Open Geospatial Consortium Inc.

Date: 2007-07-12

Reference number of this OGC® project document: **OGC 07-056r1**

Version: 0.0.9

Category: OGC® Policy Specification Discussion Paper

Editor: John R. Herring

Annex to OGC Policy and Procedures — The Specification Model — Structuring an OGC specification to encourage implementation

Contents Page

Foreword

This specification is a reflection of discussions in the OGC Architecture Board (OAB), and contains general principals of application development taken from the OAB members' collective experience.

Since this specification is a summary of collective experience, no one should claim intellectual properties rights to its contents. This standard may be used by anyone as long as its source is specified.

This standard, while it specifies the structures of other standards, does not supply them with specific content, since its level of abstraction is one level higher than any standard that might claim conformance. All content in this standard, unless specified otherwise in its title, is normative.

Since this standard specifies requirements for standards to be acceptable by the OGC TC, it is logically an annex of the TC Policy and Procedures (TC-PnP). It is however a radically different document, since the TC-PnP is a procedural set of rules, and this document is a set of testable constraints against a finished document. The mechanism for enforcement of this standard is the purview of the TC and its subgroups, and in particular the OAB.

Recent OAB discussions have identified the need for a "check list" for each candidate specification to assure its conformance to OGC policy and rules. Conformance to this document should be added to that list.

Introduction

"A really Useful Engine"

Thomas the Tank Engine,
by Rev. W. V. Awdry, 1946

Our friend Thomas, in a Zen-like moment unusual for a child's story and definitely unusual for a steam locomotive, decides that his purpose in life is to be a "Really Useful Engine[1]." A standard's usefulness and hence worth can be measured by:

- The ease with which it can be implemented.
- The number of times it has been implemented.
- The number of times those implementations are used.
- The ease with which those implementations interact.
- The number of times it has been extended through inclusion in other useful standards.

Some of these are affected by the choice of topic, but all are affected by the internal logical structure of the standard. This standard will specify generic rules for this internal structure conducive to being a true RUE (really useful engine).

A standard presents requirements, which must be satisfied by the proofs as defined by the tests of the conformance suite. These tests are organized into conformance classes, each of which represents a mechanism for partial satisfaction of the standard. These give the standard a modular structure, each module represented by a conformance class. In a well written standard, the normative clauses and any model or schema are organized in a manner that parallels the conformance clause.

[1] The capitalization is the Rev. Audrey's. See http://en.wikipedia.org/wiki/Thomas_the_Tank_Engine

Annex to OGC Policy and Procedures — The Specification Model — Structuring an OGC specification to encourage implementation

1 Scope

This standard specifies some desirable characteristics of a standards specification that will encourage implementations by minimizing difficulty and optimizing usability and interoperability.

2 Conformance

Conformance to this standard by technical implementation standards and specifications can be tested by inspection. The test suite is in Annex A.

Clause 4 formally defines the terms used in the conformance tests.

Clause 7 gives information on how this standard is to be applied to requirements, conformance clauses, UML models, XML Schemas and SQL specifications.

All numbered clauses and any Annexes that are marked as normative may contain normative language and thus place requirements on conformance or mechanism for adoption of candidate specifications to which this standard applies.

Clause 6.2 specifies the basic requirements which shall be met by a conformant standard. They are:

1. Conformance classes are Modules
2. There is a Defined Core
3. Modules test all requirements
4. Profiles (subsets of the base) are modules
5. Extensions are modules
6. Options are modules
7. Modules intersect only by reference

3 Normative references

There are no normative technical references for this document. While this document references UML, SQL and XML, and their technical specification, it does not derive any of its requirements from these documents. While this standard may be applied to extensions of those standards, conformance to them is the purview of those standards, not this one.

Since this document adds requirements to the procedure for the adoption of OGC specification, it shall be considered as a normative annex to the "current" OGC TC Policy and Procedures[2]. The effect on these procedures and a transition plan of existing and "soon to be adopted" specifications is detailed in Annex B.

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply. Terms not defined here take on their meaning from computer science or from their Standard English (common US and UK) meanings. The form of the definitions is defined by ISO Directives.

4.1

aspect

testable requirement

NOTE The implementation of an aspect is dependent on the type of specification being written. A data specification requires data structures, but a procedural specification required software implementations. The view of a standard in terms of a set of testable requirements allows us to use set descriptions of both the standard and its implementations.

Specification of an aspect are usually expressed in terms of a model of the implementation, such as a UML model, or and XML or SQL schema.

4.2

base

unique module that includes all **aspects** of a standard

NOTE A **base** is unique at any given time, but may vary over time as new extensions are added.

4.3

certificate of conformance

proof of conformance to all or part of a standard

NOTE "Certificates" do not have to be instantiated documents, having proof of conformance is sufficient. The OGC currently keeps an online list of conformant applications at http://www.opengeospatial.org/resource/products.

4.4

condition
constraint

requirement not strictly testable but which limits implementation

NOTE Conditions are generally global in nature, such as "all data types will be defined in XML schema" or "all implementations will contain WDSL and SOAP." Conditions are in effect general rules as opposed to local restrictions on a specific entity; as such they cannot be restricted to portion of the standard. There are several ways to handle conditions; to be consistent with the testable requirements; general conditions will be restricted to the core, which must always be applied to implementations.

Conditions are most often expressed in terms of items in a model, such as "class" for UML or "type" or "element" for XML.

[2] The reference to the OGC TC PnP is floating, and does not apply to a particular document, but to the policy and procedures of the Open Geospatial Consortium whether defined by an OGC document, by Roberts Rules of Order, or by common consent and practice of the membership.

4.5

conformance class;
conformance level

set of conformance tests that must be applied to receive a single certificate of conformance

NOTE The set of required **aspects** in a **conformance class** is a **module**. If a **conformance class** contains no optional **aspects**, it is a single **module**. Each optional **aspects** will be in a separate **module**.

4.6

conformance suite

set of conformance classes that test all **aspects** of a standard

NOTE The **conformance suite** is the union of all **conformance classes**. It is by definition the **conformance class** of the **base**.

4.7

conformance test

test, abstract or real, of **aspects** contained within a standard, or set of standards

4.8

core

unique module that must be implemented by any conformant system

NOTE The **core module** is unique because if there is more than one, each **core** would have to be implemented to pass any **conformance test**, and thus would have to be contained in any other **core**. The **core** may be empty, or all or part of another specification.

4.9

extension

module dependent on (including) all of the **aspects** of another **module**

NOTE Multiple **modules** may be included in a single **conformance test**. Such bundles while legal are discouraged since the may hide implementation dependencies not intended by the specification and thereby interfere with interoperability of modules within the bundle with external invocation.

4.10

module

aggregate of normative aspects of a specification that must all be implemented as a whole to satisfy an atomic set of **conformance tests**

NOTE A **module** refers to a part of a specification, but the term, and other related to it, can be used for implementations modules that implement a particular atomic set of conformance tests. Hence, the XYZ specification **module** from a **conformance suite** may be implemented by the XYZ implementation **module** from a particular software suite. Thus the term **module** is context sensitive and can be applied to a subset of a specification, to a conformance class testing that subset, and to a implementation capable of passing all the tests in that conformance class. When the context is not clear, the adjectives "specification," "conformance" and "implementation" or their equivalents are suggested.

In cognitive linguistics, this is part of a conceptual metaphor (see http://en.wikipedia.org/wiki/Conceptual_metaphor). The term **module** is part of the mapping between the target and source domain of the metaphor. In this case, the source domain (in which the term was first used) is software, although even there it is derived from manufacture and specifically the concept of interchangeable parts that enables modular construction in an assembly line. This standard links into that usage and applies the term **module** to just about anything that relates to a software **module**, including models and schemata.

4.11

profile

module defined by a subset of the **aspects** of another **module**

NOTE **Profile** has been used both as subset and **extension**, and in terms of requirements it is really just a set of requirements that can be expressed as a union of **modules,** assuming conformance to this standard**.**

In this standard, only the subset meaning is meant unless specifically stated.

4.12

standard

document containing **aspects**

NOTE This definition is included for completeness. In this standard's model, a standard is a set of testable requirements applied against implementations. In the real meaning of the word "standard" there are other conditions that may be required, but this standard has chosen to ignore them in the process of abstraction.

5 Conventions

5.1 Symbols (and abbreviated terms)

All symbols used in this document are either:

1. common mathematical symbols
2. UML 2 (Unified Modeling Language as defined by OMG)

5.2 Abbreviations

In this document the following abbreviations and acronyms are used or introduced:

ERA	Entity, Relation, Attribute (pre-object modeling technique)
ISO	International Organization for Standards (from Greek for "same")
OGC	Open Geospatial Consortium (http://www.opengeospatial.org/)
OOP	Object Oriented Programming
OOPL	OOP Language (such as C++ or Java)
RUE	Really Useful Engine
SQL	Sequel, sometimes mistakenly "Structured Query Language (relational query language)
UML	Unified Modeling Language (an object modeling language)
XML	eXtensible Markup Language
OMG	Object Management Group (http://www.omg.org/)

6 Requirements

6.1 Specification model

The primary difficulty in speaking of standards as a whole is diverse nature of the group. Some standard use UML to define behavior, others use XML to define data structures, and others use no modeling language at all. The only thing they have in common is that they define testable requirements against some design mechanism for something that could be described as an implementation of the standard. Thus, this standard lays requirements against other standards by using a set-theoretic description of those standards based on their structure as organized sets of testable requirements ("aspects"). All standards create an abstraction of the things to which they apply (called a "model") in order to provide a context for stating requirements. In this sense, this standard is a subject of its own terminology.

First this standard defines an "aspect" of a standard as an atomic testable requirement (see 4.1). The aspect will have a variety of parts spread throughout the standard. Somewhere in the normative clauses, the aspect will be defined in terms of the local modeling paradigm. In the conformance clause there will be a test defined to verify the validity of the claim that an implementation of the standard satisfies the aspect's requirement. Assuming an ISO-structure, this will be in one of the annexes, usually a "conformance test suite" that list all tests and associated structures.

Conformance tests are aggregated into conformance classes that specify how certain "certificates of conformance" are achieved; the natural inclination is to similarly aggregate aspects in this manner. The issue that blocks this approach is that some aspects' tests are listed as optional while others are listed as mandatory. To achieve a cleaner separation of aspects, this standard separates them into sets which have no optional components.

So, this standard defines a "module" as a set of aspects that must be all passed to achieve a particular level of conformance (see 4.3). Optional requirements thus become modules by themselves. This means that as far as this standard is concerned, modules may be optional, but each aspect within a module is mandatory. Some modules may contain only a single aspect.

NOTE Care must be taken, since the modules are not always in a one-to-one correspondence to conformance classes in other standards which may be the source of requirements for a standard conformant to this one. If other standard are used then their options must be specified to be useable within this standard.

Conformance classes contain dependencies on one another. These are represented by tests in one conformance class that state that another conformance class must be passed to qualify to pass this one. In terms of aspects, that says that the dependent conformance class contains tests (by the reference) for all mandatory aspects of the "included" conformance class.

Translating this into this standards view of modules, one module is dependent on another if the other is included through such a reference. In this manner, modules can be treated as sets of aspects (each physically in a single module). The set containment lattice (partial ordering) defined by:

$$[A < B] \Leftrightarrow [\{\text{aspects in } A\} \subset \{\text{aspects in } B\}]$$

This says that to pass "B", an implementation must pass "A." If "A<B", we say that A is a profile of B (see 4.11) and that B extends A (see 4.9).

A "core" module is one which is a requirement of all others (see 4.8). So, if we consider a standard "**S**" as a set of normative modules:

$$[A \text{ is core in standard } \mathbf{S}] \Leftrightarrow [\forall\ B \in \mathbf{S}\text{: } A < B\}]$$

A "base" module is one which contains all requirements (see 4.2). So, if we consider a standard "**S"** as a set of normative modules:

```
[A is base in standard S] ⇔ [∀ B ∈ S: A > B}]
```

So, a core is a "universal lower bound," and a base is a "universal upper bound."

6.2 Requirements

6.2.1 Conformance classes are Modules

Conformance classes shall not contain optional elements. Modules may be optional as a whole, but not piecemeal. This means that every implementation that passed a particular conformance class satisfies exactly the same requirements. Differences between implementations will be ascertainable by which classes are passed, not in the listing of which options within a class were chosen.

6.2.2 There is a Defined Core

Every standard shall define a core module as a separate conformance class. The core should be as simple as possible. The core module may be contained as a conformance class in another standard, in which case the current standard should define a disposition of all of the optional components in that conformance class that are required by the current standard's core module.

All general conditions will be in the core. Since the core module is contained (by reference) in each other module, general conditions are universal to all modules. Further, since the basic concept of some standard is mechanism not implementation, the core may be only constraints, and no content.

6.2.3 Modules test all requirements

All non-core aspects of the standard shall be contained in some module extending the core. Unless a requirement is contained in a conformance test and thus in a conformance class, it cannot be considered an aspect. If possible, the structure of the normative clauses of the standard should parallel the structure of the conformance tests "modules" in the conformance clause.

There shall be a natural structure on the modules so that each may be implemented on top of any implementations of its dependencies and independent of its extensions. Thus, no module shall redefine the requirements of one of its dependencies, unless that redefinition is for an entity derived from but not contained in its dependencies.

In OOP terms, subclassing is okay, redefinition is not.

6.2.4 Profiles (subsets of the base) are modules

The totality of all conformance classes specified in the standard form a base (an upper bound of all conformance modules). Subset profiles of this base shall be specified by a list of conformance classes (modules) that are to be included in that profile's conformance test.

6.2.5 Extensions are modules

Each standard that extend a standard conformant to this standard should consist of a core which is a subset profile as defined in 6.2.4, and some number of modules defined as extensions to that core. A standard conformant to this standard shall require all conformant extension to itself to be conformant to this standard.

6.2.6 Options are modules

The only options acceptable in a standard conformant to this standard shall be expressible as a list of conformance class modules to be passed.

NOTE Standards and implementations are restricted by this, but not instances of schemas. For example, a XML schema standard can specify optional element, which data instances may use or not, but schema-aware processors claiming conformance to the standard should be able to handle all element in the schema, whether they are required by the schema or not.

Requirements of the form "if the implementation does this, it must do it this way" are considered to be options and must be in a separate module.

6.2.7 Modules intersect only by reference

The common portion of any two modules consists only of references to other modules. This implies that each requirement or aspect in directly in exactly one module, and all reference to that aspect include complete conformance to its "home" module.

NOTE All general conditions are in the core module. The core module contains tests that all other modules must pass.

7 Mappings this standard to types of models

7.1 Semantics

The previous section defined requirements for conformance to this standard, but testing for that conformance may depend on how the various forms and parts of a candidate conformant standard are viewed. This clause will define how those views are to be defined in most of the common cases. Standard that take an alternative mechanism to the ones listed here must be tested solely on the structure of their conformance test suite.

Standards are often structured about some form of modeling language, or implementation paradigm. This clause looks at some common types of models and defines a mechanism to map parts of the model (language, schema, etc.) to the conformance classes used as the specification model from Clause 6.1.

As suggested in Clause 6.2.3, the structure of the normative clauses in a standard should parallel the structure of the conformance modules in that standard. The structure of the normative clauses in a well written specification will follow the structure of its model; this means that all three are parallel.

7.2 Conformance Test Suite

This is the model used in this standard. The test defined here may be applied directly to the test suite.

In each of the other profiles defined in the other Clauses to follow, some set of entities, types, elements or objects are defined and segregated into implementation modules that correspond to the test suite conformance modules. This strict parallelism of implementation and governance is the essence of this standard.

7.3 UML or other object models

UML is an OMG) and an ISO standard for expressing object models. If the organizing mechanism for the standard is an object model, then the mapping from parts of the model to the requirements shall follow the logical mechanism describe here. This clause used UML terminology, but other object modeling language and if needed, the object code itself can be mapped to a UML model. An ERA model can be considered as a partial OOP model. To be conformant to this standard, UML shall be used to express the object model, either as the core mechanism of the standard or as a normative adjunct to formally explain the standard in a standard model.

In a UML organized standard, the requirements modules shall be associated to UML packages and their content. If a requirement crosses package lines, then one of the packages will be called the source of the requirement, and the other the targets. This shall be consistent with the usual mechanism for describing the source and target of dependencies between packages. By Clause 6.2.3, only classes in the source module will be affected by the requirement. A source package shall always be dependent on the target packages (and therefore, the source module will always extent all of its target modules in the terms of this standard).

The things in a UML model are:

1. Package
2. Class or more properly "Classifier" (classes, interface, types, data types, enumerations, code lists3), contained in a package or (rarely) another class. In all cases, eventually in a package
3. Attributes of a class, consisting of a name and a type (possibly modified with a cardinality), contained in the class and dependent on its type
4. Association role, consisting of a role name (if it is visible from its containing class) and a target type (possibly with cardinality); if navigable, it is considered to be in its source class and dependent on its target class; if not navigable, it is neither (a non-navigable role while still part of an association, does not affect class implementations. All association must have at least one navigable role).
5. Operation, consisting of parameters and a return type, contained in its defining class.
6. Parameters parallel to the structure of an attribute, contained in the operation that uses it. This creates dependencies similar to an attribute.
7. Return type containing the results of the operation, considered to be part of the operation. This creates dependencies similar to an attribute.

Packages have dependencies upon one another based on usage. Any of the following conditions will produce a direct package dependency between package A and package B:

[3] Code list is an ISO TC 211 extension to UML.

a) A class in A references a class in B by using it as:

   1. the type of a attribute
   2. the target of a navigable association role
   3. the link class of an association that has one of these navigable role
   4. the key class for a navigable association role
   5. the type of a parameter used in an operation
   6. the return type of an operation

b) The package B is one of A's subpackages

Other package dependencies (indirect) will arise from the transitive closure of the above direct dependencies. That means if A depends on B, and B depends on C then A depend on C. Since these indirect dependencies will show up in the cascade of included conformance tests based solely on the direct dependencies, we can ignore them for affects on structure.

Since a package can consist solely of other packages, there is always the capability to define in a UML a single package that will correspond to a particular test suite module. All test modules in a conformant standard will be associated to one and only one UML package. If the core module contains only constraints, it may be an exception to this constraint. If two packages have a two-way dependency, they will be contained in the same module. For example, if two classes have a two-way navigable association, then these two classes must be (transitively) in the same conformance module package.

Assuming all legitimate direct packages dependencies are included in the UML model, the conformance module associated to a package A will directly reference the conformance module associated to another package B, if and only if A is dependent on B. Indirect dependencies will be dealt with as the conformance classes cascade.

7.4 XML Schema

XML Schema is defined by W3C. Each XML schema file (.xsd) carries a namespace specification. The XML Schema specification lists those namespace and schema resolution strategies that a schema aware process may use. They work in a predictable fashion if and only if the schemas are in a one to one correspondence to their namespace. Schemas may be dependent on other schemata and may contain inclusion directives that load those schemata whenever they are loaded.

XML schemas are truly independent only if they are in different namespaces. XML schema A is directly dependent on XML schema B if and only if a valid schema parser will validate schema A only if contains a reference (include) for schema B.

If a standard defines a set of schemata, it shall associate each schema with a well-defined set of namespaces (usually only one). If a standard defines constraints against a yet-to-be-defined set of schemata, it shall associate each schema to a namespace variable. A namespace set referenced in the standard's conformance test suite shall be referenced by a namespaces or namespace variable from the standard. Most text modules should contain one and only one namespace, but especially if there are two-way dependencies between namespace, it may be necessary to associated a test module to multiple namespaces. In no case shall a single namespace be directly included in two modules. By Clause 6.2.3, no module shall modify elements, types or any other aspect from a namespace to which it is not associated.

Each conformance module of an XML schema organized standard shall be associated to a particular namespace set. No namespace or namespace variable shall be assigned requirements in two different modules.

7.5 SQL

The organizing sets of SQL type definitions are schemas. An SQL organized specification will treat these schemas in an analogous manner as defined above for XML namespaces.

Each conformance module of an SQL schema organized standard shall be associated to a particular schema set. No schema or schema variable shall be assigned requirements in two different modules.

Annex A
(normative)
Abstract Conformance Test Suite

A.1 General

The conformance requirements are contained in Clause 6.

A.2 Conformance Class: Modularity

A.2.1 Conformance classes are Modules

a) Test Purpose: Verify that all conformance classes contain no optional tests

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.1

d) Test Type: Conformance.

A.2.2 There is a Defined Core

a) Test Purpose: Verify that there is a modular test that defines a core module, and that all general conditions in the standard are tested by the core.

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.2

d) Test Type: Conformance.

A.2.3 Modules test all requirements

a) Test Purpose: Verify that all requirements in the normative clauses are tested in at least one conformance class, and that no module redefines or adds tests that would restrict or modified the requirements of another module.

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.3

d) Test Type: Conformance.

A.2.4 Profiles (subsets of the base) are modules

a) Test Purpose: Verify that subset profiles are defined only by the listing of conformance test to be performed.

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.4

d) Test Type: Conformance.

A.2.5 Extensions are modules

a) Test Purpose: Verify that all extensions of the subject standard are required to conform to this standard

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.5

d) Test Type: Conformance.

A.2.6 Options are modules

a) Test Purpose: Verify that the only option in any conformance class is to apply a class' test or to not do so.

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.6

d) Test Type: Conformance.

A.2.7 Modules intersect only by reference

a) Test Purpose: Verify that each aspect is directly in only one module, and that any other use is by a reference by a module that another conformance class must also be satisfied.

b) Test Method: Inspect the document.

c) Reference: Clause 6.2.7

d) Test Type: Conformance.

Annex B
(normative)
Changes required in the OGC Specifications

B.1 Existing standards and specifications

This standard does not apply to existing specifications nor those adopted before its adoption. This standard shall apply to new major versions of such specifications.

Failure to conform to this standard should be specifically noted and reasons given for such non-compliance in the conformance clauses of any new or new version of such specifications. Adoption of such documented not compliant with this standard shall be considered as an authorized exception to the requirements of this standard by the OGC. This "exception by majority vote" will be a valid approach only before the 1-year anniversary of the adoption of this standard.

B.2 New standards and specifications

Any new standard, specification or major revision of an existing standard shall comply with this standard in the structure of its internal models and its conformance tests.

Failure to conform to this standard should be specifically noted and reasons given for such non-compliance in the conformance clauses of any new or new version of such specifications.

Adoption of such documents not compliant with this standard shall be considered as an authorized exception to the requirements of this standard by the OGC. This type of "exceptions" will be considered as an exception to the rules of the OGC and will require a 2/3 majority of all the voting members of the Technical Committee. A similar vote is required within the Planning Committee.

This vote is a vote to override the rules of the consortium, and not a vote on the merits of the proposed specification which has failed to conform to this standard. Voting to override the rules carries no intellectual property implications in addition to that consistent with a non-participating member role for the proposed specification as provided in the TC PnP.

Annex C
(informative)
Standards Metamodels

This annex contains information and arguments that first circulated in the OGC Architecture Board. It does not necessarily represent a full consensus of the OAB, but does justify the approach taken in this standard. This annex is needed since the rest of this standard is written as a standard, a lot of what to do but very little why to do it.

C.1 Abstract

One topic that increasingly the center of debate in the OGC is how to structure a standard or set of standards so that they are used to the widest extent possible and in the most useful way possible as defined by the OGC's core mission statement. To inform and expedite those discussions, we need to have a common conceptual model for how we need to put together specification for use in standardization, and how those standards should be factored, extended or profiled to meet this need.

The first step in such an endeavor is to establish a common vocabulary and a common methodology for analyzing a standard's structure – either new or existing. We need to do this because this is what a standard is – a conceptual model of a domain against which criteria and constraints can be crafted. Thus a standard for writing a standard is a model of a model or a metamodel. To document this is an attempt to explore the standards metamodels that the OGC already has in use and to determine which criteria and constraints on those metamodels best serve our purpose.

This paper is divided into three parts:

1. A proposed model of how standards are constructed and how they relate
2. A survey of the most common patterns seen in standards and how the model above describes the differences in these patterns.
3. An investigation of the perceived properties of these standards patterns as they affect our core mission – "useful standards being used usefully".

This is not a pure logical construct and the behavior patterns of both the writers and implementers of standards will affect these properties and will affect our perception of these properties. This paper will not "solve" the problem, but will point a way to the discussion that will at least arrive at a consensus of what the problems and potential problems are in the domain of writing the standards we are writing.

C.2 A little bit of history

The concept of a "standard metamodel" or a model of how to write standards arose out of discussions in the OGC Architecture Board. There were several issues that revolved around how specifications in the OGC standards baseline have been put together, how they are received and implemented, and how the resulting systems interoperate. The following surface issues all seem to have this conceptual framework at their core:

1. How can an existing standard be extended if that extension exhibits a conceptual dissonance with one of the base standards original parts? For example, how can the

new Observation and Measurement (O&M) XML based candidate specification be properly integrated with GML 3.2, which already has a different (possibly not incompatible) O&M model implementation. Can the two coexist or should GML be modified to affect the change in itself originally set forth as an extension.

2. Why is community perception of a particular standard being overly complex not diminished with the advent of simple profiles? Is there a better way than we are using to construct base standards and profiles that will be better received in the market? This issue arose separately within the context of WFS and of GML.

3. How can a standard be written so that there is a natural and easy point-of-entry application that can successfully and usefully be implemented by neophytes so that the larger structure gain "market" acceptance in the broader non-core GIS market? This may be a variation of the above issue.

C.3 Basic Model for Standards

To understand how standards should be structured and related to one another, it is logical to produce a model of the items to be constructed so that aspects of that construction can be analyzed, and best practices can be "discovered" and, in their turn, standardized.

Several fundamental relationships that create dependencies between Standards are:

1. "profile" where a profile is a proper subset of its base standard and

2. "extension" where the extension logically includes the entire core. This includes such things as "application schemas" which define extensions of the concepts in the core specification. For example, CityGML is an extension of GML, within the XML Schema metalanguage, but consistent with the GML "rules for application schemas."

3. "reference" where one standard normatively adopts in whole or in part, the aspects of another standard.

4. "factor" or "part" where a larger standard is broken down internally into parts that supply a default factorization of the aspects of the overall system being described.

5. "implement" where a software module (here represented by a UML package) implements a "part" of a standard.

Right away we are in trouble since "implement" really needs to be against an object, not some subset of an object. For this reason, we will represent the parts of a larger standard as standards in their own right, even though they are sometimes not separate documents, but specified by conformance classes within the parent standard. This includes such things as "optional" aspects in a standard, which in their nature allow bifurcation of the classic ISO concept of a conformance class – to not do an optional is one class, to do it is another. This "modularization assumption" where conformance classes (levels) can be seen in the software modularization is not a logical necessity, but it so resembles that which is done, the no discernible difference in the logic should be seen.

This basic model is given in Figure C.1. It should be noted that the targeting of the base class "Standard" by the roles "base" and "core" mean that these associations can be recursive, and thus for example, the same instance standard can be a profile of one standard, and the base of another profile. The same is true for recursion on extensions (an extension can be extended). Further, profiles can be used as core to extensions. For these reasons, the concept of "base" and "core" should not be seem as absolutes but a distinction of usage. Any standard can:

- Be a base if it is profiled.
- Be a core if it extended.

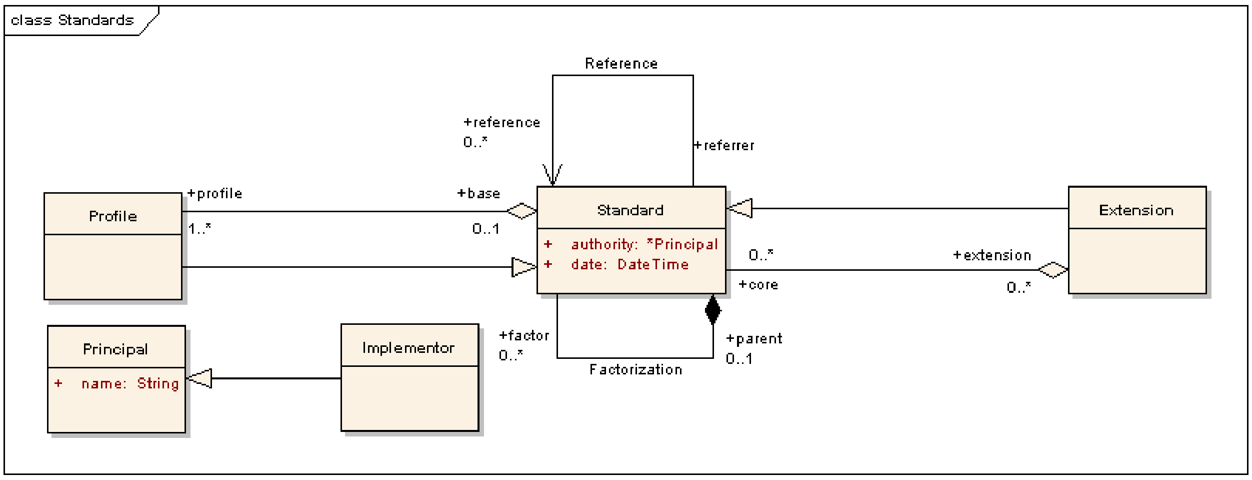

Figure C.1: Basic relationships of standards

If the factors of a standard correspond to separable conformance classes, we can now think of standards being separable. That is, if a parent standard contains conformance classes that allow a factor to be conformed to independently of the parent; we can then consider the factor as an independent standard, and the parent as an extension of the factor. For example, in ISO 19107, which is the geometry volume of the OGC abstract specification, there are conformance clauses of a variety of implementation, including one that fits Simple Features, one that fits GML, and others that add extensions for applications for Road design (include clothoids). From this we can define a canonical form of object graph where separable factors are distinguished elements while non-separable parts are not. For the purpose of this discussion, we will think of parts (factors)according to the concepts of extension and profile depending on whether the parts are separable or not. Thus a fully factorable standard is really a set of core standards (maybe only one) and a set of extensions, where a non-factorable standard, with profiles that cannot be really exist independent of the base standard, is really a base standard with some number of profiles.

Care must be taken in doing this analysis, since form is not the only thing involved in this separation. For example, the main parts of GML are in one namespace, and thus because of the defined XML Schema namespace resolution strategies, GML parts are distinguished as separate schema files within the same namespace are not separable factors. As such they must be "profiled" from the base GML standard to have a pseudo independent existence, but even then they are somewhat tethered back to GML by the "profile has same namespace" conformance criteria in the GML specification. This means that even though the various parts of GML are only semi-independent, as major modifications to only one part forces GML to step the version of the parent schema, and thus all of the parts. The caveat is that independence of parts has to take into consideration aspects that may not be visible in the parent standard, but are in normative references used as structure.

If we think of a standard as containing atomic aspects of information about its subject, then profile, extension and factoring fundamental relations are subsets of a subset relation based on these "atomic aspects" of this information. A profile or factor is a subset of its base or parent standard. A core standard is a subset of its extensions (inclusion in this case is by reference). Using a Venn diagram representation of this set metaphor, when we create a profile or extension of an existing standard, we get one of the following cases:

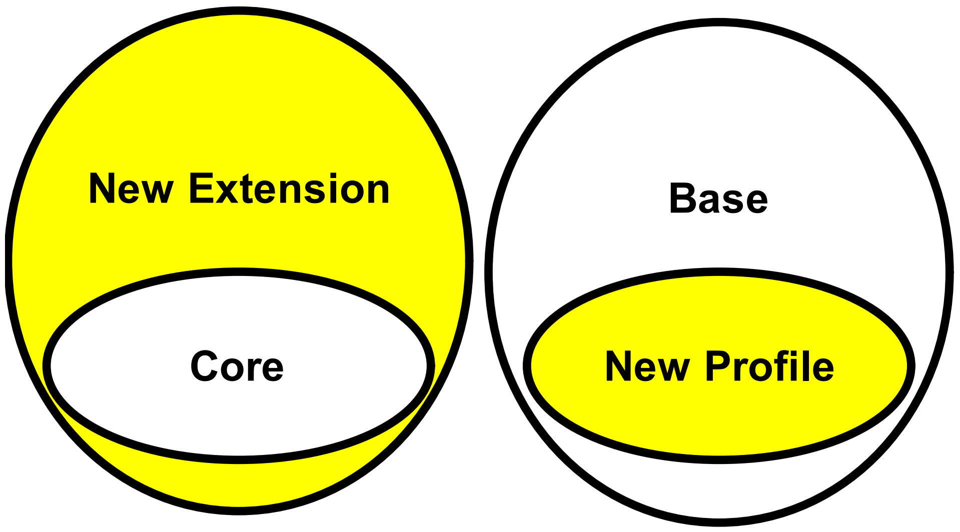

Figure C.2: Extension and Profiling as Set Relations

The real difference is in the order in which the two standards are written, and the manner in which they reference one another.

- **`Extension.date ≥ Extension.core.date`**
  `(the "smaller" core is published first;`
  `an extension cannot exist without a core)`

- **`Profile.date ≥ Profile.base.date`**
  `(the "larger" base is published first,`
  `a profile cannot exist without a base)`

This means that under normal circumstances, the inclusion of one standard in another (as sets of aspects) for a core-extension relation agrees with the age of the specification, i.e. the smaller core is older. For a base-profile relation, the smaller profile is newer.

One could argue that if all details are known, then the publication order can be controlled and should not matter, but the deciding requirement is that a standard cannot reference another standard until it exists. So we have the following driving reference requirements:

- **`X in Extension.core ⇒ X in Extension.reference`**
  `(reference are from the larger extensions to the "smaller" core)`

- **`X in Profile.base ⇒ X in Profile.reference`**
  `(references are from the smaler" profile to the "larger" base)`

Note: The terms "larger" and "smaller" refer to the size of the logical content as a collection of "aspect" information, and not to the size of the documents.

Another way to look at this issue is to consider the standard producing process model. If a standard can reference standards as yet not written, (say via a registry) then we could build an extension hierarchy (rooted at a core, and each step branching out to its extension). If we then take the base as the logical union of the leaves of the tree, and the core becomes the ultimate profile we get a profile tree as the inversion of the extension tree.

C.4 Software implementations of standards

The relationship between standards and software products, at least the default logic, should be to implement each incremental expansion of functionality by extending the existing software packaging modules in a manner that parallels the standards structure. This is not the only way to get conformance, and in some standards based on “best engineering practice” many software modules may predate the standards to which they are compliant.

For the sake of the narrative, let us consider the software design of the module a type of “private” standard, specific to the implementor of the software. As such, it has to be an extension of each standard to which the software module will be conformant.

```
If M a ModuleDesign, S a SoftwareModule then
S.design = {M} ⇒ M.core = S.implements
```

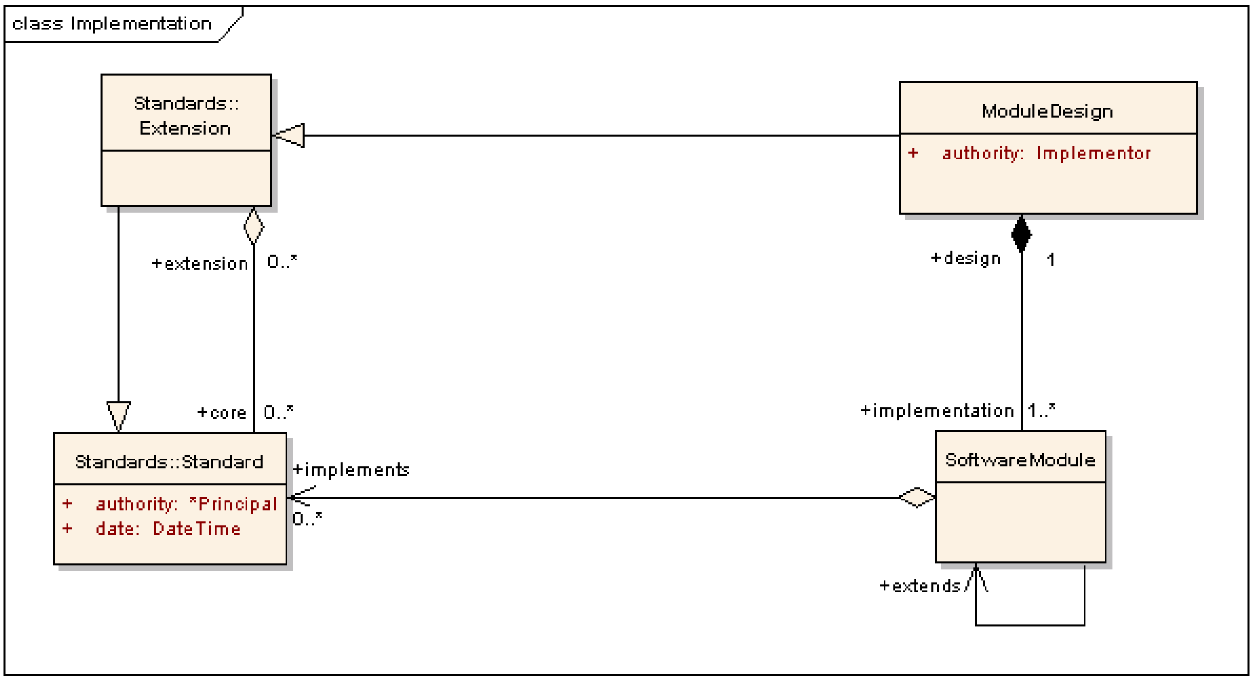

Figure C.3: Software Package implementing Standard

When we look at UML packages for the implementation of a specification, and look at core and extension we get what is referred to as a temporally covariant functor that is:

For M1 and M2, SoftwareModules, and S1 and S2, Standards then
[M1.implements S1 and M2. implements S2 and M2.extends M1]
⇒
[it is possible that S2 extends S1 (that is S2.core = S1)]

This means that it is feasible to build UML packages in the manner suggested by Figure . Note that the newer software module (which is an extension of the older module) corresponds to the newer standard, an extension of the older core standard.

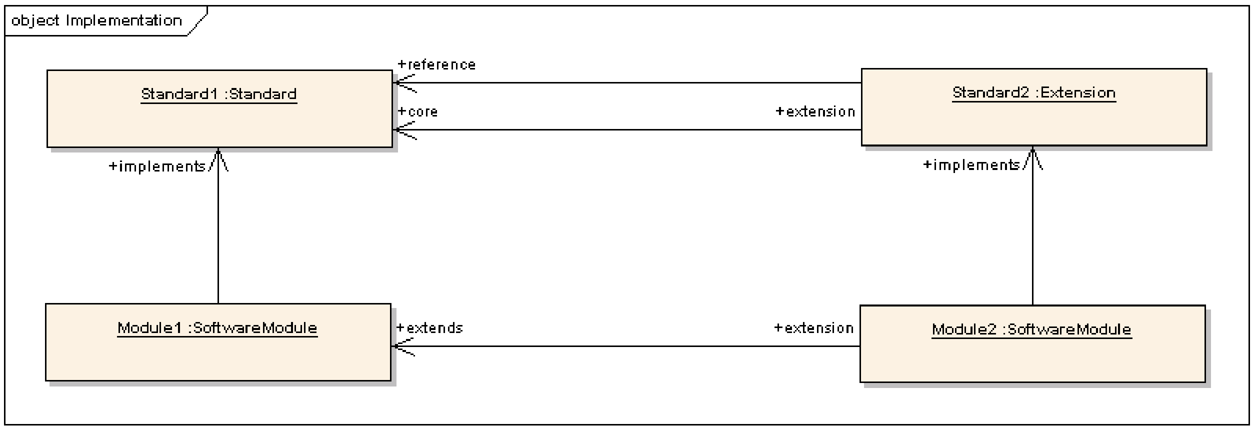

Figure C.4: Default relation between software extensions and standards

The same diagram for base and profiles is temporally contra-variant. Thus, if we are to structure software to take advantage of object extensions, we get Figure :

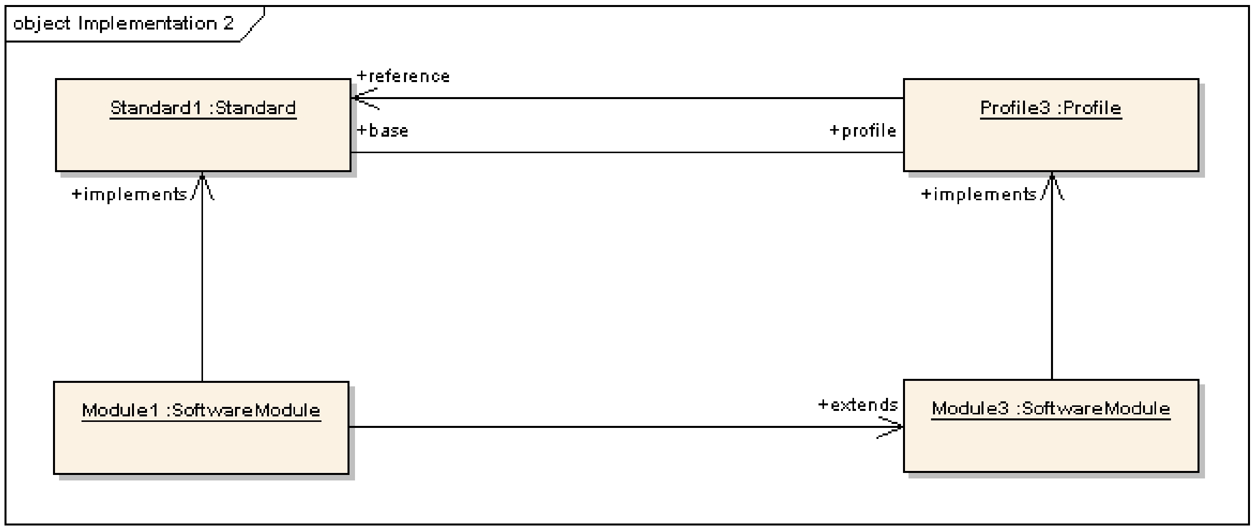

Figure C.5: Default relation between profile implementation and standards

If the larger, older standard is implemented to get maximal code reuse, it has to be implemented by the newer software module (an extension of the module implementing the profile). Under normal practice (implementation of standards in the order they are written, and a lag between base standards and profiles) this would not happen, and hence the natural modularization of the implementation of the base standard based on the way the profiles will factor it, is impossible unless all of the profile standards are published at the same time as the base standard, essentially as factors (preferably separable) of the base standard.

C.5 Standards metamodels

From what was discussed above, we will try to describe the way to specify the relationships between standards that are paralleled in the object-oriented concepts of extension. For this purpose, there are at least two simple forms of groupings for related standards, for this paper we will refer to the two metamodels as described above as:

1. Core and Extension
2. Base and Profiles

The Core and Extension model builds a minimalist "simple" core specification - one that logically everyone must use to even think about working in the arena. The Core is then augmented via extensions, each extension addressing a specific set of requirements. A user assembles the core and those extensions needed, and adds local extensions (such as feature definitions, property types, etc.) to create what 19109 calls an Application Schema. Chronologically the default logic would be that the Core is written first, and the extensions are added as needed.

The Base and Profiles model builds an all-encompassing specification - one that covers a broad area and includes solutions to all of the most common interoperability issues. The Base is then cut back to a subset (restricted) to form a Profile (as defined in ISO/TS 10000). A user selects (or constructs) a profile, and adds local extensions to create an Application Schema. Chronologically, the default logic would be that the Base is written first, and the profiles are added as needed.

These examples obviously do not form an exhaustive list of metamodels. It is possible to imagine a central standard that is both the Base of a limited number of profiles, and the Core of all future extensions. It is also common for a Base standard to be written as a collection of separable factors as described in Figure C.1 which can then be combined into profiles via a simple selection. This is essentially a core-extension model since the base is pre-factored into a useable core with predefined extension (this is how the ISO REL (rights expression language) ISO 21000-5 is structured).

Examining the extremes is useful since it begins to enumerate the parts that will get involved in the eventual balancing act of optimization (which will depend on criteria currently not determined).

The "Core and Extension" metamodel is precisely how mathematics is built. Everyone starts out learning to count - the Core is the Natural Numbers (1, 2, 3,) with a "next" function. Simple arithmetic adds addition and subtraction, which leads to issues of negative numbers which leads to the Integers. We add multiplication and then divisions (quotient and remainder style), which brings up issues on "parts" and so we get the Rational Numbers. We add symbolic problem statements (algebra) and geometry, and then coordinate geometry, and discover not all "numbers" are rational, like the square root of 2 or PI. We get to the Real Numbers. Then in turn, we add functions and get to Calculus, we add the square root of minus 1 and get Imaginary numbers, and thence complex calculus; we add multiple dimensions product spaces and get Vectors and Vector Calculus. And so on. Note that everyone uses the same counting techniques, regardless if they know any of the other stuff.

The "Base and Profile" metamodel starts with a "Theory of Everything" and culls out useful subsets as profiles. Since there is no viable "Theory of Everything." we usually get an approximation "as best as known at the moment" and so every example eventually devolves into a "really big core, profiles and extensions" as we learn things we did not know before.

C.5.1 An equivalence Lemma:

Before we begin the discussion, we need to observe one basic fact noted earlier:

> ***LEMMA***
> Given a completed set of specifications, the two metamodels can support exactly the same things in terms of the content of the standards documents.

Essentially the Base of the second model is the union of the Core and all the extensions of the first model, and the various pieces of profile in the second, will eventually define a core and set of extensions for the first. The Core is the interaction of all profiles, and the extensions are the various differences needed to construct a profile hierarchy.

One might be tempted to go for a pre-modularized Base, and build profiles based on combinations of these modules. This will work only if the factoring of the Base is *a priori* obvious and if profiles can successfully be restricted to combinations of modules. If this is possible, and there is no external consideration that would force the Base's factors to be not separable, then this design model may give the advantages of both, with a minor restriction on how profiles are to be defined (via selection of predefined parts, as opposed some uncontrolled decomposition independent of the Base's conformance classes).

C.5.2 Core and Extension advantages:

The biggest advantage to using a smallish core and multiple extensions is the ability to build in a "curriculum" of evolving functionality. Just like the Math Curriculum in schools systems, the core has to be minimalist and thereby easy to understand and implement (a metaphorical learning process). By starting with a very small core, we lay out step by step migration paths where users (here probably application implementors) can migrate to the complexity they need by picking and choosing among extensions. What you have is a set of specifications that can grow without bounds (just add a new extension if you need it, and can get the votes) but need only be implemented in parts based on the extensions. A universal implementation (one that covers everything) is universal only until the next extension hits the list. This is fundamentally the temporal argument for Figure C.4.

From a flexibility point of view, this is an ideal configuration. As new uses and requirements are discovered, new extensions come into being, and only those who need it, implement it.

C.5.3 Base and Profiles advantages:

The biggest advantage is predictability. At the beginning you define your universe - the base, and folks who want to be all things to all people implement the base. They can then handle all profiles, as long as they do not need to distinguish between them (unless they are separable factors as discussed earlier).

This works best for systems where the basic requirement is that everything must works on day-one of operation. To do this "day-one" gets postponed until everything that may affect the base specification has been handled. Welcome to the world of government procurement.

C.5.4 Default hybrid approach

As can be seen from the discussion above the distinction between the two metamodels can be only a matter of attitude. As a hybrid solution, a pre-modularized base made of separable factors recognized in the conformance classes can fit both models. This is in opposition to defining a universal, seamless base. The unfactored base standard can create two potential problems:

4. Complexity hinders Modification

5. Extensibility is Mandatory (see C.10.1.1).

It is obvious that a large complex specification is harder to fix than a small one. The suggested hybrid solution is to pre-factor the base into a core and a set of extensions. These seams act somewhat how the bulkheads in a ship work: they limit damage to small (hopefully fixable areas). They also provide a modular means to build profiles by the simple selection of the predefined modules.

C.6 Fundamental Conclusions

A more formal argument is given in C.7, but there are some fundamental conclusions that are intuitive from the discussions above.

C.6.1 The design of good modular code is core + extensions

This is object programming 101, but it does not imply that specification must be written to parallel the design of the code or system that implements it. But that is what happens most of the time. Case in point is the ISO TC 211 standards, many of which are behavioral in nature. As such they describe distinct behaviors that are exhibited by the elements of a systems, but do not imply any correspondence of that behavior to specific instances in the system.

C.6.2 The design of a good standard must fit core + extensions

Design and standards have the same form

QED

C.7 The root source of interoperability.

The most basic question is "what is the source of interoperability?" The long-accepted paradigm in standards creation has always been "common things done in a common manner." There is no reason to change that position. But this says the Base can never be the true source of widespread interoperability because it does not address the primary question of "what things are common among small groups."

In this case, the ‘80-20 rule’ usually applies, i.e. 80% of the work is covered by 20% of the design. This means a Core covering only one-fifth of the aspects of the rhetorical "theory of everything" will probably cover 80% of the interoperability needs. A Base specification is therefore a classic case of diminishing returns.

- The first 20% covers 80%,
- the second 20% covers 80% of what's left, or 16%,
- the third 20% covers 3.2%, and
- A 60% Base covers 99.2% of the interoperability problems encountered, a better score than anything we have, and still 40% to go.

But, there is a second problem. The second 20% of the Base is twice the work of the first 20%, and it gets worse. That is because the first 20% is the obvious stuff, the stuff that everyone has been doing for years, and have come to some ilk of a consensus in figuring out how to do it. That second 20% starts to get into things most folks don't do often enough to really care if they line up with other folks, and begin to contain what marketing wonks refer to as "differentiators" which are things the vendors do differently just so they can claim "cutting edge" technology - even though truth be known, it is more often "bleeding edge" technology with a fairly short life-span.

I would argue that the more a Core contains, the less likely it is to be Core-quality stuff. A small 20% Core probably reflects a complete consensus, and the extensions become the proving grounds for the "cutting edge." The cause of interoperability is better served by the 20% Core that truly reflect consensus than by a near 100% Base that is inherently unstable because it contains "bleeding-edge" technology that tomorrow will be truthfully referred to as "so yesterday."

C.8 Formal Description

The paper covered the definition of the two specification metamodels; this one extends the discussion to what impact the models have on use, implementation, market and mechanisms of adoption by various participants in the processes.

Being the Mathematical type, I'll state my conclusions as Definitions, Lemmas and Theorems, and summarize the line of reasoning behind them as informal proofs.

C.8.1 Definitions

C.8.1.1 simpler (of two alternative designs)

requires less effort to implement, including but not limited to efforts to understand the specification and to actually design and implement the part of an application for which the specification applies.

Note: Because simpler has the implication of smaller, we will treat it metaphorically like “less than.” Hence, its opposite “more complex” works like “greater than.”

C.8.1.2 simple

lower limits of comparison “simpler”

Note: That is one that is simpler than the majority, if not all of the examples,

C.8.1.3 complex

extreme opposite of simple; the upper limits of the comparison “simpler”

C.8.1.4 player:

person or group of people working in organization who are really good at doing that work

Note: From a belief that a master of a discipline will appear to act with his natural intuition in such a manner as to appear to be “playing at” the discipline for which he is a master.

C.8.1.5 core player:

player in the current market, with both a knowledge base (usually people, mostly designers and engineers), and an implementation base (code that works and is commercially viable)

C.8.1.6 implementation base

owned store of source code for the discipline that works and is commercially viable

C.8.1.7 knowledge base

store of knowledge about the discipline

Note: A knowledge base usually consists of people who are masters of the discipline, in this case mostly designers and engineers.

C.8.1.8 new player:

player who is not a core player

Note: A new player has the ability, but not necessarily the implementation or knowledge base to be a core player. In some cases this is a distinct advantage, since implementations have to be maintained, and what is perceived as knowledge may be really be "old but mistaken belief."

C.8.2 Theoretical Conclusions

C.8.2.1 LEMMA: Complexity within a specification favors adoption by core players, but is a barrier to new players.

The argument here is quite simple: a complex specification requires a larger implementation than a less-complex one. A specification that is aligned with "classical" knowledge can leverage existing code, design and expertise. Thus, a specification that uses what is commonly known allows existing players to leverage knowledge base and code base advantages in implementation. Since a complex specification is most likely to leverage common practices or concepts; they will tend to give advantage those already in the game as opposed to those not in it, i.e. potential new players.

C.8.2.2 COROLLARY: Extreme simplicity within a specification favors new players over existing core players.

This one is actually a result of the previous one, but may be non-obvious to those unfamiliar with the way most software engineers work, and a phenomena associated to the over-abundance of tools. Suppose we have a simple specification that can be implemented from scratch without much effort. Both the new player and the core player can implement the thing in a vacuum without much trouble. For the sake of argument, let's assume that the effort of this phase is about equal (we'll use that as the definition of "extreme simplicity"). Now a new player is finished, but the core player is not, because he now has to integrate the new functionality (or the new interfaces to the old functionality) into the existing commercial products, and often wait some time for the "next release" of part of that core before can release the "new product or functionality." Depending on how modular his old stuff is, this might be easy or hard, but it is not the "zero-work" issue that it is for the new player.

There is a second phenomenon at work, which might be called the "full toolbox paradox." Given a simple task, a true expert may use more tools that are needed to accomplish it whereas the neophyte might look for a simpler solution, using his limited tools and find it. For example, in the 1950's cortisone was the new arthritis wonder drug, but very expensive, because it required a long and complex chain or organic chemical reactions. In 1953, a little know company called Upjohn in Kalamazoo, Michigan trumped all the great organic chemistry labs by finding a common mold that would collapse most of the steps into a single fermentation step. Given a simple, easy to understand specification (like the chemical structure of a new drug like cortisone), there is always the chance that a single inventive solution derived from a fresh look at the problem will out-perform all existing implementations. After the process was up and running, the cost of cortisone dropped from "10 times more than the same weight of gold" to a more favorable comparison to aspirin.

C.8.2.3 THEOREM: Core and Extensions Metamodel standards give simpler specifications for the same functionality as Base and Profile Standards.

The argument is that since Base and Profile standards have to "build down" to a specific level; that they require the reader to understand not only what is being done, but also what is not being done (i.e. parts of the Base eliminated from a particular profile and why they were eliminated). Core and Extension standards only require understanding of what is being done. This may not affect the implementation effort, but does slow the understanding and design effort.

C.8.2.4 CONJECTURE: Core and Extension Metamodel standards favor the extension of the GI market to new players and thus, to some extent, undermines the advantage of current core player.

The first part of this is a direct result of the above discussion; the second part is conjectural, because it can be over-ridden to some degree by acceptance of the new rules by the designers and software engineers of the core players involved. It also can be overcome by the "head-start" that participants in the specification process can "pre-learn" and "prototype" the new paradigm while it is being designed (the period during the writing of the specification, especially if a test-bed or interoperability initiative or experiment is involved), giving them an advantage if they are willing to take it. Thus, the conjectural part is dependent on the mind-set and the level of participation of the core player. Since "forewarned is forearmed" the core players willing to participate fully in the "standards setting body" that is writing the specification, may still have a time advantage, as great or greater than the advantage dependent on their better implementation base. That means that the "some extent" of the second clause may ultimately go to zero or even flip to an advantage to the "participants" in the process - usually the core players.

Finally, there is a connection to the "core mission statement" discussions:

C.8.2.5 COROLLARY: The Core and Extension Metamodel standards approach favors the expansion of the GI market well beyond the current GI core players.

C.9 Standards Creation Scenarios

This section describes some specification-creation scenarios, and how they might work and why.

C.9.1 CASE 1: You have a viable "Theory of Everything" (a complete reference model) - at least for the industries you are worried about.

In this case, you can go either way.

A Base specification would be feasible because you have your Reference Model finished and you know which things need to be covered. It would be a bit of long job, but essentially doable. Then you get to start profiles. In each profile, you start with the Base specification and start hacking things off of it. To understand a profile properly you need a grasp of the base, and an understanding of why things were kept in the profile, and why things might have been whacked off. In other words, a proper understanding of the profile may be actually more complex then understanding the base, mainly because you are doing a negative requirements analysis, i.e. you are not only finding out what you need, you are also finding out what you don't need, in spite of the fact that the "High and Mighty Guru and Avatar of All Knowledge" (the writers of the Base) told you in the Base specification that the thing your are rejecting was a fundamental part of creation. Okay, I overcooked the metaphor, but it works for me.

In a Core and Extension specification, you could still write a Base specification, but put it in Volumes so that the seams between things are obvious. This is pretty much what ISO 21000: MPEG-21 did. They wrote a 14 part specification, and some parts had parts (ISO REL has 3 - basic, standard extensions (a bit of an oxymoron), and multimedia extensions). Building an Application schema is then fairly easy - start with the Core (identified as being used by everyone), and pick extension based on need, and create extension only if needed by the "special case" of your particular application.

One could argue that this is no different than the previous approach but, there is a difference in that the classical Base specification does not come in modules - the Core and Extension model does. If this is the case, the profile process has to inspect every aspect of the Base for usefulness, but in the Core and extension, one has the choice menu pre-divided into modules and building your brick wall of an application schema is simply a matter of picking out the bricks you need.

C.9.2 CASE 2: You do not have a viable "Theory of Everything" - at least not one that covers everyone's idea of everything.

This is actually where we are. Regardless of the "Abstract Specification" process, we have never completed our model of all of GI science - which may be an inherently impossible task, and so we may not need to be too hard on ourselves for not completing it. But we have what we have because it reflects what we know at the moment.

This is really hard for a Base and Profiles model. Since the Base is never complete, it gets modified from time to time (always more often than you want - or planned for - see Murphy's Law). This creates two distinct problems:

1. The Base is a large document, and unless modularized as "Core and Extension" can be a really nasty thing to fix.
2. The Base is the source document for all the profiles, and a modification to the Base (especially one involving a "major revision") may have to ripple through all profiles, cascading into the modification of a lot of very large documents (remember profiles are inherently bigger than extensions because of the issues discussed above).

The only really viable solution to these issues is shifting to a more "Core and extensions" model. The core should be one of the few documents that is referenced by all second tier specification, and it should be the most stable of all. Additions to the extensions list affects no current specification, and can lead to versioning of such only if the application level specification finds it needs the new functions. If extensions are part of the application specification process, most of the new additions should arise from local extension being accepted into the global pantheon. Core and extension modification is a much more elegant and much less disruptive process than a modification of the base.

C.10 Einstein’s 2-cents worth

Einstein is purported to have said:

Everything should be made as **simple** as possible, but not simpler.

Fortunately for Einstein, he did not have to define "simple" nor build a comparison operator.

Regardless of which specification metamodel you pick, you run into this problem at some point. In building profiles of a base document or building extension of a core document, you have to choose modular boundaries that make sense. One "advantage" of the base and profile model is that each profile can redefine it building blocks by refactoring the base document. One "advantage" of the core and extensions model is that you do not have to factor each time, since the modules (building blocks defined by the extension boundaries) are defined in the specification, and the application specification needs only to assemble them. So the difference is taking a marble block and chipping away that which is not David, versus building a skyscraper with understood, predefined and fixed components. (No concept of relative ease is intended as I suspect a good building is as complex as a good statue).

Consider the example of GML. It is right to point out that this "finding the modules in the marble" of GML is not a new task, and some of it was attempted in building GML as a collection of files. We probably could take the factoring of GML into multiple files as a first approximation to a core and extension alternative for GML if that is what we wanted to do. But that is not the battle we should engage in now. We should be much more interested in the question as a "go-forward" strategy as opposed to a "relive the past" exercise.

This is also reflected in the belief that the factoring of OGC technology into modules as a "do what works" program, not a "derivation exercise from first principles." For example, I am not sure I would want to factor geometry by coordinate dimension. Most of the distinction of geometric computational complexity is a question of representation, not dimension. For example, line strings and other linearly interpolations are easy in any dimension but splines are not. A 2D spline curve is much more complex computationally than a 3 or 4D line string. I believe that the flavor of Simple Features specification is much more in the restriction to linear interpolation than the restriction to 2D for version 1. Going to version 2 did not represent an issue for line strings in 3D, but we never even considered even 2D splines.

In factoring a domain into Core and Extensions, the real question is "where are the natural breaks that get reflected in folk's products?" Say for example we had some number of implementations of a generally equivalent type. We could then build a matrix cross-walk that mapped functions in each to a common descriptive base. This would give us a base categorization of the functions covered in the arena. The core is the collection of all parts that everyone does, and the various extensions are things that some do and some others don't. Building our common modules this way would mean that any two implementations that shared common functions could interoperate using the module with those function in it, and each vendor could completely categorized his functionality by listing the modules he supported.

C.10.1.1 Gödel Incompleteness Theorem and its role

Gödel (First) Incompleteness Theorem states:

> For any consistent formal theory that proves basic arithmetical truths, an arithmetical statement that is true but not provable in the theory can be constructed. That is, any theory capable of expressing elementary arithmetic cannot be both consistent and complete.

This seems a little out of left field, but it has important implication. It basically says that if you construct a formal description of a discipline (consistent formal theory) of any complexity (it is hard to avoid arithmetic, especially in a computer system design) then you either missed something that is true that you can’t prove (not complete), or you’ve got a bug in your theory (inconsistent statements that cannot both be true).

What that means for us is there is no “theory of everything” for our field or any field of any complexity. So, a base standard that represents a “theory of everything” must be flawed in some manner – it is either inherently buggy, or it missed something. This means any standard will eventually have to be extended to cover something important it missed.

For the curious, the first known near example in Mathematics is the Continuum hypothesis (or its negation, we do not know which). It is actually an undecidable in standard set theory. One could add it (or its negation) as an axiom, but a statement about trans-infinite cardinalities would seem out of place in a list of statements about set inclusion, and operations.

Bibliography

[1] Object Management Group (OMG), February 2007, Unified Modeling Language: Superstructure, version 2.1.1 , formal/07-02-05, available from OMG.org at http://www.omg.org/cgi-bin/doc?formal/07-02-05

[2] Object Management Group (OMG), February 2007, Unified Modeling Language: Infrastructure , version 2.1.1 , formal/07-02-06, available from OMG.org at http://www.omg.org/cgi-bin/doc?formal/07-02-06

[3] ISO/IEC JTC 1, ISO/IEC 9075:2003 -- Information Technology -- Datatbase Languages – SQL.

[4] ISO/IEC JTC 1, ISO/IEC 13249-3:2006 -- Information technology -- Database languages -- SQL multimedia and application packages -- Part 3: Spatial

[5] W3C, XML Schema 1.1 Part 1: Structures, 31 August 2006, available from W3C at http://www.w3.org/TR/xmlschema11-1/

[6] W3C, XML Schema 1.1 Part 2: Datatypes, 17 February 2006, available from W3C at http://www.w3.org/TR/xmlschema11-2/