

Open GIS Consortium

35 Main Street, Suite 5
Wayland, MA 01778
Telephone: +1-508-655-5858
Facsimile: +1-508-655-2237

Editor:
Telephone: +1-703-830-6516
Facsimile: +1-703-830-7096
ckottman@opengis.org

The OpenGIS™ Abstract Specification **Topic 0: Abstract Specification Overview**

Version 4

OpenGIS™ Project Document Number 99-100r1.doc

Copyright © 1999, Open GIS Consortium, Inc.

NOTICE

The information contained in this document is subject to change without notice.

The material in this document details an Open GIS Consortium (OGC) specification in accordance with the license and notice set forth on this page. This document does not represent a commitment to implement any portion of this specification in any companies' products.

While the information in this publication is believed to be accurate, the Open GIS Consortium makes no warranty of any kind with regard to this material including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The Open GIS Consortium shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

The Open GIS Consortium is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks, or other special designations to indicate compliance with these materials.

This document contains information which is protected by copyright. All Rights Reserved. Except as otherwise provided herein, no part of this work may be reproduced or used in any form or by any means (graphic, electronic, or mechanical including photocopying, recording, taping, or information storage and retrieval systems) without the permission of the copyright owner. All copies of this document must include the copyright and other information contained on this page.

The copyright owner grants member companies of the OGC permission to make a limited number of copies of this document (up to fifty copies) for their internal use as a part of the OGC Technology Development process.

Revision History

Date	Description
15 March 1999	Carried over previous version 98-100r1 and updated for 1999; add new Section 3.2, per change proposal 99-010 (w/ friendly amendments), adopted February 9, 1999; update relationships of Topic Volumes to WGs and SIGs in Section 2.4; update for new Topic Volumes 10, 15 and 16; updated description of the Abstract Specification, Section 2.2 and changed use of terms “Abstract Specification” and “Abstract Model” elsewhere to be consistent; added Sections 3.3, 3.4 & 4.1 from change proposal 98-072, approved February 9, 1999;
23 June 1999	Create new Section 4 (Shared UML Packages) and populate with content from: Appendix B (Unit of Measure) and C (Basic Data Types) of 99-102r1, Accuracy Package from change proposal 99-038r1 (approved at April 1999 TC), and former Section 3.6 (Programming Language) from this document. Minor updates to Sections 2.4, 2.5.1 and 3.1.6.

This page is intentionally left blank.

Table of Contents

1. Introduction.....	1
1.1. The Open GIS Consortium.....	1
1.1.1. Overview.....	1
1.1.2. Vision	1
1.1.3. Specifications.....	1
1.2. The Open GIS Consortium Process.....	1
1.2.1. The Consensus Process	1
1.3. References for Section 1.....	1
2. The Abstract Specification.....	2
2.1. Background.....	2
2.1.1. The OGC Bookshelves.....	2
2.1.2. Core and Domain Bookshelves	2
2.2. Purpose	2
2.3. Topic Volumes	3
2.4. Topics and Their Proponents	3
2.5. Relationships Between the Topic Volumes.....	4
2.5.1. Central Themes: Sharing Information and Providing Services	4
2.5.2. Providing Geospatial Services.....	4
2.5.3. Sharing Geospatial Information	4
2.5.4. Prerequisites for Sharing Geospatial Information.....	4
2.5.5. Miscellaneous Topics	5
2.6. References for Section 2.....	5
3. Guidelines and Conventions	6
3.1. The Structure of Individual Topic Volumes	6
3.1.1. Introduction to the Topic.....	6
3.1.2. The Essential Model of the Topic	6
3.1.3. The Abstract Model of the Topic.....	6
3.1.4. Future Work	6
3.1.5. Glossary and Acronyms Supporting the Topic	6
3.1.6. Well Known Structures Supporting the Topic.....	6
3.2. The Use of Object Modeling Notation	7
3.3. Presentation and Notation	7
3.4. Packages and Dependencies	10
3.5. References	10
4. Shared UML Packages	11
4.1. Programming Language	11
4.1.1. Set Theory and Collection Interfaces.....	11
4.1.2. Set and Bag.....	11
4.1.3. Sequence	12
4.1.4. Dictionary	13
4.1.5. Basic Language Types.....	14
4.2. Basic Data Types	15

4.2.1. Class name: Real	17
4.2.2. Class name: Float.....	17
4.2.3. Class name: Double.....	17
4.2.4. Class name: int32	17
4.2.5. Class name: uint32.....	17
4.2.6. Class name: Byte	18
4.2.7. Class name: Boolean.....	18
4.2.8. Class name: CharacterString.....	18
4.2.9. Class name: Vector.....	18
4.2.10. Class name: Name.....	19
4.3. Unit of Measure Package	20
4.3.1. Class name: Unit Of Measure.....	20
4.3.2. Class name: UomScale.....	21
4.3.3. Class name: UomLength.....	21
4.3.4. Class name: UomAngle.....	21
4.3.5. Class name: UomTime	21
4.3.6. Class name: UomArea.....	21
4.3.7. Class name: UomVelocity	22
4.3.8. Class name: Measure	22
4.3.9. Class name: Length.....	22
4.3.10. Class name: Angle.....	23
4.3.11. Class name: Area.....	23
4.3.12. Class name: Velocity	23
4.3.13. Class name: Time	24
4.3.14. Class name: Scale.....	24
4.4. Accuracy Package.....	24
4.4.1. Class name: AccuracyMeasure.....	25
4.4.2. Class name: CovarianceMatrix.....	26
4.4.3. Class name: CovarianceElement	27
4.4.4. Class name: ErrorEstimate	27
4.4.5. Class name: SphericalError.....	27
4.4.6. Class name: CircularError	28
4.4.7. Class name: LinearError	28
4.4.8. Class name: CompoundAccuracy	28
4.4.9. Class name: CEandLE	29
4.4.10. Class name: ThreeLE.....	29
4.4.11. Class name: TwoLE	29
5. Future Work.....	30

1. Introduction

1.1. The Open GIS Consortium

1.1.1. Overview

The Open GIS Consortium is a not-for-profit organization dedicated to open systems geoprocessing. For more information on the OGC and its mission and activities see our World Wide Web site at <<http://www.opengis.org>>. In this section we very briefly introduce the vision of the OGC and to its major technical body, the OGC Technical Committee.

1.1.2. Vision

OGC envisions the full integration of geospatial data and geoprocessing resources into mainstream computing and the widespread use of interoperable geoprocessing software and geodata products throughout the information infrastructure.

1.1.3. Specifications

The OGC Technology Development Process [1] creates two types of specification products: Abstract and Implementation specifications. The purpose of the Abstract Specification is to create and document a conceptual model sufficient enough to allow for the creation of Implementation Specifications. Implementation Specifications are unambiguous technology platform specifications for implementation of industry-standard, software application programming interfaces.

1.2. The Open GIS Consortium Process

1.2.1. The Consensus Process

The Open GIS Consortium uses a process of consensus gathering among its membership in order to achieve specifications for software components related to Geographic Information Systems and Geoprocessing.

Through the activities of Special Interest Groups and Working Groups, members of the Consortium develop Abstract Specifications. These are brought before the Technical Committee, where consensus is achieved.

When individual topics of the Abstract Specification are sufficiently mature, the OGC membership may issue a Request for Proposal (RFP) for Implementation Specifications. Responses to such a proposal provide implementation-level specifications in a platform-neutral fashion.

Upon consensus acceptance by the OGC membership, the Implementation Specifications become a part of the OGC suite of specifications. Software that is nominated by members is checked for conformance to the Implementation Specification, and if found conforming, is distinguished by an OGC trademark.

For detailed information on the OGC Technology Development Process refer to OGC documents [1], [2] and [3]. The OpenGIS™ Guide [4] provides a more detailed overview of the OGC vision, process and scope of technology.

1.3. References for Section 1

- [1] Open GIS Consortium, 1997. The OGC Technical Committee Technology Development Process, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/development.htm>>.
- [2] Open GIS Consortium, 1997. OGC Technical Committee Policies and Procedures, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/development.htm>>.
- [3] Open GIS Consortium, 1997. The OGC Technical Committee Request Plan & Schedule, Wayland, Massachusetts. Available via the WWW as <<http://opengis.org/techno/development.htm>>.
- [4] Open GIS Consortium, 1997. The OpenGIS™ Guide: Introduction to Interoperable Geoprocessing, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/guide.htm>>.

2. The Abstract Specification

2.1. Background

2.1.1. The OGC Bookshelves

The Open GIS Consortium maintains its consensus in a number of “Bookshelves.” This document (Topic 0) is an overview of the OpenGIS™ Abstract Specification Bookshelf . This bookshelf contains the “Topic” volumes that, together, form the OpenGIS™ Abstract Specification [2].

Additional bookshelves include, or will include: The Implementation Specification Bookshelf (holding the implementation specifications that arise from the RFP consensus process), and other collections of standards that may originate in the Object Management Group (OMG), the International Organization for Standards (ISO), the Federal Geographic Data Committee (FGDC), and others, on the basis of the consensus of the OGC membership.

2.1.2. Core and Domain Bookshelves

Bookshelves are called “Core Technology” if they pertain broadly to many GIS disciplines. All of the current Topics are intended to be Core.

Bookshelves are envisioned that will be specific to particular disciplines or domains. These will be called “Domain Technology” generically, and will identify their intended specific domain. In this way, the OGC will perhaps publish a “Electrical Conductor Network” Topic volume.

2.2. Purpose

The purpose of the Abstract Specification is to create and document a conceptual model sufficient enough to allow for the creation of Implementation Specifications. The Abstract Specification consists of two models derived from the Syntropy object analysis and design methodology [2].

The first and simpler model is called the Essential Model and its purpose is to establish the conceptual linkage of the software or system design to the real world. The Essential Model is a description of how the world works (or should work).

The second model, the meat of the Abstract Specification, is the Abstract Model that defines the eventual software system in an implementation neutral manner. The Abstract Model is a description of how software should work. The Abstract Model represents a compromise between the paradigms of the intended target implementation environments.

The purpose of the Abstract Specification is:

- To relate software and system design to real world situations.
- To capture and precisely state requirements and domain knowledge so that all stakeholders may understand and agree on them.
- To think about the design of the system.
- To capture design decisions in a mutable form separate from the requirements.
- To generate usable work products (such as prototypes and proof of concept implementations).
- To organize, find, filter, retrieve, examine and edit information about large systems.
- To explore multiple solutions economically.
- To master complexity.

The Abstract Specification, and specifically the Abstract Model, is used in all these capacities. Additionally, it provides an implementation neutral, but technically complete “language” to discuss issues of interoperability.

2.3. Topic Volumes

There are currently 17 “books” in the OGC Abstract Specification (including this one, Topic 0). Each is called a “Topic,” and given a descriptive name. The Abstract Specification refers to the collection of Topic Volume documents referenced by [2].

The Abstract Specification is broken into Topics in order to assist parallel developments of different topics by different Working Groups of the OGC membership. Figure 1 shows some dependency relationships between topics.

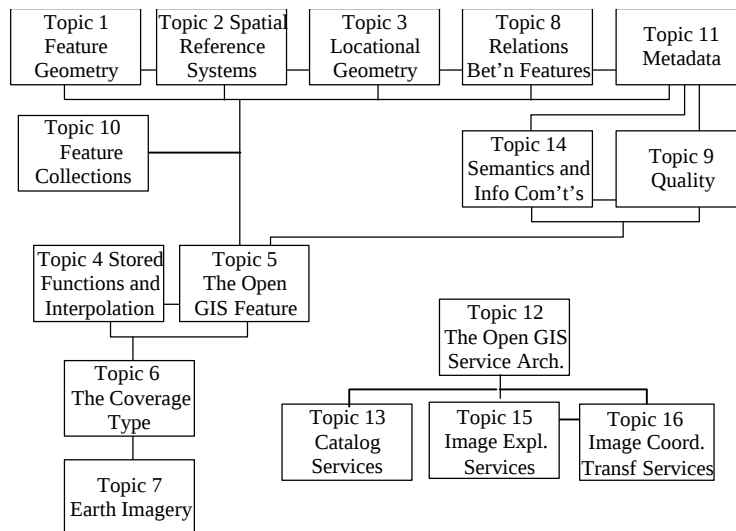


Figure 1: Abstract Specification Topic Dependencies

The Abstract Specification is organized into separate topic volumes in order to manage the complexity of the subject matter and to assist parallel development of work items by different Working Groups of the OGC Technical Committee. The topics are, in reality, dependent upon one another—each one begging to be written first. *Each topic must be read in the context of the entire Abstract Specification.*

The topic volumes are not all written at the same level of detail. Some are mature, and are the basis for Requests For Proposal (RFP). Others are immature, and require additional specification before RFPs can be issued. The level of maturity of a topic reflects the level of understanding and discussion occurring within the Technical Committee. Refer to the OGC Technical Committee Policies and Procedures [3] and Technology Development Process [4] documents for more information on the OGC OpenGIS™ standards development process.

2.4. Topics and Their Proponents

The 17 Topics and the Work Groups (WGs) or Special Interest Groups (SIGs) primarily responsible for them are:

Topic Volume	Volume Name	Work Group or Special Interest Group
Topic 0	Abstract Specification Overview	Core Task Force Editor
Topic 1	Feature Geometry	Geometry WG
Topic 2	Spatial Reference Systems	Coordinate Transformation WG
Topic 3	Locational Geometry Structures	Coordinate Transformation WG
Topic 4	Stored Functions and Interpolation	Coverages WG
Topic 5	The OpenGIS™ Feature	Feature SIG
Topic 6	The Coverage Type	Coverages WG
Topic 7	Earth Imagery Case	Coverages WG and Image Exploitation Services SIG
Topic 8	Relationships Between Features	Feature Identity and Relationships WG and Feature SIG
Topic 9	Quality	Metadata SIG

Topic 10	Feature Collections	Feature SIG
Topic 11	Metadata	Metadata WG
Topic 12	The OpenGIS™ Service Architecture	The Services Architecture SIG
Topic 13	Catalog Services	Catalog WG
Topic 14	Semantics and Information Communities	Core Task Force
Topic 15	Image Exploitation Services	Image Exploitation Services SIG
Topic 16	Image Coordinate Transformation Services	Image Exploitation Services SIG and Coordinate Transformation WG

Additional Topic volumes are expected to emerge as follows:

Topic Volume	Topic Name	Work Group or Special Interest Group
Tbd	Tbd	Telecommunications SIG
Tbd	Tbd	WWW Mapping SIG
Tbd	Tbd	Transportation SIG

Additional topics will be added as the scope of the OGC widens, at the consensus of the OGC membership. Already, as the last two rows to the list above indicates, there is movement toward Topics centered on the needs of the Telecommunications and Transportation industries.

2.5. Relationships Between the Topic Volumes

2.5.1. Central Themes: Sharing Information and Providing Services

The OGC has two central technology themes: Sharing Geospatial Information, and Providing Geospatial Services.

Topics 12,13, 15 and 16 are concerned with providing geospatial services. The remainder are centered on sharing geospatial information.

2.5.2. Providing Geospatial Services

Topic 12 specifies a comprehensive set of geospatial services. Topics 13, 15 and 16 describe, in greater detail, specific categories of geospatial service.

A great many additional services are foreseen. See Section 4 for a list of suggested services that need open interfaces for interoperability.

2.5.3. Sharing Geospatial Information

Topics 5, 6, and 7 are centered on sharing geospatial information. These Topics are the OpenGIS™ Feature, the Coverage Type, and Earth Imagery. The Coverage is a special case of an OpenGIS™ Feature, and an Earth Image is a special case of a Coverage. These special cases are broken out into separate Topics because they are large subjects, and have their own OGC Working Groups.

Each of these Topics (5,6 and 7) is fundamentally concerned with the handling and exposing of Geospatial information, whether it is modeled using Features With Geometry, Coverages, or with the Imagery information type.

2.5.4. Prerequisites for Sharing Geospatial Information

Topics 1, 2, 3, 8, and 11 directly support Topics 5, 6, and 7. Topic 1 provides the Geometry structures for Features With Geometry. Topics 2 and 3 deal with the same subject (geolocation), but in two different technology domains. The general subject is relating model coordinates to Earth coordinates. Topic 2 provides the Spatial Reference Systems by which features are related to positions on the Earth in the discipline of Geodesy. Topic 3 adds tools for providing geospatial referencing to image coordinates, raster coordinates, and indirect referencing systems that are not found in Geodesy texts. Topic 8 provides for the modeling and exposure of relationships between features. Topic 11 provides for the modeling and query of metadata. Each of these Topics is essential before any geospatial information may be shared.

Topic 4, on Stored Functions, is necessary to support Topic 6, Coverages. Most Coverages depend on two stored functions. The functions map respectively “to” and “from” a mathematical coordinate space called the Coverage Extent, or a Spatial Domain. The first function, the Coverage Generation Function, maps from Earth coordinates to Coverage Extent Coordinates, and provides

geolocation. The second, the Schema Mapping, maps from the Coverage Extent to some range of values. In general, both functions might be stored.

2.5.5. Miscellaneous Topics

The remainder of the Topics fall into the “Miscellaneous” category. These are:

Topic 10	Transfer Technology
Topic 9	Quality
Topic 14	Semantics and Information Communities

All three of these Topics have strong information theoretic content that is not strongly tied to geospatial issues. Perhaps, in the long run, this means that these Topics will be endowed with interoperable interfaces by authorities other than the OGC. In the immediate future, however, these Topics must be enabled with interoperating technology by OGC, as they are necessary for the understanding of and access to geospatial information.

2.6. References for Section 2

- [1] Open GIS Consortium, 1999. The OpenGIS™ Abstract Specification, OpenGIS Project Documents 99-100 through 99-116, available through WWW as <<http://www.opengis.org/techno/specs.htm>>.
- [2] Open GIS Consortium, 1997. The OGC Technical Committee Technology Development Process, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/development.htm>>.
- [3] Open GIS Consortium, 1997. OGC Technical Committee Policies and Procedures, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/development.htm>>.
- [4] Cook, Steve, and John Daniels, Designing Objects Systems: Object-Oriented Modeling with Syntropy, Prentice Hall, New York, 1994, xx + 389 pp.
- [5] Open GIS Consortium, 1999. Topic 0, Abstract Specification Overview, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/specs.htm>>.

3. Guidelines and Conventions

3.1. The Structure of Individual Topic Volumes

Each Topic volume ordinarily contains six sections:

Section 1	Introduction to the Topic
Section 2	The Essential Model of the Topic
Section 3	The Abstract Model of the Topic
Section 4	Future Work
Appendix A	Glossary and Acronyms supporting the Topic
Appendix B	Well Known Structures supporting the Topic

Each section may conclude with a “References” subsection in order to simplify references to other Topics, and to other sources of technology.

3.1.1. *Introduction to the Topic*

Each Topic document should have an Introduction to the subject. The Introduction has a brief description of the purpose and scope of the Abstract Specification. The Introduction also states the subject and scope of the topic volume. The Introduction should also provide a history and status of work on the topic. The Introduction should answer questions like: Has the Topic been submitted for an RFP? Has an implementation specification been written?

The Introduction of each Topic provides space to describe how each Topic is related to other Topics and to other standards. The Introduction should also reference the central documents of the Consortium and, at a minimum, the Abstract Specification Overview [1].

3.1.2. *The Essential Model of the Topic*

The Essential Model, should explain in real world terms the objects, interfaces, behaviors, and parameters that are the subject of the Topic. Diagrams, use-case analysis, structured graphical or lexical models are welcome, if they assist understanding of the central message of the Topic.

3.1.3. *The Abstract Model of the Topic*

Section 3, the Abstract Model, is the heart of each Topic document. Structured graphical or lexical languages for the expression of the conceptual schema are encouraged.

Typically, the Abstract Model identifies the classes and subclasses of interest, identifies and describes their relationships, and abstractly specifies the interfaces that are to be implemented in software.

At the most fundamental layer of information sharing, well known structures (WKS) are necessary. To make the WKS easily available to system developers, they are not ordinarily placed in the Abstract Model, but rather in an Appendix of the appropriate Topic volume.

The Abstract Model section of each Topic should provide references to foundation information for any advanced technology introduced.

3.1.4. *Future Work*

The OpenGIS™ Specifications will never be completely finished. This is because Geospatial Information, like other information, is constantly growing, changing, and being integrated into new environments. Moreover, the Geospatial marketplace is expected to be a dynamic source of new requirements. Therefore, each Topic volume will have a Section 4 titled “Future Work.”

Where it is possible, each Topic should identify the most important “next” developments that should be undertaken by the Working Group responsible for the Topic. This list of needs and future requirements constitutes the Future Work section.

3.1.5. *Glossary and Acronyms Supporting the Topic*

Terminology and acronyms specific to the Topic Volume should be defined in an Appendix.

3.1.6. *Well Known Structures Supporting the Topic*

Well Known Structures are to be expressed using Unified Modeling Language (UML) and, optionally, in Interface Definition Language (IDL) and placed in an Appendix of the appropriate Topic volume.

Where WKS are clearly needed, but not yet formally expressed, a list identifying and describing such WKS shall be placed in an Appendix along with the formally expressed WKS, if any. The Future Work section may reference this list.

3.2. The Use of Object Modeling Notation

The Unified Modeling Language (UML) shall be used as the notation for all object oriented models included in OGC Abstract Specification and Implementation Specification documents. Specifically UML provides the notation for:

- Use-Case Models
- Class Diagrams
- Object Diagrams
- State Diagrams
- Sequence Diagrams
- Collaboration Diagrams
- Activity Diagrams
- Component Diagrams
- Deployment Diagrams

UML is defined in a series of documents from the Object Management Group, specifically [2], [3], [4], [5], [6], [7] and [8].

3.3. Presentation and Notation

In the Abstract Specification, conceptual schemas are presented in the Universal Modeling Language (UML).

UML supports the definition of interfaces to objects as attributes operations, or associations. The fact that an interface has been modeled in one way or another in this standard should not be considered a restriction on implementations. Attributes may be implemented either directly as data or as a pair of “accessor” and “mutator” operations for getting and setting values. Most diagrams in the Abstract Specification are “context diagrams” which center about a single class and display its attributes, operations, and important relationships. UML does not require all relationships to be displayed in all diagrams, and some of the more trivial ones have been left out of some diagrams to keep them simple.

Attributes and operations are presented in the UML diagrams in compliance with the UML Notation Guide. UML notation for an attribute has the form:

`<<stereotype>> visibility name : type = initial {property, ...}`

UML notation for an operation has the form:

`<<stereotype>> visibility name (parameter : type, ...): return-type`

where the various parts of the above syntax are as follows:

- `<<stereotype>>` — use tag for the attribute or operation being defined (see below)
- visibility — public (+), private (-) or protected (#) indicating the visibility of this attribute or operation from outside the object, since this is an interface standard, we define all attributes and operations as public. If the visibility includes “/”, then the attribute is derived from some other part of the model.
- name — the name of the attribute or operation

- parameter — name of a parameter to the operation, usually indicative of the role of the parameter in the operation being defined. Note that the syntax structures for an operation and for an attribute are identical except that an operation includes a parameter list.
- type — the type, either object or value of the preceding parameter or attribute
- return-type — the type of the return value or object for the operation, essentially the type of the operation.
- property — additional information about the attribute or operation, such as NOT NULL or UNIQUE, not used in this standard
- ... — the preceding can be repeated any number of times
- initial — default value of the attribute, used when a new object is constructed unless specifically overridden by the constructor's parameter list.

In the text, notation from the Object Constraint Language (OCL) is used with some slight modifications. The "::" is a resolution operator indicating the name space of that which follows. In most cases in OCL, the name space is the class in which the operation is defined, but it can also include the package name in which a class is defined. In this document all name spaces are class identifier and can take only one of two forms:

```
type ::= class-name | package-name::class-name
```

Unless there is a potential of confusion or a need for emphasis, the package name is not included. In this standard, all class names have followed a standard naming convention (a two letter package identifier followed by an underscore "_") and are unique within the model. This avoids the need for package names resolution in type and class names for all classes unique to this model. Profiles of this part are encouraged to retain this convention if possible. For attributes and operations, the text description is as follows:

```
attribute-name : attribute-type
type-1::function-name(name-2 : type-2, name-3 : type-3, ... ) :
return-type
```

Object-oriented operator notation (such as would be used in C++) places the first parameter before the operation as in a method declaration as follows:

```
return-type type-1::operation(type-2, type-3 ... )
```

Such methods are restricted in name space, in the sense they are available only if the "object-type-1" name space is available. In addition, during invocation, the identity of the implicit parameter of type "type-1" is known. In OCL, this object is identified as "self." In C++, this object is identified as "this." In non-object languages or for free functions in an object language, the functional notation for an operation does not distinguish the first parameter in any manner and is written:

```
return-type operation(name-1 : type-1, name-2 : type-2, name-3 :
type-3 ...)
```

These notations are equivalent (except for emphasis) and both can be used in profiles of this standard.

These operation definitions are called "operation signatures" or "protocols." This distinguishes the operation from the invocation mechanism. In UML, the formal notation defines protocols, and the operations associated to them are defined only informally in the associated documentation, which can include OCL constraints.

In the view that attribute can be considered a type of operation (mutator and accessor pairs), this term can be extended to include attribute "signatures." The definition of a signature includes the operation name; the parameter names and types; and the return type. Methods or attributes can be overridden by providing a new method whose signature is the same as the original except that some of the types have been replaced, usually by subtypes of the originals. The reuse of signatures is called "polymorphism." Polymorphism arising from class inheritance is called "structural." Polymorphism arising from semantic similarity is called "natural" or "generic." For example, in the

geometry and topology classes, the common protocol for “boundary” is a natural polymorphism in that it arises from an operational constraint based on the definition of topology. It is not a structural polymorphism, since the two packages do not share a common superclass ancestor. Assuming that the class inheritance hierarchy is based on semantics of the objects, then structural polymorphism is natural. Polymorphism that does not depend on semantic similarity is “ad-hoc.” For example, the use of “+” in numbers to denote addition and in character string classes to denote concatenation is ad-hoc polymorphism. Ad-hoc polymorphism is semantically confusing, and is therefore not used in this standard, and should be avoided in profiles of this standard.

Most operations are defined in a functional style, that is all parameters are passed as read-only, and the only modification or creation of objects is done by using the return type in an assignment operator. In describing interfaces, the adjective “this” will indicate the entity whose object interfaces are being invoked. If an object is passed as a parameter to the method of another object, it will be referred to as a “passed” object. In OCL, this object is referred to as “self.”

Each association in the model is given an association name, and each class that participates in the association is given a role name. In the case where “Class1” has an association “Relation” with

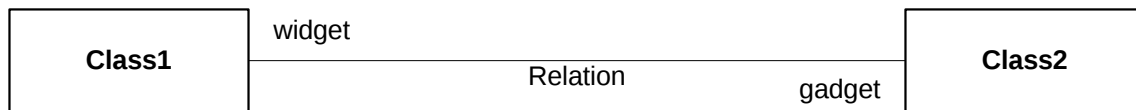


Figure 3-1. UML Example Association

Then the two classes when implemented would normally include as “attributes” references to the other class named for the roles of the relation such as:

```

Class1::gadget : Set(Class2) :    // this is the many to many case
Class2::widget : Set(Class1)
  
```

Note that the role name for Class2 is used as the variable name in Class1 that points to Class2. This notation will be used where appropriate in the text below, but this is not meant to imply a particular implementation of associations.

Most entities in a UML model can be described by a “stereotype” which is included near the name of the object and enclosed in guillemets “<<” and “>>”. The stereotype allows the model to extend UML to include descriptions of elements of the model.

Several collection types are required to make the standard consistent, but these types do not have to be specific in terms of their interfaces. The most common of these interfaces is the finite set. If we have a type “T,” we denote a new parameterized type called “Set(T)” to consist of all finite, unordered set of entities of type “T.” Implementation environments often supply several common collection types such as arrays, and we do not wish to try to impose a universal interface on these types. Section 3.4 includes an example interface definition for these types. This standard does not restrict the use of logically equivalent types native to particular implementation environments. These classes include the following:

- Set(T) — a possibly infinite set; restricted only to values (see definition).
- Set(T) — a finite set, usable for object types.
- Sequence(T) — an ordered finite set of objects, possibly with repeated values.
- Number, float, integer, real — various simple value types usually instantiated as programming language primitives within the environment.
- Length, area, distance — various scalar values, usually associated to a particular unit of measure such as the meter or acre.

Several of the operations defined in the Abstract Specification use NULL and EMPTY as possible values. NULL means that the value asked for is undefined. This standard assumes that all NULL values are equivalent. If a NULL is returned when an object has been requested, then the assumption is that no object matching the criteria specified exists. EMPTY refers to objects that can be interpreted as sets of one form or another, and means that the set in question contains no elements. Unlike programming systems that have strongly typed aggregates, this standard uses the mathematical tautology that there is one and only one empty set and that any object representing that empty set is equivalent to any other set doing the same. Other than being empty, such sets lack any inherent information, and so a NULL value is assumed equivalent to an EMPTY set in appropriate context.

3.4. Packages and Dependencies

The Abstract Specification Topic Volumes are organized in terms of UML packages. A package is a set of related types and interfaces that form a consistent component of a software system design. Packages do not usually form a complete system since they often invoke the services provided by other packages in the system. When one package, acting as a client, uses another, acting as a server, to supply needed services, the client package is said to be **dependent** on the server package. This is indicated in package diagrams by:



Figure 3-2. UML Example Package Dependency

Because of this client-server relation, inter-package dependencies define the criterion for viable application schemas. An application schema that contains any package defined from the Abstract Specification shall also contain all of its dependencies.

3.5. References

- [1] Open GIS Consortium, 1999. Topic 0, Abstract Specification Overview, Wayland, Massachusetts. Available via the WWW as <<http://www.opengis.org/techno/specs.htm>>.
- [2] OMG Document ad/98-08-03, UML Summary version 1.1, 1 September 1997
- [3] OMG Document ad/98-08-04, UML Semantics version 1.1, 1 September 1997
- [4] OMG Document ad/98-08-05, UML Notation Guide version 1.1, 1 September 1997
- [5] OMG Document ad/98-08-08, Object Constraint Language Specification version 1.1, 1 September 1997
- [6] OMG Document ad/97-08-09, OA&D CORBAfacility Interface Definition version 1.1, 1 September 1997
- [7] OMG Document ad/98-08-07, UML Extension for Business Modeling version 1.1, 1 September 1997
- [8] OMG Document ad/98-08-06, UML Extension for Objectory Process for Software Engineering version 1.1, 1 September 1997

4. Shared UML Packages

This section specifies a set of UML packages that are, or will be, used by (or in) the abstract object models of multiple Topic volumes. These shared UML packages define data type classes for data structures and elements that are used in multiple places within the OGC specifications. These UML packages do not define services, beyond any services that may be needed to retrieve or change specific data values.

4.1. Programming Language

Most of this specification for programming language has been made consistent with the UML Object Constraint Language (OCL).

4.1.1. Set Theory and Collection Interfaces

The major difference between this language specification and OCL is the inclusion of the TransfiniteSet. Protocols that do not depend upon the explicit instantiation of the membership of a set have been moved to this superclass level to define the behavior of sets of possibly infinite size. Instantiations of this sort of set would have to be done implicitly, via a Boolean test, as opposed to an explicit enumeration of members.

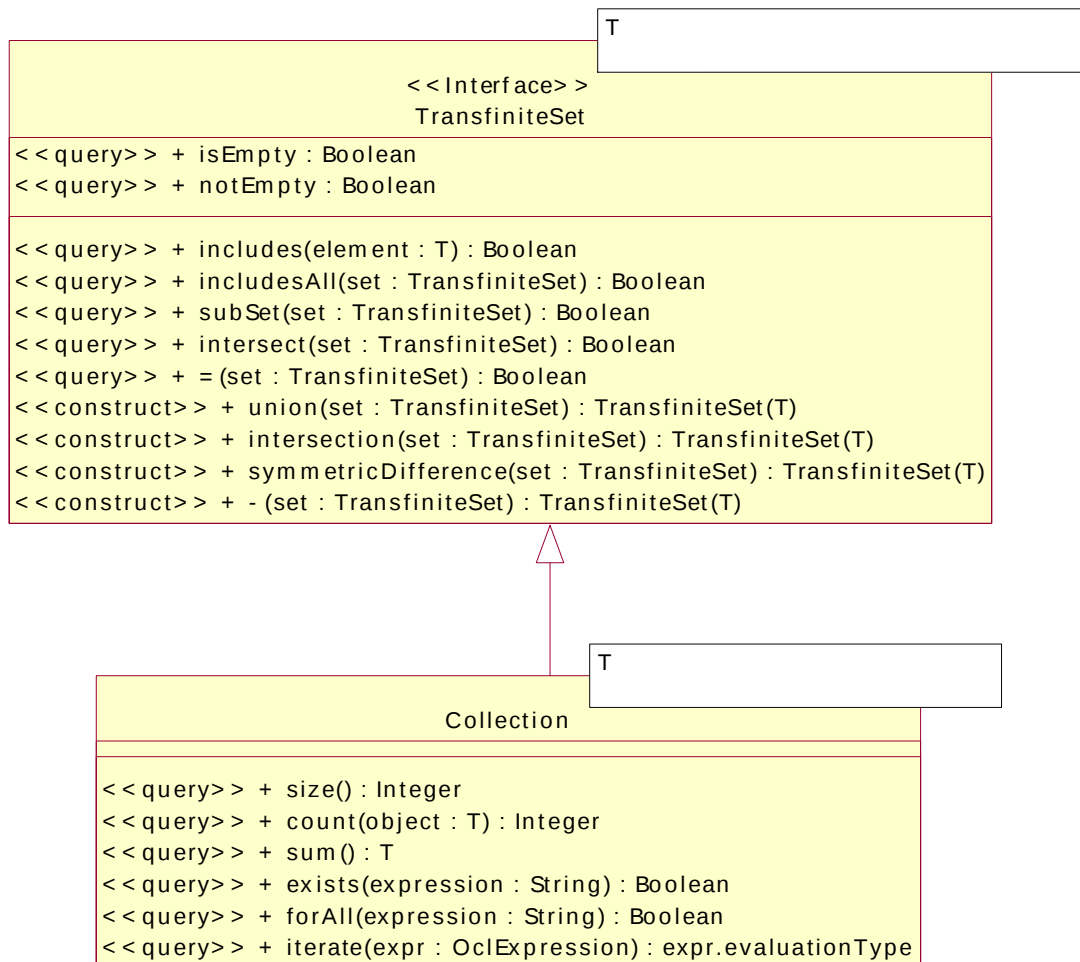


Figure 4-1: Set Theory and General Collection Interfaces

4.1.2. Set and Bag

A Set is a finite set of objects, where each object is considered to be in the collection once. A Bag is similar, but, an instance count of its members allows them to be in the bag more than once. Bags are most often implemented through the use of proxies or reference pointers.

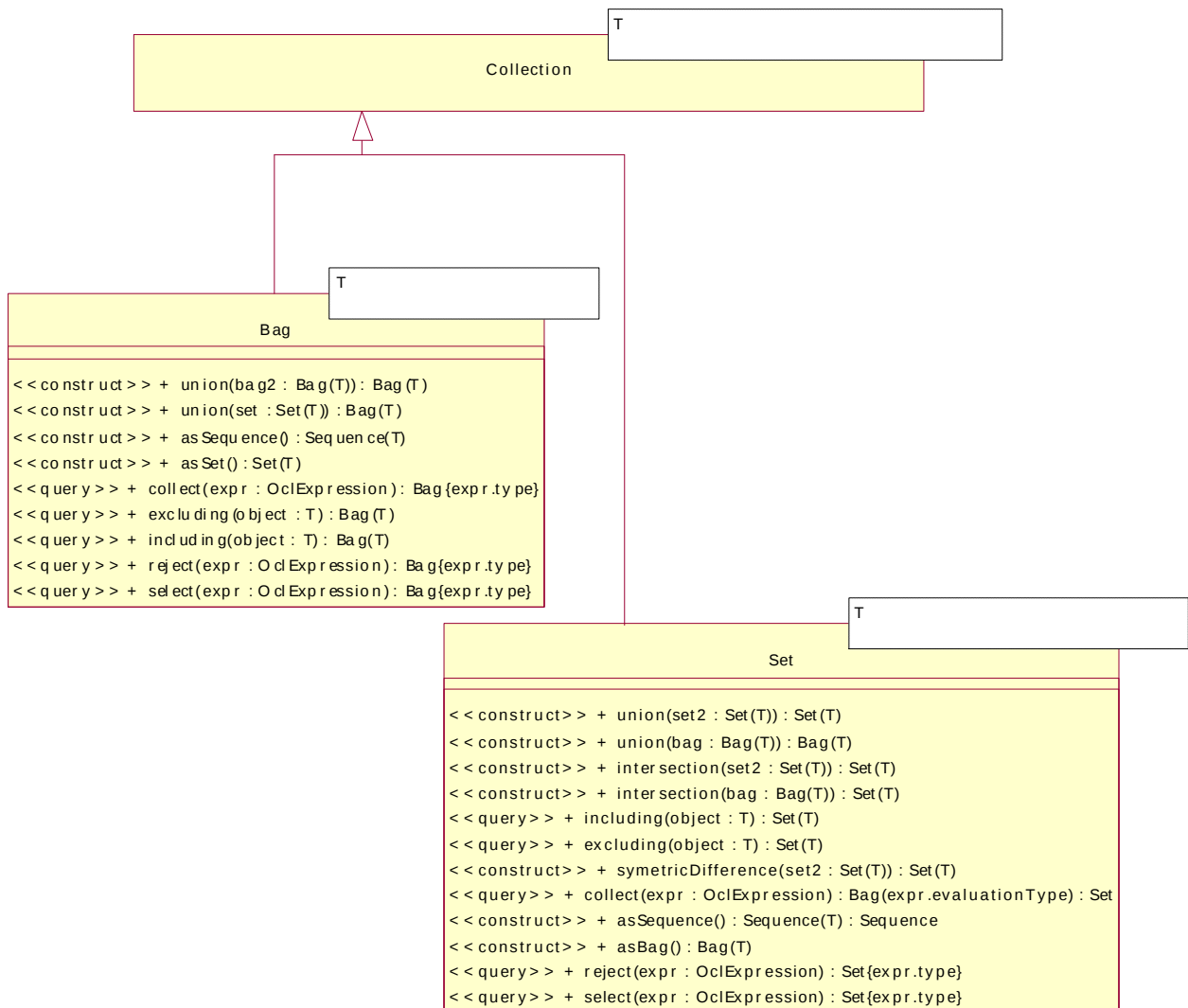


Figure 4-2: Bag and Set Classes

4.1.3. Sequence

A Sequence is a Bag-like structure that orders the element instances. Sequences may be used as either lists or arrays. Arrays in programming languages are sequences that can be directly indexed by the offset from the beginning of the sequence. Although there are lists that are not semantically equivalent to arrays, the details of this separateness is at the implementation, and does not affect the polymorphic interface defined here.

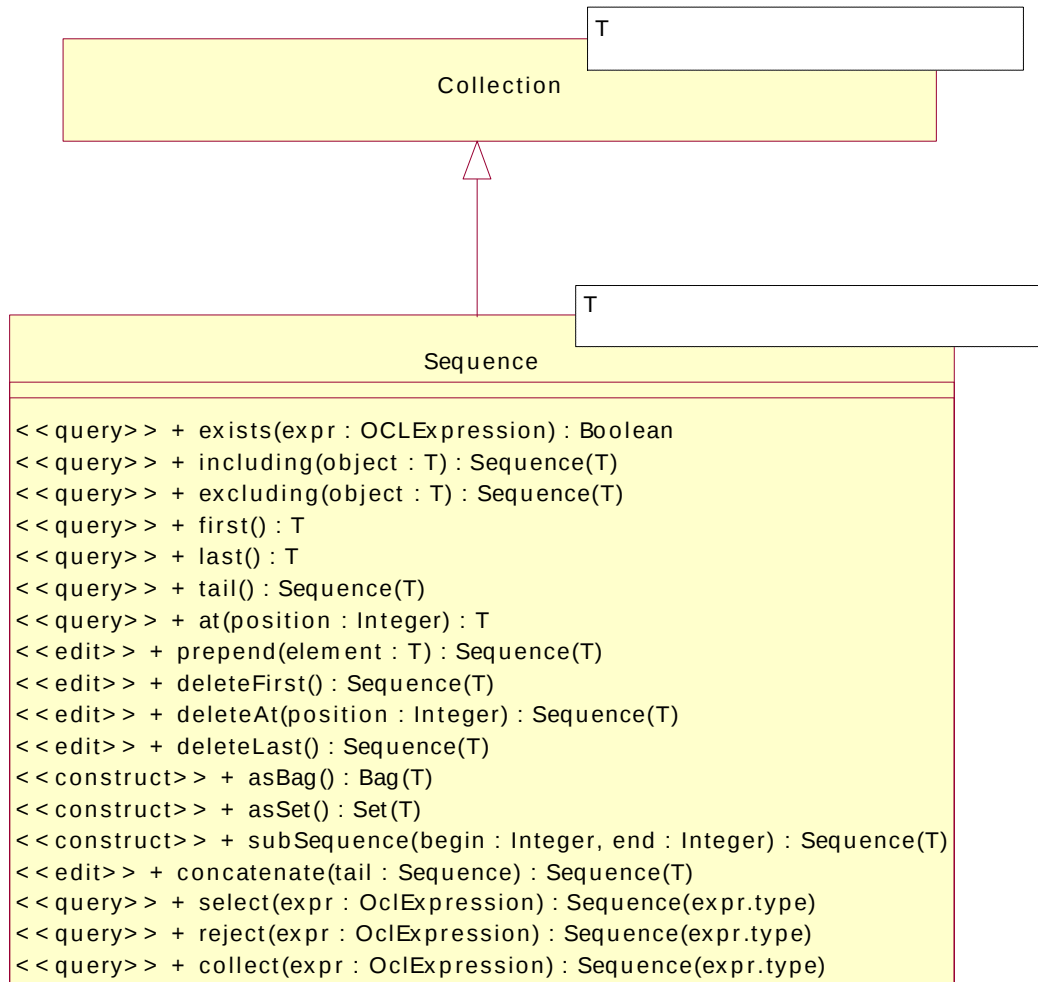


Figure 4-3: Sequence Class

4.1.4. Dictionary

A dictionary is similar to an array, except that the lookup index is not the integers. Although any index type (KeyType) or return type (ValueType) is acceptable, the application within this standard is to use character strings to represent names of parameters and numbers to return values.

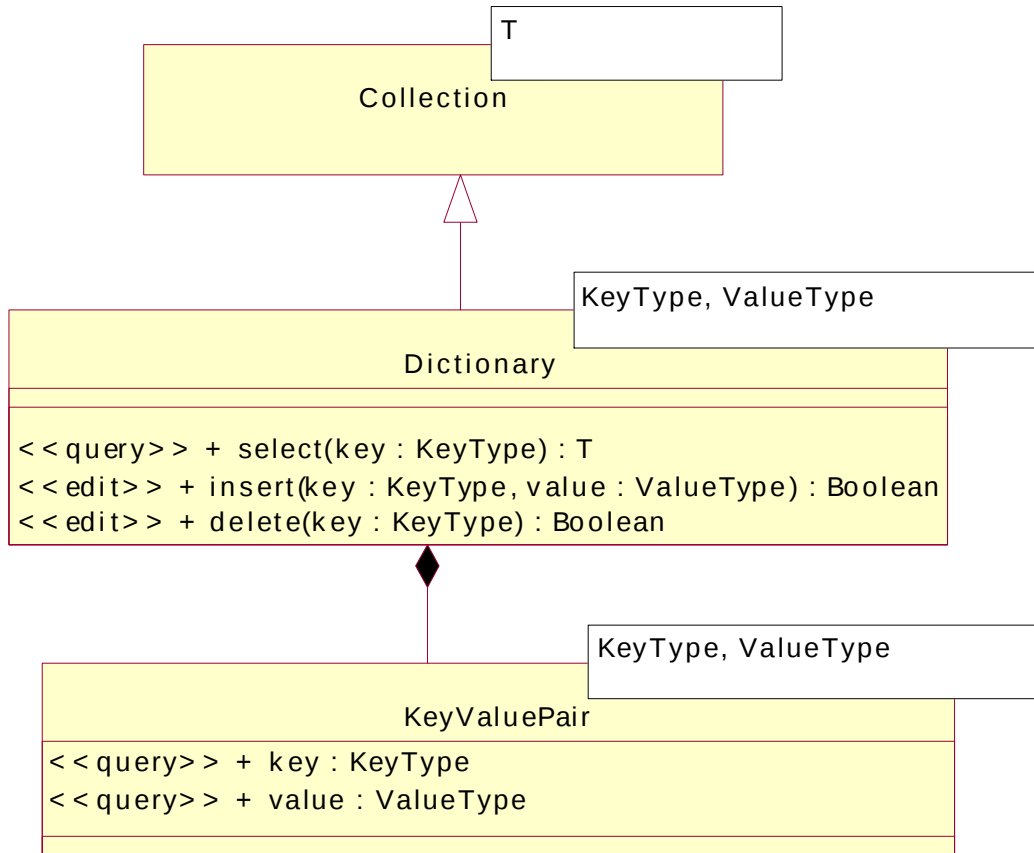


Figure 4-4: Dictionary Class

4.1.5. Basic Language Types

Basic data types are consistent with programming language types for numbers, Booleans, character strings, date and time.

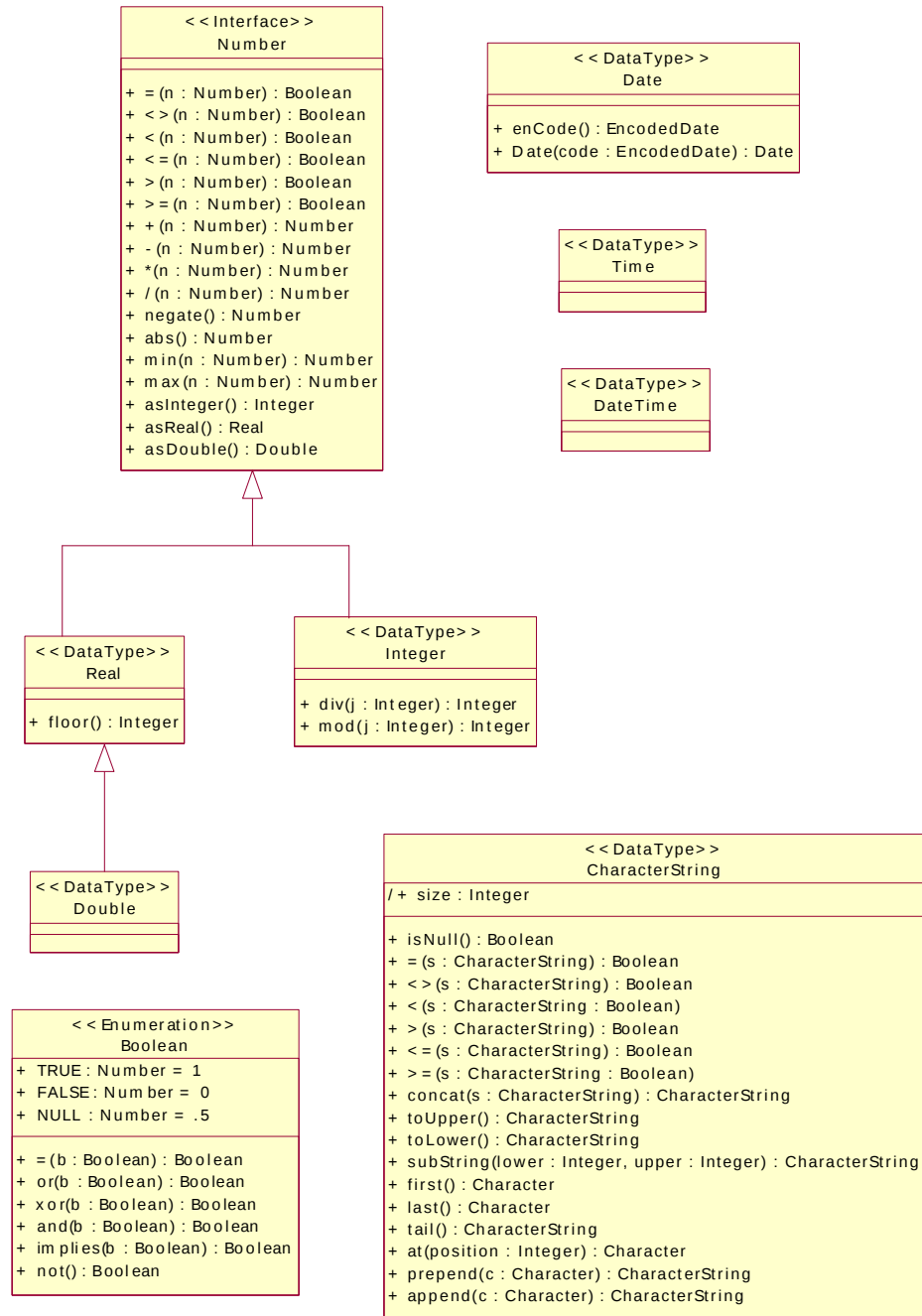


Figure 4-5: Basic Language Types

4.2. Basic Data Types

This package is another support package for the Coordinate Transformation object model. It contains classes that would probably be used as part of implementing a CT Service.

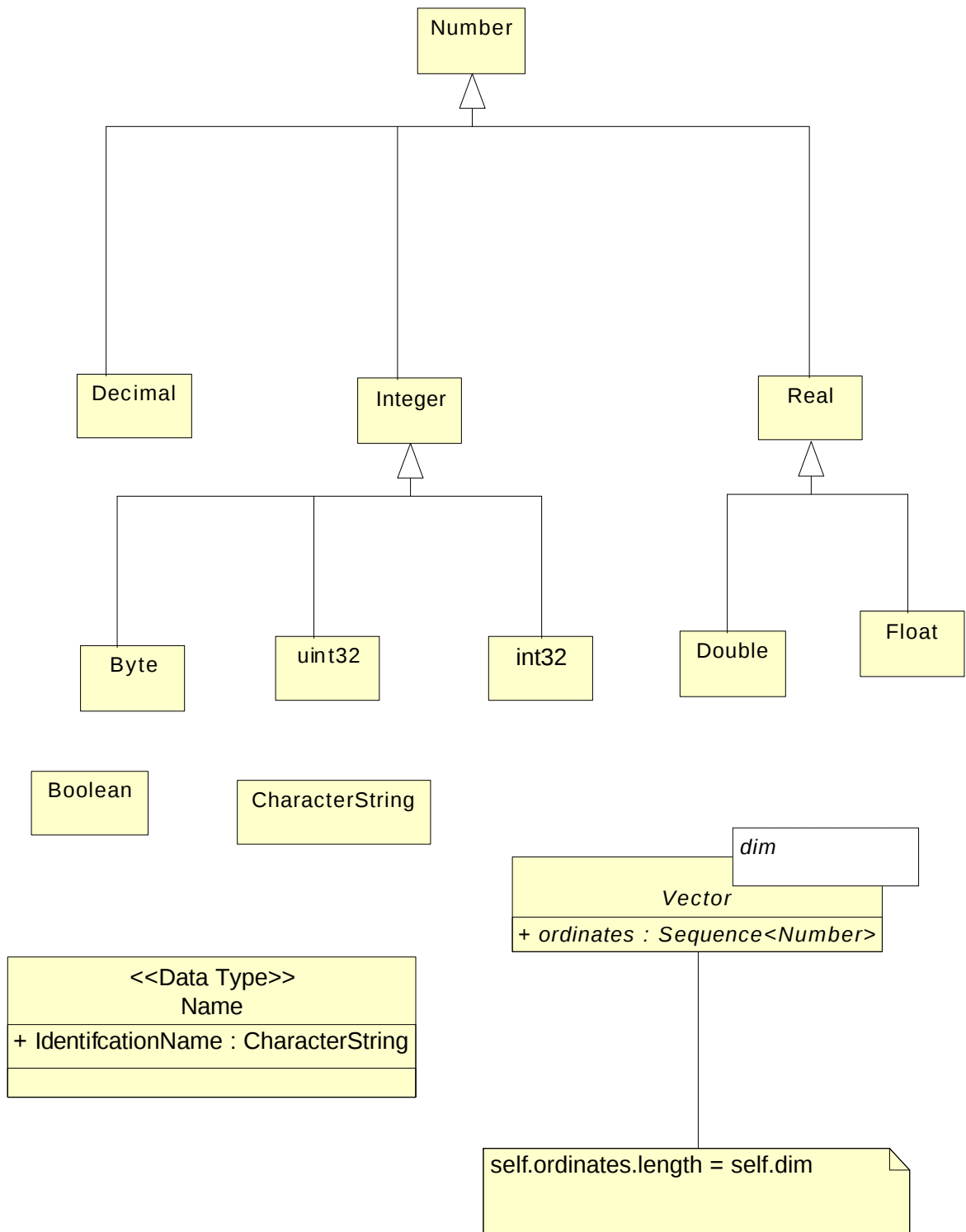


Figure 4-6. Basic Data Types Class Diagram from OGC Geometry Model

4.2.1. Class name: *Real*

Package the Class belongs to: Basic Data Types

Documentation:

Any number that corresponds to a point on a number line.

Any real can be represented by a (potentially infinite) decimal expansion in any chosen radix. Most computer systems use a radix 2 (binary) expansion of fixed length with an offset exponent.

Hierarchy:

Superclasses: Number

4.2.2. Class name: *Float*

Package the Class belongs to: Basic Data Types

Documentation:

A limited precision real number used mainly to represent numbers of limited accuracy, such as a surveyed length over long distances.

Usually represented by an IEEE 32 bit real.

Hierarchy:

Superclasses: Real

4.2.3. Class name: *Double*

Package the Class belongs to: Basic Data Types

Documentation:

An extended precision floating point (real) number. Most often, a Double is a 64 bit (8 byte) double precision data type that encodes a double precision number using the IEEE 754 double precision format.

More modern processors, or ones requiring a higher precision could be using a 128 bit floating point. Again, the type is meant to be descriptive as opposed to prescriptive.

Hierarchy:

Superclasses: Real

4.2.4. Class name: *int32*

Package the Class belongs to: Basic Data Types

Documentation:

An extended precision integer used where +/- 2 billion is necessary and sufficient for a range. Can be used in direct positions where 1 cm accuracy for a worldwide coordinate reference system is sufficient.

A signed 32 bit integer give a range of -2,147,483,647 to + 2,147,483,648.

Hierarchy:

Superclasses: Integer

4.2.5. Class name: *uint32*

Package the Class belongs to: Basic Data Types

Documentation:

An extended precision unsigned integer used where 0 to 4 billion is necessary and sufficient for a range. It can be used in direct positions where 1 cm accuracy for a world wide coordinate reference system is sufficient.

Normally, an Unsigned Integer (unit) is a 32 bit (4 byte) data type that encodes a non-negative integer in the range of 0 to 4,294,967,295.

Hierarchy:

Superclasses: Integer

4.2.6. Class name: *Byte*

Package the Class belongs to: Basic Data Types

Documentation:

An integer used to represent for very small whole numbers (i.e., usually an integer less than 64).

Normally, a byte is the smallest directly addressable unit of storage; the amount of memory space used to store one character, or one very small integer is usually 8 bits.

Hierarchy:

Superclasses: Integer

4.2.7. Class name: *Boolean*

Package the Class belongs to: Basic Data Types

Documentation:

In this context, a binary variable that usually takes a value of zero (FALSE), one (TRUE) or NULL (undecided or unknown).

The value that results from doing a Boolean operation in which each of the operands and the result is a logical truth value.

Hierarchy:

Superclasses: none

4.2.8. Class name: *CharacterString*

Package the Class belongs to: Basic Data Types

Documentation:

A group or sequence of printable characters, usually from a defined character set or alphabet, such as Roman (Latin-based in common use in Europe) or Cyrillic (Greek based, used in Russian and other Slavic Languages).

A character may be use as a letter, digit, or other symbol that is used to represent information.

Hierarchy:

Superclasses: none

4.2.9. Class name: *Vector*

Package the Class belongs to: Basic Data Types

Documentation:

A ordered list of numbers, used to represent either a point in a Cartesian coordinate system or a vector (distance and direction), either free floating or attached to a point.

Each number in the list is referred to as an ordinate. The list together as a whole is a coordinate (coordinated set of ordinates).

Hierarchy:

Superclasses: none

Generic parameters:

Integer dim

Public Interfaces:

Attributes: ordinates

Operations: add

multiply

Attributes:

ordinates : Sequence<Number>

An ordered list of numbers

Operations:

add

multiply

4.2.10. Class name: *Name*

Package the Class belongs to: Basic Data Types

Documentation:

A generic data type class that is currently being used to identify a specify class in a given model being developed.

Hierarchy:

Superclasses: none

Public Interfaces:

Attributes: identificationName

Attributes:

identificationName : CharacterString

A character string used to identify the class (e.g., the name of coordinate transformation or coordinate reference system

4.3. Unit of Measure Package

This package is a support package for the Coordinate Transformation object model. It contains classes that would probably be used as part of implementing a CT Service.

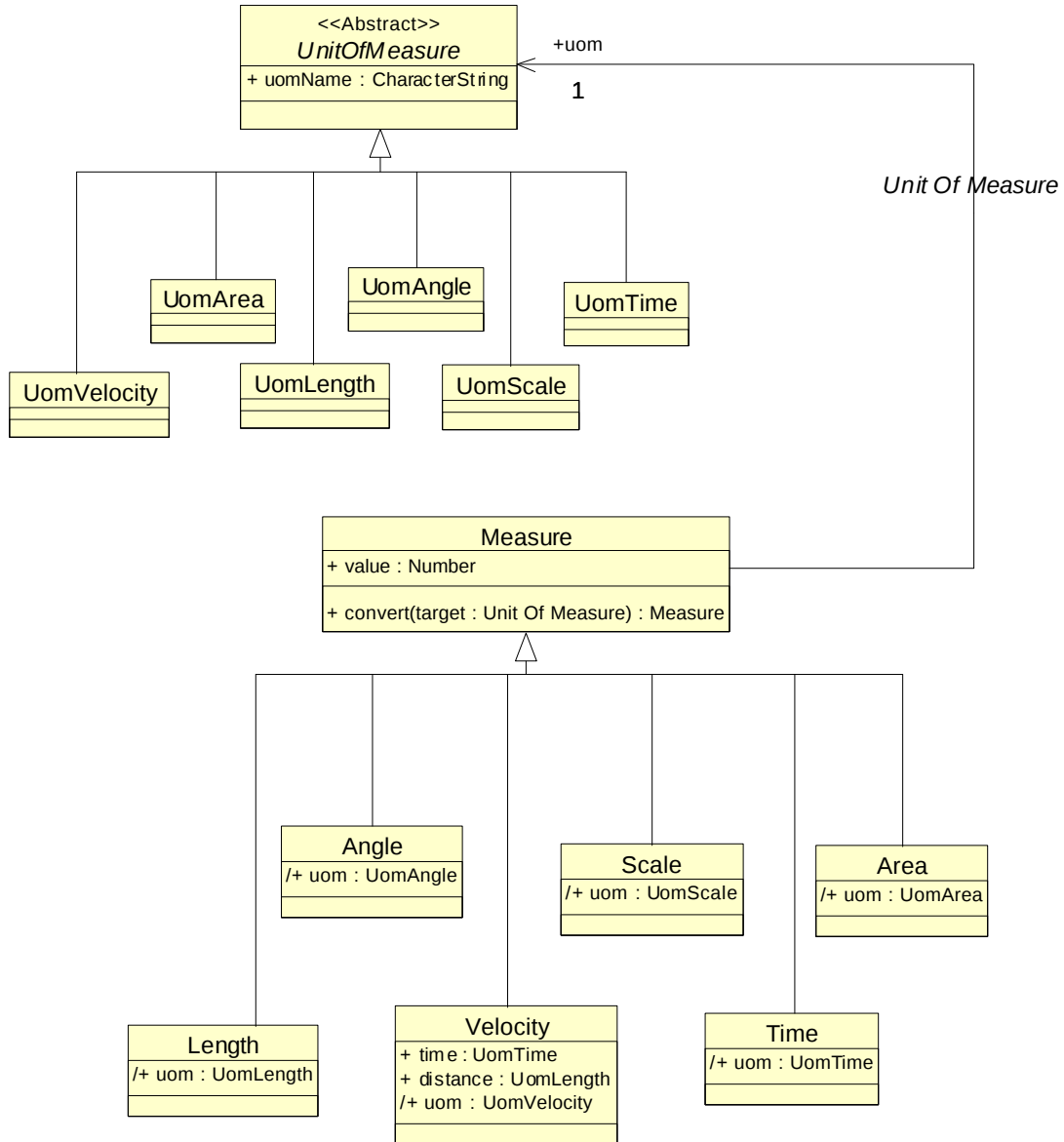


Figure 4-7. Main Class Diagram for Unit of Measure Package

4.3.1. Class name: **Unit Of Measure**

Package the Class belongs to: Unit of Measure

Documentation:

Any of the systems devised to measure some physical quantity such as distance or area or a system devised to measure such things as the passage of time.

Stereotype: Abstract

Hierarchy: Superclasses: none

Associations: <no rolename> : Measurement in association UOM

Public Interfaces: **Attributes:** uomName

UomName: CharacterString

The name(s) assigned to a particular unit of measure. Examples would include the following: 1) for uomArea - square feet, 2) for uomTime - seconds, 3) for uomArea - miles and 4) uomAngle - degrees.

4.3.2. Class name: *UomScale*

Package the Class belongs to: Unit of Measure

Documentation:

Any of the measuring systems commonly used to measure scale, or the ratio between quantities with the same units. Scale factors are often unitless.

Hierarchy:

Superclasses: UnitOfMeasure

4.3.3. Class name: *UomLength*

Package the Class belongs to: Unit of Measure

Documentation: Any of the measuring systems to measure the length, distance between two entities. Example are the English System of feet and inches or the metric system of millimeters, centimeters and meters.

Hierarchy:

Superclasses: UnitOfMeasure

4.3.4. Class name: *UomAngle*

Package the Class belongs to: Unit of Measure

Documentation:

The measuring system(s) commonly used to measure angles. In the U.S., the sexagesimal system of degrees, minutes and seconds is frequently used. The radian measurement system is also used to a limited degree. In other parts of the world the grad angle measuring system is used.

Hierarchy:

Superclasses: UnitOfMeasure

4.3.5. Class name: *UomTime*

Package the Class belongs to: Unit of Measure

Documentation:

Any of the systems or methods of measuring or reckoning the passage of time and/or date, (e.g., seconds, minutes, days, months).

Hierarchy:

Superclasses: UnitOfMeasure

4.3.6. Class name: *UomArea*

Package the Class belongs to: Unit of Measure

Documentation:

Any of the measuring systems commonly used to measure area. Common units include square length units, such as square meters and square feet. Other common unit include acres (in the U.S.) and hectares.

Hierarchy:

Superclasses: UnitOfMeasure

4.3.7. Class name: *UomVelocity*

Package the Class belongs to: Unit of Measure

Documentation:

Any of the systems or methods used to measure motion, usually measured as the change in position during a given time interval (e.g., m/sec^2).

Hierarchy:

Superclasses: UnitOfMeasure

4.3.8. Class name: *Measure*

Package the Class belongs to: Unit of Measure

Documentation:

The result from performing the act or process of ascertaining the extent, dimensions, or quantity of some entity.

Subclasses of Measure will have to specify which class of UnitOfMeasure that they use. That is, a length measure must use a length unit of measure, and so forth. This restriction is given in the model by a specialization of the role name "uom" which is logically a derived attribute or pseudo-attribute (see UML-RM p166). Note that by substitutability, a specialized member definition must be consistent with the member definition at any superclass (see UML-RM p287, and p 458).

Hierarchy:

Superclasses: none

Associations:<no rolename> : Unit Of Measure in association UOMPublic **Interfaces:****Attributes:**
value

Operations: convert

Attributes:

value : Number

The numerical value quantifying the extent or size of some quantity, in the units specified by the associated UnitOfMeasure class.

Operation: **convert(target : Unit Of Measure) : Measure**

Public member of: Measure

Return Class: Measure

Arguments: Unit Of Measure target

Documentation:

An operation to convert from one unit of measure to another such as feet to meters, or degrees to radians.

4.3.9. Class name: *Length*

Package the Class belongs to: Unit of Measure

Documentation:The measure of a distance, either directly or as an integral. An example, would be the length of curve, the perimeter of a polygon as the length of the boundary.

Hierarchy:

Superclasses: Measure

Public Interfaces:

Attributes: uom

Attributes:

uom : UomLength

This attribute contains the unit of measure for the Length class, generally expressed as one of the following quantities:

- 1) meters;
- 2) feet;
- 3) kilometers; or
- 4) miles.

4.3.10. Class name: *Angle*

Package the Class belongs to: Unit of Measure

Documentation:

The amount of rotation needed to bring one line or plane into coincidence with another, generally measured in degrees, sexagesimal degrees or grads are generally used.

Hierarchy:

Superclasses: Measure

Public Interfaces:

Attributes: uom

Attributes:

uom : UomAngle

This attribute contains the unit of measure for the Angle class, generally expressed as either radians or degrees.

4.3.11. Class name: *Area*

Package the Class belongs to: Unit of Measure

Documentation:

The measure of the physical extent of any 2-D geometric object.

Hierarchy:

Superclasses: ;, Measure

Public Interfaces:

Attributes: uom

Attributes:

uom : UomArea

This attribute contains the unit of measure for the Area class. Common unit of measures include square length units, such as square meters and square feet. Other common units include acres (in the U.S.) and hectares.

4.3.12. Class name: *Velocity*

Package the Class belongs to: Unit of Measure

Documentation: The measure of motion in terms of speed in a particular direction. It is usually calculated using the simple formula, the change in position during a given time interval.

Hierarchy:

Superclasses: Measure

Public Interfaces:

Attributes: uom
time
distance

Attributes:

time : UomTime The designation of a time interval on a selected time scale, astronomical or atomic. It is used in the sense of time of day.
distance : UomLength

The extent or amount of space between two objects (e.g., points, lines).

uom : UomLength

This attribute contains the unit of measure for the Velocity class, generally expressed as the change of distance for given time interval (e.g., m/sec).

4.3.13. Class name: Time

Package the Class belongs to: Unit of Measure

Documentation: The designation of an instant on a selected time scale, astronomical or atomic. It is used in the sense of time of day.

Hierarchy:

Superclasses: Measure

Public Interfaces:

Attributes:

uom : UomTime

This attribute contains the unit of measure for the Time class, commonly expressed in one of the following units; seconds, hours or days.

4.3.14. Class name: Scale

Package the Class belongs to: Unit of Measure

Documentation: The ratio of one quantity to another, often unitless.

Hierarchy:

Superclasses: Measure

Public Interfaces:

Attributes: uom

Attributes:

uom : UomScale

This attribute contains the unit of measure for the Scale class (e.g., inches/mile). This class may also not have a unit of measure attribute value, as the scale measure can be unitless.

4.4. Accuracy Package

The Accuracy package defines a set of data types for recording the accuracy of many numerical quantities and data structures. More specifically, this package defines an Accuracy Measure class, with multiple subtypes to be used for different forms of accuracy information.

Figure 4-8 is a UML class diagram of the Accuracy package. The Accuracy package depends on (or uses) the Unit Of Measure and the Basic Data Types packages, as defined in sections 4.2 and 4.3 above.

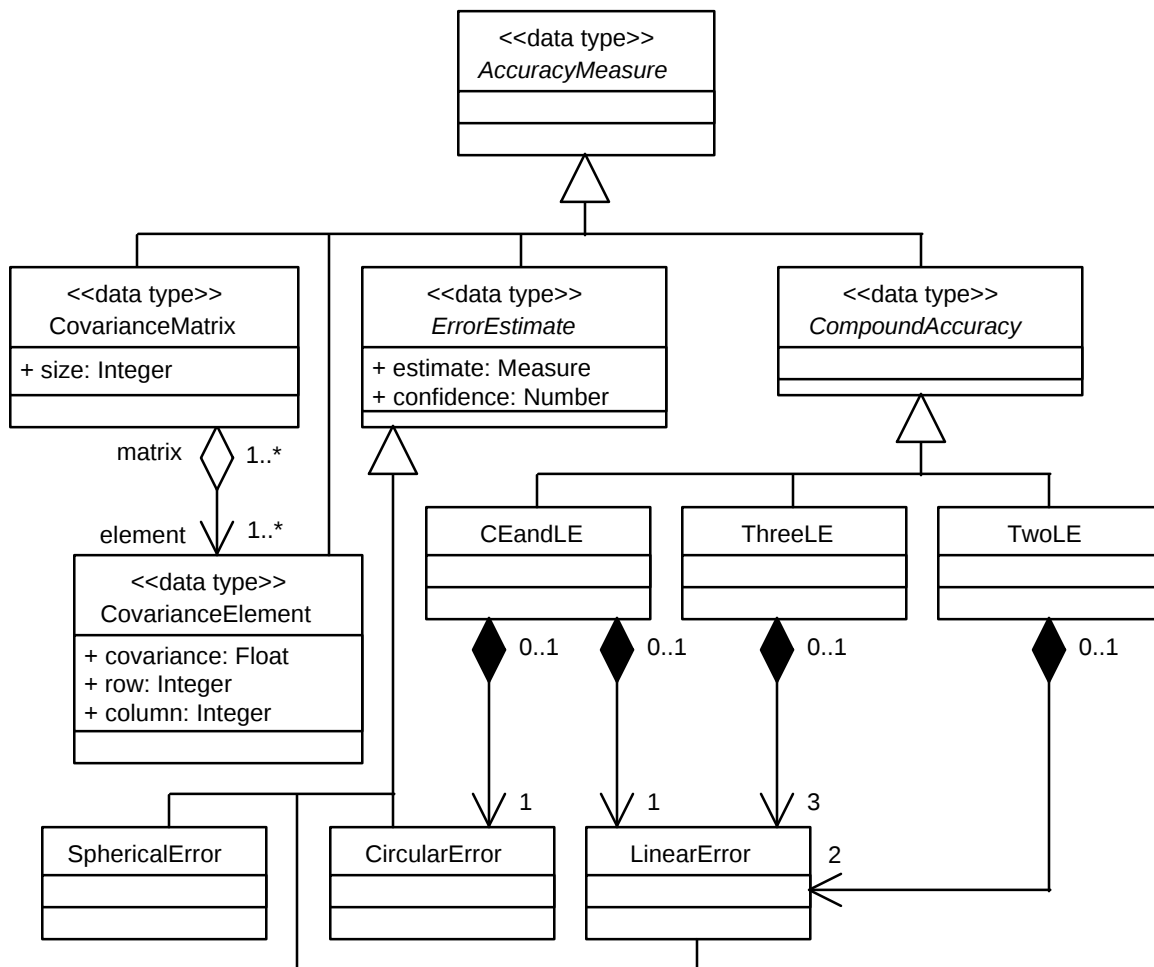


Figure 4-8. Class Diagram of Accuracy Package

Note that both the Covariance Matrix and Covariance Element classes are considered subclasses of the Accuracy Measure class. Also, the Linear Error class is a subclass of the Error Estimate class. Objects of the Linear Error class can be used in a composition association by an object of one of the Compound Accuracy subclasses, or can be used independently. Objects of the Circular Error class can be used in a composition association by an object of the CEandLE class, or can be used independently.

This Accuracy package makes use of the Measure class defined in the Units of Measure package. Other than that use, there are no standard associations between the classes in these two packages. Elements from these two packages are expected to be used together when a Measure needs an associated Accuracy Measure. However, elements from these two packages can be used separately. For example, a Measure object may not need an associated Accuracy Measure object. Alternately, an Accuracy Measure object may not be (directly) associated with a specific Measure object, or a set of such objects.

The following subsections more completely define the Accuracy package classes and associations.

4.4.1. Class name: AccuracyMeasure

Package Class belongs to: Accuracy

Documentation:

Abstract class for describing the accuracy of one quantity or a related set of quantities.

Superclass: none

Stereotype: data type

Associations: none

Attributes: none

Operations: none

4.4.2. Class name: *CovarianceMatrix*

Package Class belongs to: Accuracy

Superclass: AccuracyMeasure

Stereotype: data type

Associations: matrix (of 1..*) CovarianceElement

Note: Since a complete covariance matrix is symmetrical, only one of each pair of symmetrical matrix elements needs to be recorded. Each use of a Covariance Matrix must specify whether expected matrix elements can be missing, and how to interpret a missing expected matrix element (missing values are unknown or are zero).

Attributes:

size: Integer

Documentation:

Specifies the accuracy of a single quantity or a related set of quantities in the form of a covariance matrix, sometimes called a variance-covariance matrix. In a covariance matrix, the diagonal elements are the error variances of the corresponding coordinates, or the squares of the standard deviations. The off-diagonal elements are the covariances between the errors in the corresponding coordinates; these covariances will be zero when the errors in different coordinates are not statistically correlated.

A complete covariance matrix is always square and symmetrical, meaning that the same matrix element values appear on both sides of the diagonal elements. Only one of each pair of symmetrical matrix elements needs to be recorded. Other matrix elements may also be missing, meaning the value of that element is either unknown or zero. Each use of a Covariance Matrix must specify whether expected matrix elements can be missing, and how to interpret a missing expected matrix element (missing values are unknown or are zero).

For the three ground coordinates of one point, a covariance matrix is a 3 by 3 matrix, with the matrix rows and columns each corresponding to the three coordinates. For just the two horizontal ground coordinates, a covariance matrix is a 2 by 2 matrix, with the matrix rows and columns each corresponding to the two horizontal coordinates. The matrix elements are the expected average values of the product of the error in the row coordinate times the simultaneous error in the column coordinate.

Covariance matrices can be used to record absolute and/or relative accuracies. A partial covariance matrix for relative accuracy uses the 3 coordinates of one point for matrix rows and the 3 coordinates of the second point for matrix columns. A complete covariance matrix for N specific points would contain 3N rows and 3N columns.

Note that a 3 by 3 covariance matrix provides six independent numbers, while horizontal CE plus vertical LE are only two independent numbers. The information contained will be equivalent when:

1. The errors in the three ground coordinates are not statistically correlated.
2. The errors in the two horizontal coordinates have the same statistics.

Number of elements along each axis of (square) covariance matrix.

Operations: none

4.4.3. Class name: *CovarianceElement*

Package Class belongs to: Accuracy

Documentation:

Specifies the value of an element of a covariance matrix. If this element is off the matrix diagonal, specifies the values of both symmetrical matrix elements.

Superclass: AccuracyMeasure

Stereotype: data type

Associations: element (1..*) CovarianceMatrix

One CovarianceElement object can be contained in multiple CovarianceMatrix objects, when those matrices overlap.

(describes) (2) (Any Object Class)

In use, each CovarianceElement object must be directly or indirectly associated with two other objects, each of which provides a value and has an associated UnitOfMeasure object. If the two associated objects are the same object, this CovarianceElement represents the variance of the error in the value represented by that object.

Attributes:

covariance: Float

Specifies the value of this element of a covariance matrix. The units of this value are the product of the units of the two (directly or indirectly) associated objects that each provide a value and define the appropriate UnitOfMeasure. A zero value means no expected error.

row: Integer

The row position of this element in the containing covariance matrix. The value can range from 1 up to the value of the “size” attribute of the associated Covariance Matrix.

column: Integer

The column position of this element in the containing covariance matrix. The value can range from 1 up to the value of the “size” attribute of the associated Covariance Matrix.

Operations: none

4.4.4. Class name: *ErrorEstimate*

Package Class belongs to: Accuracy

Documentation:

Abstract class for describing the accuracy of one quantity or a related set of quantities in the form of an error estimate. This error estimate consists of a Measure (usually a Length measure) and the confidence probability that the actual error is less than the specified Measure.

Superclass: AccuracyMeasure

Stereotype: data type

Associations: none

Attributes:

estimate: Measure

The value of this error measure, with the corresponding units of measure.

confidence: Number

The percentage of actual error values expected to be less than the specified error estimate, in percent units. The legal values range from 0 to 100. Commonly used values are 50 (percent), 67 (percent), 90 (percent), and 95 (percent). For a Linear Error with a normal probability distribution, a confidence of 67 (percent) corresponds to the standard deviation.

Operations: none

4.4.5. Class name: *SphericalError*

Package Class belongs to: Accuracy

Documentation:

The accuracy of a set of three closely related quantities, expressed as the expected magnitude of the three-dimensional error vector. These three quantities are usually the three coordinates of a point position. These three quantities might or might not be recorded in the same units of measure. The spherical error estimate value can be expressed in different units.

Superclass: ErrorEstimate
Stereotype: data type (inherited)
Associations: none
Attributes: only inherited
Operations: none

4.4.6. Class name: CircularError

Package Class belongs to: Accuracy

Documentation:

The accuracy of a set of two closely related quantities, expressed as the expected magnitude of the two-dimensional error vector. These two quantities are usually the two horizontal coordinates of a ground point position, or the two coordinates of an image point. These two quantities are usually recorded in the same units of measure, but they could use different units. The circular error estimate value can be expressed in different units.

Superclass: ErrorEstimate
Stereotype: data type (inherited)
Associations: (contained in) (0..1) CEandLE
Attributes: only inherited
Operations: none

4.4.7. Class name: LinearError

Package Class belongs to: Accuracy

Documentation:

The accuracy of one quantity expressed as the expected magnitude of the one-dimensional error vector. The linear error estimate value can be expressed in different units of measure than the value of the quantity.

Superclass: ErrorEstimate
Stereotype: data type (inherited)
Associations: (contained in) (0..1)CEandLE
(contained in) (0..1)ThreeLE
(contained in) (0..1)TwoLE

Note: These three associations are mutually exclusive. However, a Linear Error need not be contained in an object of any other Accuracy Measure class.

Attributes: only inherited
Operations: none

4.4.8. Class name: CompoundAccuracy

Package Class belongs to: Accuracy

Documentation:

Abstract class for describing the accuracy of a set of three or two related quantities by using three or two error estimates. The multiple quantities whose accuracy is specified usually specify a position in three or two dimensional space.

Superclass: AccuracyMeasure
Stereotype: data type
Associations: none
Attributes: none
Operations: none

4.4.9. Class name: *CEandLE*

Package Class belongs to: Accuracy

Documentation:

The accuracy of a set of three related quantities, expressed as the combination of one Circular Error (for the first two quantities) and one Linear Error (for the third quantity). The three related quantities usually specify a position in three dimensional space. The Circular Error is then used for the horizontal position, and the Linear Error is used for the elevation. The Circular Error and the Linear Error will often have the same confidence probability and the same units of measure, but that is not required.

Superclass: CompoundAccuracy
Stereotype: data type (inherited)
Associations: (contains) (1) CircularError
 (contains) (1) LinearError
Attributes: none
Operations: none

4.4.10. Class name: *ThreeLE*

Package Class belongs to: Accuracy

Documentation:

The accuracy of a set of three related quantities, expressed as the combination of three Linear Error objects, one for each quantity. The three related quantities usually specify a position in three dimensional ground space. The three Linear Error objects will usually have the same confidence probability and the same units of measure, but that is not required.

Superclass: CompoundAccuracy
Stereotype: data type (inherited)
Associations: (contains) (3) LinearError
Attributes: none
Operations: none

4.4.11. Class name: *TwoLE*

Package Class belongs to: Accuracy

Documentation:

The accuracy of a set of two related quantities, expressed as the combination of two Linear Error objects, one for each quantity. The two related quantities usually specify a position in two dimensional ground or image space. The two Linear Error objects will usually have the same confidence probability and the same units of measure, but that is not required.

Superclass: CompoundAccuracy
Stereotype: data type (inherited)
Associations: (contains) (2) LinearError
Attributes: none
Operations: none

5. Future Work

There is much to do. Following is a list of services that may be required in some federal agencies, for example.

- Geospatial Information Access Services
- Feature Generalization Services
- Geospatial Information Extraction Services
- Geospatial Coordinate Transformation Services
- Grid Services
- Geospatial Annotation Services
- Imagery Manipulation Services
- Imagery Exploitation Services
- [Image] Exploitation Mensuration Services
- Geospatial Analysis Services
- Geospatial Registration and Adjustment Services
- Geospatial Symbolization Management Services
- Geospatial Dissemination Services
- Imagery Dissemination Services
- Image Map Generation Services
- Image Synthesis Services
- Image Understanding Services
- Geospatial Display Services