

Open Geospatial Consortium Inc.

Date: 2009-01-15

Reference number of this document: OGC 08-126 (used to be 99-105r2.doc)

Version: 5.0

Category: OpenGIS[®] Abstract Specification Topic Volume

Editors: Cliff Kottman and Carl Reed

The OpenGIS[®] Abstract Specification

Topic 5: Features

Copyright © 2009 Open Geospatial Consortium, Inc. All Rights Reserved.
To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>.

Document type: OpenGIS[®] Abstract Specification
Document subtype: Topic Volume
Document stage: Approved
Document language: English

Revision History

Date	Description
20 January 1998	renumber and update copyrights for 1998; add sections 2.11 and 2.12 (RFP1 status); renamed 98-105r1
28 January 1998	update with CORBA conclusions; incorporate Grant Ruwoldt’s clarifications on feature, feature type, feature collection , etc. into section 3; remove feature type and feature collection object models; renamed 98-105r1a
11 January 1999	Updates to 98-105r1a from project document 98-056, approved December 1998; remove references to “OID”; remove section 3.8 of 98-056 as suggested by author; remove future work items as suggested by author of 98-056; reconstitute from 98-105r1 lost feature collection content; use new template format; update copyrights for 1999; renamed as project document 99-105
15 February 1999	Added UML diagrams for FT_Feature as discussed in Atlanta TC, some informal text representing future work in Section 3 moved to Section 4. Renamed 99-105r1, fixed cross-references in Figure 2-14.
24 March 1999	Renamed to 99-105r2; updated for new document template following guidance of change proposal 99-010 w/ friendly amendments to remove Section 1 boilerplate from individual topic volumes, approved 9 February 1999; added Appendix C, from change proposal 98-072 (approved February 9, 1999), and reference to it in Section 2.10.
August 2008	Bring into line with other OGC AS Topic Volumes and OGC Standards Baseline and relevant ISO 19xxx series documents.

This page is intentionally left blank.

Table of Contents

1. Introduction.....	1
1.1. The Abstract Specification	1
1.2. Introduction to Features	1
1.3. Information Models.....	1
1.4. Terms and Definitions.....	2
1.5. References for Section 1.....	2
2. The Essential Model for Features	3
2.1. General Notion of Geographic Information	3
2.2. Introduction to the Notion of a Information Community	4
2.3. The Real World	5
2.4. The Conceptual World.....	6
2.5. The Geospatial World.....	7
2.6. The Dimensional World.....	9
2.7. The Project World (or The “World View”).....	10
<i>2.7.1. Introduction to the Modeling of Geospatial Features.....</i>	<i>10</i>
<i>2.7.2. Introduction to the Project World from the “Feature With Geometry” Point of View.....</i>	<i>10</i>
<i>2.7.3. Project World Feature Instances</i>	<i>11</i>
<i>2.7.4. Project World Feature Types</i>	<i>12</i>
<i>2.7.5. Project World Geometry.....</i>	<i>12</i>
<i>2.7.6. Project World Envelope.....</i>	<i>12</i>
<i>2.7.7. Project World Geometry Schema.....</i>	<i>12</i>
<i>2.7.8. Coordinate reference System</i>	<i>12</i>
<i>2.7.9. Project World Attribute-Value Pairs.....</i>	<i>13</i>
<i>2.7.10. Project World Attribute Schema and Feature Schema</i>	<i>14</i>
<i>2.7.11. Project Schema.....</i>	<i>15</i>
2.8. The OpenGIS Point World.....	17
2.9. The OpenGIS Geometry World.....	18
2.10. The OpenGIS Feature World	20
<i>2.10.1. Feature Collections</i>	<i>21</i>
<i>2.10.2. Interfaces on Feature Collections.....</i>	<i>22</i>
<i>2.10.3. Background Notes on Feature Collections.....</i>	<i>22</i>
2.11. Summary of the Nine Layer Model.....	23
2.12. An Alternative Perspective to the Nine Layer Model.....	24
2.13. Persistent Feature Identifiers.....	25
<i>2.13.1. Problem Statement</i>	<i>25</i>
2.14. Resolving Scoped Feature Identifiers.....	26
<i>2.14.1. Fundamental Ideas.....</i>	<i>26</i>
<i>2.14.2. Key concept.....</i>	<i>26</i>
<i>2.14.3. Key supporting concept</i>	<i>26</i>
<i>2.14.4. Principles</i>	<i>27</i>
<i>2.14.5. Resolver Services</i>	<i>28</i>
<i>2.14.6. Further Example</i>	<i>29</i>
<i>2.14.7. Hints.....</i>	<i>29</i>
<i>2.14.8. Well Known Scopes and URNs</i>	<i>29</i>

2.14.9. <i>Uniform Resource Names</i>	30
2.14.10. <i>What resolution does</i>	30
2.14.11. <i>Resolution, discovery and access</i>	30
2.14.12. <i>Handles and descriptors</i>	31
2.14.13. <i>Permanent scope objects</i>	31
2.14.14. <i>Uniqueness of identifiers</i>	31
2.14.14.1. <i>The same feature can be in several Scopes</i>	31
2.14.14.2. <i>An identifier cannot be a leaf in more than one scope</i>	31
2.14.15. <i>Uniqueness and equality</i>	31
2.14.16. <i>Internal identifiers and internal scopes</i>	31
2.14.17. <i>Scope aliases</i>	32
2.14.18. <i>Published identifiers</i>	32
2.14.19. <i>FeatureCollection</i>	32
2.14.20. <i>Methods of Feature Identity Scope</i>	32
2.15. Feature Identifier Change Registries: Incremental Publishing	33
2.15.1. <i>Conflict</i>	33
2.15.2. <i>Fundamental Ideas</i>	33
2.15.3. <i>New concept</i>	34
2.15.4. <i>Incremental Publishing</i>	34
2.15.5. <i>Incremental Publishing</i>	34
2.15.6. <i>Non-tree client network</i>	35
2.15.7. <i>Distributed Editing</i>	35
2.15.8. <i>Responsibilities and compositions</i>	36
2.15.9. <i>Local cleanliness</i>	36
2.15.10. <i>Out of control copying</i>	36
2.15.11. <i>Handles for servers</i>	37
2.15.12. <i>Identifier update packet</i>	37
2.15.13. <i>Relationships</i>	37
2.15.14. <i>GIS requirements</i>	37
2.15.15. <i>Last word</i>	37
2.16. Status of the Feature Specification	Error! Bookmark not defined.
2.17. References to Section 2	37
3. The Abstract Model for Feature, Feature Identifier, Identifier Scope, Identifier Change Registry, Feature Repository and Feature Collection	40
3.1. FT_Feature	40
3.1.1. <i>FT_FeatureType</i>	40
3.1.2. <i>FT_FeatureAttribute</i>	40
3.1.3. <i>FT_FeatureIdentifier</i>	43
3.1.4. <i>FT_Feature Persistence</i>	43
3.1.5. <i>FT_Feature Instances</i>	43
3.2. FT_Identifier	43
3.2.1. <i>Characteristics of FT_Identifier</i>	43
3.3. FT_IdentifierScope	44
3.3.1. <i>Characteristics of FT_IdentifierScope</i>	44
3.3.2. <i>Methods of FT_IdentifierScope</i>	44
3.4. FT_IdentifierChangeRegistry	45
3.4.1. <i>Change Registry Characteristics</i>	45
3.5. FT_FeatureRepository	45
3.5.1. <i>FT_FeatureRepository Characteristics</i>	45
3.6. FT_FeatureType	45
3.6.1. <i>FT_FeatureType Characteristics</i>	45

3.7. FT_FeatureCollection	46
<i>3.7.1. Characteristics of FT_FeatureCollections</i>	<i>46</i>
4. Future Work.....	47
5. Appendix A: Well Known Structures.....	48
5.1. Well-Known Structures	48
5.2. References to Appendix A	48
6. Appendix B. The ISO TC211 General Feature Model.....	49
6.1. References	50

1. Introduction

1.1. The Abstract Specification

The purpose of the Abstract Specification is to document a conceptual model sufficient enough to allow for the creation of OGC Implementation Standards. The Abstract Specification consists of two models derived from the Syntropy object analysis and design methodology [1].

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 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, such as those ISO documents that are used as part of the Abstract Specification. . The level of maturity of a topic reflects the level of understanding and discussion occurring within the OGC Technical Committee. Refer to the OGC Technical Committee Policies and Procedures [2] document for more information on the OGC standards development process.

Refer to Topic Volume 0: Abstract Specification Overview [3] for an introduction to all of the topic volumes comprising the Abstract Specification and for editorial guidance, rules and etiquette for authors (and readers) of OGC specifications.

1.2. Introduction to Features

From ISO 19101, “A feature is an abstraction of a real world phenomenon”; it is a geographic feature if it is associated with a location relative to the Earth. Vector data consists of geometric and topological primitives used, separately or in combination, to construct objects that express the spatial characteristics of geographic features. Raster data is based on the division of the extent covered into small units according to a tessellation of the space and the assignment to each unit of an attribute value. Attributes of (either contained in or associated to) a feature describe measurable or describable properties about this entity. Unlike a data structure description, feature instances derive their semantics and valid use or analysis from the corresponding real world entities’ meaning. Documenting feature instances, types, semantics and their properties is often detailed in an information model.

1.3. Information Models

An information model details how to take real world ideas or objects and make them useful to a computer system. In the geospatial world the focus is on depicting things in the real world as points, lines, or polygons (the geometry “primitives” we use to assemble location data about those real world objects) and their attributes (information about those objects). When linked together, a pair (geometry and attributes) representing one or more real world objects, is called a feature.

In ISO 19101, there is the concept of the *Domain reference model*. *The DRM* provides a high-level representation and description of the structure and content of geographic

information. For the purposes of this document, the information and domain reference models are synonymous.

1.4. Terms and Definitions

The following terms and definitions used in this document are from various ISO documents, including 19101 [4], 19111 [6] and 19107 [5]

- **Coordinate:** one of a **sequence** of N-numbers designating the position of a **point** in N-dimensional space [6]
- **Coordinate System:** set of mathematical rules for specifying how **coordinates** are to be assigned to **points** [6]
- **Coordinate Reference System:** **coordinate system** that is related to the real world by a datum [6]
- **epistemic** - Of, relating to, or involving knowledge; cognitive. Adj. of or relating to epistemology; "epistemic modal"
- **Feature:** abstraction of real world phenomena
- **Feature Collection** (see Record)
- **geographic information:** information concerning phenomena implicitly or explicitly associated with a location relative to the Earth [4]
- **Point:** **0-dimensional geometric primitive, representing a position**
- **Record:** finite, named collection of related items (**objects** or values) [5]

1.5. References for Section 1

- [1] Cook, Steve, and John Daniels, Designing Objects Systems: Object-Oriented Modeling with Syntropy, Prentice Hall, New York, 1994, 389 pp.
- [2] Open Geospatial Consortium, 2008. OGC Technical Committee Policies and Procedures, Wayland, Massachusetts. Available via the WWW as <
<http://www.opengeospatial.org/ogc/policies> >.
- [3] Open Geospatial Consortium, 2005. Topic 0, Abstract Specification Overview, Wayland, Massachusetts. Available via the WWW as <
<http://www.opengeospatial.org/standards/as> >.
- [4] ISO 19101:2005 - Geographic information. Reference model
- [5] ISO 19107:2003. Geographic information -- Spatial schema.
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=26012
- [6] ISO 19111: Spatial Referencing by Coordinates

2. The Essential Model for Features

2.1. General Notion of Geographic Information

Geospatial information is anything that you can learn by looking at maps -- not just traditional maps, but new, creative, digital maps and earth visualization systems. A map, after all, is simply a metaphor for the Earth itself. We therefore accept raster Earth imagery as a kind of map, and even less structured collections of samples of Earth phenomena with any kind of instrumentation as acceptable maps. For the purposes of this document, we use the term "map" in a very broad sense – encompassing all earth metaphors from traditional paper maps to 3d earth visualization systems.

We can learn about phenomena that vary with time by looking at special maps designed to reveal temporal differences and events. However, the study of the temporal aspect of geospatial information in 4-space is postponed for a later version of this document. For now, we will assume that phenomena do not change, or that temporal aspects of geospatial information can be held as attributes of features.

The fundamental unit of geospatial information is called a *feature*. Features may be defined recursively, so there can be considerable variation in feature granularity. For example, depending on the application or interests of the information gatherer, any of the items in Figure 2-1 could be a feature.

Potential Features

A segment of a road between consecutive intersections	A numbered highway consisting of many road segments	A dynamically segmented road
A georeferenced satellite image	A single pixel from the image mentioned at the left	A drainage network
A temperature overlay on a weather map	A triangulated irregular network	A set of seismic event magnitude contours

Figure 2-1: Example Features

The collection and use of geospatial information has one purpose: to communicate knowledge about phenomena that have location. For example, the knowledge imparted by the map answers two kinds of questions: "where" and "what." Maps can tell us where things are, both in relation to other nearby things. Maps also can tell us what things are, either through symbology (e.g., by use of color or line pattern whose meaning is explained in a legend) or through text or tabular annotations or multi-media links. The same goes for attributes that modify or extend our knowledge of things.

Digital geospatial information is geospatial information that has been encoded into a digital form. The encoding is done so that computer resources can be applied to automate the business of geospatial information processing: storage, transmission, analysis, visualization and so forth.

There are many different ways to create digital representations of geospatial information. This richness of alternatives is more a curse than a blessing since it has created the confusing and apparently chaotic variety of Geographic Information System (GIS) data structures and formats now confronting GIS users.

This Abstract Specification exists to help bring order to this chaos and to provide the foundation model upon which the OGC standards are based.

2.2. Introduction to the Notion of a Information Community

Typically, geographic features – such as “road” - are defined in the context of an Information Community. Broadly speaking, an Information Community (IC) is a collection of systems or individuals who share a common understanding of information and processes for their specific domain. This implies that the members of the information community share common understanding of their domain world, including definitions, vocabularies, interests, mutual awareness, and common technology sufficiently that they have the potential – and perhaps the capability - to share and use the information. For example, a state Department of Transportation is an Information Community. Any given DoT has a shared agreement on how they define any geographic feature within their IC, such the definition and semantics for a feature type “road”.

Information sharing between two individuals not belonging to the same information community is usually impeded by any of three conditions.

1. Ignorance of the existence of information outside one’s own information community.
2. Modeling of phenomena not of mutual interest.
3. Modeling of phenomena in two representations different from each other such that each is not recognized by the other.

The third condition is very, very typical in the geospatial community. Continuing our example of roads, different Departments of Transportation (different information communities) define and collect road features differently. The result is that a State boundary, a road may have different names, different semantics, different accuracy metadata and so forth.

The OGC provides a forum for enabling ICs to articulate and discuss their domain of interest. This is critical. A key aspect of interoperability is for one or more information communities to interact and drive to consensus for a common understanding (vocabulary and semantics) for their information domain. These information models define feature types and their properties. The goal to is define an information model for their community. Technology is then deployed that uses the information model. For example, in the US, each state DOT is essentially an independent information domain with their own processes, policies, funding sources, etc. Even so, if we are to address broader interoperability issues, such as cross state sharing of transportation data, the State DoTs need to collaborate and define a common information model.

This document explores how an IC can formalize and unify its information-theoretic foundations to ensure that information sharing within its community is straightforward. As we will see, the process of formalization of information models for a community actually benefits communication between widely separated ICs, because it exposes the semantics of specialists within each community in a structured way.

Following we discuss several levels of abstraction implicit when modeling real-world facts to create information models.

This section discusses the layers of abstraction between real-world facts and their representation as a collection of Features with Geometry. Nine layers of abstraction are identified, with eight interfaces between them. The layers of abstraction, their names, the languages they use, their interfaces, and the methods that support navigation through the interfaces are all identified in Figure 2-2.

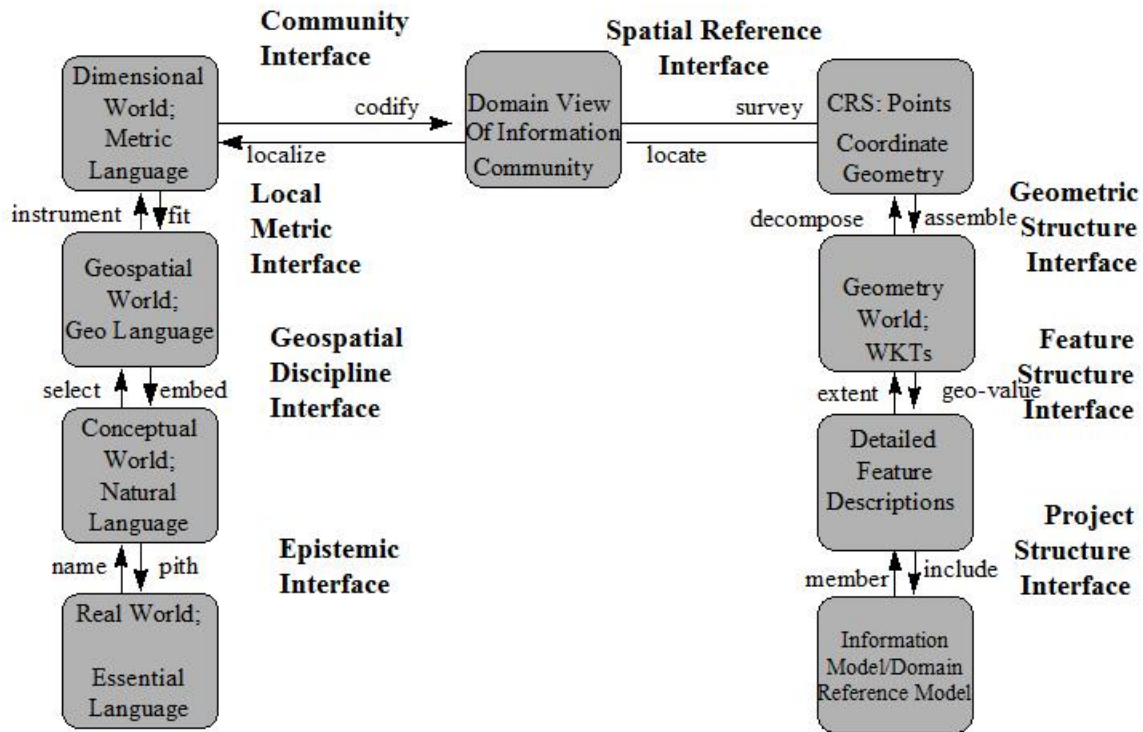


Figure 2-2: Nine Layers of Abstraction

We will discuss the nine layers one at a time. The first five layers, from the Real World to the Project World, deal with the abstraction of real world facts, and are not modeled in software. The final four layers, from Points to Feature Collections, deal with mathematical and symbolic models of the world and are meant to be modeled in software. Even so, this Essential Model of the final four layers assumes that they are real-world objects, and gives no specification, however abstract, for their implementation. The final layer is the abstraction of reality specified in the language an information – the geometric and semantic description of a set of features or Feature Collection.

2.3. The Real World

By “The Real World” we mean the collection of all facts, whether they are known by mankind or not. Facts in the real world are understood in terms of their essence. For example, a tree is something that belongs to the category of things that have “treeness.” Figure 2-3 is meant to represent the Real World. The cloud-like texture that occupies most of the figure represents the wilderness of unknown facts that occupy the chaos of the universe. Only a few of these facts are recognized as familiar patterns, and some of these are represented in the drawing.

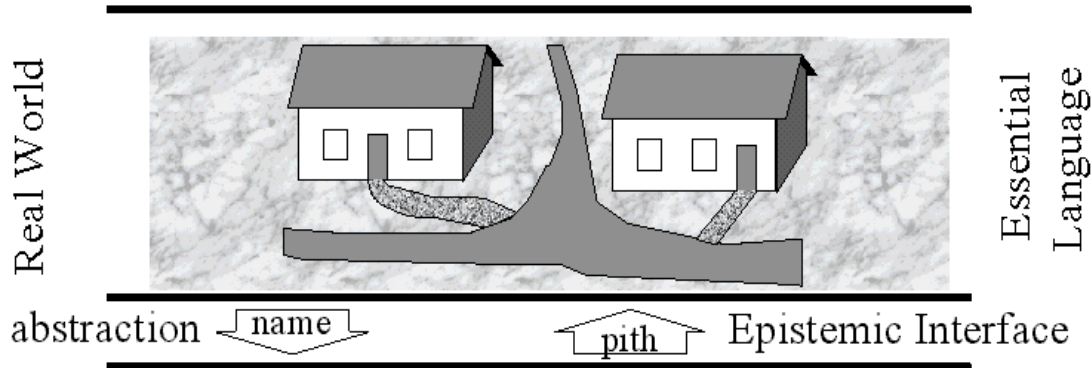


Figure 2-3: The Real World

Figure 2-4 is an abstraction of Figure 2-3, using the Syntropy notation of [Cook94]. An object type is represented by a rectangle, with a name at the top of the rectangle.

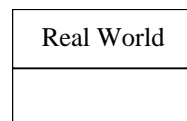


Figure 2-4: The Real World Object Type

Figure 2-4 is an essential model in the sense of [Cook94], that is, it is intended to help understand a situation. The situation is the first step toward abstracting a part of the real world into feature collections. Along the way, we will refine the concept of a Information Community for geospatial feature collections.

Human discourse does not take place at the Real World level. Instead, humans give names to things (abstract them) and communicate with each other using these names. This naming process is exactly the method by which one interfaces to the next level of abstraction: the *name* method. The names are the proper and common nouns and pronouns of our natural language, and they are the language of the Conceptual World.

2.4. The Conceptual World

The conceptual world is the world of our natural language. We see and recognize the things we can name, and this set of facts is often called the Universe of Discourse. In Figure 2-5, which represents the Conceptual World, the clouds representing the chaos of the universe are not present because this chaos is usually invisible in the context of our natural language. Instead, the drawing shows things we can easily identify: doors, paths, bricks, a roof, the house of a friend, and so forth. The method by which we interface back to the Real World is the extraction of the essence of a fact. We call this the *pith* of the fact. Because we give names to things that we know, and can sense the essence of these same known things, we call the interface between the Real World and the Conceptual World the *Epistemic Interface*.

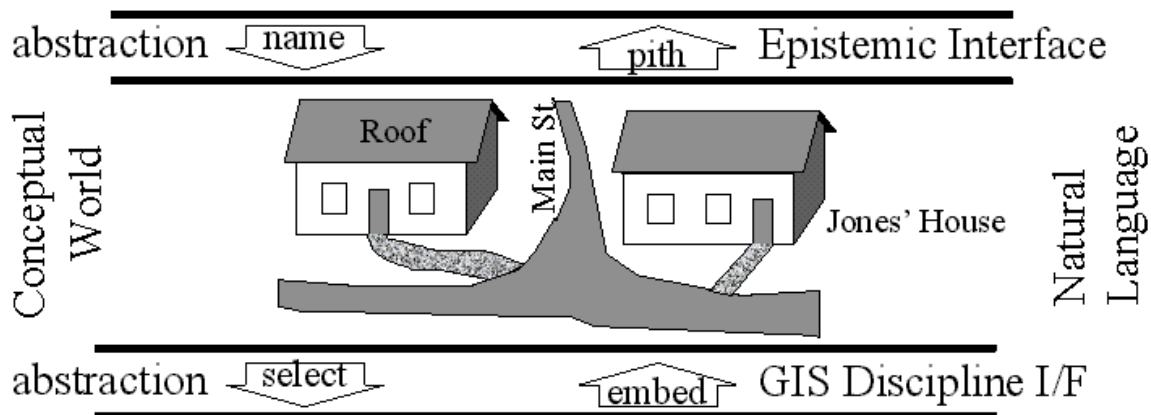


Figure 2-5: The Conceptual World

The Conceptual World of our natural language is not sufficiently abstract for information modeling of geospatial features. In most Geospatial Technology Systems, only a simplified subset of the Conceptual World is of interest. This subset is called the Geospatial World. The method by which one interfaces to the Conceptual World is *select*.

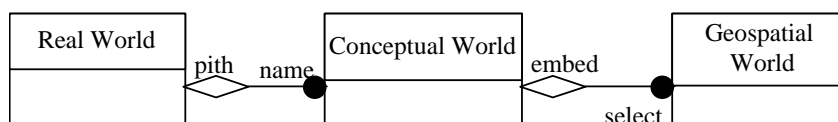


Figure 2-6: Associations between the Real World and the Geospatial World

In Figure 2-6 there are three object types, each represented by a rectangle, and each with a name at the top. Lines between the rectangles represent associations between objects, and each of these carries a role name at each end to explain the association. The diamond represents aggregation; for example, the existence of the Conceptual World depends on the existence of the Real World. The solid circle means there is a set (instead of a single object) at that end of the association. For example, each Conceptual World embeds a set of distinct Geospatial Worlds.

2.5. The Geospatial World

Practitioners of geospatial technology are used to thinking of the world in an abstraction that is almost cartoon in character. This is because the world, even at the conceptual level, is too full of complex shapes, patterns, details, and change to model realistically. These complexities are eliminated in the Geospatial World, and replaced with abstractions that are usually temporally and spatially static as well. Please note, however, that an increasing number of applications are utilizing dynamic abstractions.

Our little slice of the universe is redrawn at the Geospatial World level in cartoon fashion in Figure 2-7. Notice that some features have vanished and others have become greatly simplified. For example, some windows, walls, and roofs of buildings have vanished. This is because they are not of interest to the information communities geospatial view of the world. They have become “invisible” to the community. Of course, there is no universal definition of exactly what features are of interest to a given community, and perhaps sometimes a roof may be of interest. The cartoon merely indicates that the Geospatial World is a subset and a simplification of the Conceptual World.

Euclidean metric space that is assumed to exist (at least locally) throughout the Geospatial World.

2.6. The Dimensional World

The Dimensional World is an abstraction of the Geospatial World where we become equipped with measuring tools such as tape measure, theodolites, GPS, and/or compass. The facts that are recognized at this level of abstraction include unary relations (such as length of an arc, and bearing of a horizontal line) and binary relations (such as distance between two points, buffer zone, and the “contains” relation) that are abstractions of the features themselves. Other examples include North, overlap, touch, dimensionality, intersect, and so forth.

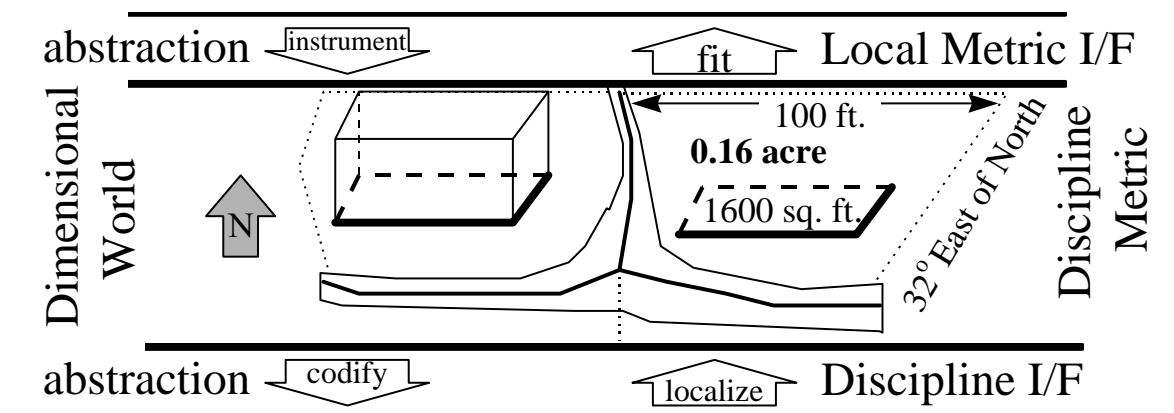


Figure 2-8: The Dimensional World

The method by which the Dimensional World interfaces to the Geospatial World is called *fit*. A distance between two telephone poles is a fact that belongs to the Dimensional World. A wire of this length *fits* the span between them as seen in the Geospatial World. We include in Figure 2-8 some of the abstractions that are present in the Dimensional World.

The Dimensional World is the last of the “generic geospatial” abstractions of the Real World. The next abstraction is called a Domain World (some authors use the term “World View” with a similar meaning) that occurs only in the context of an actual domain implementation. Each implementation is specific to a particular discipline or sub-discipline (or a combination of a few of them). In each actual implementation, only a subset of the Dimensional World is recognized. Often, the subset is determined by map neat lines and by the particular phenomena that have been instrumented.

In addition to the elimination of all but the features of specific interest to the particular domain (discipline or disciplines), there is another abstraction at the Project World level: we introduce the notion of a project-wide Coordinate Reference System (CRS). The most common CRS establish a coordinate structure around the abstraction we call the Earth’s surface, but there are other indirect reference methods, such as the use of a linear reference technique to identify points with a single parameter (e.g., highway mileposts). Whatever the method, we shall insist that all the coordinates that define the geometry of the features abstracted into the Project World carry an abstraction that can be defined by the Coordinate Reference System abide by the coordinate system framework. Hence, coordinates are defined to be a collection of points sufficient for the geometric construction of the geospatial extent of any feature of interest. For the purposes of this document, the term “coordinate” is meant to be generic; it includes “poles” (for spline

construction), “knots” (for NURB construction), “center” (for circle construction), and so forth.

The interface between the Dimensional World and the Project World is called the Community Interface. The method for invoking the Community Interface from a feature in the Dimensional World is called *codify*, and results in a coordinate system being invoked in terms of which all the features’ corners can be abstracted as an n-tuple (n is usually 2 or 3) or into an equivalent abstraction. Conversely, a feature with known coordinates in the Project World can invoke the *localize* interface to result in the feature’s proper placement relative to other features in the Dimensional World, and with appropriate measurements.

2.7. The Project World (or The “World View”)

2.7.1. Introduction to the Modeling of Geospatial Features

There are three popular approaches for the modeling of geospatial features.

The first models the spatial extent of a feature with point, lines, polygons, and other geometric primitives that come from a list of well-known types. Features modeled in this fashion are called “Features with Geometry.”

The second approach is called a “Feature as Coverage”. This technology includes images as a special case.

The third approach is “Feature as Observation”. An Observation is an action with a result which has a value describing some phenomenon. The observation is modelled as a Feature within the context of the General Feature Model [ISO 19101, ISO 19109]. An observation feature binds a result to a feature of interest, upon which the observation was made. The observed property is a property of the feature of interest.

All these primary Features types are intimately related, yet have distinct concepts

This specification will complete the nine layers of abstraction for the case of Features with Geometry. The notion of Coverages is introduced and compared to Features with Geometry in Section 2.13.

2.7.2. Introduction to the Project World from the “Feature With Geometry” Point of View

The use requirements of geospatial content for an information community always reflect the specific discipline or disciplines of their owners. Examples of geospatial disciplines in which geospatial abstractions are frequently found include forest management, soil mapping, cadastre management, base cartography, surface hydrology, wetlands, transportation modeling, and so forth. Each of these disciplines has many sub-disciplines, and an information community can involve any combination of these.

It is exactly the multiplicity of languages at the Project World level that leads to the most difficult problems in interoperability between geospatial information stores. This is the source of much of the splintering that artificially divides the Geospatial World. The situation is manageable, however, if sufficient care is taken in the formalization of the language constructs. We will see that each Geospatial Information Community is intimately related to an equivalence class in the set of all possible Project Worlds under a special relation.

Let’s look at three different Project World views of our slice of the universe. They are shown in Figure 2-9. They reflect the views of a cartographer, a cadastre manager, and a pavement manager, respectively. (A cadastre is a collection of parcel ownership and parcel extent records.) Notice that each Geospatial Information Community sees an abstraction of a particular subset of the whole Geospatial World.

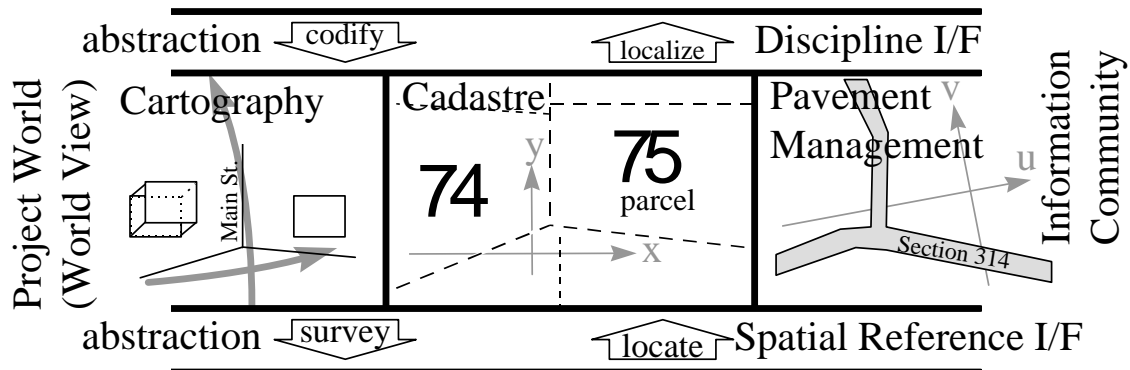


Figure 2-9: The Project World, or The World View

Using the Syntropy notation, the associations from the Geospatial World to the Project World are represented in Figure 2-10.

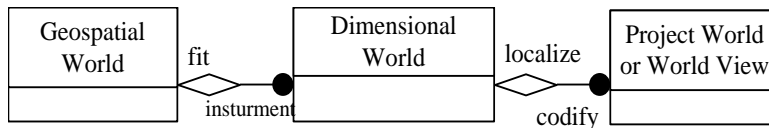


Figure 2-10: Navigation between the Geospatial World and the Project World.

To give the next four levels of abstraction sufficient rigor for unambiguous representation in software, we insist on imposing a great deal of structure upon the language of the Project World (which we call the Information Community Language.) It is through these structures that individuals certify that they are observing common phenomena, and have abstracted the Real World in a repeatable way. Specifically, we impose nine formal and rigorous structures: Feature Instances, Geometry, Coordinates, Geometry Schema, Coordinate Reference System, Feature Types, Attribute-Value Pairs, Attribute Schema, and Project Schema. All of these are to be understood as real world facts, not to be modeled in software at this level of abstraction. We forbid all language that lies outside these structures. Because all nine structures are essential for information sharing, we will discuss each of them, one at a time.

2.7.3. Project World Feature Instances

The primary structure we impose on the Project World in that of the feature. The Project World is completely defined by the instances of the phenomena it recognizes: features. This specification is intended to enable the geospatial engineer to conceptualize and model features in a manner appropriate to the task at hand, and does not prejudice the engineer with predefined thematic classes. This specification is standardizing geometric classes, but does not assume that two engineers will apply them identically to model the same information.

The next eight structures are, for the most part, substructures of the feature structure. We insist that the notion of a feature be completely definitive in a Project World. We demand that our conceptual model of the Project World explicitly recognize any hierarchical, network, geometric, topological or other relationship between the phenomena recognized as features, and hold them as features themselves or as attributes of features. Examples of such relations are “is stacked on” and “is a part of.”

The features of the Project World carry less, but more structured, information than was held at the Geospatial World level, through the process of abstraction and simplification. To make the Project World definitive, we demand that the rules for the inclusion of features from the Geospatial World into the Project World be explicit. Such rules are often called “capture criteria,” or “instance conditions.” Each instance of a feature in the Project World fits a category in the capture criteria.

2.7.4. Project World Feature Types

The categories of features captured in a Project World are feature types. Each feature type can be thought of as a “template” to be populated at each occasion of a feature instance corresponding to that type. The template is populated with information specific to the feature instance. More details are at Section 2.7.8.

2.7.5. Project World Geometry

The feature of the Project World is a simplification and abstraction of the cartoon information carried in the Geospatial World. We recommend that features with spatial extent in the Project World be conceptualized using simple primitive geometric shapes. For interoperability, we recommend that the primitive shapes be instances of the Well-Known-Types (WKTs) of geometry, such as polygons, line strings, polyhedrons, and other shapes. The current enumeration of geometry types can be found in the OGC Abstract Specification Topic Volume 5 (also ISO 19107 – Features). We call instances of WKTs Well-Known-Structures (WKSs). We further demand that rules for representing each feature type with WKTs is explicit (for example, a rule may specify that a brick house is seen as a polyhedron.) Finally, we demand that the conceptual model of the WKSs carries sufficient information to enable the reconstruction of the extents of the features to which they contribute. That is, we insist that the geometry components “know” how they contribute to complex geometries (for example, the highway segment geometries must know the sequence in which they concatenate to become an entire highway feature.

2.7.6. Project World Envelope

The primitive spatial concept in a Project World, from which all phenomena with spatial extent are spatially conceptualized, is “envelope”. An envelope is a geometric primitive that (directly or indirectly through a geometric construct, such as a spline) defines (in whole or in part) the spatial extent of one or more of the phenomena recognized within the Project World.

Each OGC WKS can be constructed from a finite set of points that are its corners. We strongly recommend that the corners of an envelope be explicit in the Project World. We further demand that the model for the envelope carries sufficient information to allow for the construction of the WKSs. For example, we recommend that the sequencing of envelope along a polyline is explicitly available within the Project World.

2.7.7. Project World Geometry Schema

The collection of rules implicit in sections 2.7.5 and 2.7.6 are called the *geometry schema* of the Project World. Examples include:

- for each feature type, the geometry WKTs that are to be used in its representation
- for each WKS, the feature instances to which it contributes
- for each feature instance, the WKSs that contribute to its extent
- for each WKS, the structured list of corners which specify it
- for each corner of an envelope, the WKSs to which it contributes

2.7.8. Coordinate reference System

We introduced the notion of a Coordinate reference System at the end of Section 2.6. Here, as there, we are treating a Coordinate reference System as a real world fact. A good example is a well-monumented point-of-beginning used by a surveyor. The point-of-beginning, together with the tools and rules for surveying, enable one to attach coordinates to every point of interest in a project. The points of interest are precisely the coordinates needed for the geometric construction of the geospatial extent of any feature recognized by the project.

Because the introduction of a Coordinate reference System is central to the notion of a project world, the methods for traversing the Community Interface between the Dimensional World and the Project World are named for the actions relating to it. Invoking the method *codify* causes points in the Dimensional World to obtain their Coordinate reference System coordinates or parameters. Invoking the method *localize* removes the coordinates and recovers points in correct relative positions.

A detailed model for Spatial Referencing by Coordinates can be found in ISO 19111, OGC Abstract Specification Topic Volume 2.

2.7.9. Project World Attribute-Value Pairs

Section 2.7.3 began by insisting that the Project World recognize only features. This section puts additional structure on the concept of feature. We are going to replace our rich and complex natural language descriptions of features with an abstracted stick-figure language of attribute-value pairs.

So far, each Project World feature is associated with two structures: the feature type of which it is an instance, and the geometry consisting of an OGC WKS. The geometry is expressed relative to a Coordinate Reference Systems, and includes all coordinates needed for its construction. We now additionally insist that each feature instance be described using a strict grammar and vocabulary. The purpose of the grammar and vocabulary is to unambiguously and in a repeatable manner describe the attributes of interest for each feature within the Project World.

In the Project World, each feature type is associated with a list of attributes. Our grammar demands that each attribute in the list have a formal attribute name, and that each attribute take a value of a type specific for that attribute. A sample grammar for feature types is shown by the sample template in Figure 2-11.

Feature Type: RoadSegment	
<u>Property Name</u>	<u>Value Type</u>
RoadSegmentStartCoordinate	Point Type
RoadSegmentEndCoordinate	Point Type
RoadSegmentLength	real (meters)
RoadSegmentName	string
RoadSegmentGeometry	ISO/WKT polyline type
RoadSegmentSurfaceMaterial	member of SurfaceMaterial domain

Figure 2-11: Template for [AttributeName /Value Type] Pairs for Feature Type: RoadSegment

Our vocabulary, therefore, is quite limited. We are allowed only the following terms:

1. the names of the feature types recognized by the Project World
2. attribute names
3. attribute values.

Of course, each of these terms can be used only where our grammar permits.

Every feature may publish a unique, persistent. Most feature types have an attribute that can take geometry values. This is the attribute whose value is the WKS that models the physical extent of a feature instance of this type. Relationships between features may not be carried as an attribute-value pair, but must use the facilities described in Topic 8: Relationships Between Features.

2.7.10. Project World Attribute Schema and Feature Schema

The collection of all feature types recognized by a Project World (including the capture criterion for that feature type) together with the list of Attribute/ValueType pairs for each one, is the *attribute schema* of that Project World.

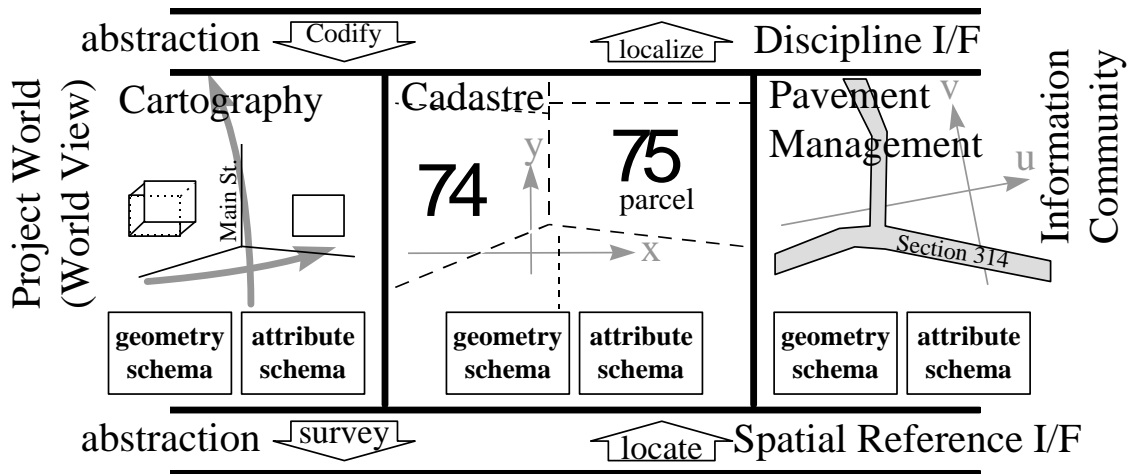


Figure 2-12: Another look at the Project World

The attribute and geometry schemas constitute the *feature schema* of the Project World. See Figure 2-12.

Figure 2-12 is a refinement of Figure 2-9. It explicitly includes the feature schema as a part of the Project World.

The table of attribute-value pairs in a Project World, and more importantly, the semantics behind its terms, reflects the interests of the scientists who conceptualize and formalize it. We insist that the schema be extremely formal, so that all the individuals participating in a Project World observe the same phenomena, and record them in exactly the same way. This rigor mandates that the attribute schema must be completely specific about the

names and meanings of its attributes, as well as the meaning (to include types and units) of its values.

This implies the existence of a dictionary in each Project World where the semantics of the terms in the vocabulary are made clear. The semantics are usually explained in a natural language, with many examples, and have many more dimensions than indicated in this paper. The semantics include aspects of: feature definition, classification discriminators, accuracy, sources, methods, intended uses, and so forth.

2.7.11. Project Schema

The OGC is, to a large extent, about enabling commerce in geospatial information and process. We make the basic assumption that an important unit of trade in this commerce is a feature collection. Each distinct and unique feature collection, therefore, needs to carry enough meta-information to allow the feature information it carries to be understood and exploited. We hinted at some of this meta-information when we spoke of a dictionary for the Project World vocabulary, but there is more. The Project Schema is the formalization of this meta-information, and it is best defined in the Project World context. Figure 2-13 shows the relationships between a feature collection, its features, and the meta-information needed for commerce. The shaded object types denote information that supports the modeling of feature instances within the feature collection. The white object types all hold meta-information of some kind. The white objects, collectively, are termed metadata. We have already introduced the Feature Schema; it is simply the union of the Geometry Schema (See Section 2.7.7) and the Attribute Schema (See Section 2.7.10).

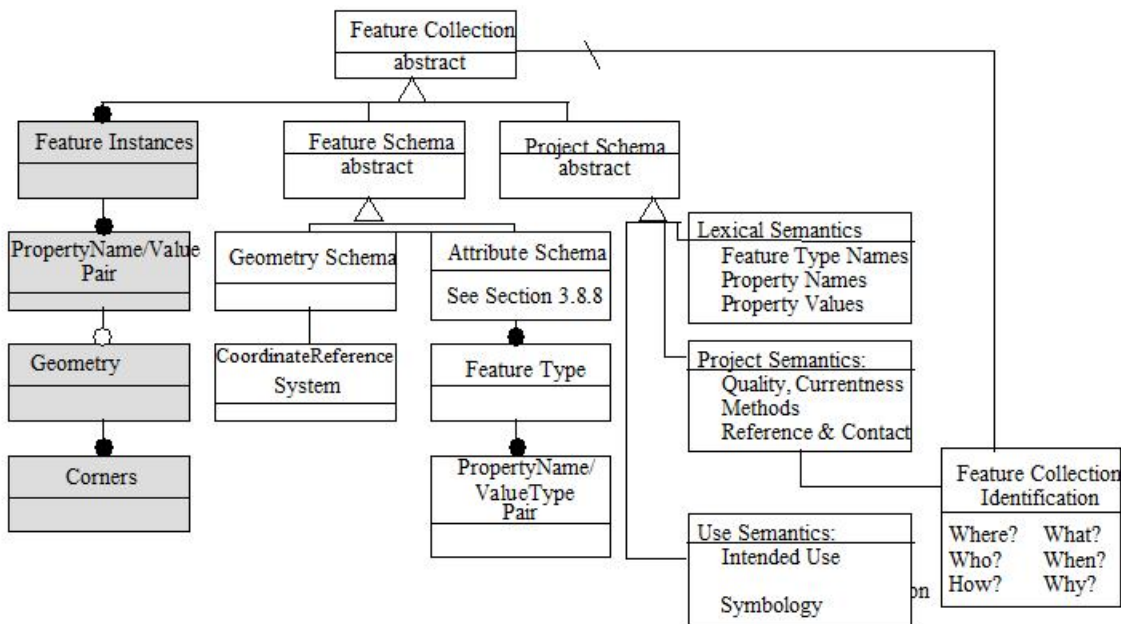


Figure 2-13: The Role of the Project Schema in a Feature Collection

The Project Schema consists of four parts:

1. Feature Collection Identification
2. Lexical Semantics
3. Project Semantics
4. Use Semantics.

We will briefly explain each of these objects, below.

The Feature Collection Identification is a simplified selection from the Project Semantics that is designed to be queryable. For now, it suffices to say that this object provides a unique identifier to the Feature Collection, and provides a high-level description of it:

Where: the region of the Earth covered

What: the approximate scale, the thematic key words

Who: the responsible agent, instructions for access to the Feature Collection

When: the date the data was created and the date of its sources

How: the method to acquire the feature data

Why: the intended use

We will see that the Feature Collection Identification (metadata) is used to support discovery. Since the Feature Collection Identification often plays a role that makes it seem to be intimately connected to the Feature Collection itself, we show a derived relationship between these two object types.

Lexical Semantics are basically the dictionary of terms used in the attribute schema. The Lexical Semantics must provide sufficiently rich definitions and enough examples that there be no ambiguity concerning the proper schema for each feature instance.

Project Semantics describe how the Project World was conceptualized, and how the Feature Collection was generated. Included here is the physical extent of the project, such as the map neatlines, or the region imaged from a space platform.

Use Semantics provide details on how to exploit the feature collection. Included are the intended uses for the information as it was collected, any indices that improve access to the information,

Figure 2-14 is a last look at our slice of the universe in conceptual and literal terms. Further abstractions will be symbolic and mathematical representations.

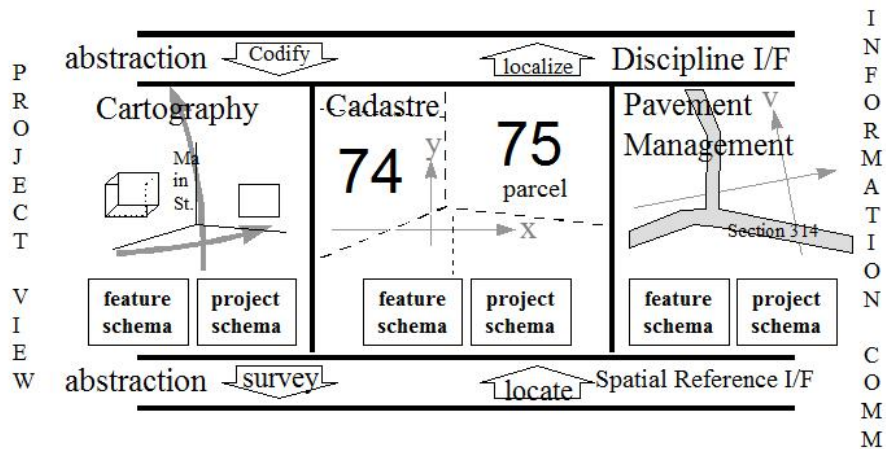


Figure 2-14: A Final Look at the Project World and Its Fundamental Schema

We now have the tools with which to define a Geospatial Information Community. But first let us complete the process of abstracting to features and feature collections.

2.8. The Point World

Each Project World has a project-wide coordinate reference system. With the coordinate reference system, we are able to interface to the next level of abstraction: the Point World. The Point World is perhaps best imagined as Cartesian space populated with a finite collection of special points. Starting from any coordinate of any feature in the Project World, we assume a method named *survey* that abstracts the coordinates of the corner to arrive at a special point in the Point World. Conversely, for each special point in the Point World we assume a method named *locate* that locates that point as a coordinate of a feature in the Project World. Of course, the *survey* and *locate* methods work for points other than the feature coordinates, but to recognize these additional points, one must enlarge the schema and allow new phenomena to be recognized.

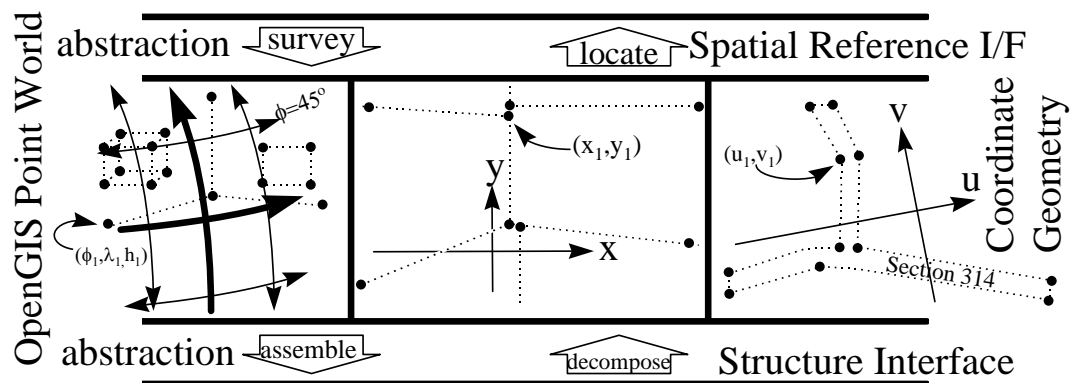


Figure 2-15: Point World

Figure 2-15 represents the situation. There are three Point Worlds represented: one each for cartography, cadastre, and pavement management. Each Point World consists of a coordinate reference system together with a finite set of points (which are at the coordinates of geometries representing features of interest to those three specialties.)

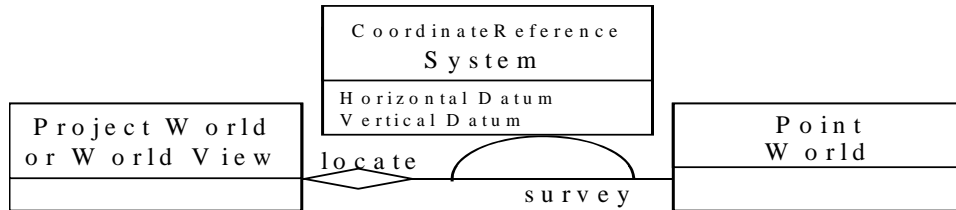


Figure 2-16: The Abstraction of Points from the Project World

Using the notation of [Cook94], the situation is represented in Figure 2-16, where an attribute of an association is shown attached to the association line with an oval. The association of a coordinate in the Project World to a coordinate pair (or n-tuple) in the Point World has an attribute called the Coordinate Reference System. Perhaps more properly, there is an association type that specifies the structure of the association with additional information, including the horizontal and vertical datums. (A horizontal datum, in this setting, may be taken to be an unambiguous assignment of longitude and latitude values to real world locations in a manner that agrees with observations of lengths and angles. A vertical datum similarly assigns elevations to locations.)

2.9. The Geometry World

The careful reader noticed that Figure 2-15 contained more than isolated points and coordinate axes. There are also dotted lines that carry the “memory” of how the points are assigned to specific coordinates of more complex geometry, and combine to recover the geometry components from which the features are to be modeled. The dotted lines did not actually belong in Figure 2-15, because they are not a part of the Point World.

The function of the OGC Geometry World is to recover from earlier abstractions the information needed to construct the OGC Geometry WKTs that model the Project World.

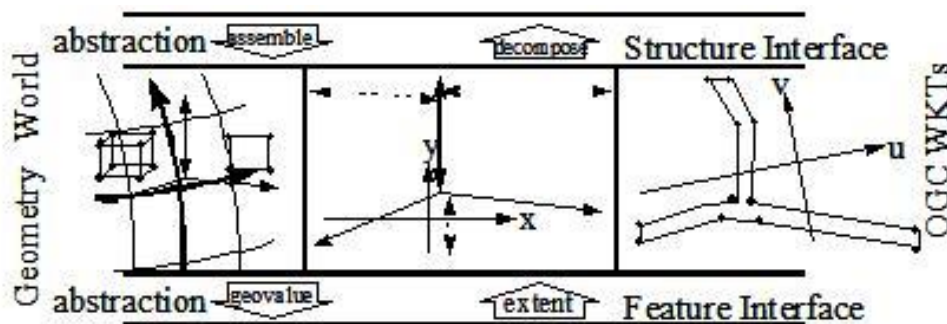


Figure 2-17: The OGC Geometry World

Figure 2-17 shows some representations of the Geometry World, which is populated with points, polygons, polyhedra, and so forth, all in the context of an abstract coordinate system.

The role of the geometry schema is shown more explicitly in Figure 2-18. Here we see that the association between the Point World and the Geometry World is actually a derived association (as indicated by the diagonal line on the association line.) The navigation path from Points to Geometries is through the geometry schema captured in the Project World abstraction.

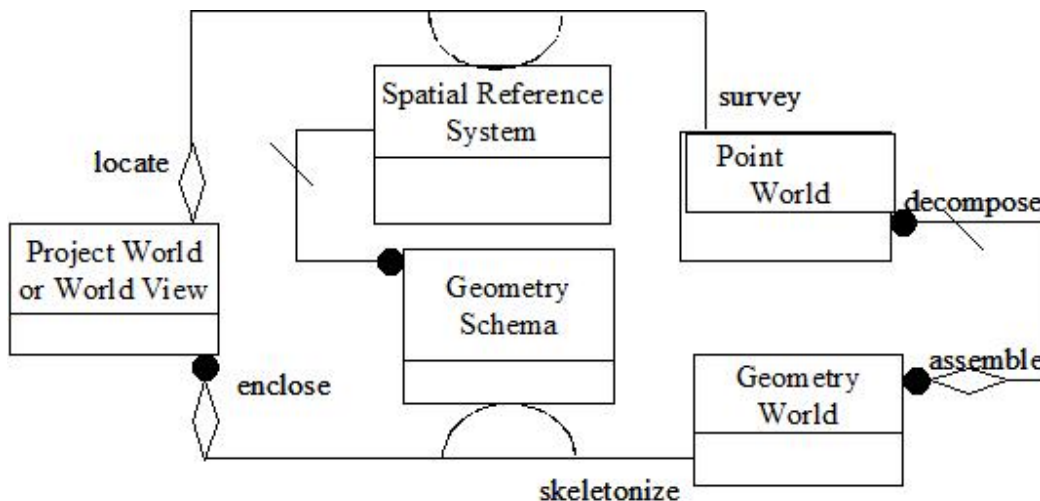


Figure 2-18: Navigation From Points to Geometry

The Geometry Schema Interface lies between the Project World and the Geometry World. To traverse the interface from the Project world, one invokes the *skeletonize* method to create the OGC WKS equivalent to any recognized geometry in the Project World. Traversing the interface in the reverse direction is invoked by the *enclose* method, whereby an OGC WKS is replaced with the cartoon outline of the actual phenomenon represented by that WKS. The interface between the Point World and the OGC Geometry World is invoked by the *decompose* and *assemble* methods, which are defined by the paths through the Project World.

We also show a derived association between the Coordinate Reference System and the Geometry Schema. This is to indicate that the Coordinate Reference System often seems to behave as if it operates on complete geometries, and not just on coordinates.

In Figure 2-18 we used the geometry schema to build the Geometry. Next, we will use the attribute schema to build Features.

2.10. The Feature World

At the level of an Essential Model, each Feature type is specified by its attribute set. An attribute set is a list of attribute/value pairs that is applied to all features of the corresponding type. Every feature instance in the Project World belongs to the one of the recognized types. The type specifies the list of attributes that distinguish the feature, as specified by the Attribute Schema. Most geospatial features have an attribute called “Geometry,” (or, “[FeatureTypeName]Geometry”) and, if so, its value must be the OGC WKS that was conceptualized in the Geometry Schema. Figure 2-19 is a representation of the situation.

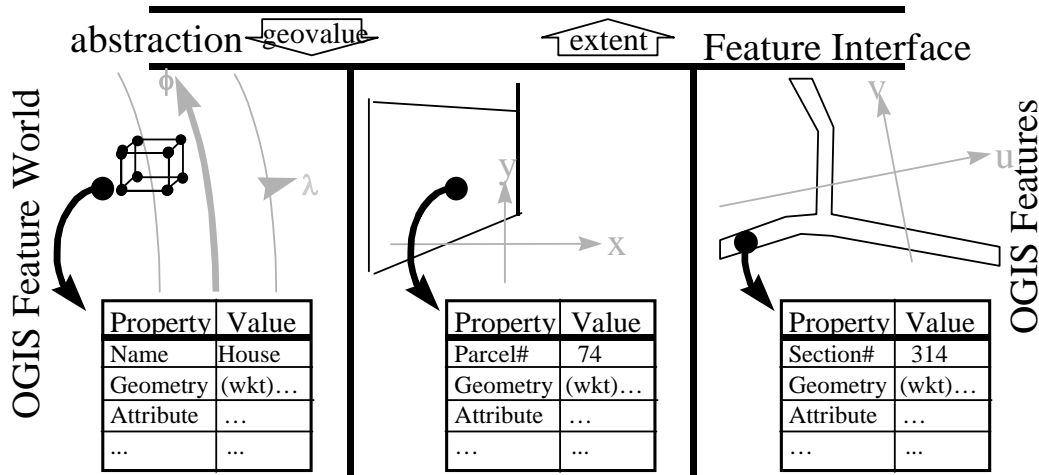


Figure 2-19: The Feature View

Figure 2-19 shows a typical feature in each of the cartographic, cadastre, and pavement management domains. These features did not suddenly come to exist for the first time in the Feature World. The features are the same ones formulated in the Project World. There are two chief components to a feature: its attribute schema, and its geometry. Figure 2-20 shows how the two components of the feature come together.

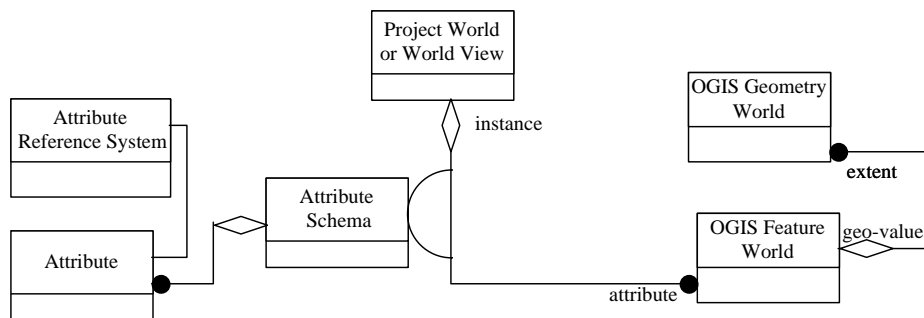


Figure 2-20: Navigation Near a Feature

Notice that the association between Geometry and Feature has no slanted line on it. This is in recognition that this is not a derived association, but rather the primary navigation path for geometry. The path from the Project World to the Feature does not carry geometry, but it does carry all the rest of the Feature Schema.

The interface between the Geometry World and the Feature World is the Feature Structure Interface. It is traversed by the *extent* and *geometry-value* (or *geo-value*, for short) methods. These methods are clear: the extent of a feature is the space it occupies, and this is modeled by an OGC WKS. Likewise, every feature with extent has an attribute named geometry, whose value (that is, the geo-value) comes from the Geometry World.

The interface between the Project World and the Feature World is called the Attribute Schema Interface. It is traversed by transparent methods named *attribute* and *instance*.

There are two new object types in Figure 2-20: Attribute and Attribute Reference System. By “Attribute” we mean a (AttributeName, ValueType) pair, as defined in Section 2.7.9, and as spelled out in the Attribute Schema. The Attribute Reference System is (sometimes) needed to give meaning to the attribute value. For example, a feature may be a temperature coverage defined by a triangulated irregular network (TIN) from a finite number of temperature samples. The mechanics of returning a temperature at a specified point (a coordinate perhaps not in the finite set of coordinates with temperatures) is accomplished by the Attribute Reference System.

Refer also to Appendix B. The ISO TC211 General Feature Model for a similar representation of features, their attributes, functions and relationships.

2.10.1. Feature Collections

An Feature Collection is an abstract object consisting of Feature Instances, their Feature Schema, and Project Schema. Figure 2-21 shows Feature Collections in a Syntropy diagram.

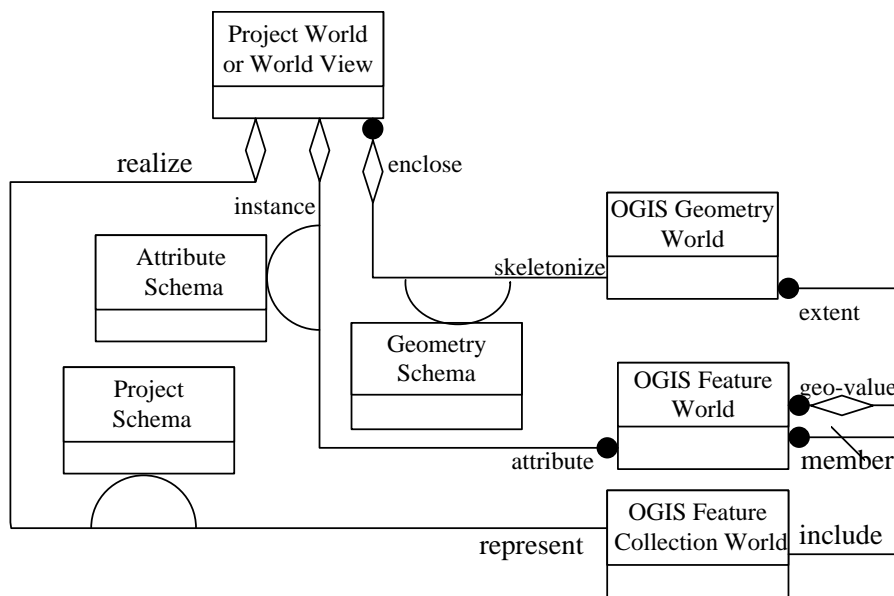


Figure 2-21 Navigation Near the Feature Collection World

Feature Schema are the key to Feature Collections.

Figure 2-21 can be simplified by collapsing the Geometry Schema and the Attribute Schema into the Feature Schema. Doing this yields Figure 2-22.

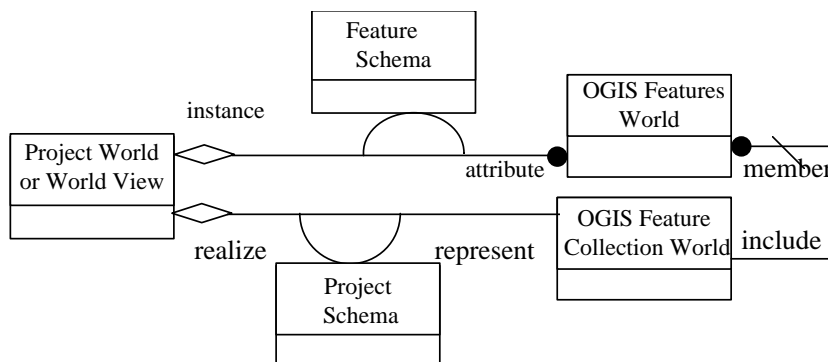


Figure 2-22 Feature Collections Consist of Features, their Feature Schema, and a Project Schema

2.10.2. Interfaces on Feature Collections

The interface between the Feature Collection World and the Feature World is called the Project Structure Interface. This interface is transparent, but it is derived from paths through the Project World. This is because the members of the Feature Collection World correspond directly to phenomenon in the Project World, not to a feature in the Feature World. Nevertheless, the methods across the Project Structure Interface, called *member* and *include*, are obvious.

The interface between the Feature Collection World and the Project World is called the Project Schema Interface. Its methods are also obvious, and are called *realize* and *represent*.

2.10.3. Background Notes on Feature Collections

The Open GIS Consortium has not yet achieved consensus on many issues surrounding Feature Collections. On the one hand, perhaps Feature Collections are not needed at all. This is because:

- a feature can be a composite of other features, connected by relationships of a particular type (see Topic 8)
- a “tile” may be a feature composed of (related to) the features it contains
- a feature may have one of its geometry values “divided” by tile boundaries, and thus be split into one feature with two geometry values or two features each with part of the geometry, yet need to be “reassembled” on demand by an interface on Feature or by a service.

On the other hand, the real world seems full of Feature Collections that need to be addressed. These include:

- projects, which have assignment boundaries and feature capture criteria and thresholds
- products from Government agencies, such as VPF, ADRG, SDTS, ATKIS, and similar files
- GIS database files
- the persistent or non-persistent ad hoc collection of features present at any moment in a GIS workspace.

Feature Collections seem to need important interfaces in order to support the needs of Catalogs and Catalog Services. These interfaces seem to be tightly coupled with Feature Collection Metadata. Formal definition of feature identifier scope is required, and every scope implicitly has an association with the collection of features whose identifiers it can resolve.

2.11. Summary of the Nine Layer Model

As a summary of the preceding sections, and as a reformulation of Figure 2-2, we present Figure 2-23.

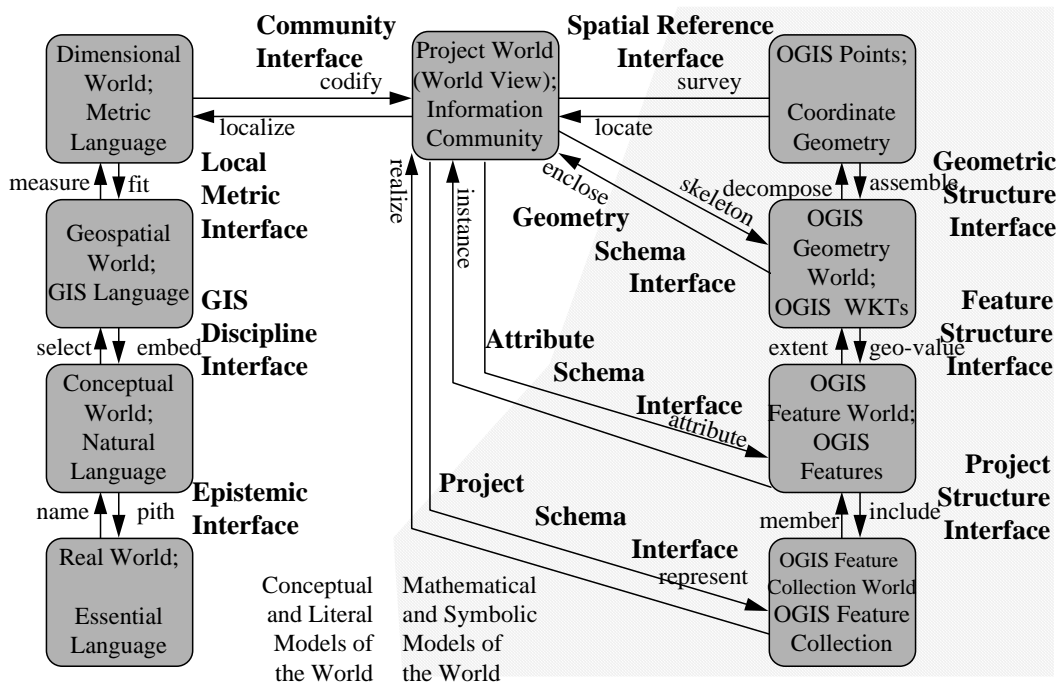


Figure 2-23 Nine Layers of Abstraction with Additional Interfaces

Using the notation of [1], we present Figure 2-23.

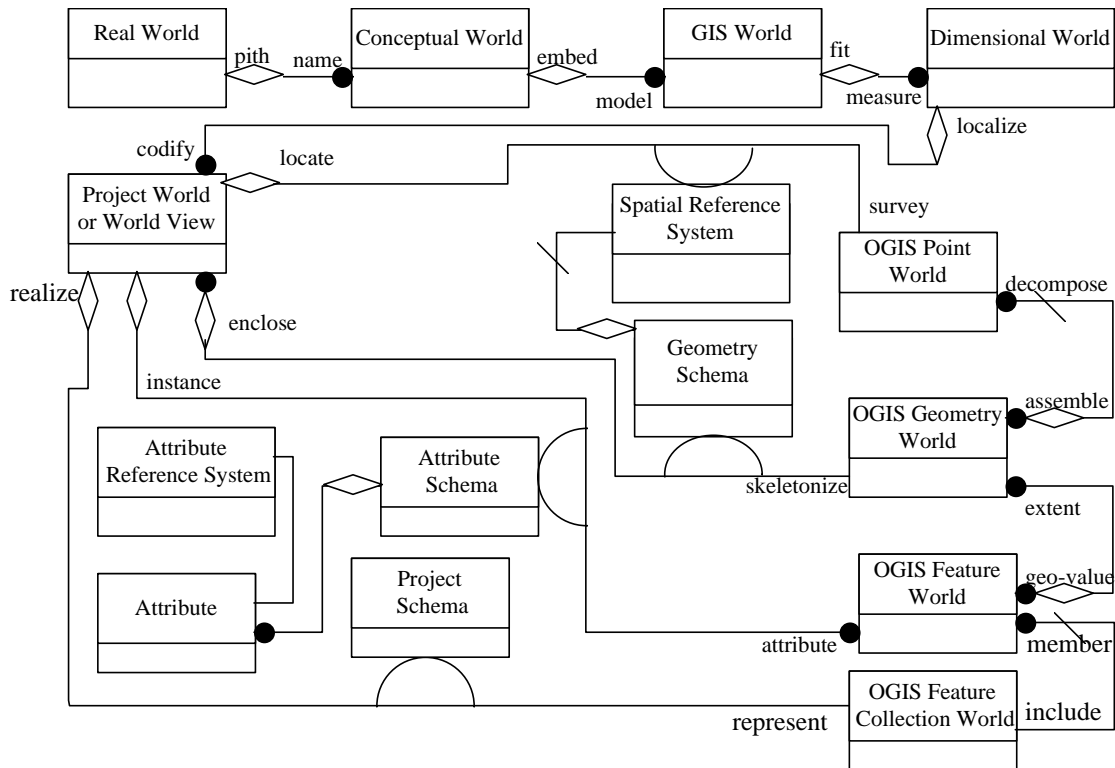


Figure 2-24 The Object Types of the Nine Layer Model

2.12. An Alternative Perspective to the Nine Layer Model

Figure 2-25 presents an alternative view of the various ways of defining multiple worlds, but from a language perspective. This view is equally valid but different. In particular, the nine-layer model implies that the essence of all features is tied up in their spatial extents (they are all drawn from a 'Project World' which is a 'codification' of the 'Dimensional World'). However, we define features to be able to have many spatial extents - or none - and where the spatial extent of the feature may be of less importance than its relationships to other features (see Topic 8).

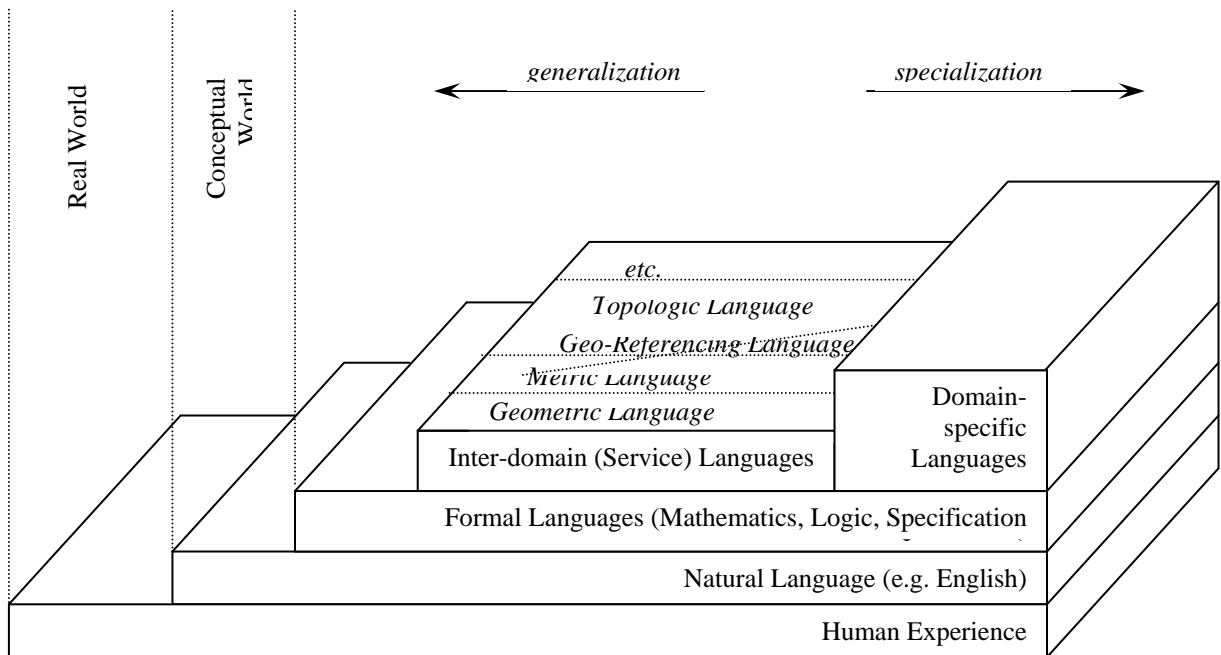


Figure 2-25 Conceptual Model of the ‘Real World’ modelling process

2.13. Persistent Feature Identifiers

2.13.1. Problem Statement

Managers of complex geographic information systems require capabilities for incremental update, distributed update, copying and partially modifying data. These capabilities require a more careful examination of the issue of feature identity than has previously been achieved.

Identity has implementation, logical, syntactic and philosophical aspects. A complete analysis is not attempted here.

Some handling of feature identifiers is necessary in order to represent relationships between features. This is discussed in Topic 8 of the Abstract Specification

A troublesome issue in the community is the lack of ability to support value-added information in a cost-effective manner [Hair97]. Value added information is an extension of a primary database which needs persistent feature identifiers in order to allow:

- additional information to a base dataset is required for a one-off or transient application
- a base of information exists but there is a long term need for additional information that has not been captured in the past
- partnerships are established based on areas of expertise but sharing some common data, e.g. between telecommunications and electricity utility companies
- incremental updating and incremental publishing
- maintenance of complex or aggregate objects (currently using relationships, see Topic 8).

- Reference to a geographic feature within a repository from outside the repository and perhaps outside any computing system, e.g. as a string of text in a hand-written letter.

We also need persistent, immutable feature identifiers in order to support:

- Tracking of feature identity during updates and collaborative working
- Proper management of versions
- Lineages of data transformations

The differences between geographic feature identifiers and many other types of persistent identifiers are of scope and scale. A single geospatial data repository may contain hundreds of millions of features. CORBA Persistent Object Identifiers are intended to identify executing software objects in much smaller numbers. Similarly, the IETF Unified Resource Names are designed for smaller numbers of objects. However, the design of IETF URNs may be appropriate for geographic features even if current implementations of identifier resolvers may not be [URNDNS].

2.14. Resolving Scoped Feature Identifiers

2.14.1. *Fundamental Ideas*

Every feature has one feature identifier (name) within its immediate scope (namespace).

A feature identifier is persistent and immutable.

A namespace, a "Scope", in which an identifier can be resolved, is itself a software object and does itself require a permanent identifier.

Scopes can be nested, but do not form a strict tree: a directed acyclic graph (DAG) is possible.

At the "bottom" are the leaf-objects, which are explicit identifiers which do not then need resolving in any other scope.

At the "top" we need to have some "Well Known Scopes" written into the implementation specification.

Leaf objects can be feature identifiers, scope identifiers, or any other type of identifier. We just deal with feature and scope identifiers here. Note that we only deal with identifiers.

Using a fully-resolved "bottom" leaf feature identifier to actually retrieve a feature's data is access not resolution and is a different issue.

2.14.2. *Key concept*

Scope is a permanent object with a persistent, immutable name that allows it to be located indefinitely, which can answer queries about identifiers "in its scope": resolving them to more explicit identifiers.

Finding a "scope object" is thus necessary. CORBA POAs and IETF URNs would be appropriate for the identifiers of scope objects.

2.14.3. *Key supporting concept*

Any scoped identifier (simply called an "identifier" or "name") is conceptually and perhaps actually constructed in two parts:

- A suffix, which may be opaque or may be an identifier in some grammar.

- A prefix, which defines the scope in which the suffix has meaning.

There is also always a third, invisible, part: the scope in which the whole thing, prefix and suffix, should be interpreted. This is because an identifier is always "inside" a scope.

2.14.4. Principles

Longevity, delegation and independence [URNres].

Identifiers must be persistent (last a long time)

It must be possible to delegate both authority and responsibility for both naming and resolution

The various delegations must be capable of being independent of one another

There are three aspects to naming: identification, location and mnemonics (semantics) [URIres].

The resolution system must be separate from the way identifiers are assigned [URNDNS]. This separation allows multiple naming approaches and resolution approaches to interact and to compete.

These identifiers are generally distinct from Human Readable Names (HRNs). However, a Scope can be defined in which identifiers are Human Readable, and then HRNs can be resolved in the same way as any other scoped identifier.

We need to be able to "grandfather in" existing naming schemes and Scopes [URNDNS].

Resolver services must be able to support scaling in three dimensions: number of features, number of data publishers, and complexity of delegation [URIres]

"One cannot make global, systemic guarantees except at an expense beyond reason." [URIres].

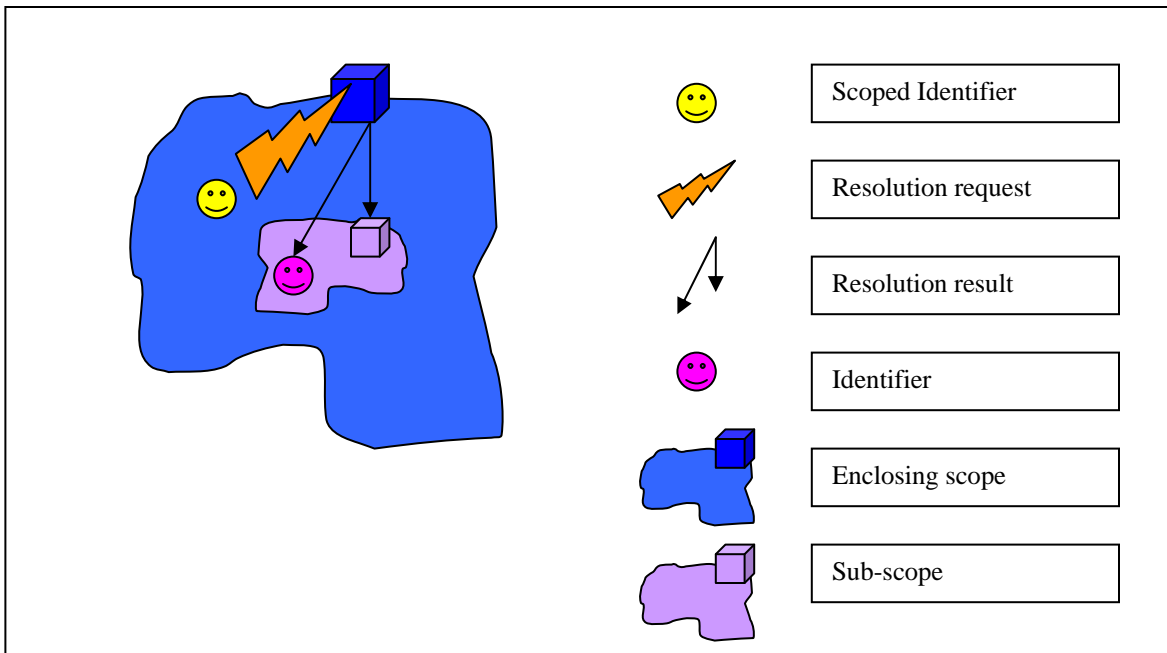


Figure 2-26. Resolution of a scoped identifier in a scope yielding an identifier inside an enclosed scope.

2.14.5. Resolver Services

Resolvers and similar systems can offer several services. The following list is taken from IETF work which use the term "name" for "identifier", "namespace" for "scope", and "characteristics" for "metadata" [URNDNS]:

N2L	Given a name (identifier), return a location (URL)
N2Ls	Given a name (identifier), return a set of locations (URLs)
N2R	Given a name (identifier), return the feature ("resource") itself
N2Rs	Given a name (identifier), return multiple instances of the feature ("resource")
N2C	Given a name (identifier), return its characteristics (metadata), e.g. a URC. Characteristics are always potentially a list (plural).
N2Ns	Given a name (identifier), return all the names that are identifiers for that feature
N=N?	Are these two names identifying the same feature?
L2R	Given a location (URL), return the feature ("resource")
L2Ns	Given a location (URL), return the names that identify that location
L2Ls	Given a location (URL), return a list of locations that contain the same feature
L2C	Given a location (URL), return its characteristics (metadata), e.g. a URC.
L=L?	Are these two locations the same?
R=R?	Are these two features the same? (Assuming they have different names)

Figure 2-27 Resolver Services

Notes for Figure 2-27:

- Using the above naming convention, "C2N" would be a metadata query, not a resolver query.
- N2C, even at the "lowest" level scope, is often all that can be achieved if the "resource" is off-line or in hard-copy form only.
- For the purposes of feature identification, we need all scoped identifier resolvers to offer only the following service: the critical service N2Ns (which we expect usually to return a single new name in a "lower" scope).
- For the purposes of feature identification, we need the "lowest" scoped identifier resolvers to offer only the following services: N2R, or N2Rs, and/or N2C as appropriate.
- This list is not exhaustive and many more services can be envisaged. Rather than freeze the set of services as function calls in a standard Interface, it would be

preferable to set up an architecture whereby new services could be registered, discovered and advertised. This is the approach which will eventually be followed by IETF [URIs], but at present such an approach would be premature.

2.14.6. *Further Example*

"The Handle System®" from CNRI is an example of a 1-level scope system. An identifier looks like this:

`hdl://cnri.test/abcd/efg/ijk#123`

Where:

- the suffix is: `abcd/efg/ijk#123`
- the prefix is: `hdl://cnri-test/`
- and the scope in which it makes sense as an identifier is The Handle System® which exists in within the Internet.

The Handle System includes a large number of scopes, but all of them have "`hdl://`" as the first part of their prefix.

2.14.7. *Hints*

The above example illustrates another useful but non-essential characteristic: the invisible scope in which the identifier should be resolved is indicated in a non-formal way, a "hint" [URNres] by the initial part of the text string of the prefix. Any informed person reading "`hdl://`" has a pretty good idea that we are dealing with a variety of URL (a URI to be pedantic [URI]) and we need to find which protocol "`hdl:`" represents. This capability is most use at the "top" level when we have to decide which Well Known Scope we have to start with. Lower scopes could be pure binary for prefixes and suffixes.

Hints are much more general facility than simply being an interpretation on prefixes; they are "anything that helps in the resolution of an identifier" [URNres]. Hints may themselves have metadata attached to them and may be an intermediate step in the resolution of an identifier. However they are also only hints: they may be out of date, temporarily invalid etc.

2.14.8. *Well Known Scopes and URNs*

A Well Known Scope is a Scope Environment, e.g. X.500, Corba Locator Service, DNS (machine identifiers), URL (file identifiers if the location is permanent).

Other Scope Environments might be:

- a FeatureCollection
- a database
- a computer
- a geodata repository service
- a website
- URNs

These can be self-referential and mutually referential. Within any scope environment there will be many resolvers. Each environment will need to have the services of a Resolver Discovery Service (RDS) - but the RDS for one environment does not itself have to be hosted in that environment, e.g. an RDS for X.500 resolvers could run in the WWW environment.

2.14.9. Uniform Resource Names

A URN always has the following syntax:

urn: <NID> : <NSS>

(i.e., urn: <namespace> : <suffixname-which-can-contain-colons>)

The standard names are NID and NSS: NID is the namespace identifier (scope identifier) and NSS is the namespace-specific string [URNres].

Thus the "invisible", third, enclosing scope for a URN is always visible because all URNs begin with the same literal string "urn:"

NSSs are expected to be partitioned into delegated and subdelegated namespaces where the delegated namespaces are free to choose their own syntax for the variable part of the namespace [URNres].

URNs can have access controls applied to their resolution.

If a scoped identifier is a Uniform Resource Name or URN [URN], there is a proposed mechanism already whereby resolvers for those names can be located using DNS [URNDNS] and HTTP [URNhttp]

The "hint" to find a resolver can be encoded in a rewrite rule inside Domain Name Servers [URNDNS]. Replicated repositories and resolvers can be supported by this means.

URNs are a specific design of scopes which can be arranged in nested scopes in a directed acyclic graph.

URNs need not include the more-resolved scoped identifiers lexically in the successive suffix parts (opaque or not), and they can support rewrite rules which enable automatic grammar-directed mapping between names (identifiers) in one namespace (scope) and another namespace.

Every resolution of an identifier (name) occurs in URN resolvers, possibly even in the same resolver software running on the same machine, but the path of resolution can trace out a directed .acyclic graph.

URNs already have had preliminary studies made on their security and privacy [URNres].

2.14.10. What resolution does

An identifier is resolved by giving it to the Scope object in which it is defined (the invisible one), which (conceptually) strips off the prefix, finds the Scope object that the prefix names, and hands the suffix to that Scope object. When the process reaches a leaf identifier, the stack unwinds and the leaf identifier is returned together with whatever information is necessary for the original enquirer to make use of it. This is a sequence on N2N operations.

2.14.11. Resolution, discovery and access

For software specification it is best to keep orthogonal functions specified entirely separately as distinct interfaces. Any implementation may choose to support one or several interfaces. We have three quite separate specifications to think about:

- Issuing or assigning scoped identifiers
- Resolving scoped identifiers N2N
- Discovering objects that match certain properties (catalog query using metadata) C2N

- Accessing data (data repository query), N2R

Experience with these types of systems has produced a major principle: that the resolution system must be separate from the way identifiers are assigned [URNDNS].

2.14.12. Handles and descriptors

We understand a feature descriptor to be something which is resolved by a Catalog Service. A feature descriptor contains "sufficiently unique metadata".

This discussion of scoped identifiers covers what we previously called a feature handle.

2.14.13. Permanent scope objects

Anything could offer an identifier resolution service within an "understood" scope, but we need to ensure that scoped-identifiers are only issued containing scope object prefixes for scope objects with certain minimum required qualities of permanence and immutability.

Catalog services are something which we could also cover by this same scoped identifier mechanism.

2.14.14. Uniqueness of identifiers

2.14.14.1. The same feature can be in several Scopes

There does not seem to be any way to avoid this since anyone can set up an identifier resolution service which then defines its own new scope.

Thus the same feature (a specific software representation of a real world object) can have several scoped identifiers. It will have the same leaf identifier (in the *immediate* scope of the feature), but this could appear in published form hidden inside several different scoped identifiers via different scopes.

Thus we lose equality by value for published identifiers; however we can define an equality method on identifiers if the method resolves them down to the lowest common denominator scope. Because scopes form a DAG not a tree, and because identifier length does not tell you anything about how many prefixes may be hidden inside opaque suffixes, it is impossible to tell how far down this is until the resolution is actually performed.

We need to make the following restriction:

2.14.14.2. An identifier cannot be a leaf in more than one scope

This is tricky, since data repositories can always be copied. So we need to think about how to define scopes for copies, and to consider replication possibilities to see if we can relax this condition under certain carefully-controlled circumstances.

2.14.15. Uniqueness and equality

The idea of equality is very specific to individual information communities. Often the same feature *does* require multiple names, even at the most fully-resolved level. The classic example is a weather map:

- The map of 26 October 1998
- Today's weather map

Both are useful names, and at some time they may both point to the same resource [URIRes]. A resolver may support equality of feature R2R as well as equality of fully-resolved identifier N2N, but if it does it will do it in an opaque way.

2.14.16. Internal identifiers and internal scopes

All identifiers discussed in this background description are published, external identifiers. Any repository can use any internal identification mechanism it likes so long as it can support permanent, immutable external identifiers.

As an example, consider a system which uses 8-digit numbers as internal feature identifiers. Such a system might want to be able to represent foreign identifiers (e.g. to implement some feature-feature relationships) and it might do so by internally using identifiers of the form "9xxxxxxx". The initial "9" indicating an external identifier. Such a system would then have a bit of software in its export subsystem which published its own identifiers as scoped-identifiers and replaced the foreign identifiers with the original foreign scoped identifier (stored in some local registry).

The interesting thing about this example is that the initial "9" can itself be considered to be a scope-prefix inside the proprietary system. This illustrates the general points that

- a scope "inside" another scope can actually be "larger" than the containing scope, and that
- a resolving service may only be able to cope with a subset of the identifiers which actually exist in a "sub"-scope.

The important property of an identifier is that it is resolvable, not that it is readable.

2.14.17. Scope aliases

Some scopes may exist only to provide short-forms of identifiers, e.g. in the above example a single digit "9" represented the entire outside world. A deeply-nested identifier could accumulate a very long string of prefixes, so within an organisation or information community, a scope resolver which provided short-form aliases could be useful.

Scopes can cross-reference each other, thus the opaque part of an X.500 scope may be a URN which resolves to a Handle System identifier.

2.14.18. Published identifiers

Any identifier published, e.g. in an email or quoted by some other piece of general purpose software for any value-adding purpose, must quote scope prefixes all the way back to the most-global Well Known Scope. This is where it is particularly useful if the prefixes understood by Well Known Scopes are not opaque but are readable as "hints". The examples of the hints "hdl://" and "urn:" have already been mentioned.

2.14.19. FeatureCollection

Conceptually, there exists a FeatureCollection of all the features whose identifiers are in the Scope, but

this FeatureCollection is not a Scope,

neither is a Scope a FeatureCollection,

they are just implicitly associated. The FeatureCollection may not be instantiatable even in theory for some Scopes.

2.14.20. Methods of Feature Identity Scope

A Scope Object has these methods:

- Given an identifier, a scope object returns a sub-scope identifier and a sub-identifier. ("Traverse" method) N2N
- Given two identifiers, a scope object resolves them to a common scope, obtains a canonical form, and then responds by saying whether they are equal or not. ("IsEqual" method) N=N

- Given an identifier, determine if it is a leaf identifier in this scope N2R?
- Given a leaf identifier, return the object. This is a repository service method, not a scope service. N2R
- Given a leaf identifier which we are pretty sure is a feature, not some other kind of identifier, return us the feature [as a feature handle?]. Again, this is a repository service method. N2R
- We give it an object {handle?} and ask for a scoped-identifier in the current scope. R2N
- We give it an object {handle?} and ask for a public, publishable scoped identifier that contains all the scopes out to a Well Known Scope. (The "ComposeId" method). R2N+!N2N

This last method is arguably implementable because we always know where we are in the scope sequence because we must always have come "down" some route through the scopes' DAG to get to the scope we are "in" now. Being able to produce a publishable identifier is clearly a requirement, how it is done should be left to responses to an RFP.

A Scope is probably an Interface not a Class, (using Java nomenclature), i.e. the set of Scope methods could be supported by many different objects of different classes.

It has been provisionally decided that we do not want or need a method *on Feature* where you give it a scope and ask what its publishable identifier *would* be from that scope.

2.15. Feature Identifier Change Registries: Incremental Publishing

2.15.1. Conflict

Incremental update requires permanent feature identifiers, but also intrinsically means that some identifiers will not be permanent because they will be deleted ("retired").

This conflict can be managed if designed properly. But we cannot suggest a solution which requires any kind of centralised international database.

"In case that the source supports versioning, a registry of the relationship between the ID of the original version and that of subsequent ones should be maintained. [...] Case 4:... the client requires to be notified of any updates on the retrieved objects. A broadcast mechanism can be designed such that the source sends an update alert on the network." [\[Bishr99\]](#)

"These references include such things as value-added attributes, additional feature relationships, etc. While it may be possible to resolve all of these references immediately in a small centralized database, it is intractable considering the number of geospatial databases and the volume of features currently in existence. Therefore, one must employ a technique that supports on-demand resolution of these references at any point in the future. This technique must also support tracing the lineage of a feature back in time through changes in delineation, or forward to the present from some historical date." [\[Hair97\]](#)

We address those feature identifiers which would be used to manage incremental updates. These are almost certainly the same identifiers used to implement relationships (see Topic 8). Compound objects are assumed to be implemented using relationships. We assume a scope mechanism for resolution of scoped identifiers.

2.15.2. Fundamental Ideas

When we have permanent, immutable feature identifiers which are to be used by third-party software, we must have some registry somewhere which records their replacements by updated identifiers.

Separating the specification of read-only access from update-editing simplifies things.

Hierarchical decomposition across scopes in which identifiers are managed to be consistent is one good way to make identifier registries scalable and workable.

Resolution and clean-up, across a local domain, should be done as often as is sensible in order to keep registries manageably small.

2.15.3. *New concept*

The concept of published- versus dirty-feature identifiers is introduced.

This turns out to be sufficient. We do not need to specify the full semantics of a formal system to specify long-transaction or version control architectures.

2.15.4. *Incremental Publishing*

A key idea is that incremental publishing is what we want to achieve. We do not need at this time an all-singing, all-dancing global peer-to-peer transactional universal historical feature identifier monster.

This section is structured like this:

1. Incremental publishing requirement
2. Distributed edit/update requirement
3. Both together.

We can start simply, with read-only incremental updates to read-only GIS clients.

We then go on to consider a tight community which is collaboratively updating a GIS dataset, e.g. within a single cartography publisher's organisation. This has lower priority within OGC at this time.

We then show that these two architectures can be composed in one specific way which works and has good global properties. There may be other ways of carefully composing these two architectures.

2.15.5. *Incremental Publishing*

With incremental publishing, there is a notional single master database, the "publisher" which periodically issues incremental updates. The clients have read-only copies of the data which they can use in various value-added ways [Arctur98]. The thing they do that is relevant to this discussion is that they create references to feature identifiers in other software, web-pages, other GIS packages etc.

Example: a hydrographic agency distributes CD-ROMS of charts to pleasure-boat owners with daily "notices to mariners" broadcast using the GSM telephone short-message service. Some boat owners have add-on software which presents the data as a 3D visualisation and correlates it with on-shore harbour information purchased from a yachting club.

Thus every feature identifier which is sent out from the master is published and the master takes responsibility for ensuring that it continues to be useful so long as clients follow some simple instructions and implement a local registry.

Assume for the moment that the client only works within a closed group and no reference is made to the data except via the client, i.e. a strict tree.

The client local registry contains an index of every feature identifier that the client has used in its value added activities.

When an update packet arrives from the master, it contains the geodata update in some file format and a feature identifier registry update, probably in some XML encoding. This

says which identifiers are being split, merged and deleted [Arctur98]. (It probably lists all the new ids too.) If the client has used any of these IDs, it has responsibility for:

1. either finding where they were used and replacing them,
2. or since the client always services the geodata requests for its value-added activities, it can resolve outdated ids on-the-fly when it services the request.

When we resolve updated identifiers we have a choice:

1. replace all references to the old identifier with references to the new identifier, or
2. if the GIS can handle it, add the new object, create a relationship between the old and new object, and update those references to the old object which require the "latest" copy but keep references to the old object if they are intended to point to that old object.

The second option is one that often happens when there is a mixture of archive data which must be kept consistent, e.g. compound objects represented using relationships, and current data which must reflect the current configuration. Other software systems distinguish the two by defining this behaviour as an aspect of the relationship type, like cardinality or bidirectionality.

2.15.6. Non-tree client network

Now we relax the condition that access to the geodata is through a strict tree. This is more than we need at this point in OGC since our user-base wants controlled incremental publishing first.

Any set of clients which are synchronized in their updates can use each others value-added software because the feature ids will be identical. [All clients treat the geodata as read-only, remember.]

Now consider that one of these clients has created some additional information about an object whose geodata is in the public domain. That client publishes a web-page containing feature identifier references. A reader of that web-page may not want to go to that client to get the public data, but wherever the reader goes to get it may not be synchronized so some identifiers will be meaningless.

We can fall back on a universal master in this case (which could be replicated). It would have to have interfaces appropriate for readers coming to it with identifiers from any one of the historical update packages. If we say that clients must always quote their current update package level ("patch number") whenever they quote a set of identifiers, then the reader will know that and can get the appropriate resolution and thus correct data from the master.

Note that when dealing with read-only data it is always possible to set up replicated servers and client-side caches to improve scalability and performance.

2.15.7. Distributed Editing

Whereas incremental publishing must be able to cope with clients which may number in hundreds of thousands who are largely out of control, our current requirements for distributed editing involve a few tens of editing clients which are under close control.

It is the editing process which splits, merges and creates identifiers.

Example: a mapping organisation has 20 groups working on different segments of data. These segments may be defined by tiles, irregular spatial boundaries or by theme (feature class).

Each editor checks out a writeable segment from the common master database and works on it, checking versions of his segment back in from time to time. Each editor creates dirty feature identifiers which it has to reconcile with the master. It may reconcile its

segment by keeping a copy and updating the identifiers with those it gets from the master, but more likely it will just delete its local segment and re-check-out a new version of the segment from the master.

After a while all editors have resolved their changes back into the master. At this point the master can be published. [Note that this means that the version network has achieved closure. This requirement will be relaxed later.]

All the dirty identifiers never leave the organisation in which they were created and no editing client performs any value-added activity in which any identifier is used at all. As segments are checked back in, new identifiers are either kept or renumbered if they conflict with another, but they are never clean until the dataset as a whole is published (perhaps as an incremental update, perhaps as a whole).

2.15.8. Responsibilities and compositions

So long as a client operates on an identifier strictly according to either the read-only incremental publishing protocol or the distributed editing role, it can do both at once. So some of its identifiers (those it got from an incremental update from the master) are published and some (those it creates itself) are dirty. It can use the published ones in value added activities, but it can't use the dirty ones. It has to wait until it gets them back from the master in published form.

2.15.9. Local cleanliness

Alternatively, the editing client can use its own dirty identifiers if it takes full responsibility for them for its sub-clients who use them, i.e. it acts like a master which a whole subsidiary architecture of feature identifier update packets and identifier registries at its sub-clients. If you issue a dirty identifier, it is your responsibility to clean up afterwards.

This introduces the notion of "local cleanliness". A sub-subclient would not be aware that it had a dirty identifier, and indeed, so long as it only talked to other sub-clients of the same client (or to its master - which we call an editing-client), the identifier would be effectively published. It only appears dirty outside that little group. Everything is fine so long as that little group achieves local closure, i.e. eventually the editing client reconciles itself with its master and then cleans up all its sub-clients: then the sub-clients can be normal clients of the master.

Dirtiness is thus a relative, not an absolute concept: it is only dirty outside the "isolation ward". This is OK so long as we maintain our context properly which is why we do need some Uniform Resource Name (URN) or handle system to keep track of who we are talking to. DNS itself would be sufficient (if awkward) because it can define machine aliases, but the extra level of indirection from a URN or handle system is well worth it [Sargent99].

2.15.10. Out of control copying

How do we cope with out-of-control copying of data as we might find in the open Internet ?

This is the situation where we cannot assume that a client has had all (or any) of the interim update packets, or where anyone copies a dataset from a client without re-registering with the master. This is fundamentally always possible, even if we were to devise some Kerberos security architecture, if we do not use military-style no-write-down no-read-up controls which are impossible.

This is the same problem as the "non tree client network" we discussed above, with the same solution, unless the uncontrolled copy was made of dirty data. In that case the original master won't be able to help, and the edit-update client may be unaware of the copy and may have reconciled all its sub-clients and deleted all historical records of dirty

identifiers. It will always delete them because that is the whole point of reconciliation, to save space in what otherwise would be an exponentially expanding identifier list.

This remaining unresolvable problem will always be with us in some form: someone can always take a copy of data, change it in an arbitrary way, then give it to someone else and vanish. Thus the fact that we can't deal with this case is not important as no system can deal with it.

2.15.11. *Handles for servers*

The handle (URN) for a master which issues update packets needs to be written into the header of the packet so that a client which needs the next update always knows where to find it.

The individual identifiers in the packet should also be annotated with their server's URN if they are being passed on from another master server. That is the same thing as saying that an update-editing client which put out its own dirty identifiers would use its own URN as a prefix for the dirty ones, while using its master's URN as a prefix for the published ones. Thus a sub-client within an "isolation ward" would in fact be able to tell the difference between globally published and locally published (globally dirty) identifiers.

2.15.12. *Identifier update packet*

What would an update packet look like ? The geodata part would be some established file-format which had feature identifiers, e.g. an object form of SDTS [Arctur98], or a set of OGC Simple Feature transactions. The identifier part of the packet might look use some XML language.

2.15.13. *Relationships*

Between which objects can we support relationships ?

Easily between objects which are in the same update packet.

Also between pre-existing objects and objects in the update packet after published identifiers have been resolved. Though if we have bidirectional relationships, this may be the same thing as the previous point since related objects would be updated.

Any relationship which uses a dirty identifier is a dirty relationship and subject to the same usage constraints.

2.15.14. *GIS requirements*

What must a GIS be able to do in order to participate in this scheme?

Simple. The minimum and maximum requirements are the same: it must support permanent, immutable feature identifiers within itself. Everything else, the identifier registries etc., can all be handled by external, add-on software which translates the identifiers into some internal form for the GIS and which can handle URNs to talk to master servers and read update packets.

2.15.15. *Last word*

The version control architecture described here is over-simple with a single binary distinction between published and dirty identifiers. OGC will eventually need a proper long-transaction protocol, version semantics etc. This is a short-cut: everything which really should be logically distinct and separately specified at different levels of abstraction has been bundled together for the sake of speed and simplicity. This is not a long-term solution, but it may be good enough for now.

2.16. References to Section 2

- [1] [Cook94] Cook, Steve, and John Daniels, Designing Object Systems: Object-Oriented Modelling with Syntropy, 1994, 389 pp. Prentice Hall Press; URN:ISBN:0132038609
- [2] [OGCSF] Open GIS Consortium, 2006. The OpenGIS® Simple Features Implementation Specification Parts 1 and 2. Available at <http://www.opengeospatial.org/standards/>
- [3] <No longer applicable. Part of the TC Policies and Procedures>
- [4] [Bishr99] A Globally Unique Persistent Object ID for Geospatial information Sharing, Yaser A. Bishr, Interop'99 submission.<http://www.opengis.org/members/fid.wg/index.htm>
- [5] [Sargent99] Feature Identities, Descriptors and Handles, Philip Sargent, Interop'99 submission.

- [6] [Arctur98] Issues and prospects for the next generation of the spatial data transfer standard (SDTS), David Arctur, David Hair, George Timson, E.Paul Martin, Robin Fegeas. IJGIS (1998) 12 (4) 403-425.
- [7] [Hair97] Feature Maintenance Concepts, Requirements, and Strategies, Version 3.0 May 28, 1997, David Hair, EROS Data Center, George Timson, Mid-Continent Mapping Center, Paul Martin, Rocky Mountain Mapping Center. Published by U.S. Geological Survey/National Mapping Division. <http://wagic.wa.gov/Framework/cadastre/presentations/acsm1999paper.htm>
- [8] [UML] Unified Modeling Language Documentation Set Version 1.1, Santa Clara, California.

- [9] [Z39.50] Information Retrieval (Z39.50): Application Service Definition and Protocol Specification, ANSI/ISO Z39.50-1995, Official Text, July 1995, Z39.50 Maintenance Agency.
- [10] [LDAP] K. Zeilenga, "Lightweight Directory Access Protocol: Technical Specification Road Map ", Internet Engineering Task Force (IETF) Request For Comment (RFC) 4510, 2006. <http://tools.ietf.org/html/rfc4510>
- [11] [URNDNS] Daniel, R., Mealling, M.: Resolution of Uniform Resource Identifiers using the Domain Name System, Internet Engineering Task Force (IETF) Request For Comment (RFC) 2168, June 1997, <http://www.ietf.org/rfc/rfc2168.txt>
- [12] [URNhttp] Daniel, R.: A Trivial Convention for using HTTP in URN, Internet Engineering Task Force (IETF) Request For Comment (RFC) 2169, June 1997, <http://www.ietf.org/rfc/rfc2169.txt>
- [13] [URN] Moats, R.: Uniform Resource Name Syntax, Internet Engineering Task Force (IETF) Request For Comment (RFC) 2141, May 1997, <http://www.ietf.org/rfc/rfc2141.txt>
- [14] [URNres] Sollins, K.: Architectural Principles of Uniform Resource Name Resolution, Internet Engineering Task Force (IETF) Request For Comment (RFC) 2276, January 1998, <http://www.ietf.org/rfc/rfc2276.txt>
- [15] [URires] Mealling, M., Daniel, R.: URI Resolution Services Necessary for URN Resolution, Internet Engineering Task Force (IETF), March 1999. <http://www.ietf.org/rfc/rfc2483.txt>
- [16] [URI] Berners-Lee, T, Fielding, R., Irvine, U.C., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax, Internet Engineering Task Force (IETF) Request For Comment (RFC) 3986, August 2005, <http://www.ietf.org/rfc/rfc3986.txt>
- [17] [Shklar97] New approaches to cataloging, querying and browsing geospatial metadata, L.Shklar, C.Behrens, E.Au, IEEE Metadata Conf., 1997. <http://computer.org/conferen/proceed/meta97/papers/lshklar/lshklar.html>

[18][UMLguide] The Unified Modeling Language Guide, G.Booch, J. Rumbaugh, I. Jacobson, Addison-Wesley 1999. <http://www-01.ibm.com/software/rational/>

3. The Abstract Model for Feature, Feature Identifier, Identifier Scope, Identifier Change Registry, Feature Repository and Feature Collection

3.1. FT_Feature

3.1.1. FT_FeatureType

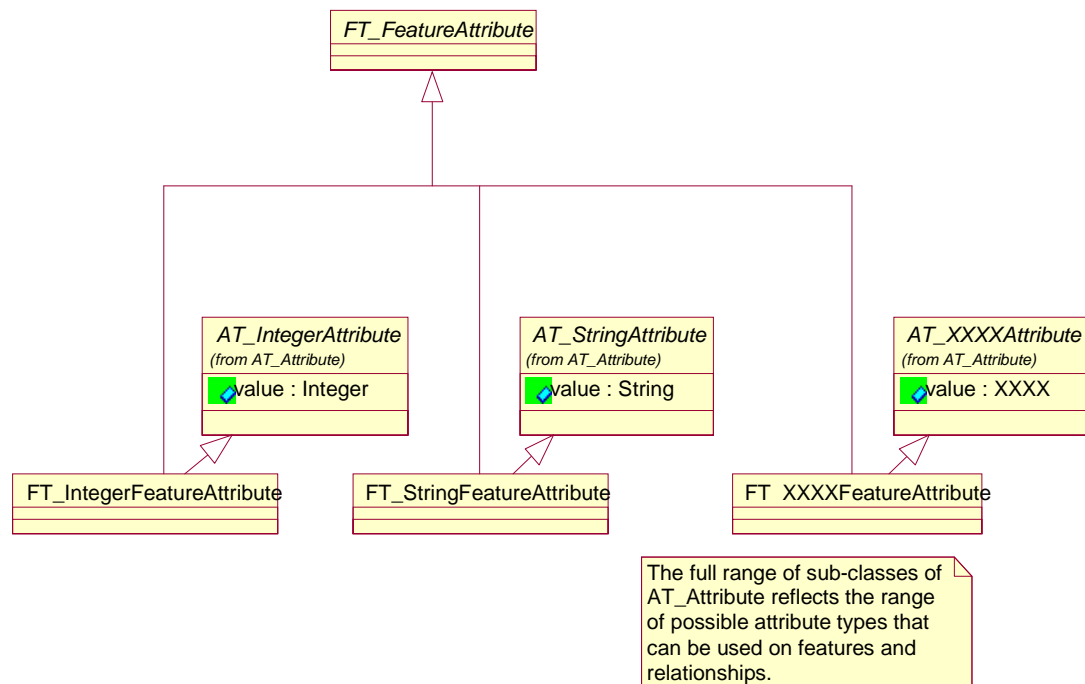
An FT_Feature is of one FT_FeatureType. An FT_Feature must yield its type on demand to an OpenGIS client in a ‘well known’ format.

The term “well known” in this context means defined using some means commonly understandable by OpenGIS clients. This could be explicitly defined in the implementation specification but preferably some means available through the underlying distribution technology will be used .

An FT_Feature may yield its type (when demanded) directly or by passing a reference to a ‘FT_FeatureType’ object.

3.1.2. FT_FeatureAttribute

An FT_Feature has associated FT_FeatureAttributes. Each FT_FeatureAttribute is has a value within the valid domain of the attribute. Names and types of FT_FeatureAttributes



are defined by the classes in the AT_Attribute package they inherit from.

Figure 3-1 The FT_FeatureAttribute

A subset of the attributes of an FT_Feature may be geometric. This subset may, among other things, represent the spatial extent of the FT_Feature. This subset may be empty for FT_Features of some FT_FeatureTypes. In this specification, the UML takes precedence over text when the UML is more specific.

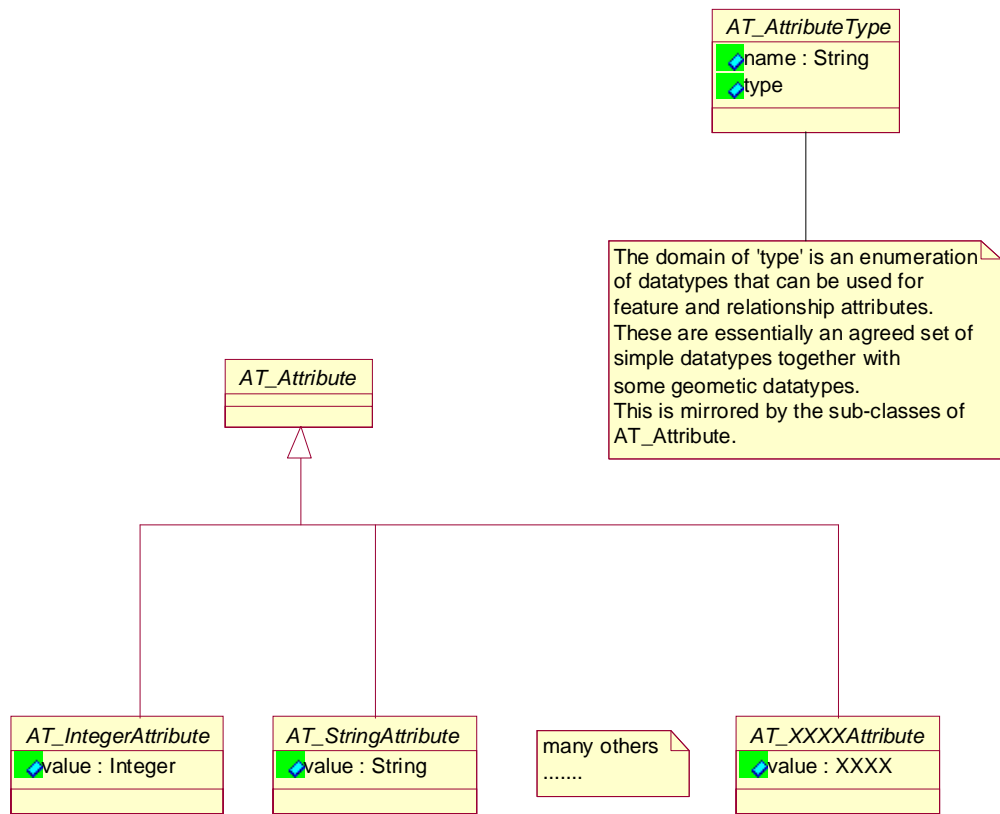


Figure 3-2 The AT_Attribute Package

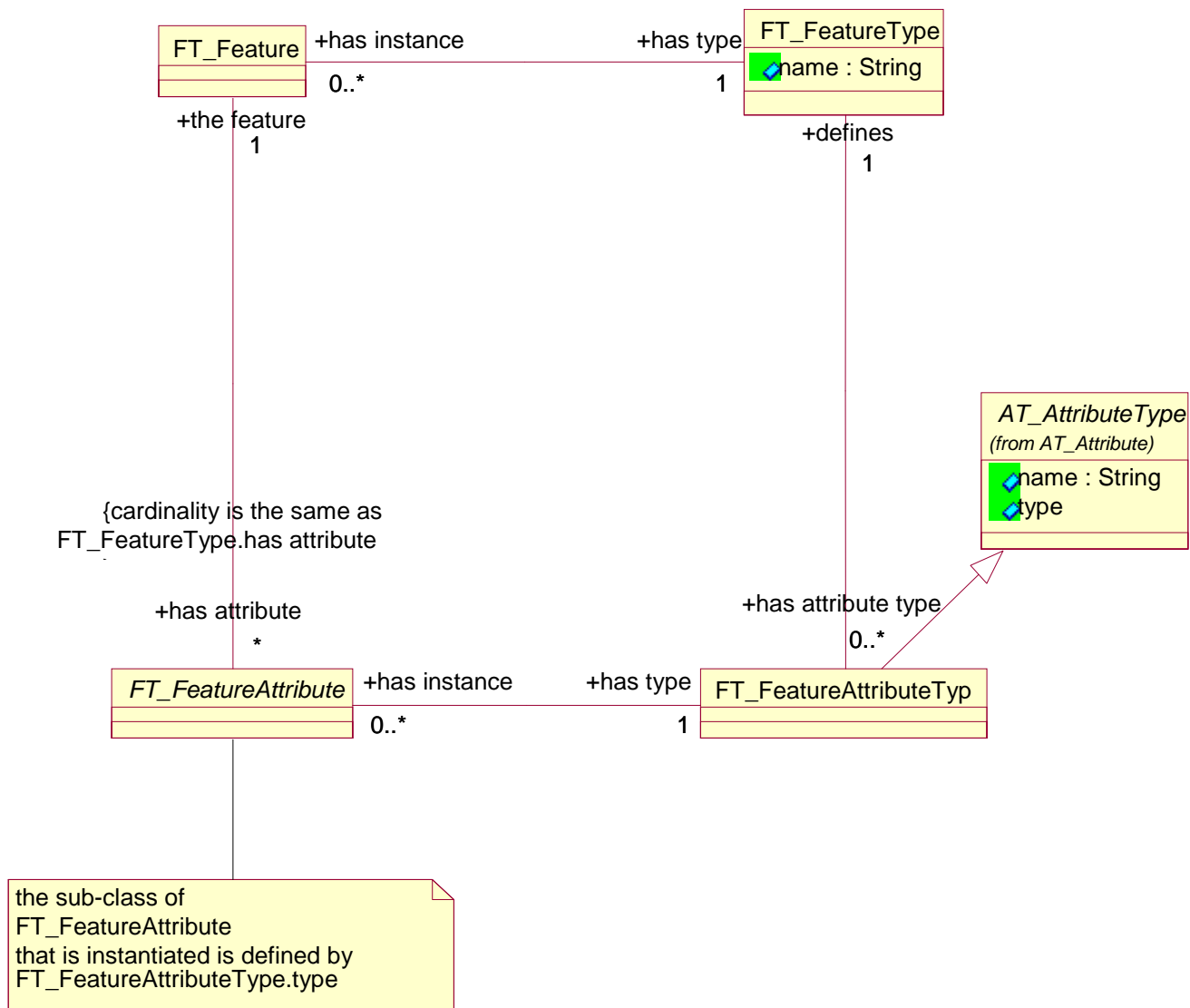


Figure 3-3 The FT_FeatureType

3.1.3. FT_FeatureIdentifier

An FT_Feature has a single identifier that is unique within a single FT_IdentifierScope and, in general, independent of the value of any or all of its associated attributes

Basing identity on a key of one or more attribute values, a technique sometimes used by RDBMSs, is not precluded. Such a technique, however, is not mandated. Many systems have a concept of identity that cannot be adequately addressed by a candidate key and this independence must be preserved across OpenGIS databases. Many of the uses of FT_FeatureIdentifiers, e.g. for incremental publishing or for relationships between FT_Features in different FT_IdentifierScopes, or the requirement for persistent, immutable identifiers, cannot be met by a candidate key approach without imposing severe usage limitations, e.g. read-only access forever.

Requiring unique identity within a Scope does not allow for such devices as 'temporary' or 'alias' identities within the same Scope. It does allow for identification of temporary FT_Features: e.g. identity for FT_Features that only exist (and are therefore only reachable) within the context of a database session, i.e. in another Scope with limited persistence.

3.1.4. FT_Feature Persistence

An FT_Feature is generally persistent. An FT_Feature can be "reached" (accessed or a route to access returned) from its FT_LeafIdentifier. If this is no longer reachable, then the FT_Feature, if reached by other means, e.g. a query on attribute values, has to be considered to be a different FT_Feature with a different identifier.

3.1.5. FT_Feature Instances

An FT_Feature may also be referred to as An FT_Feature Instance.

3.2. FT_Identifier

3.2.1. Characteristics of FT_Identifier

An FT_Feature's identity is represented by An FT_Identifier. This may be either a *leaf* identifier (FT_LeafIdentifier) or a *scoped* identifier (FT_ScopedIdentifier).

An FT_Identifier has meaning only within an FT_IdentifierScope (definition of identifier)

A published FT_ScopedIdentifier of an FT_Feature, together with the FT_ScopedIdentifier of the FT_IdentifierScope, can be used to refer to an FT_Feature or to assert an informal and uncontrolled relationship with an FT_Feature in a data repository which may have no management authority in common with the agency using the identifier.

Given an FT_LeafIdentifier, return some means of accessing or obtaining the object, e.g. a reference to or description of the repository holding the object.

Given an FT_Feature within this FT_IdentifierScope, ask for its FT_LeafIdentifier in this scope.

Given an FT_Feature within this FT_IdentifierScope, ask for a public, publishable FT_ScopedIdentifier that contains all the scopes out to a Well Known Scope. (The "composeId()" method).

Implement some procedure for maintaining consistency for published FT_ScopedIdentifiers through an FT_IdentifierChangeRegistry architecture or some equivalent mechanism.

3.4. FT_IdentifierChangeRegistry

3.4.1. Change Registry Characteristics

An FT_IdentifierChangeRegistry is a persistently available cache which records which FT_Identifiers were replaced or retired. It only need contain information on published FT_ScopedIdentifiers.

3.5. FT_FeatureRepository

3.5.1. FT_FeatureRepository Characteristics

An FT_FeatureRepository is an FT_FeatureCollection with certain responsibilities for the issuing and maintenance of FT_Identifiers.

An FT_FeatureRepository is an FT_FeatureCollection which implements repository functions.

Given an FT_LeafIdentifier which we are sure identifies an FT_Feature and not some other kind of object, return us the FT_Feature or an interface to it.

An FT_FeatureRepository manages access to FT_Features.

An FT_FeatureRepository collaborates with FT_IdentifierScopes and FT_IdentifierChangeRegistries (or similar) to maintain published FT_Identifiers when such identifiers change.

3.6. FT_FeatureType

3.6.1. FT_FeatureType Characteristics

Every FT_Feature instance has an FT_FeatureType.

Each FT_FeatureType has an FT_Identifier.

An FT_FeatureType is defined by a set of attribute definitions and role definitions (see Topic 8). Either set (or both) may be the null set.

Each FT_FeatureAttributeType definition has a name and a type (inherited from AT_AttributeType). The type is a defined domain of valid attribute values. Attribute types are either 'well known' or are able to expose their structure in a 'well known' format on demand. They may be simple basic types (longs, floats, strings, etc.) or geometries. They may not be FT_Feature identifiers (e.g. a reference to an FT_Feature of a particular type), such inter-FT_Feature references should use the Relationships facilities described in Topic 8.

An FT_FeatureType may also define a set of other FT_FeatureTypes for which it is substitutable. Whether this is achieved through single or multiple inheritance, aggregation

or some other means is an implementation issue. For example, if FT_FeatureType B is defined as being substitutable for FT_FeatureType A then each instance of Type B can be used wherever an instance of Type A is permitted, e.g. in a relationship (see Topic 8).

The FT_FeatureAttributeType set of an FT_FeatureType is a superset of the corresponding sets for all FT_FeatureType for which it is substitutable.

3.7. FT_FeatureCollection

3.7.1. Characteristics of FT_FeatureCollections

An FT_FeatureCollection is an FT_Feature instance that groups other FT_Features.

As an FT_Feature Collection is also an FT_Feature, FT_FeatureCollections have FT_FeatureType, FT_Identifier, an associated set of FT_FeatureAttributes. There are many examples of an FT_FeatureCollection where it seems counter-intuitive to describe it as an FT_Feature (e.g. scope of a query). The countering argument is that many of the mechanisms defined for FT_Features (attributes, type) are equally applicable to FT_FeatureCollections and that defining an FT_FeatureCollection as an FT_Feature allows for the construction of complex hierarchies or networks of FT_Features.

Uses of FT_FeatureCollections include the representation of a physical or logical repository of FT_Features (an FT_FeatureRepository), a complex or composite FT_Feature; the result of a query; the scope of a query; an ad-hoc collection created for a particular purpose. An FT_FeatureCollection representing a persistent complex or composite FT_Feature should use explicit FT_FeatureRelationships (see Topic 8) to represent this.

FT_FeatureCollections may be transient or persistent.

4. Future Work

- As Implementation Specifications evolve, this topic must be edited to maintain alignment with them.
- Reconcile this with the Topic 13 and the Catalog proposals.
- Much of the discussion of Feature in the first half of section 2 concerns Geometries, which are an attribute domain for Features. This material should be removed to Topic 1.
- The whole area of *schema discovery* needs to be tackled, e.g. what FeatureTypes are allowed in any dataset, what attribute domains are supported etc. This overlaps with the Catalog work (Topic 13) and is particularly relevant for Feature Relationships (Topic 8). When we describe schema discovery interfaces we must be careful not to rule out those GISs which support *dynamic schema updates*.
- The 9-layer model has significant flaws, in particular the level at which we put Geometry is not in accord with the specifications we wish to produce.
- An FT_FeatureRepository (Datastore) is an FT_FeatureCollection with certain responsibilities for the issuing and maintenance of FT_Identifiers. Do we need some constraints on the FT_FeatureTypes of FT_FeatureCollections ?
- More work on FT_FeatureCollections is needed if we wish to issue RFPs requiring them. What are the fundamental classes and subclasses of Feature Collection. How do they behave? What are the relations between them and FT_Features, Catalogs, Metadata, Schema, and other objects, and between themselves?
- In future, an FT_Feature may participate in various processes. The set of valid processes in which it may participate will be defined (directly or indirectly) by the FT_Feature's FT_FeatureType. This concept of process participation is intended to be broad enough to include object methods, static functions and stored procedures.

5. Appendix A: Well Known Structures

5.1. Well-Known Structures

Feature Geometry (that is, the AttributeValue for the AttributeName “OGCGeometry”) have their WKS detailed in Topic 1: Feature Geometry. [1]

Coordinate reference systems will have their WKS listed in Topic 2: Coordinate reference Systems. [2]

Other WKS associated with Features are to be found in the Implementation Specifications for OGC Simple Features. [3]

Identifier Update Packets may need a WKS; which is more likely to be an XML encoding than a binary format.

5.2. References to Appendix A

- [1] Open Geospatial Consortium and ISO, 2001. Abstract Specification Topic 1: Feature Geometry. Available at <http://www.opengeospatial.org/standards/as>
- [2] Open Geospatial Consortium and ISO, 2004. Abstract Specification Topic 2: Spatial Reference by Coordinates. Available at <http://www.opengeospatial.org/standards/as>
- [3] Open Geospatial Consortium, 2006. The OpenGIS® Simple Features Implementation Standard Part 1 - Common Architecture. Available at <http://www.opengeospatial.org/standards/sfa>

6. Appendix B. The ISO TC211 General Feature Model

The ISO TC211 15046 Part 9: Rules For Application Schema standard [1] describes a general feature model that places the geometry in the context of its use to describe the location of features. Although this model does not overtly affect the geometry model, it does form the basis for assumptions on the use of this model in individual application implementations. Figure 6-1 places the feature packages in context with the geometry packages.

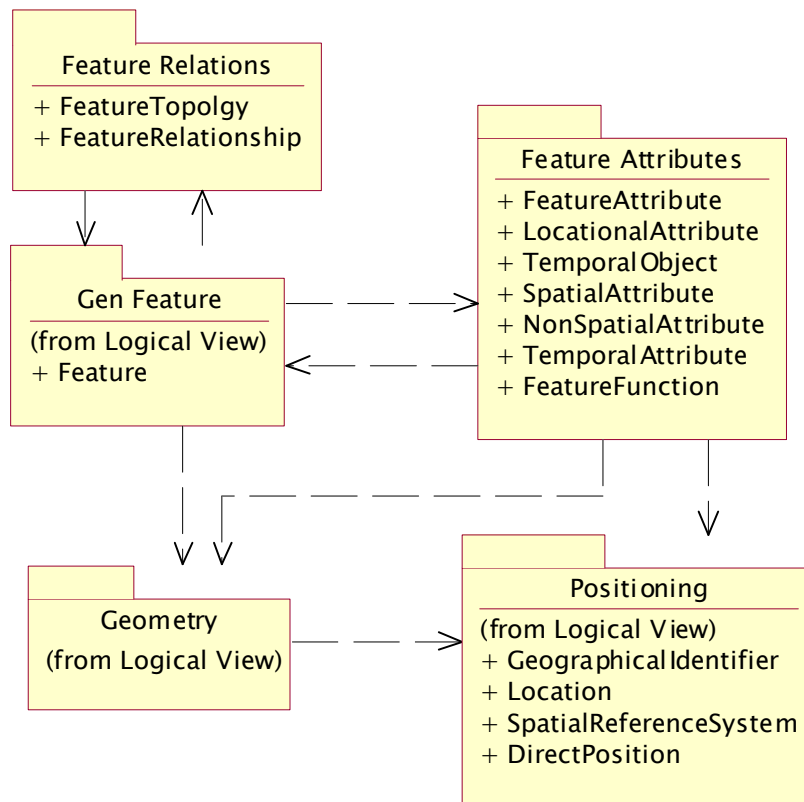


Figure 6-1. Feature Model Packages (from Rules for Application Schemas)

Figure 6-2: General Feature Model (from Rules for Application Schemas) details the internal structure of the feature packages as derived from Part 9: Rules For Application Schema [1].

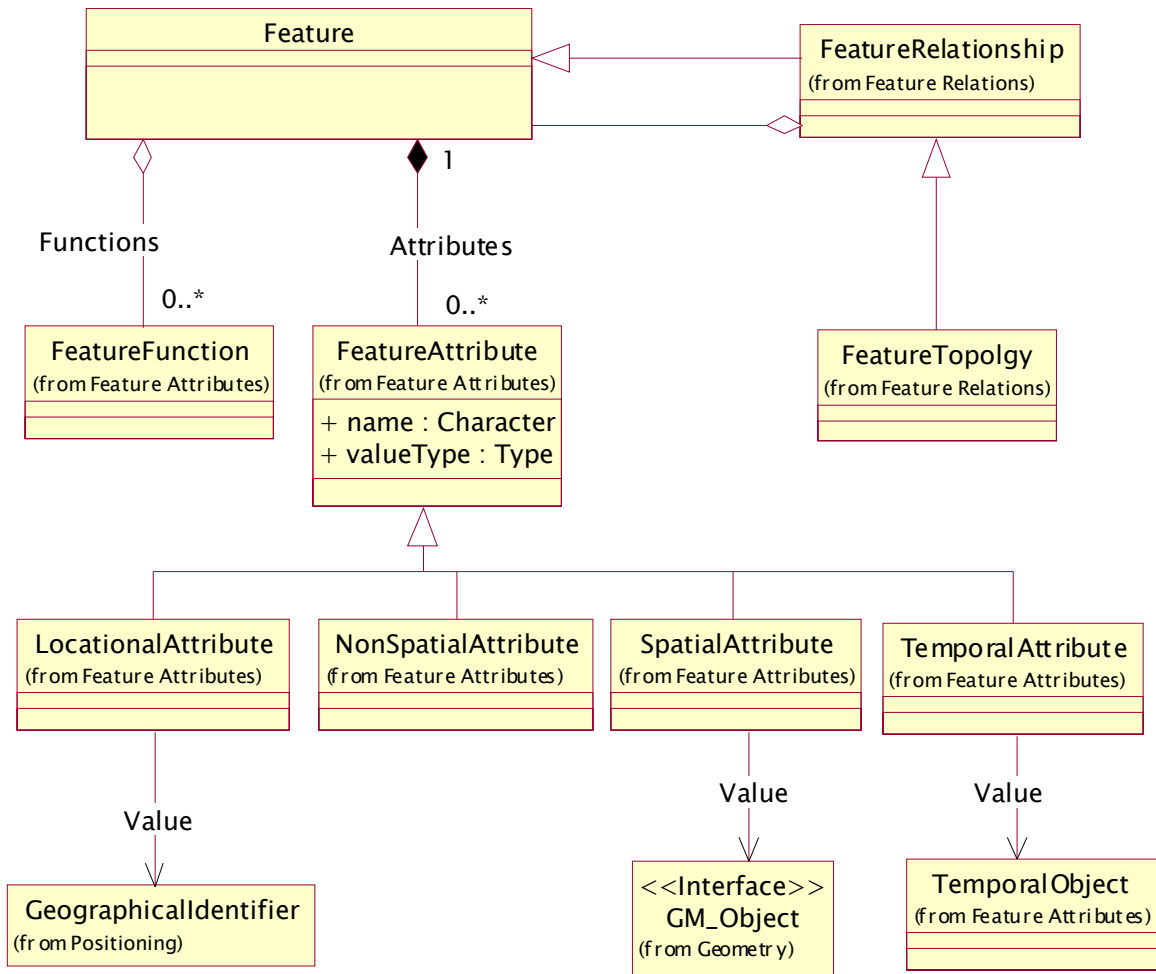


Figure 6-2: General Feature Model (from Rules for Application Schemas)

6.1. References

- [1] International Standard ISO 15046-9, Geographic information — Part 9: Rules for application schema, Technical Committee ISO/TC 211 Geographic Information/Geomatics.