed information model is that *association role* is an abstract notion that can be defined as required by each sponsoring organization for registry content. The roles can be declared, modified, or queried via standardized access methods, and the Association entity can hold an arbitrary number of association instances for each registered object.

3.4 Classifications

We define a *classification scheme* to be a fixed hierarchy of nodes. Others may define a classification scheme either more or less generally, but we believe that a fixed hierarchy of nodes provides a very useful structure for defining classification semantics that can be used to partition a given population into various collections, each with specific properties. The *classification* of a population according to a given classification scheme is accomplished by assigning to each object in the population a reference to a single node of the classification scheme hierarchy.

In the following paragraphs we present several different types of classifications schemes. It is our intent that each of these types can be captured by our general definition of a classification scheme. Following the examples, we give a mathematical definition of a classification scheme and specify an XML DTD for representing it.

Simple 1-level classification scheme

A simple 1-level classification scheme is a list of distinct values that can be used to partition a collection of objects. The identifier of the classification scheme can be viewed as the root of the hierarchy, with each of the distinct values considered as a node at the first level. An example of such a simple classification scheme is StudentStatus for a student population, which can be used to partition the students into Freshmen, Sophomores, Juniors, Seniors, and Special students.

Each node of a classification scheme will allow distinctions to be made between *itemValue* and *itemName*. An itemValue will always be considered as a reference to be used in place of the itemName. With this capability, we could refine the StudentStatus classification scheme to specify the item values FR, SO, JR, SR, SP as references and replacements for the longer and more descriptive item names. A classification scheme defines only one item value for each item name.

The most common usage of a simple 1-level classification scheme is to define an enumeration domain. Enumeration domains are used extensively in the LTSC LOM specification [7].

Reference [6] shows how the LOM enumeration domains can be re-defined as registered classification schemes to support the classification of learning objects in many different ways.

Multi-level naming classification scheme

A multi-level naming classification scheme uses scoped names to identify the nodes in a classification hierarchy. Each name is meaningful only if the name of its parent node is known. Names need be unique only as children of the parent node, so in order to uniquely identify a node it is necessary to know the names of each node in a path from the root to the given node.

An example of a multi-level naming classification scheme is the scheme used by biologists to classify all living things. The scheme consists of seven levels: Kingdom, Phylum, Class, Order, Family, Genus, and Species, each with a list of recognized values. However, it is not required that names within each level be unique; there could be a species for tree that is the same as the species of some animal, because trees and animals are in different kingdoms, and thus in different branches of the hierarchy. A complete classification of all living things would thus require a value for each of the seven levels.

A classification scheme can be used for classification of a population taken from just one of its nodes. For example, since primate is an instance of Order, a classification of all primates could consist of just three values, one for each of the levels Family, Genus, and Species. Similarly, a classification of all modern-day-trees could consist of values for just Genus and Species. Currently our model does not support the definition of one classification scheme in terms of another, but such capabilities are highly desirable and will be provided in a later version of this model.

In using a naming classification scheme for classification of a given population, it is necessary to identify the following items: 1) a globally unique name or other reference for a classification scheme, 2) a unique identifier for each level within the scheme, and 3) a value for each level. This is accomplished by defining a *classification* to be a reference to a classification scheme, i.e. a SchemeURN, and a set of level-value pairs. For trees the level-value pairs would be

{(Genus, <genus name>), (Species, <species name>)}.

In order to accommodate multi-level naming classification schemes, the Classification class in our model has a dependent subclass called LevelValuePair. Each classification instance contains a collection of level-value pairs that uniquely identify a node in the classification scheme hierarchy. In a simple 1-level classification scheme the level is superfluous, so by default it is called the *leaf* level.

Multi-level coded classification scheme

A multi-level coded classification scheme uses a string of codes to represent a path down the classification scheme hierarchy. Each node has a code that is unique under its parent; then, in an N-level hierarchy, a sequence of node codes from Level 1 to Level N uniquely determines a definition path through the tree. As above, each code is considered to be an itemValue that represents an itemName. As in the named classification scheme, item names do not need to be

unique. However, the codes are chosen so that it is convenient to represent the path from the root to the given node as a short string of codes.

In a coded classification scheme, the itemValue is defined to be the sequence of item codes from the root to the given node. Then a classification using this scheme need only supply a value for a single node. The values for each item in the path can be inferred from the sequence, thereby identifying the name for each item in the path.

An example of a coded classification scheme is one for newspaper articles that uses a 3-level scheme with 2 digits to identify the Level-1 subject (e.g. Sports, Business, News) and three digits each to identify Level-2 subjects and Level-3 subjects. The coded value "15052003" thus represents a named classification path as Sport (15), followed by Ski Jumping (052), followed by K180 Flying Jump (003). A complete version of this classification scheme has hundreds of nodes (cf. [11]).

In coded classifications, if the structure of the code path is known, i.e. 2:3:3 digits for the three levels, then the level name is unimportant. An application receiving the itemValue “15052003” would know to break it up into three codes 15, 52, and 3 to retrieve the three item names for the item values “15”, “15052”, and “15052003”. If the code structure is not known, then it could be implied by using a separator between the codes in the path, e.g. “15:052:003”, or one could begin with the leaf node and successively find each parent node until reaching the root. In any classification scheme, we will always assume that if any node of the scheme is known, then the sequence of parent nodes, and their associated level, can always be determined.

The Classification and LevelValuePair classes in our model support multi-level coded classification schemes in the same way they support multilevel named classification schemes. However, in the coded case where a single item value uniquely determines a node in the classification hierarchy, the classification is represented by a single pair, e.g. (leaf, 15052) to classify a Ski Jumping article. As above, the default *leaf* level indicates that the item value uniquely identifies a node in the hierarchy.

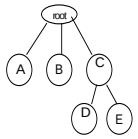
Subset classification scheme

A subset classification scheme presents a list of options to pick from, and then classifies a member of a population based on the specific subset of those options assigned to that member. For example, a classification scheme may list 5 hardware/software options as pre-requisites for being able to use an application program. Each potential user is classified by the specific subset of the pre-requisites they satisfy, from 0 to all 5. Since there are 32 possible subsets of the 5 options, the 5 options become a collection of 32 different classification nodes.

In a subset classification scheme it is always possible to choose the empty subset as a classification. In our model we need a way to indicate the empty set as an explicit choice, and as something distinct from no classification at all. Using a keyword like EMPTY is one option; others are also under consideration.

The subset classification scheme generalizes to any previously defined classification scheme provided that it is possible to assign multiple classification node references to any member of the population being classified. If the initial classification scheme has N possible classification

items, then the derived subset classification scheme has 2**N possible classification values. For simple 1-level classification schemes and for multi-level coded classification schemes this assignment is straight-forward. For multilevel named classification schemes one has to use some convention, e.g. nested sets, to separate the node references. For example, if the base hierarchy of a subset classification scheme is given by the hierarchy to the right, then the subset classification consisting of the nodes labeled A and D is represented unambigously by the nested set of level-value pairs {{(1,A)}, {(1,C), (2,D)}}.

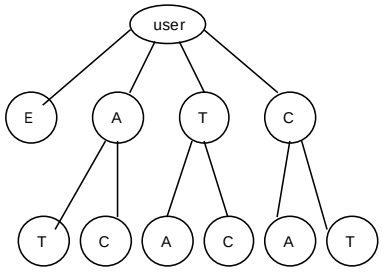


We support subset classification schemes in our information model for all subset schemes derived from base schemes whose item values uniquely identify a node at each level. We do this by allowing a classification to consist of multiple item values at each level of the classification scheme hierarchy. For example, if in the above hierarchy the nodes are identified by their labels, then the subset classification discussed above is represented by the set {(1,A), (2,D)}. An additional complication for multilevel subset schemes is that a classification may have multiple representations, e.g. {(1,C)} and {(2,D), (2,E)} represent the same classification.

Prioritized list classification scheme

A prioritized list classification scheme presents a list of options to pick from, and then classifies a member of a population based on the specific priorities given to each item. Each member of the population is first classified by choosing a subset of the items as in a subset classification scheme and then ordering the subset by priority. If the initial set of choices is N items from a 1-level classification scheme, then a prioritized list of K items could be viewed as a K-level classification scheme where each level identifies the position of the item in the list. This approach provides SUM(N!/(N-k)! | k=0,…K) nodes in the K-level prioritized classification scheme. We support prioritized list classification schemes in our information model by allowing a 1-level classification scheme to be re-specified as a prioritized K-level scheme.

A simple example of a prioritized classification scheme is taken from LTSC LOM [7], where an educational learning object is classified by prioritizing the users for whom it is designed. Beginning with a simple 1-level scheme determined by {Adults(A), Teens(T), Children(C)}, a prioritized 2-level scheme is represented by the hierarchy on the right. A classifier could choose any one of the 10 nodes for the classification, where E identifies the EMPTY node. A product designed for Teens but acceptable for older children would be classified as {(1,T), (2,C)}.



User-provided classification value

In some classification schemes, the scheme-definer identifies a specific partition of the population, and then wants to offer the classifier an opportunity to provide a refinement that is not pre-specified. We want to be able to define a classification scheme that offers such an alternative in a standard manner.

For example, consider the student classification scheme introduced above, i.e. {Freshman (FR), Sophomore (SO), Junior (JR), Senior (SR), Special (SP)}, where the scheme-definer wants to encourage the classifier to provide refinements for various types of special students. If the classification scheme definition specifies a node to be USER_INPUT, then the classifier could enter an arbitrary value that that satisfies the datatype of itemValue, and the result would be treated as a legal itemValue by any conformant registry/repository. The refined classification could be reported by two level-value pairs as {(Primary, SP), (Secondary, MyRefinement)}.

Mathematical definition of classification scheme

We define a classification scheme S to be a pair (N, ≤) where N is a set of nodes and ≤ is a partial ordering over N, with the additional requirement that the set of predecessors of every node is linearly ordered by the partial ordering and has a unique first element. Every node, x, is assigned a level number by the expression Level(x) = Card(Pred(x))+1. All nodes that have no predecessors are at level 1. The number of levels in the classification scheme is defined to be the maximum of {Level(x) | x in N}.

If N is a finite set, then every classification scheme S with N as its set of nodes has exactly K levels for some integer K between 1 and N inclusive. In the information model, we allow a scheme-definer to specify an itemValue and an itemName for each node and a levelCode and levelName for each level, and a schemeName for each classification scheme S.

Classification scheme as an XML DTD

The following XML ClassificationScheme specification can be used as an XML DTD both to define a new classification scheme or to represent an existing classification scheme.

```
<!ELEMENT ClassificationScheme
  (  Comment?,
     ClassificationLevel*,
     ClassificationNode+   )>
<!ATTLIST ClassificationScheme
      schemeName   CDATA   #IMPLIED  >

<!ELEMENT ClassificationNode
    (( ClassificationItem, ClassificationNode*) | USER_INPUT )>

<!ELEMENT ClassificationItem ( Comment? )>
<!ATTLIST ClassificationItem
      itemValue         CDATA #REQUIRED
      itemName          CDATA #IMPLIED
      levelNbr          CDATA #IMPLIED
      levelCode         CDATA #IMPLIED >

<!ELEMENT ClassificationLevel  ( Comment? )>
<!ATTLIST ClassificationLevel
      levelCode         CDATA #REQUIRED
      levelName         CDATA #IMPLIED
      levelNbr          CDATA #IMPLIED >

<!ELEMENT USER_INPUT EMPTY >
```

Semantic Rules

1. The nested hierarchy of ClassificationNode elements determines the partial ordering of a classification scheme over those nodes. The itemValue and itemName attributes identify the itemValue and itemName of each node.
2. The ClassificationLevel elements, if present, must be equal in number to the number of levels in the classification scheme derived from the nested hierarchy of ClassificationNode's. The levelCode and levelName attributes identify the levelCode and levelName of each level.
3. The schemeName, if present, identifies the CommonName of the classification scheme.
4. If USER_INPUT is specified as a ClassificationNode sub-element, then the itemValue in any classification that references this classification scheme can be any value that satisfies the datatype for itemValue.

Classification XML

A registered object may be classified according to any number of classification schemes. Each such classification is represented as the following XML element definition.

```
<!ELEMENT Classification  (LevelValuePair+)>
<!ATTLIST Classification
       schemeURN   CDATA  #REQUIRED
       schemeName  CDATA  #IMPLIED >

<!ELEMENT LevelValuePair (Comment?)
<!ATTLIST LevelValuePair
       levelCode  CDATA  "leaf"
       itemValue  CDATA  #REQUIRED
       levelName  CDATA  #IMPLIED
       levelNbr   CDATA  #IMPLIED
       itemName   CDATA  #IMPLIED >
```

Semantic Rules

1. The schemeURN references a registered classification scheme.
2. The set of (levelCode, itemValue) pairs must reference a single node of the classification scheme identified by schemeURN.

3.5 Registry Package

A registry package is a set of pointers to registry entries. Note that there are two levels of indirection here! A package is a set of pointers, not a set of registry entries; thus a registry entry can be represented as an element of many different registry packages without being copied multiple times. A registry package will often represent a collection of registered objects, but the registered objects can be determined only by first going to the registry entry that references that registered object. This indirection is purposeful. It allows registered objects to be withdrawn, or superseded or replaced by other registered objects without changing the content of the registry package. The user of a package always has the option to check if a registry entry element of the package still references the original object. The status of the registered object can be determined from the registrationStatus attribute of the registry entry.

The XML representation of a registry package is given by the following DTD:

```
<!ELEMENT  RegistryPackage (PackageMember*)>
<!ATTLIST RegistryPackage
       packageURN    CDATA  #REQUIRED
       packageName   CDATA  #IMPLIED
       regEntryID    CDATA  #IMPLIED >

<!ELEMENT PackageMember EMPTY
<!ATTLIST PackageMember
       memberURN     CDATA  #REQUIRED
       memberURL     CDATA  #IMPLIED
       regEntryID    CDATA  #IMPLIED >
```

Semantic Rules

1. The packageURN identifies a registry entry whose objectType attribute is *registry package*.
2. The packageName, if present, is the commonName of the registry entry identified by packageURN.
3. The objectURL of the registry entry identified by packageURN points to a registry package in the local repository.
4. Each memberURN identifies a registry entry in the local registry.
5. Each registry entry identified by a memberURN participates as the assocItem in an association instance where the association role is References and the givenItem is the registry entry identified by the packageURN.

3.6 Other Metadata

The OtherMetadata class in Figure 3 is just a place holder for a number of other dependent UML classes that contain additional metadata for a registered object. We believe that it will be very helpful to have at least four other classes that give helpful structure for capturing various kinds of additional information. The ContextNames class is needed to capture different names for a registered object in different environments, including URN's assigned by other registries. The TranslatedDescription class is needed to provide an opportunity for translations of the original description attribute of the RegistryEntry class into any number of different human readable languages. The RelatedData class is very helpful for capturing references to other informational items that are strongly related to the registered object but not important enough to be registered themselves, and the ProductionCredits class provides an opportunity to list the creators of a registered object. Figure 4 provides the details of the RegistryEntry class together with other classes that are dependent on it. The following paragraphs give additional discussion for usage of these classes.

Related data

Related data is a conceptual notion used to reference data objects that are related to a registered object. This category of metadata is reserved for things like graphic visualizations, example sets, white papers, usage scenarios, extended documentation, vendor propaganda, etc. Sometimes related data objects will be very important, e.g. usage documentation, and will themselves be registered objects. At other times, related data objects will be less important support information for a registered item and will not be registered. In the later situation, the registry may maintain a simple list of references to the related data. Each such reference becomes an instance of the RelatedData class.

In many cases, related data objects will not be registered, yet they are valuable supplements to a registered object and should be optionally available. In these cases the registration authority will keep a list of the name and type of the related data object, with just enough additional

information so that they can be presented as options on a web page. In those cases the related objects and their associated metadata will be created, held, and maintained by the submitting organization or some other external repository.

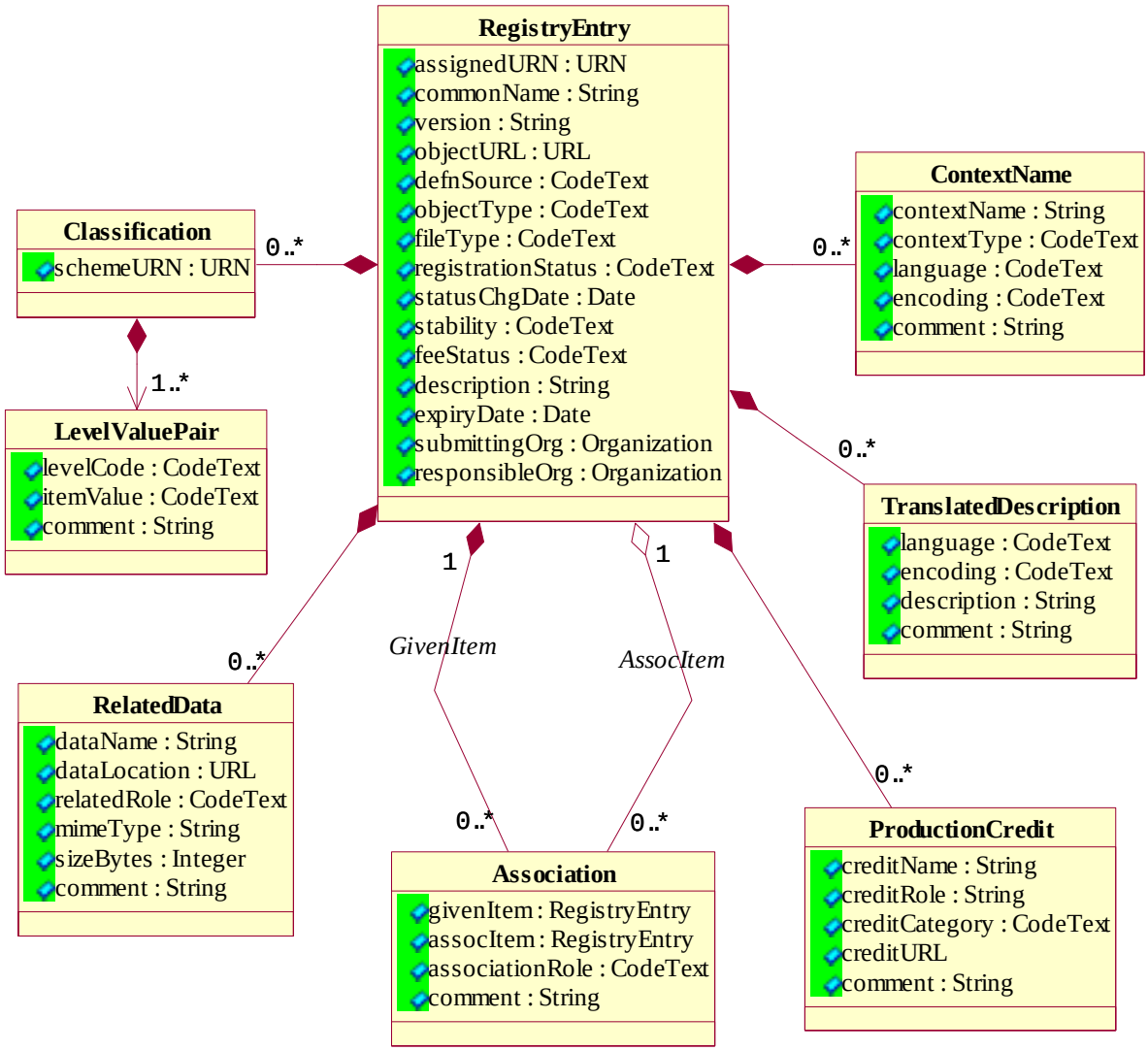


Figure 4 - RegistryEntry with dependent classes

Each instance of the RelatedData class consists of a human readable name, a URL, a standardized classification or *related role* as a supporting object, a MIME file representation type, the size of the file in bytes, and a short human readable comment. The *related role* may be defined by the registry's sponsoring organization, thereby giving a very specific reference to required or optional documentation. As with *association roles*, an advantage of the proposed information model is that the related roles can be declared, modified, or queried via standardized access methods.

Context names

Context name is a conceptual notion used to represent alternate or alias names for a registered object. Especially significant are names used in special circumstances or contexts, e.g. short names for local identifiers in a specific programming language context, or globally unique qualified names that satisfy a specific hierarchical qualification structure. In many cases the alternate names will include the globally unique names assigned to the registered object by other registration authorities. Alternate names can also be used for names in different human languages with characters encoded in unusual character sets.

A ContextName instance consists of the context name, an abstract *context type* to identify the contextual category in which that name is used, and a human readable comment that further explains how that name should be used. A context name instance is also tagged with optional language and character set codes that apply to the context name and any optional comments. As with *association roles* and *related types*, the *context type* of a context name may be defined by the sponsor of a registry specification, thereby allowing unique usage but retaining standardized access methods.

Translated descriptions

A translated description is a translation of the description attribute given in the RegistryEntry instance. The intent is that each translated description instance is a direct translation of the original description in a different human language. Every instance is tagged with a human language code and a character set code that apply to the text of the description and to any optional comments.

A description is intended to play the role of an abstract and keyword list commonly required for library resources. We are assuming that the keyword list, if any, will be buried in the description so that a search engine will find them for indexing. If necessary, we could make keywordList a separate attribute of both the RegistryEntry class and the TranslatedDescription class

Production credits

Production credit is an abstract notion used to identify people, places, or organizations that contributed in any way to the creation of a registered object. This notion is particularly significant for library resources and educational or learning materials (cf [4] and [9]). ProductionCredit instances are different than Contact instances, although they could overlap. Contacts are more like salespeople, who can speak to the final product, its availability, usage, registration status, and future plans, whereas ProductionCredit instances will be much more like movie credits, giving credit even for very specialized contributions to the creation of the registered resource, e.g. illustrator, technical support, etc. The intent is that production credits be presented in much the same way as credits at the end of a motion picture. It usually suffices to give a name and the role played.

A ProductionCredit instance consists of the name of the entity deserving recognition, the role it played in the production, an abstract *credit category* with valid values defined by the sponsor of the specialization being used, an optional URL to help locate the home page of the named entity, and an optional comment that might further explain the role played by that entity in the

production of the registered object. As with *association roles*, an advantage of the proposed information model is that the *credit category* can be declared, modified, or queried via standardized access methods.

4. Administration Facilities

An information model for a registry/repository is concerned with more than just the metadata for registered objects. It is also expected to provide support for maintaining some degree of version control for these objects as well as administrative accountability for the submission and management of registry entries. The UML class diagram in figure 5 provides a structure whereby all of these administrative expectations can be met. We assume that each class plays two roles: it identifies the structure of each instance and it gives the name of a container for all persistent instances of that type.

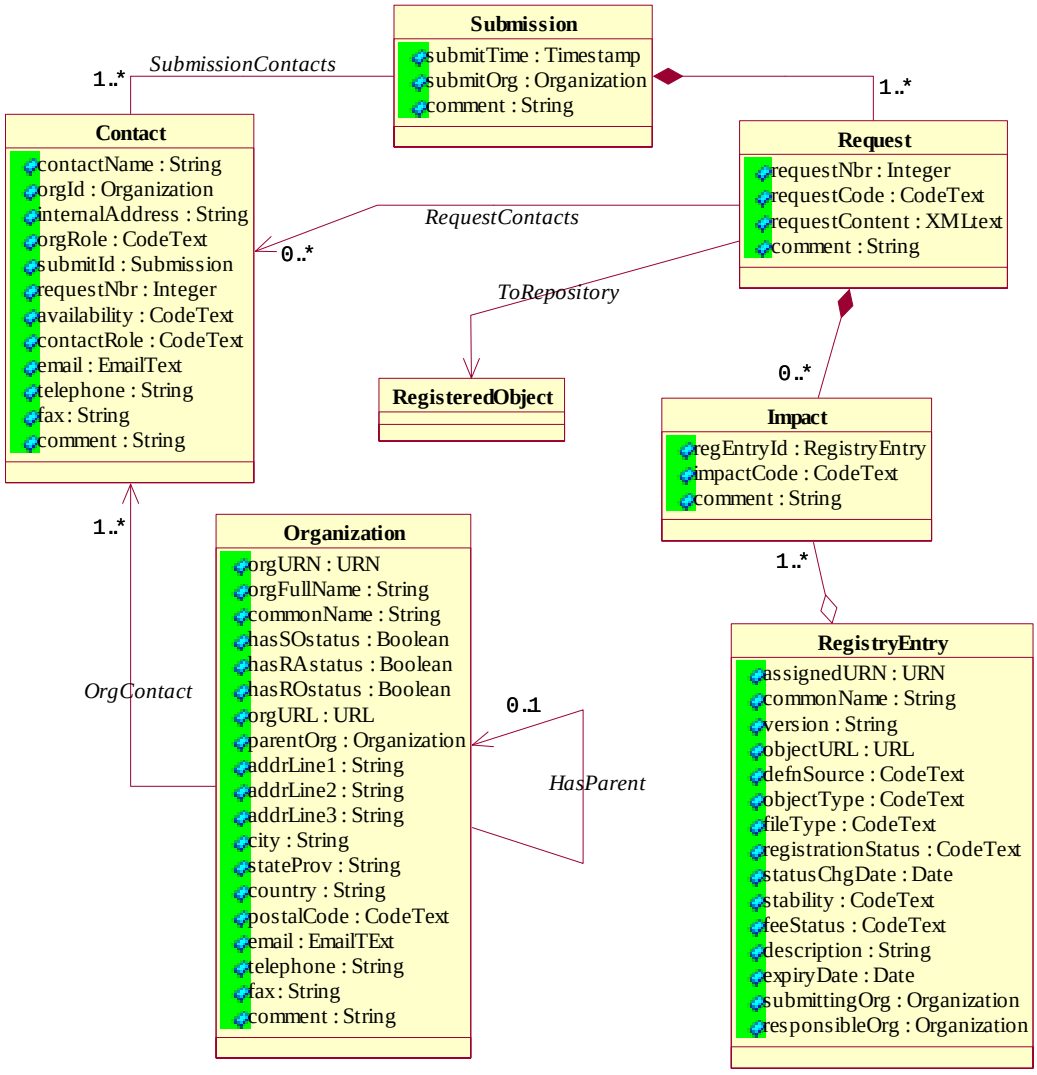


Figure 5 - Registry Administration

The administration facility must keep track of the following:

- An organization contact for each Organization instance.
- A submission contact for each Submission instance.
- Optional administrative and technical contacts for each Request instance.
- An Impact instance for each impact that a Request has on any registry entry.

4.1 Who submits objects for registration

Different registries will have quite different rules for who is allowed to submit objects for registration. Some professional organizations like IMS [8] will want to establish registries for presenting the intellectual efforts of their membership to the rest of the world, so it will be open to their entire membership. Others will be strongly manufacturing oriented, trying to automate the manufacturing process using registries for parts and suppliers as well as DTD's for machine-to-machine messaging and schemas for machine processes. These registries may have a very limited number of submitters. Some registries will be open to the public, soliciting and registering reviews of products or services. And as we can see from our bibliography, a major application of registry/repository implementations will be for supporting the workflow and business processes of electronic commerce. As such they will be open to new input from nearly any legitimate business or trade organization.

We believe that it is sufficient for a registry to maintain a list of all of the organizations it is prepared to do business with, with a minimum amount of contact information for each one. Any additional information needed could be required by submittal of a company profile for registration. Minimum contact information includes a full legal name, address, telephone, and one or more contact points with contact names and email addresses. The registry will pre-load the organization entity with known organizations and then let others apply for status to make submissions. We follow the lead of ISO/IEC 11179 [12] and consider three types of organizations: registration authority (RA), responsible organization (RO), and submitting organization (SO). A registration authority will be certified by some central authority to maintain a conforming registry/repository; they will trust one another and freely share registry information. Responsible organizations will often be standards committees, consortia, or trade associations that develop and maintain specifications; they may or may not maintain registries for their own products. Submitting organizations will be involved with the bulk of registry interaction by submitting new information. There needs to be a bootstrap registry service that allows organizations to request that they be recognized as submitting organizations.

4.2 Submission workflow

A submission is a collection of requests, in the form of a message, sent from a submitting organization to a registry. We assume that some transport service has delivered the message to the registry and that some authentication service has authenticated the submitter as a legitimate submitting organization. The registry/repository then swings into action.

The registry logs each submission, assigns it a persistent object identifier, timestamps it with the date and time received, adds one new entry to the Submission container and one or more new entries to the Contact container. It then considers each request separately.

Each request is assigned a request number to distinguish it from other requests in the same submission. Since a request is part of a submission it is not necessary to assign each request a separate unique identifier. Each request is assigned a *request code* from the following list.

Registry Request

Request Code	Request Name
addAssoc	AddAssociation
addClassif	AddClassification
addCtxName	AddContextName
addPrdCrd	AddProductionCredit
addRelData	AddRelatedData
addTrnDesc	AddTranslatedDescription
defClassif	DefineClassificationScheme
defRegPkg	DefineRegistryPackage
delAssoc	DeleteAssociation
delClassif	DeleteClassification
delCtxName	DeleteContextName
delPrdCrd	DeleteProductionCredit
delRelData	DeleteRelatedData
delTrnDesc	DeleteTranslatedDescription
modClassif	ModifyClassificationScheme
modRegPkg	ModifyRegistryPackage
modRegEntry	ModifyRegistryEntry
regObj	RegisterObject
regSO	RegisterSubmittingOrg
repRegObj	ReplaceRegisteredObject
supRegObj	SupersedeRegisteredObject
wdrRegObj	WithdrawRegisteredObject

The request codes determine a simple 1-level classification scheme for requests. They are in a one-to-one correspondence with registry service requests that can effect a change of registry content. This classification will support queries on registry content to determine which requests result in objects being superseded or replaced and which requests alter other metadata content. The request code and the XML content of each request are stored in the Request container. After some period of time, the XML content of each request instance may be deleted, but the remainder of the new request instance is kept permanently as part of the administrative record.

4.3 Impact workflow

A request may have an impact on one or more registry entries. For example, a request to supersede registered object A with a new registered object B will have impacts on the registry entries for both A and B. For accountability and versioning control, the registry must retain a record of all such impacts.

If a request creates a new registry entry, we want to link that request to the entry it creates so that we will have an administrative record of the date and time the entry was created and who owns it. Similarly, if new associations or new related data items are submitted for an existing registry entry, we will want an administrative record of the date and time of each addition. As part of registry version control requirements we especially want to maintain a record of all requests that result in replacement of one registered object by another, the registration of a new version that supersedes a previous version, or the deprecation or withdrawal of a registered object. The classification, context name, translated description, and production credit instances may be submitted by organizations other than the owner of the registry entry they are linked to, so the registry should maintain an administrative record of all such additions or modifications.

We can maintain an administrative record of all new versions and replacements and all other additions or modifications to registry metadata by maintaining a many-to-many relationship between Request instances and RegistryEntry instances. We define a new class, the Impact class, to record this information. The registry will populate the Impact container as it performs the actions of each request. All impact instances are created and modified solely by the registration authority. Each impact instance is assigned an *impact code* from the following list.

Registry Impact

Impact Code	Impact Name
AAS	Add Association
ACF	Add Classification
ACT	Add Contact
ACN	Add Context Name
APC	Add Production Credit
ARO	Add Registered Object
ARE	Add Registry Entry
ARD	Add Related Data
ASO	Add Submitting Organization
ATD	Add Translated Description
DAS	Delete Association
DCF	Delete Classification
DCT	Delete Contact
DCN	Delete Context Name
DPC	Delete Production Credit
DRO	Delete Registered Object
DRE	Delete Registry Entry
DRD	Delete Related Data
DSO	Delete Submitting Organization
DTD	Delete Translated Description
UAS	Update Association
UCF	Update Classification
UCT	Update Contact
UCN	Update Context Name
UPC	Update Production Credit
URO	Update Registered Object
URE	Update Registry Entry
URD	Update Related Data
USO	Update Submitting Organization
UTD	Update Translated Description

The impact codes determine a simple 1-level classification scheme for impact instances. They are in a one-to-one correspondence with add, delete, and update operations on the ten basic classes that make up the registry/repository information model. This classification will support queries on registry content to determine which requests result in impacts of a given type on registered objects. All impact instances should be permanently maintained as part of the administrative record.

5. Registry Services

From figure 1 in the Introduction, we see that registry services include Submission(), Query(), and two special methods: GetRegistryEntry() and GetRegisteredObject(). A Submission is a collection of requests such as the following:

- register a new object,
- add to or modify the associations of a registered object,
- add to or modify the classifications of a registered object,
- add to or modify the other metadata instances of a registered object,
- register a company or organization as a Submitting Organization,
- modify the content of a previous submission,
- withdraw a previously registered object,
- supersede an existing registered object,
- replace an existing registered object.

A Query asks for the return of data from the registry/repository. The two special "Get" methods are designed for implementations that do not support Query.

We specify XML DTD's for the request and the response of each of these services. We assume that XML elements have already been specified for each of the UML classes defined herein. For details of the Oasis specification, see [15].

5.1 GetRegisteredObject()

```
<!ELEMENT GetRegisteredObject EMPTY >
<!ATTLIST GetRegisteredObject
       assignedURN   CDATA   #REQUIRED >

<!ELEMENT GetRegisteredObjectResult (RegisteredObject) >
<!ATTLIST GetRegisteredObjectResult
       assignedURN         CDATA   #REQUIRED
       commonName          CDATA   #REQUIRED
       objectURL           CDATA   #REQUIRED
       objectType          CDATA   #REQUIRED
       fileType            CDATA   #REQUIRED
       registrationStatus  CDATA   #REQUIRED
       stability           CDATA   #REQUIRED >
```

5.2 GetRegistryEntry()

```
<!ELEMENT GetRegistryEntry
   ((   WithDescription
      | WithClassifications
      | WithAssociations
      | WithRelatedData
      | WithContextNames
      | WithProductionCredits
      | WithTranslatedDescriptions )* )>
<!ATTLIST GetRegistryEntry
      assignedURN   CDATA   #REQUIRED >

<!ELEMENT WithDescription             EMPTY >
<!ELEMENT WithClassifications         EMPTY >
<!ELEMENT WithAssociations            EMPTY >
<!ELEMENT WithRelatedData             EMPTY >
<!ELEMENT WithContextNames            EMPTY >
<!ELEMENT WithProductionCredits       EMPTY >
<!ELEMENT WithTranslatedDescriptions  EMPTY >

<!ELEMENT GetRegistryEntryResult
   ( RegistryEntryInstance ,
     Classification* ,
     Association* ,
     RelatedData* ,
     ContextName* ,
     ProductionCredit* ,
     TranslatedDescription* ) >
```

5.3 Submission Services

```
<!ELEMENT Submission ( Request+, Contact+ )>
<!ATTLIST Submission
      SubmitOrgURN   CDATA   #REQUIRED >

<!ELEMENT Request
  (  (
         AddAssociation
       | AddClassification
       | AddContextName
       | AddProductionCredit
       | AddRelatedData
       | AddTranslatedDescription
       | DefineClassificationScheme
       | DefineRegistryPackage
       | DeleteAssociation
       | DeleteClassification
       | DeleteContextName
       | DeleteProductionCredit
       | DeleteRelatedData
       | DeleteTranslatedDescription
       | ModifyClassificationScheme
       | ModifyRegistryPackage
       | ModifyRegistryEntry
       | RegisterObject
       | ReplaceRegisteredObject
       | SupersedeRegisteredObject
       | WithdrawRegisteredObject    ),
    Contact*,
    Comment? )>
```

The XML Element definitions for the listed Request alternatives are still under development, but it is expected that they will be analogous to the Request elements defined in [15].

5.4 Query Services

The Query services for this specification are still under development (cf [13]).
For information contact michael.kass@nist.gov.

6. Conformance Alternatives

An implementation may claim conformance to this specification at any of several different levels, including RegistryOnly, RegistryRepositoryBasic, and RegistryRepositoryQuery. In addition, the two repository levels may be specified with or without Validation.

RegistryOnly

If an implementation claims conformance at the RegistryOnly level, then it must suppport the RegistryEntry, Association, Classification, RelatedData,, ContextName, ProductionCredit, TranslatedDescription, Organization, Contact, and Submission classes via implementation of the XML Submission DTD with explicit support for the following registry services:

- RegisterSubmittingOrganization, RegisterObject without object content, ReplaceRegisteredObject, SupersedeRegisteredObject, WithdrawRegisteredObject, ModifyRegistryEntry, AddAssociation, AddClassification, AddContextName, AddRelatedData, AddProductionCreddit, AddTranslatedDescription, DeleteAssociation, DeleteClassification, DeleteContextName, DeleteRelatedData, DeleteProductionCredit, DeleteTranslatedDescription, and

- GetRegistryMetadata(assignedURN : URN)

RegistryRepositoryBasic

If an implementation claims conformance at the RegistryRepositoryBasic level, then it must satisfy the requirements for RegistryOnly conformance. In addition, it must support the RegisteredObject, RegistryPackage, and ClassificationScheme classes via implementation of the following additional registry services:

- RegisterObject with object content, DefineRegistryPackage, DefineClassificationScheme, ModifyRegistryPackage, and

- GetRegisteredObject(assignedURN : URN)

RegistryRepositoryQuery

If an implementation claims conformance at the RegistryRepositoryQuery level, then it must satisfy the requirements for RegistryRepositoryBasic conformance. In addition, it must support full implementation of the Impact class and implementation of the following registry services:

- ModifyClassificationScheme, GetSchemeSubtree, and
- Query()

with Validation option

An implementation may claim conformance at any RegistryRepository level with or without Validation. If with Validation is specified, then the implementation must support validation of any registered object whose fileType is declared to be text/xml and that has a validatesTo association with a registered object of type text/xml/dtd or text/xml/schema. In addition the implementation shall state any other fileType's for which it supports validation.

7. Conclusions

We have specified a proposed registry/repository information model. It is represented by a UML class model and an XML services interface. This general-purpose model is capable of supporting a wide range of specialized implementations, each catering to the specific requirements of some sponsoring group. The model provides a standardized registry services interface that can be used by abstract agents to assist a human or some other software process to register new objects, provide appropriate metadata for those objects, browse or query registry content, filter out irrelevant references, and retrieve the content of selected items.

The primary advantage of this generalized approach to registry/repository specification is that the model is simple enough so that it can be easily implemented with existing off-the-shelf data management technologies, yet flexible enough to support the required specializations of many different kinds of sponsoring groups. The static logical structures of the model are able to support a standard registry services interface across all specializations so that the registry content is accessible to virtually any electronic agent.

A reference implementation of this model is currently under development at NIST.

Bibliography

1. BizTalk, *BizTalk Framework Specification*, version 1.0, 9 December 1999, http://schemas.biztalk.org/BizTalk/gr677h7w.xml.

2. CommerceNet, *The eCo Specification*, CommerceNet, Inc., September 1999, http://eco.commerce.net/specs/index.cfm.

3. cXML, *cXML Specification and UsersGuide*, version 1.1, June 2000, http://www.cxml.org, 122 pages plus DTD's.

4. Dublin Core, *Metadata Element Set*, ANSI/NISO Z39.85-200x, ISSN: 1041-5653, for Z39 ballot July 1 through August 15, 2000, http://purl.org/dc.

5. ebXML, Registry and Repository project, http://www.edxml.org/project_teams/registry/registry.html.

6. Gallagher, L. and L. Carnahan, *Representing Learning Object Metadata in a Generic Registry/Repository*, draft NIST report, 1 September 2000, ftp://xsun.sdct.itl.nist.gov/regrep/IMSrepresentation.pdf, 27 pages.

7. IEEE Learning Technology Standards Committee (LTSC), IEEE P1484.12, Learning Object Metadata, Working Draft Document, version 3.8, dated 7 November 1999.

8. IMS, *Meta-Data Best Practice and Implementation Guide*, Final specification, version 1.1, http://www.imsproject.org/metadata/mdbestv1p1.html, 5 May 2000, 37 pages.

9. IMS, *Learning Resource Meta-data - Information Model*, Final specification, version 1.1, http://www.imsproject.org/metadata/mdinfov1p1.html, 5 May 2000, 24 pages.

10. IMS, *Learning Resource - XML Binding Specification*, Final specification, version 1.1, http://www.imsproject.org/metadata/mdbindv1p1.html, 5 May 2000, 25 pages.

11. IPTC, *Subject Reference System for News Items*, April 2000, IIM Guideline 3, Version 4, http://www.iptc.org/SubjectView.zip.

12. ISO 11179, *Information Technology - Data Management and Interchange - Metadata Registries* - Part 3: Registry Metamodel, ISO/IEC CD 11179-3, document SC32 N490, 30 June 2000, 124 pages.

13. Kass, Michael, *Information Model Query DTD*, 16 October 2000, ftp://xsun.sdct.itl.nist.gov/regrep/InfoModelQuery.dtd.

14. NIST, *Candidate Oasis Information Model*, NIST contribution to Oasis, Draft 8 June 2000, Revised 10 July 2000, ftp://xsun.sdct.itl.nist.gov/regrep/OasisModel.pdf, 54 pages.

15. NIST, *Candidate Oasis Registry/Repository XML Definitions*, NIST contribution to Oasis, Draft 8 June 2000, Revised 21 September 2000, ftp://xsun.sdct.itl.nist.gov/regrep/OasisNewXML.pdf, 67 pages.

16. OASIS, Organization for the Advancement of Structured Information Standards, Registry and Repository working group, http://www.oasis-open.org and http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html.

17. OBI Specification, *Open Buying on the Internet*, Technical Specifications, release v2.1, 1999, http://www.openbuy.org/obi/specs/xyx.pdf, 381 pages.

18. OMG, *Meta Object Facility (MOF) Specification*, Version 1.3, Object Management Group, dated March 2000, http://cgi.omg.org/cgi-bin/doc?formal/00-04-03.pdf, 522 pages.

19. RosettaNet, *Standards - Partner Interface Processes (PIPs), Dictionaries, Implementation Framework, Product and Partner Codes*, http://www.rosettanet.org.

20. UDDI, *Universal Description Discovery and Integration*, 6 September 2000, http://www.uddi.org/specification.html, Data Structure Reference, 30 pages, API Specification, 60 pages.

21. XBRL, *eXtensible Business Reporting language*, http://www.xfrml.org.