Open Geospatial Consortium Inc.

Date: 26-SEPT-2005

Reference number of this OGC® Project Document: **OGC 05-084**

Version: 0.0.1

Category: OpenGIS® Discussion Paper

Editor: Vincent Delfosse (Ionic Software)

Catalog 2.0.1 Accessibility for OWS3 – Best Practices

An OGC Interoperability Program Report

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of

Document type: OpenGIS® Discussion Paper
Document subtype:
Document stage: Approved
Document language: English

any relevant patent rights of which they are aware and to provide supporting documentation.

Contents

i. Preface

The Open Geospatial Consortium (OGC) is an international industry consortium of more than 300 companies, government agencies, and universities participating in a consensus process to develop publicly available geo-processing specifications. This Interoperability Program Report (IPR) is a product of the OGC Web Services, Phase-3 (OWS3) project.

The OGC Web Services Initiative, Phase 3, is part of the OGC's Interoperability Program: a global, collaborative, hands-on engineering and testing program designed to deliver prototype technologies and proven candidate specifications into the OGC's Specification Development Program. In OGC Interoperability Initiatives, international teams of technology providers work together to solve specific geo-processing interoperability problems posed by Initiative sponsors.

ii. Submitting organizations

This draft Interoperability Program Report – Engineering Specification is being submitted to the OGC Interoperability Program by the following organizations:

Ionic Software
18, Rue de Wallonie
4460 Grace-Hollogne
Belgium

Document Contributor Contact Points

All questions regarding this submission should be directed to the editor or the submitters:

Vincent Delfosse
Ionic Software

iii. Revision history

Date	Release	Description
26-SEPT-2005	0.0.0	▪ Initial version (incomplete)
28-SEPT-2005	0.0.1	▪ Initial complete version
April 2006	0.0.1	▪ CNR _ Fix copyright, add reference, general check and edit

Foreword

The OGC Catalog-Web Profile is a complex specification that implies usage of many concepts, such as ressources, metadata, registry, registry information model, harvesting, etc. This document is a user-friendly introduction to these concepts. It will help the understanding of the Catalog specification in general and of the Catalog Web profile with ebRIM in particular.

Attention is drawn to the possibility that some of the elements of this part of OGC 05-109 may be the subject of patent rights. Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

1 References

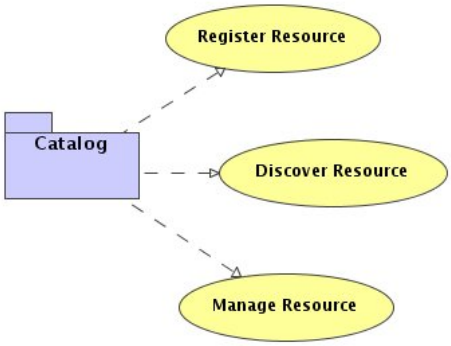
OpenGIS® Catalogue Service Implementation Specification Version 2.0.1 with corrigendum, OGC document 04-021r4, August 2004.

Oasis Document, *Oasis/ebXML Registry Information Model, V2.5*, http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrim-2.5.pdf, 2003

2 Introduction

An OGC Catalog is a collection of descriptive information (metadata) about data that can be geographically referenced. It can be seen as a web-enabled database (or repository), in the form of an OGC service. Typically, the needed functionnalities can be devided in three groups :

- registering a resource in the catalog (i.e., added resources to the repository)
- discovering a resource (i.e., searching withing the repository)
- managing the resources (i.e., to be able to update or delete some resource, adding classification to a service, etc).



To achieve these required features, the OGC WRS (Web Registry Service) specification defines a set of operations that must be supported by a catalog implementation. Examples of such standard operations are of RegisterResource, GetRecord, or DeleteResource, etc.

The problem we want to address in this document is the format of the data exchanged between the catalog server and its client : we need one format for expressing the resources in a catalog (as the result of discover resource, or a register resource), and one format for expressing the queries a client application wants to send to a catalog.

Let's only consider the format to express the resources themselves for the moment. We are facing two contradictory requirements:

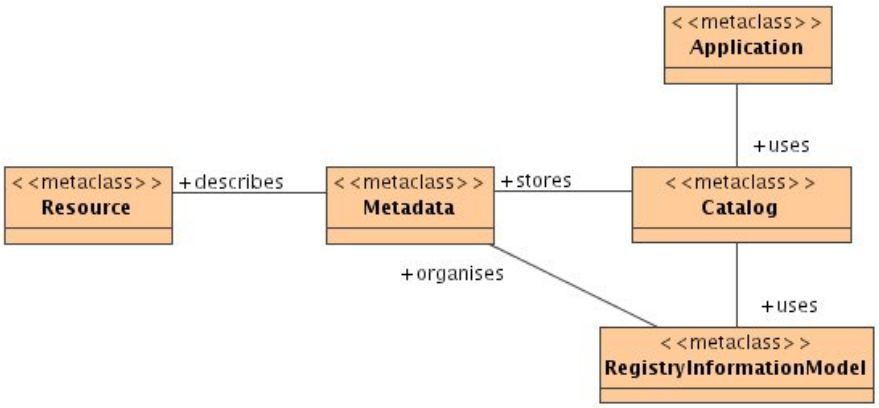
- as the catalog is designed to manage a large variety of different resource types, we need a format flexible and extensible enough to support this;

- at the same time, because the goal here is to reach 100% interoperability, we need to completely and unambiguously describe how a specific resource type will be described in a catalog.

The OGC did wisely agree on a two-step approach to solve this problem :

- first defining a generic way of expressing any type of data. The usual name for such generic data description framework is RIM - Registry Information Model.

- then for every specific type of resource, we will have to provide a precise mapping between the resource structure and its representation withtin the RIM.



We can then draw a high level view of the main concepts used throughout this document :

- Resources, defined very verbosely in http://www.webreference.com/html/tutorial2/1.html. It is just something, *anything* that is *unique*. In this RFQ, some resources has been identified as being important to the OGC community, i.e. Data, Services, Sensors, UML models, XML Schemas, SLD, etc.

- Metadata. In this case, Metadata is a group of information that describes a resource. We clearly see in the diagram that metadata role is to describe some resources. In this RFQ, the metadata are ISO 19115 for the Data, ISO 191119 for the Services, SensorML for the Sensors, a part of the UML model (it may contain its own metadata), a part of the XML Schema (idem), a part of SLD (idem). As we see, some resources are self-describing in a sense they contain their own metadata.

- Catalog, introduced as a storage mechanism for metadata. There are two possibilities here, metadata can be stored in the Catalog component or in a metadata repository linked to the Catalog, at this level of abstraction, it does not matter. In any case, the

Catalog will have to store at least a part of the metadata to be able to answer the searches.

- Finally, the Registry Information Model is probably the more complex component to understand. It is used by a Catalog to organize the Metadata and their relationship and thereby organize the resources amongst themselves. This is a key component as it will drive the registration process and the search process. In this RFQ, three Registry Information Model are present, ebRIM, ISO19115/19119 and, by default in the Catalog Service-Web specification, Dublin Core.

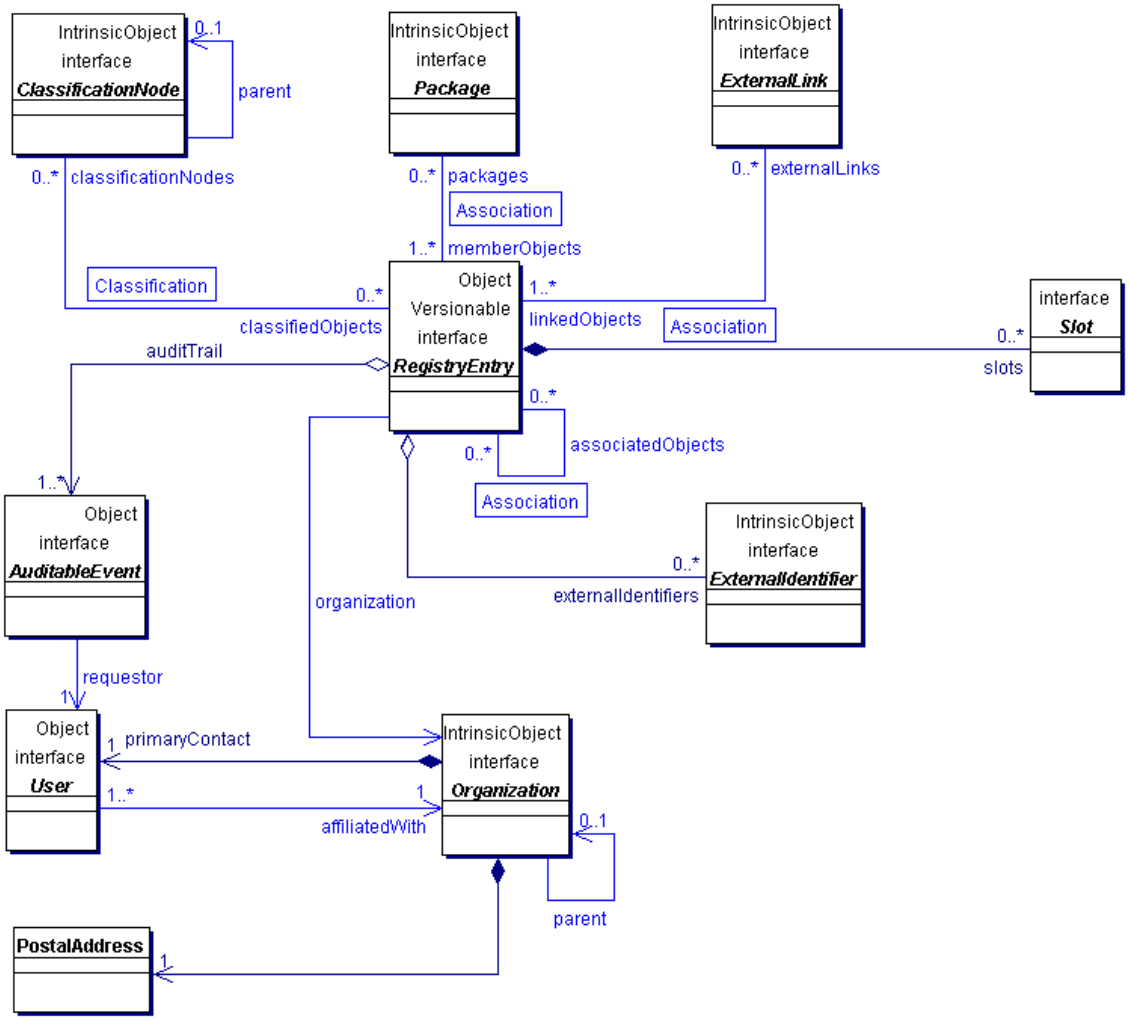
3 Registry Information Model (RIM)

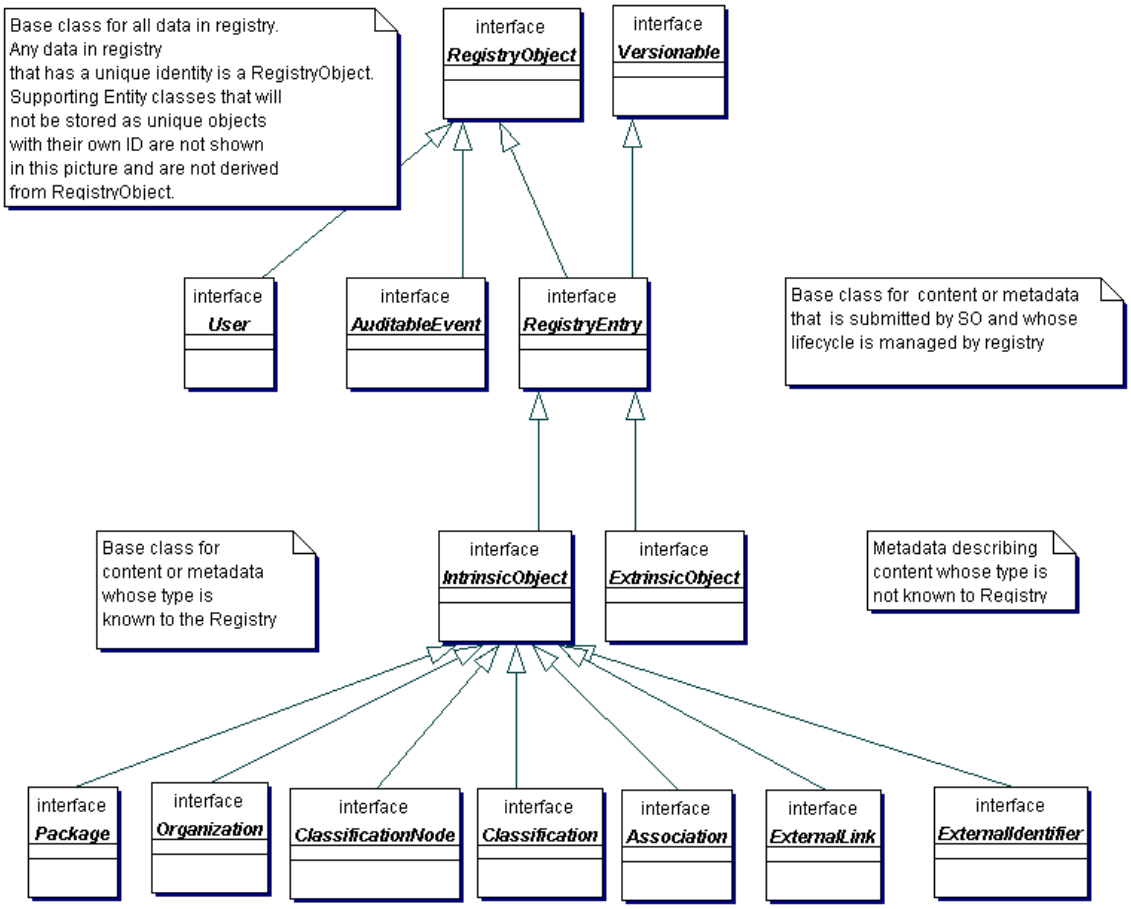
The registry information model (RIM) is a conceptual metamodel that specifies the structure of metadata within a registry; its main purpose is to provide a formal structure representing metadata resources and their interrelationships. The model also establishes a flexible basis for defining implementation and processing requirements. While the metamodel need not be physically implemented as specified, it must be possible to unambiguously map between the implementation and the metamodel in both directions. The importance of the metamodel lies in its specification of a common model for understanding, sharing, and reusing of the contents of registry implementations.

In effect, a registry can serve as a ‘metadata hub’ that presents a common information model. Various wrappers or transformers may be employed to interoperate with different metadata repositories. A repository item is associated with a set of standard metadata defined as attributes of the RegistryObject class and its sub-classes as described in the core ebXML registry information model (ebRIM). The RIM is a data model that describes the possible states of the registry; it is used to explain the behavior of the operations that must preserve the constraints of the information model. One benefit of adopting the general ebRIM model is that every registry instance is built around the same model, so linking or federating registries becomes feasible. Furthermore, it is possible to accommodate diverse registry implementations by defining additional mappings as needed.

4 Overview of the ebRIM structure

Here are two UML schemas of the different ebRIM elements.





The Registry Information Model provides a blueprint or high-level schema for the ebXML *Registry*. Its primary value is for implementers of ebXML *Registries*. It provides these implementers with information on the type of metadata that is stored in the *Registry* as well as the relationships among metadata *Classes*.

You will find a detailed description of ebRIM at http://www.ebxml.org/specs/**ebRIM**.pdf. We will quickly go through some of the important classes in there.

- RegistryEntry serves as a base class, providing basic functionnalities like links to associations, classifiactions, and slots. It also has some general attributes like name
- Slot provides a dynamic way to add arbitrary attributes to RegistryEntry.
- Association are used to define one-to-one associations between objects in the information model.
- Classification are used to classify RegistryEntry with a ClassificationNode within a ClassificationScheme.

5 EbRIM in details

We will go now more in the fundamental features of EbRIM. You will see an example using these features in the next section.

5.1 RegistryEntry and Slot

Slots provide a dynamic way to add arbitrary attributes to RegistryEntry instances. This ability to add attributes dynamically to RegistryEntry instances enables extensibility within the Registry Information Model.

If, for example, you want to store a PublicationDate to a given RegistryObject, you will actually create a Slot within this object. All the RegistryObject can hold any number of slots. A Slot object has a name, a type, and a collection of values.

In general, a Slot will be used to map ‘primitive attributes’, or attributes that can accept a String representation.

5.2 Associations

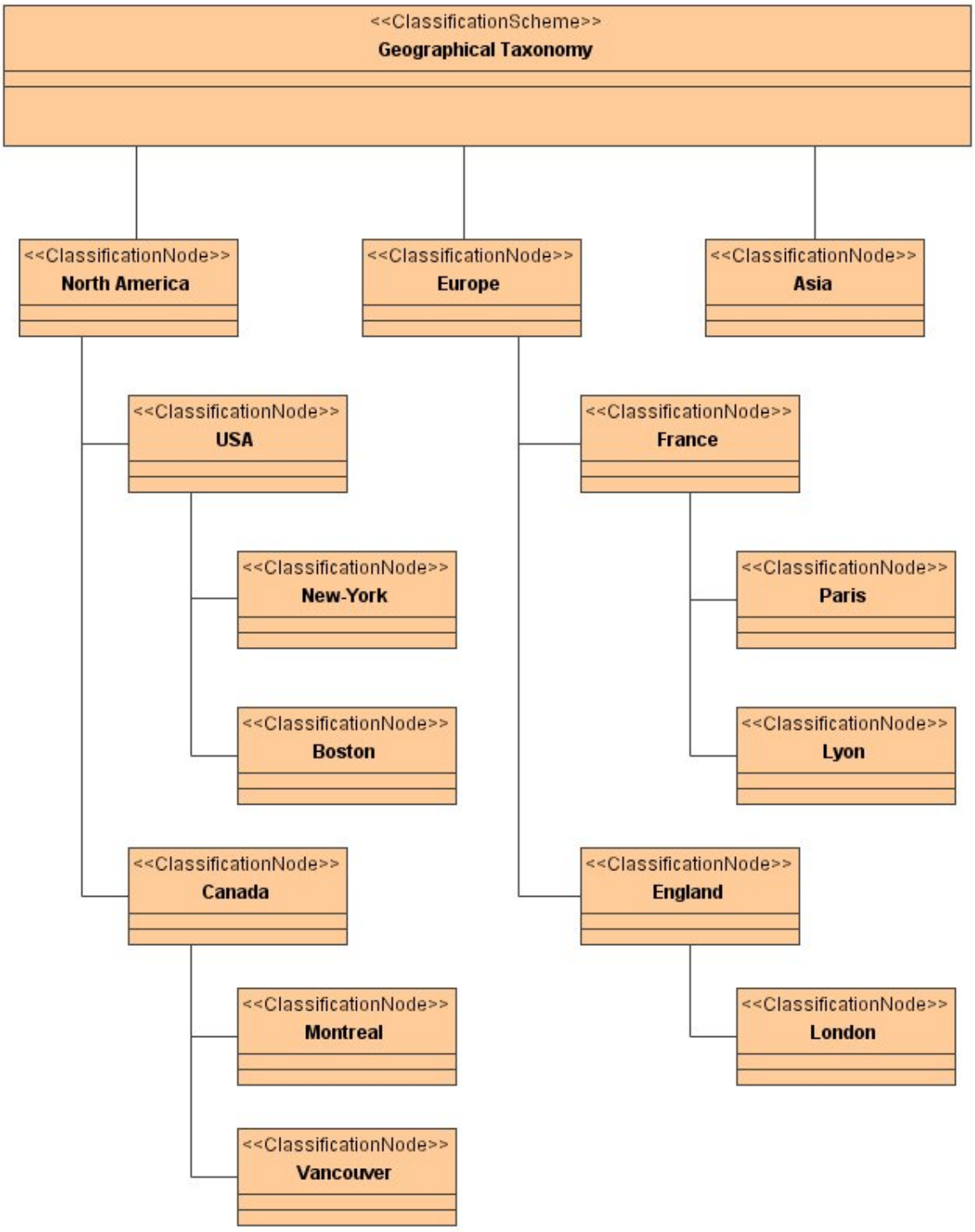
Association instances are RegistryEntries that are used to define associations between objects in the information model.

In the WFS profile, we are linking the service object and the extrinsicObject representing the FeatureType by an association. An Association has a type (represented by a Concept object), a source object and a target object. We see that an Association is directed. In our example, the Service is the source of the association and the feature type is the target.

5.3 Classifications

This is a more specific feature. First, we need to expore the notion of ClassificationNode and ClassificationScheme. These objects are making hierarchical trees of concepts, also known as Taxonomies.

For example, we can create a geographical ClassificationScheme, with concepts representing the mutliple continents, then multiple countries, and finally some cities. Here is a subset of such a ClassificationScheme :



You can note one root, of type ClassificationScheme, and the hierarchical sub-nodes, of type ClassificationNode (also known as Concepts). Once your classification shcemes are defined (or imported from existing taxonomies), you can classify your data in the ebRIM according to it. For instance, we could classify all the Organization defined in an ebRIM according to

multiple ClassificationSchemes: geographically, or by sector of activity, etc. This brings mutliple advantages. You could for instance browse your taxonomy, going down from the root to the node you're interested in, an only then ask for all data classified according to this node.

A classification is represented by a Classification object. Classification has two main attributes: the 'classifiedObject' and the 'classificationNode'.

5.4 Linked documents

ExtrinsicObjects are elements that can have a content, whose structure is not expressed in ebRIM. For instance, you may want to import a jpeg document in your repository, or attach some metadata described in XML within the repository.

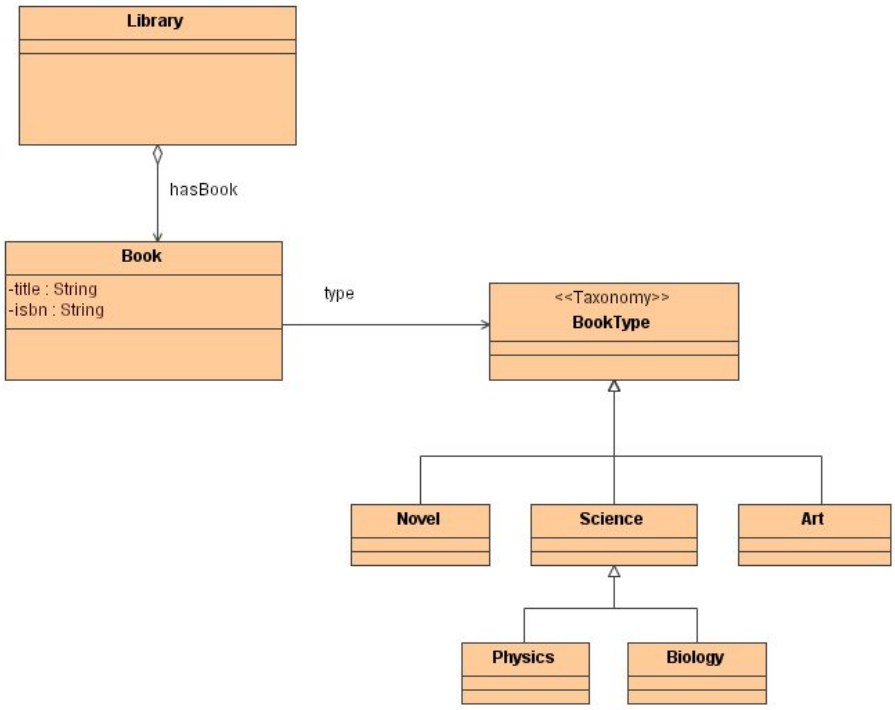
The main attributes of an ExtrinsicObject are a contentURI and a mimeType. For instance, the Ionic catalogue creates a thumbnail per layer in a WMS ebRIM description. It stores the a Jpeg in an extrinsicObject (given in the contentURI). This object is linked to the corresponding layer through an Association.

6 A methodology for describing resources in ebRIM

If we want to register a new kind of resource in the catalog, we need to define a 'profile' for this type of resource, i.e., to define its mapping within an ebRIM. Fortunately, it is quite straightforward to convert a UML schema into an ebRIM representation. So, the procedure to produce an ebRIM profile can be:

- describe the resource in UML
- translate the UML schema in an ebRIM profile
- if necessary, let the profile be reviewed by the community using the resource
- iterate over these steps until you reach the required consensus for standardization.

Let's now consider an example of a UML schema from which we want to infeer an ebRIM profile.



In this example, we represent a “Library” resource, storing books, each books having two string attributes, and possibly classified according to a given taxonomy BookType. Here is how to map some of the basic construct of UML:

- a class is usually described by an ExtrinsicObject, except if there is already the same object type defined in the ebRIM (for instance, Organization, ClasificationScheme or ClassificationNode). In our example, both Library and Book will become ExtrinsicObjects.

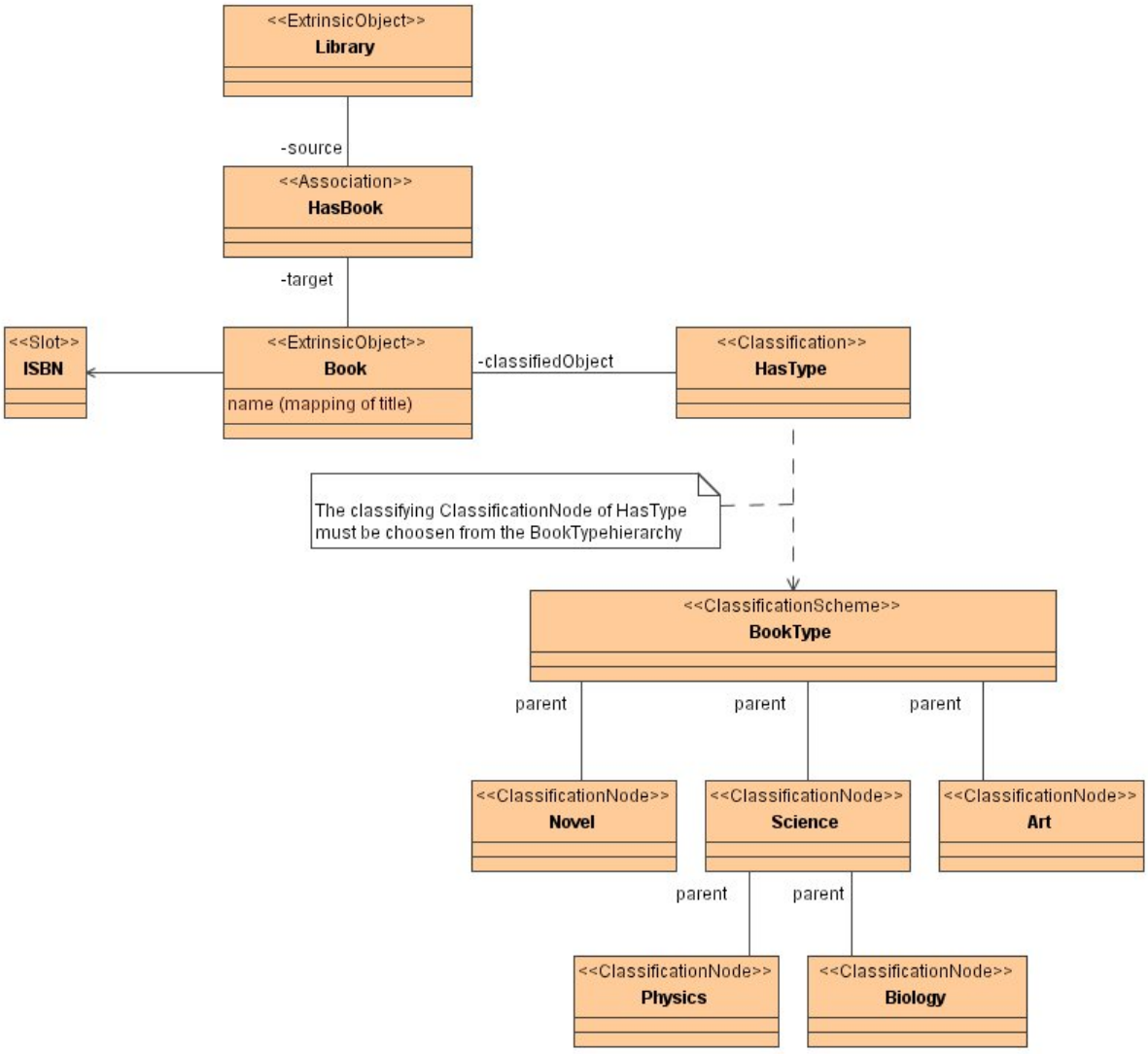
- a primitive attribute usually becomes a Slot attached to the relevant registry element. In our example, the ExtrinsicObject Book will accept one slot ISBN. For the title property, we prefer to use one of the predefined ebRIM attribute: name.

- An association is usually represented as an Association object in the ebRIM. The ebRIM Association has a source and a target ebRIM element, and so are directed (but can be seen as navigable both way all the time). In our example, the ‘hasBook’ association will become an ebRIM association, an instance of such an association being created per Book in the Library.

- We will give a separate treatment to the BookType hierarchy. It actually represents a Taxonomy, or ClassificationScheme, and has predefined constructs built in the ebRIM. So, the class BookType will become an ebRIM ClassificationScheme, when all descendant classes will become ebRIM Concepts.

- When a UML association represents a ClassificationNode classifying an object within a Taxonomy, we will represent this by an ebRIM Classification element. In our example, the 'type' association becomes an ebRIM Classification.

We can now represent our ebRIM profile. To do so, we propose to use the UML notation again, but completely dedicated to the ebRIM model, i.e., through the extensive use of the stereotype mechanism, every element will get its precise ebRIM type:



While still close to the original resource model, this schema completely and unambiguously describes how the resource can be represented in ebRIM.

7 Harvesting a resource

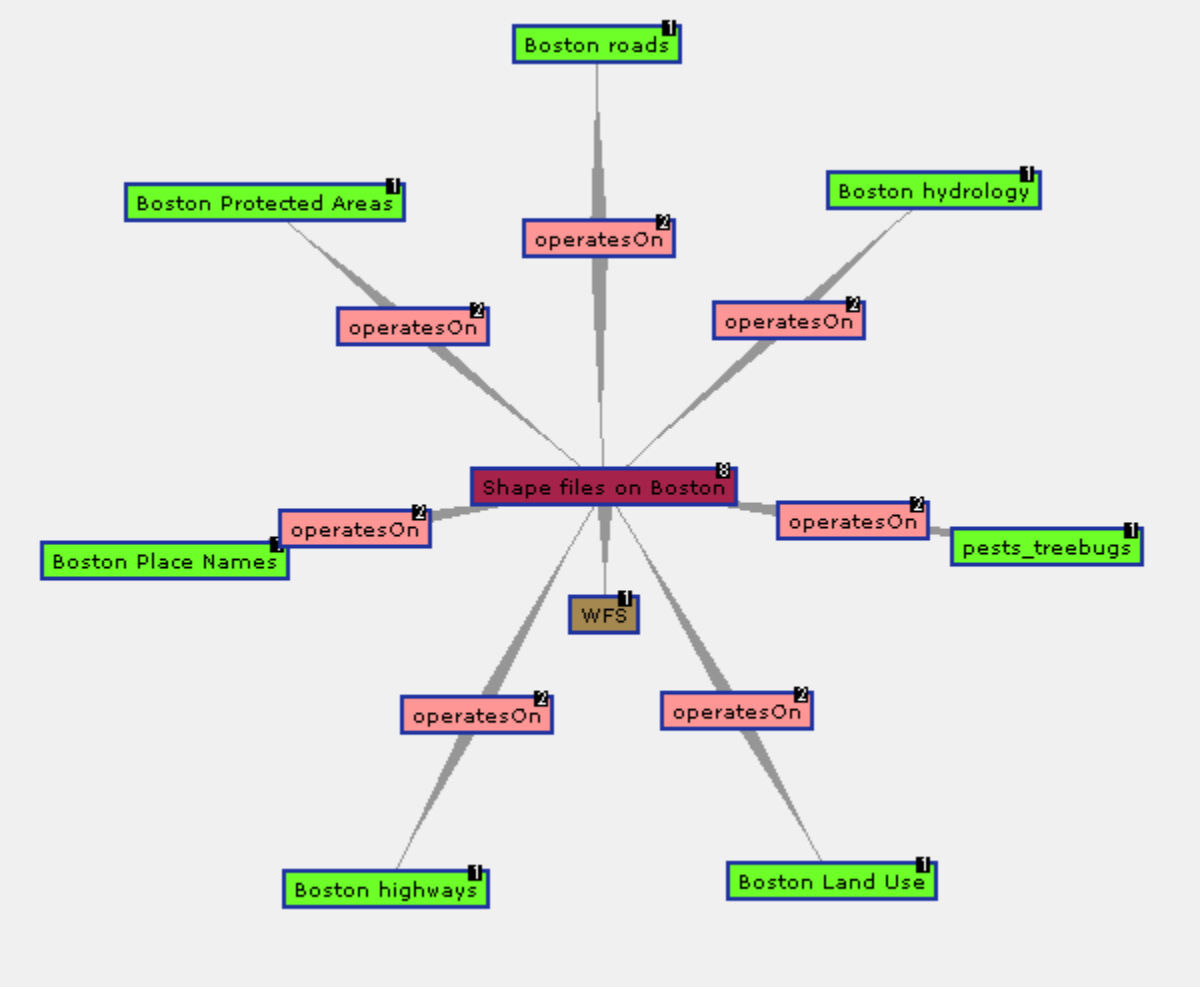
For a specific type of resource supported by a catalog, we call ‘harvesting’ the high level operation consisting of :

- accessing and parsing one instance of such a resource, usually distributed in multiple documents (capabilities, metadata associated to different layers of the resource, etc);
- importing this resource in the catalog repository, in a format compliant with a given profile);
- returning the set of ebRIM objects representing the resource in its profile.

With a request as simple as

```
http://localhost:8080/ionicwrs/wrs/WRS?
request=Harvest
&resourceType=WFS
&source= http://demo.ionicsoft.com/ionicweb/wfs/BOSTON_SHAPE
```

you will add the whole set of records in the registry representing the specified WFS within the given catalog, i.e., in this case, one service (red), some associations (pink), some wfs feature types (green), and one serviceBinding (brown). Also, all the feature types, represented by ExtrinsicObjects, will see



8 Discovering a resource

The act of discovering information is the other key functionality of the Catalogue. See the OGC Catalogue Specification for more information.