Open Geospatial Consortium, Inc.

Date: 2010-09-27

Reference number of this document: OGC 11-107

Version: 1.0

Category: Public Engineering Report

Editors: Rob Atkinson, James Groffen

OGC® OWS-8 Domain Modelling Cookbook

Document type:	OGC® Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

Preface

Use Case modeling has been undertaken to establish the scope and business requirements for system and domain information modeling. INSPIRE has elucidated on this process (INSPIRE, 2007) and similar approaches have been used for domain models such as GeoSCIML (Sen and Duffy, 2005).

This document assumes formal domain modeling undertaken according to the ISO 19103 Conceptual Schema Language (ISO, 1999) and 19109 Rules for Application Schema (ISO, 2000).

A number of additional perspectives may be added from the experience of these and other domain modeling exercises.

Contents

Page

Figures

OGC® OWS-8 Domain Modelling Cookbook

1 Introduction

1.1 Scope

This OGC^{TM} document describes best practices for building and maintaining inter-related domain models, which have dependencies upon multiple systems. It describes how to build interoperable, maintainable domain models, the challenges and pitfalls faced in building these models, the techniques and patterns that should be applied, and specific tools that can be used. The theory of domain modelling is addressed, followed by practical step-by-step instructions on how to use of the tools. Examples are provided from Aeronautical Information Exchange Model (AIXM) and Farm Markup Language (FarmML) as they were refined in the OGC's OWS-8 testbed.

In line with OGC testbed principles, this document is provided in draft form. There are areas where extra sections can be incorporated and the described tools are under rapid development and should be revisited in future document versions.

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

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

1.2 What is a Domain Model?

A domain model represents the vocabulary and key concepts of a real world system. It identifies entities and attributes of the entities, along with the relationships between entities. For instance, a domain model of a city may include buildings, streets, and city furniture. A building may be broken down into a garage and house, and the house entity may describe a roof and geometry of the building.

Domain models are important for describing, analysing, visualising, and sharing information about all sorts of systems; from describing house plans, to analysing impacts of climate change on river systems.

The challenge with domain modelling is that most systems are inter-related with other systems. This leads to challenges with inter-dependency – aligning concepts between models and stability of versions of subcomponents.

1.3 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
James Groffen (editor)	LISAsoft
Rob Atkinson (editor)	CSIRO
Johannes Echterhoff	iGSI
Cameron Shorter	LISAsoft

1.4 Revision history

Date	Release	Editor	Primary clauses modified	Description
21 JUN 2011	0.1	Rob Atkionson	throughout	Document framework
26 JUN 2011	0.2	Jim Groffen	throughout	Expanded on framework. Draft ready for OWS-8 Aviation thread.
27 JUN 2011	0.3	Cameron Shorter	throughout	Added Pre-Deliverable Disclaimer
8 AUG 2011	0.4	Jim Groffen, Rob Atkinson	Throughout	Describe concept / apply process approach taken. Document first two phases of this.
15 AUG 2011	0.5	Jim Groffen	throughout	General review and added detail
26 SEP 2011	0.6	Cameron Shorter	Scope,	General review

1.5 Future work

tbd

1.6 Forward

This document is a deliverable of the OGC Web Services (OWS) Initiative - Phase 8 (OWS-8). It describes a recommended approach to domain modelling following ISO and OGC standards. The approach described supports GML application profiles defined using formal UML. Also described are tools to support the modelling process from conceptual design (UML) to physical implementation (XML Schema).

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- Digital NOTAM Event Specification v1.0
- INSPIRE, E., 2007. D2.6: Methodology for the development of data specifications. INSPIRE Drafting Team "Data Specifications"
- ISO, 1999. ISO 19103 Geographic information - Part 3: Conceptual schema language. International Organization for Standardization (ISO).
- ISO, 2000. ISO 19109.3 Geographic information - Rules for application schema. International Organization for Standardization (ISO).
- Sen, M., Duffy, T., 2005. GeoSciML: Development of a generic GeoScience Markup Language. Comput. Geosci., 31(9): 1095-1103.
- Solid Earth and Environment GRID (SEE GRID) community website; https://www.seegrid.csiro.au/wiki

3 Terms and definitions

Features

Attributes

Relationships

Abstraction

Structure

Data Products

Separation of Concerns

Application Schema

Implementation Schema

Subversion

HollowWorld

Enterprise Architect

4 Conventions

4.1 Abbreviated terms

AIXM	Aeronautical Information Exchange Model - A GML Application Profile that defines feature types and standardised encoding methods for aeronautical information.
DNES	Digital NOTAM Event Specification -
GML	Geographic Markup Language - as a modeling language for geographic systems as well as an open interchange format for geographic transactions.
NOTAM	Notice to Airmen - A NOTAM is filed with an aviation authority to alert aircraft pilots of any hazards *en route* or at a specific location. The authority in turn provides a means of disseminating relevant NOTAMs to pilots.
UML	Unified Modelling Language - a standardized general-purpose modeling language that includes a set of graphics for visual representation.

4.2 UML notation

Diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r3].

This document introduces the concept of formal UML notation that requires conformance of the UML model to an ISO / OGC standardized approach. Conformance can be checked with a provided tool – Solid Ground.

5 A Case For Domain Modelling

Domain modelling is expensive and difficult. Why are you going to do it? What are the business requirements to be met by domain modelling and are they worth it?

Much literature exists around the purpose and value of formal information modelling in the general “enterprise” or “development” spaces. Domain modelling is focused on being able to share information semantics across a “universe of discourse” (ISO19101) - i.e. to specify concepts and information organisation (schemas) between different actors, typically operating in different enterprise and governance contexts. In the OGC context this is further extended to imply platform independence - information models in an interoperability context.

In short, interoperable systems require formal information modes to specify the interfaces and data structures they expose, and increasingly, the content and definitions involved in the semantics of information exchange.

Formal domain models are required in order to be able to interpret such specifications - if a standard approach and formalism is not used, such specifications will be harder to read and interpret.

The underlying issue is that similar concepts occur across multiple domains - and if every domain "reinvents the wheel" it becomes increasingly costly and difficult to decide what the scope of a domain is, and many competing, overlapping models will arise, making it impossible to build interoperable components.

The solution is a good formalism, methodology and infrastructure to allow re-use of model components.

The formalism used here is ISO 19103 (Conceptual Schema Language) and ISO 19109 Rules for Application Schema (including the General Feature Model, or **GFM)**.

5.1 Re-use of common concepts

Re-use of common concepts is a high level business concern – impacting:

- Cost of development;
- Control over lifecycle;
- Ability to interoperate with related domains;
- Availability of tools to operate on the model.

Thus the first challenge for any domain is to determine how best to re-use existing concepts. In ISO 19100 context used by the OGC this is achieved formally through a **package import** between **Application Schemas.**

Choosing a package to import is however problematic, but becoming less so with the advent of publication of libraries ISO Harmonised Model and SWE.

In practice, one can only import modules from a context that is at least as stable, and recognised by the domain – for example a national or international context may readily use elements from the ISO or OGC baselines, but not an unmaintained output from an academic project, regardless of how excellent or commonly used it is.

Even if a suitable library is not available, projects and technology-based models may provide useful clues as to how to structure reusable components.

Rule of thumb: even if a suitable candidate is not immediately available, assume common concepts will be available at some time in the future and partition into separate modules.

The next question, then, is how to partition models. This can be achieved by analysing the governance and interdependencies to determine which parts belong in separate packages. For example a domain concerned with water quality might include concepts of chemical species, but wont "own" these concepts, therefore they should be in a separate package.

5.2 Governance

It is critical to understand the scope of each component of a domain model. The best way to do this is to establish the rules for maintaining a module – who are the stakeholders and why.

Avoid including any definitions in a package that cannot be maintained by the designated package maintainer – these can be put in separate interim packages and imports, pending identification of a suitable candidate.

This leads to a practical problem, which has stopped people from following these principles in practice - it is very hard with "off the shelf" modelling tools to swap in different versions of imported packages as they become available. Fortunately, tools to perform these functions are now available, through CSIRO's freely available SolidGround plugin to Enterprise Architect. Similar functions could be developed for other modelling platforms, or provided as infrastructure services in future.

For more on Governance look at the Sustainable Model Management section.

5.3 Assumed knowledge

What is often missing from Use Case based efforts to define the scope of application domains is a detailed analysis of what questions an information model supports, and what knowledge the user must have to invoke such a question.

Relationships between classes should be viewed in the context of whether they are "traversable" within the context of the Use Case.

There is thus a strong relationship between the intended deployment architecture (services) and the information model, based on the relationships between FeatureTypes.

It may be necessary to define specialised FeatureTypes that represent denormalised views of the concepts that may be exposed by service interfaces. This in turn raises questions about the degree of abstraction of an information model. Implementation models are very hard to re-use across domains, except where they represent a specific metadata-rich data product and services that may be consumed by external domains. Simple models are safe when an application is expceted to have access to all the relevant metadata in advance.

It is recommended that conceptual, reusable elements are modelled separately to "data product" oriented implementation views. How these are best related is an ongoing research topic, but pragmatism suggests using simple sharable concepts as data types within implementation packages

Domain Modelling in Practice

Throughout this document, a working example of performing domain modelling will be presented using two real-world examples. These examples both form small parts of larger, complicated domains in different stages of modelling.

The first example is of progressing an existing domain modelling effort - the Aeronautical Information Exchange Model or AIXM. A portion of this model is being used to migrate the existing Notice to Airmen or NOTAM messaging system to a digital process encoded using AIXM - the Digital NOTAM Event Specification. This effort is already well underway. This document will take the existing work and apply the practices detailed in this document to it.

The second example is of starting a new domain modelling effort - FarmML. FarmML is being created to deal with a need of Australian Government to manage agricultural land use across state and federal organisations.

New concepts presented in each section will be put into practice and documented along with the concepts, based on these two examples.

6 An Environment for Modelling

To begin with we will need an environment to develop domain models. This must respect the rules from ISO 19109, and this can be done by using a formal UML Profile and the dta type libraries from ISO. Libraries from OGC and other domains of interest may also be required.

While domain modelling can be done with a variety of tools, following the procedures described in this document is best achieved by taking advantage of the tools that have been developed for Enterprise Architect and Subversion.

Two approaches are suggested. The first is describe in Appendex A, and based on a series of Subversion repositories, and is widely used under the label of the "HollowWorld" recipe. The second approach is currently experimental based on a model registry concept, and supported as part of the SolidGround toolset.

https://wiki.csiro.au/confluence/display/solidground/Solid+Ground+Toolset

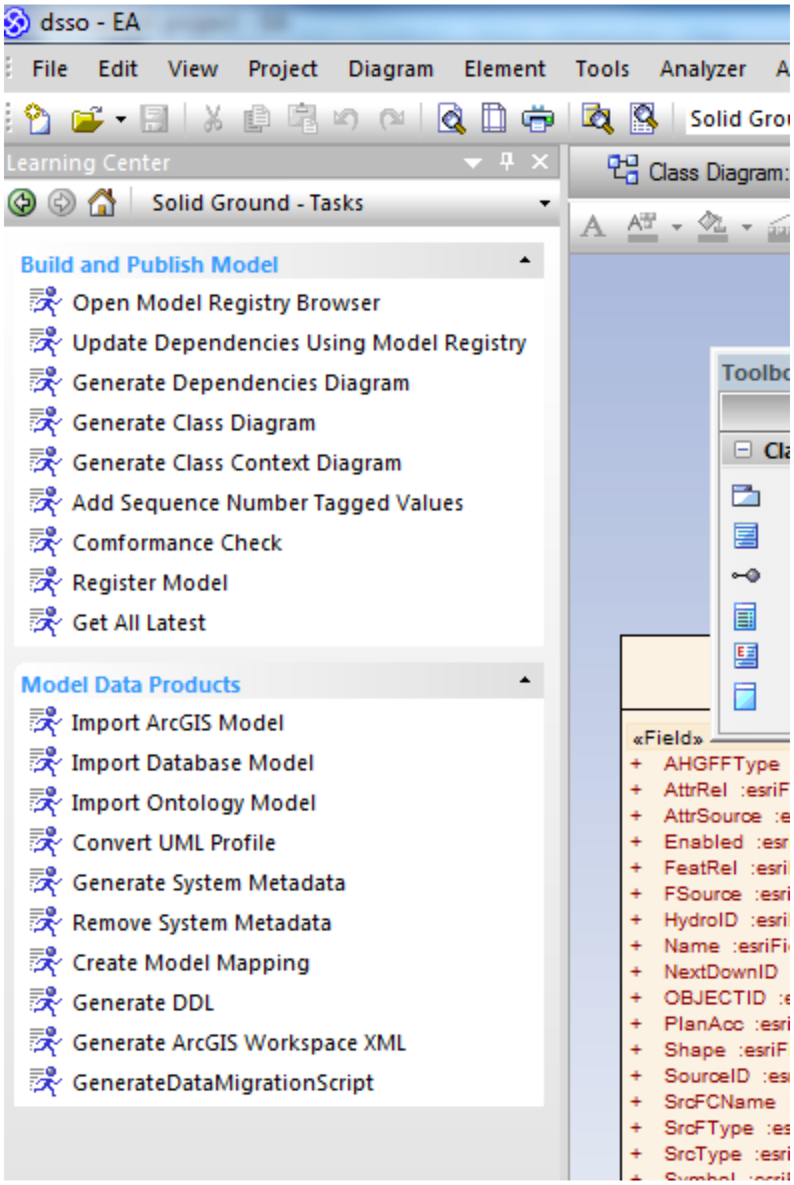
Modelling taking into account all the issues discussed here can be undertaken manually (i.e. using a generic UML editing environment), although this has been found to require a great deal of skill and familiarity with UML. SolidGround provides a suite of tools for automating various modelling tasks needed for domain modelling.

The tools can be applied in a step-wise process, depending on the context. Two common contexts are:

1. build a model from a library of model components

2. reverse-engineer a data product from an implementation

Refactoring an existing model can be performed using the same tasks, and the full extended menu of tools and manual processes, but will vary in execution depending on the nature of the starting model.



Build and Publish Model

- Open Model Registry Browser
- Update Dependencies Using Model Registry
- Generate Dependencies Diagram
- Generate Class Diagram
- Generate Class Context Diagram
- Add Sequence Number Tagged Values
- Comformance Check
- Register Model
- Get All Latest

Model Data Products

- Import ArcGIS Model
- Import Database Model
- Import Ontology Model
- Convert UML Profile
- Generate System Metadata
- Remove System Metadata
- Create Model Mapping
- Generate DDL
- Generate ArcGIS Workspace XML
- GenerateDataMigrationScript

«Field»
+ AHGFFType
+ AttrRel :esriF
+ AttrSource :e
+ Enabled :esr
+ FeatRel :esri
+ FSource :esri
+ HydroID :esri
+ Name :esriFi
+ NextDownID
+ OBJECTID :e
+ PlanAcc :esri
+ Shape :esriF
+ SourceID :es
+ SrcFCName
+ SrcFType :es
+ SrcType :esri

Figure 1 – Solid Ground Tasks in the Learning Centre

6.1 Abstraction

Choosing the right level of abstraction for a model is a very challenging task. A low level of abstraction will generate an implementation-specific model at the cost of lower general applicability. A high level of abstraction will allow other domains to specify interoperability at the semantic level, but may be harder for domain users familiar with specific data products to read.

As a rule of thumb, model either data products (to be exposed via service interfaces) or definitions of things for re-use (registers of features).

If the goal is to share identified features across multiple users, then the model needs to reflect the requirements of the registration process – it must be focused on the needs of the identifying party. Choose a high-level of abstraction, including only those attributes and relationships needed to assist in distinguishing between different individuals.

For convenience, this model can be extended with additional attributes, or mapped to a simpler structure, to assist data transfer. This will lead to related models, but a simpler and clearer focus for each model.

Data products should be modelled around familiar and convenient packaging and services.

This separation of concerns is commonly understood within the database design world, with normalised databases accessed via views (or data warehouses accessed via "data marts").

In either case, however, modularity has a role to play.

6.2 Patterns

A number of key modelling patterns are common and have profound implications.

The first to consider is the "root class for the domain" model. In this pattern all classes in a domain inherit from a single class, which contains common metadata.

Two examples of this are in the ESRI workspace model, where FeatureClasses are derived from geometry objects (Point).

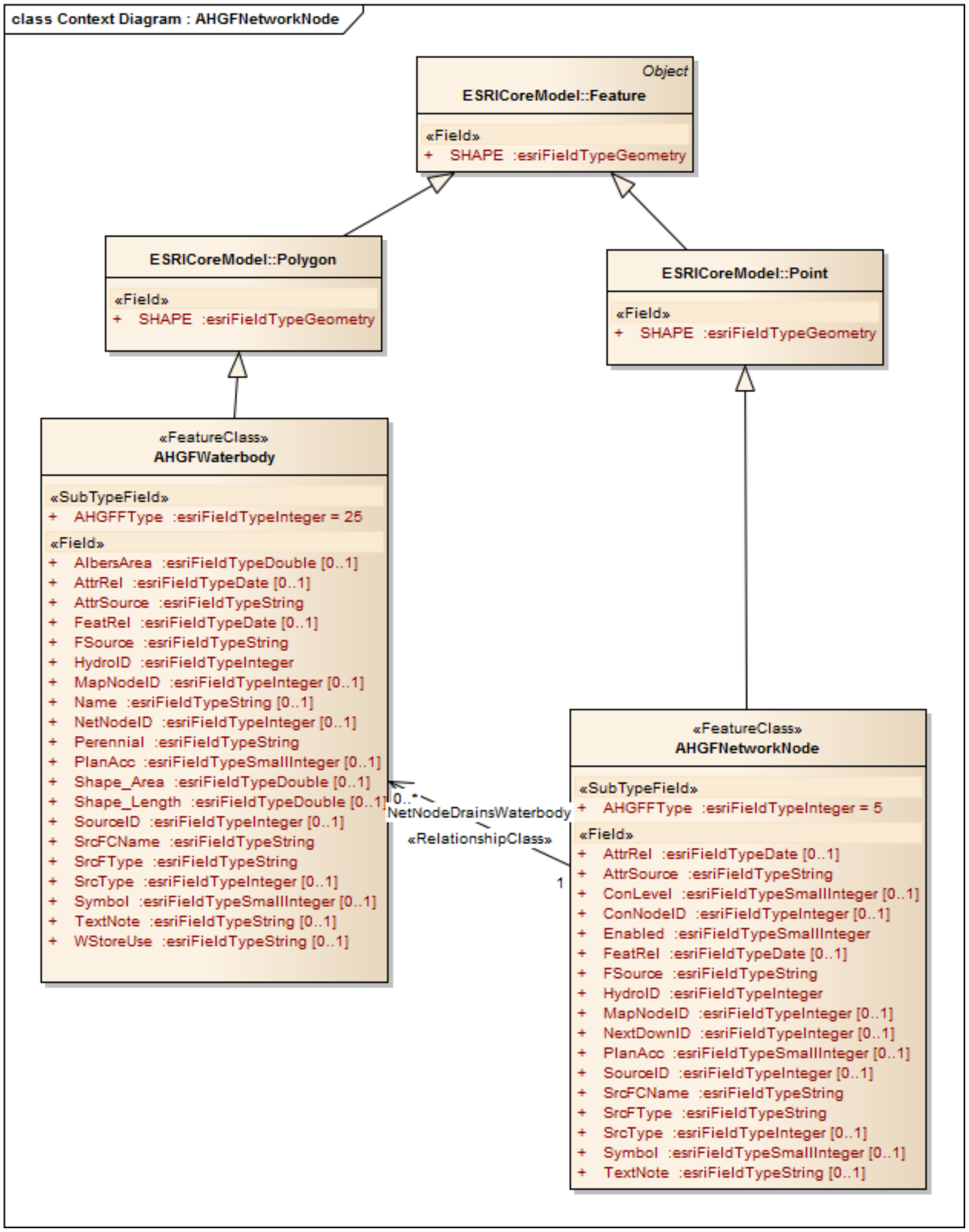

class Context Diagram : AHGFNetworkNode
Object
ESRICoreModel::Feature
«Field»
+ SHAPE :esriFieldTypeGeometry
ESRICoreModel::Polygon
«Field»
+ SHAPE :esriFieldTypeGeometry
ESRICoreModel::Point
«Field»
+ SHAPE :esriFieldTypeGeometry
«FeatureClass»
AHGFWaterbody
«SubTypeField»
+ AHGFFType :esriFieldTypeInteger = 25
«Field»
+ AlbersArea :esriFieldTypeDouble [0..1]
+ AttrRel :esriFieldTypeDate [0..1]
+ AttrSource :esriFieldTypeString
+ FeatRel :esriFieldTypeDate [0..1]
+ FSource :esriFieldTypeString
+ HydroID :esriFieldTypeInteger
+ MapNodeID :esriFieldTypeInteger [0..1]
+ Name :esriFieldTypeString [0..1]
+ NetNodeID :esriFieldTypeInteger [0..1]
+ Perennial :esriFieldTypeString
+ PlanAcc :esriFieldTypeSmallInteger [0..1]
+ Shape_Area :esriFieldTypeDouble [0..1]
+ Shape_Length :esriFieldTypeDouble [0..1]
+ SourceID :esriFieldTypeInteger [0..1]
+ SrcFCName :esriFieldTypeString
+ SrcFType :esriFieldTypeString
+ SrcType :esriFieldTypeInteger [0..1]
+ Symbol :esriFieldTypeSmallInteger [0..1]
+ TextNote :esriFieldTypeString [0..1]
+ WStoreUse :esriFieldTypeString [0..1]
0..*
NetNodeDrainsWaterbody
«RelationshipClass»
1
«FeatureClass»
AHGFNetworkNode
«SubTypeField»
+ AHGFFType :esriFieldTypeInteger = 5
«Field»
+ AttrRel :esriFieldTypeDate [0..1]
+ AttrSource :esriFieldTypeString
+ ConLevel :esriFieldTypeSmallInteger [0..1]
+ ConNodeID :esriFieldTypeInteger [0..1]
+ Enabled :esriFieldTypeSmallInteger
+ FeatRel :esriFieldTypeDate [0..1]
+ FSource :esriFieldTypeString
+ HydroID :esriFieldTypeInteger
+ MapNodeID :esriFieldTypeInteger [0..1]
+ NextDownID :esriFieldTypeInteger [0..1]
+ PlanAcc :esriFieldTypeSmallInteger [0..1]
+ SourceID :esriFieldTypeInteger [0..1]
+ SrcFCName :esriFieldTypeString
+ SrcFType :esriFieldTypeString
+ SrcType :esriFieldTypeInteger [0..1]
+ Symbol :esriFieldTypeSmallInteger [0..1]
+ TextNote :esriFieldTypeString [0..1]

Figure 2 — Example of defining a root class for the domain.

The second, and related pattern, that can be seen here is to include implementation oriented metadata into a class - such as “Created By”, “Modification Date” etc.

In general, such system metadata could be included into an application schema during the process of encoding into an implementation-specific model, in much the same way as gml:name is added to all FeatureTypes when encoding a UML model of the feature type.

These two patterns should be confined to implementation schemas, rather than models to express the underlying concepts and features in a domain.

In a Platform Specific Model, such as a ESRI Geodatabase specific profile, this may be realised by a stereotype rather than direct inheritance, as in:

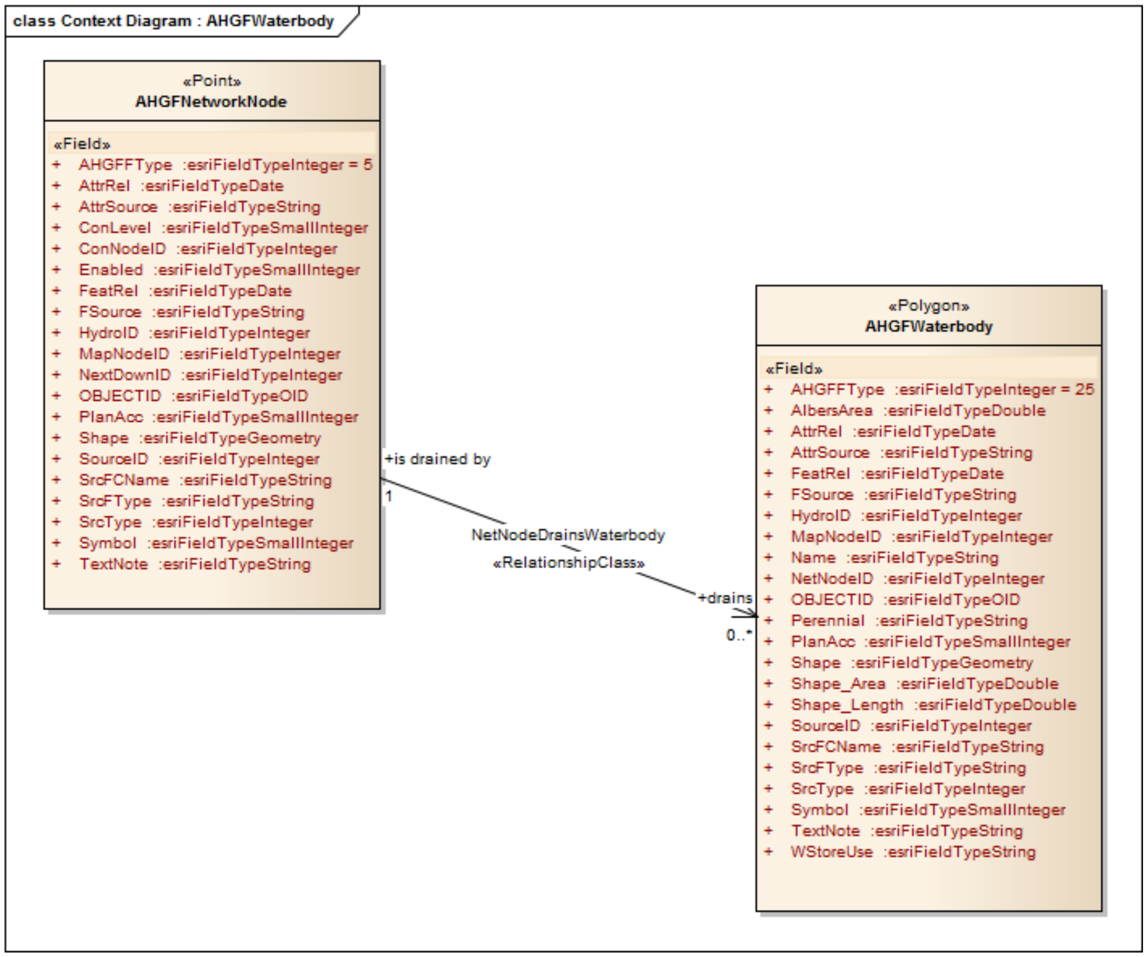

class Context Diagram : AHGFWaterbody
«Point»
AHGFNetworkNode
«Field»
+ AHGFFType :esriFieldTypeInteger = 5
+ AttrRel :esriFieldTypeDate
+ AttrSource :esriFieldTypeString
+ ConLevel :esriFieldTypeSmallInteger
+ ConNodeID :esriFieldTypeInteger
+ Enabled :esriFieldTypeSmallInteger
+ FeatRel :esriFieldTypeDate
+ FSource :esriFieldTypeString
+ HydroID :esriFieldTypeInteger
+ MapNodeID :esriFieldTypeInteger
+ NextDownID :esriFieldTypeInteger
+ OBJECTID :esriFieldTypeOID
+ PlanAcc :esriFieldTypeSmallInteger
+ Shape :esriFieldTypeGeometry
+ SourceID :esriFieldTypeInteger
+ SrcFCName :esriFieldTypeString
+ SrcFType :esriFieldTypeString
+ SrcType :esriFieldTypeInteger
+ Symbol :esriFieldTypeSmallInteger
+ TextNote :esriFieldTypeString
+is drained by
1
NetNodeDrainsWaterbody
«RelationshipClass»
+drains
0..*
«Polygon»
AHGFWaterbody
«Field»
+ AHGFFType :esriFieldTypeInteger = 25
+ AlbersArea :esriFieldTypeDouble
+ AttrRel :esriFieldTypeDate
+ AttrSource :esriFieldTypeString
+ FeatRel :esriFieldTypeDate
+ FSource :esriFieldTypeString
+ HydroID :esriFieldTypeInteger
+ MapNodeID :esriFieldTypeInteger
+ Name :esriFieldTypeString
+ NetNodeID :esriFieldTypeInteger
+ OBJECTID :esriFieldTypeOID
+ Perennial :esriFieldTypeString
+ PlanAcc :esriFieldTypeSmallInteger
+ Shape :esriFieldTypeGeometry
+ Shape_Area :esriFieldTypeDouble
+ Shape_Length :esriFieldTypeDouble
+ SourceID :esriFieldTypeInteger
+ SrcFCName :esriFieldTypeString
+ SrcFType :esriFieldTypeString
+ SrcType :esriFieldTypeInteger
+ Symbol :esriFieldTypeSmallInteger
+ TextNote :esriFieldTypeString
+ WStoreUse :esriFieldTypeString

Figure 3 – Platform specific details embedded in class

After removing the implementation patterns, and converting to the ISO idiom this might look like:

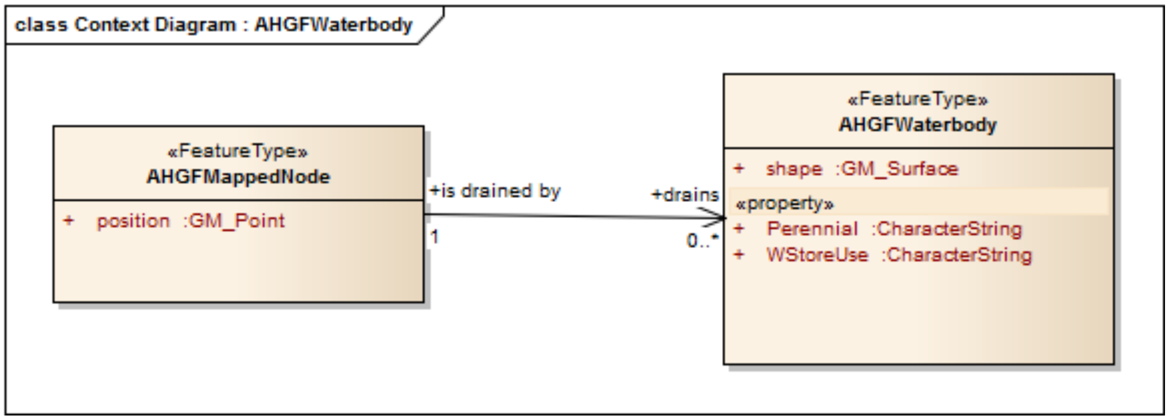


Figure 4 – Underlying concepts realised by stereotype

6.3 Application Schemas

Application Schemas are model packages that may be re-used. If you want to reuse a concept, you must import the containing Application Schema, so we need to design these carefully.

Large monolithic Application Schemas are expensive to build, difficult to test and very fragile, and not re-usable, so should be modularised following the well-known principles of simplifying complexity through **encapsulation.**

Having multiple modules creates more dependencies, but like in software development it is easier to develop, maintain and test discrete modules.

Application Schemas should contain any mutually interdependent definitions, and may contain:

- no FeatureTypes (i.e. just ties together other Application Schemas)
- a single FeatureType (i.e. is a container that specifies the governance of the definition)
- many FeatureTypes
- too many FeatureTypes (that could be logically separated).

Application schemas should ideally contain in the order of 5-10 classes, or just import a set of such application schemas.

6.4 Definitions

Models should be self-contained in terms of having documentation in the UML slots provided.

Citations (references to the original source) should be included (using tagged values for these to make the machine processable is being considered).

Data dictionaries should be imported if pre-existing, or maintained by automated export processes from the model as the point of truth.

(NB SolidGround provides tools for import/export of definitions to Excel and export to SKOS formats.)

Pay particular attention to relationships, putting notes on the target ends of UML associations (the semantics of an association property in the General Feature model).

6.5 Example Application of Modelling Practices

Let's take the modelling practices summaries above and apply them to our example scenarios. First we need some background on the example scenarios.

6.5.1 Digital NOTAM Event Specification - DNES

"Digital Aeronautical Information Update (Digital NOTAM) - a data set made available through digital services containing information concerning the establishment, condition or change in any aeronautical facility, service, procedure or hazard, the timely knowledge of which is essential to systems and automated equipment used by personnel concerned with flight operations."

6.5.1.1 Abstraction

Digital NOTAM is unusual in domain modelling in that it is has truly global relevance. This simplifies determining the appropriate level of abstraction. While pilots regularly deal with local conditions particular to some airspace, expected conditions are not in the scope of a NOTAM. NOTAM's can inherently apply to any air service, as they govern notices to a globally performed activity.

One of the primary goals of digitizing NOTAM is to improve automation of delivering notices relevant to the recipient. Currently a great deal of effort goes into evaluating which NOTAM's are relevant to the recipient and which can be safely ignored. This highlights a level of abstraction that must be catered for - all NOTAM must be filterable at a common level.

NOTAM's describe some event that causes an expected or unexpected effect that airmen need to be aware of. While the effects vary based on the scenario, the concept of a NOTAM being a message that notifies of the event and the impacts of that event is common.

6.5.1.2 Abstraction Patterns

AIXM provides a variety of features that are directly appropriate to Digital NOTAM. AIXM also deals with concepts pertinent to Digital NOTAM including:

- Temporality - NOTAM's describe a temporary or sometimes permanent change to a set of features. AIXM deals with this by allowing the definition of time slices that describe the normal (BASELINE), temporary changes (TEMPDELTA), permanent changes (PERMDELTA) or current state (SNAPSHOT).

The existing NOTAM system already includes a variety of specialisations such as SNOWTAM, BIRDTAM and the various different scenarios that a NOTAM can describe, such as an unserviceable navigational aid or a runway closure. These correlate directly to specialisations of a NOTAM message.

AIXM provides the bulk of implementation necessary for Digital NOTAM. Digital NOTAM introduces an Event schema and defines scenarios represented in specialised AIXMBasicMessage structures.

7 Model Hygiene

Why does it matter that a model is accurate and void of idiosyncratic detail? Remember that the goal of a model is documenting an agreement, but formal models are designed for processability - they need to be consistently interpretable by tools, and most likely will be transformed into XML (GML) schemas. Tools are emerging for creating ontology viewpoints, database schemas and documentation views based on formal models. [ref SolidGround]

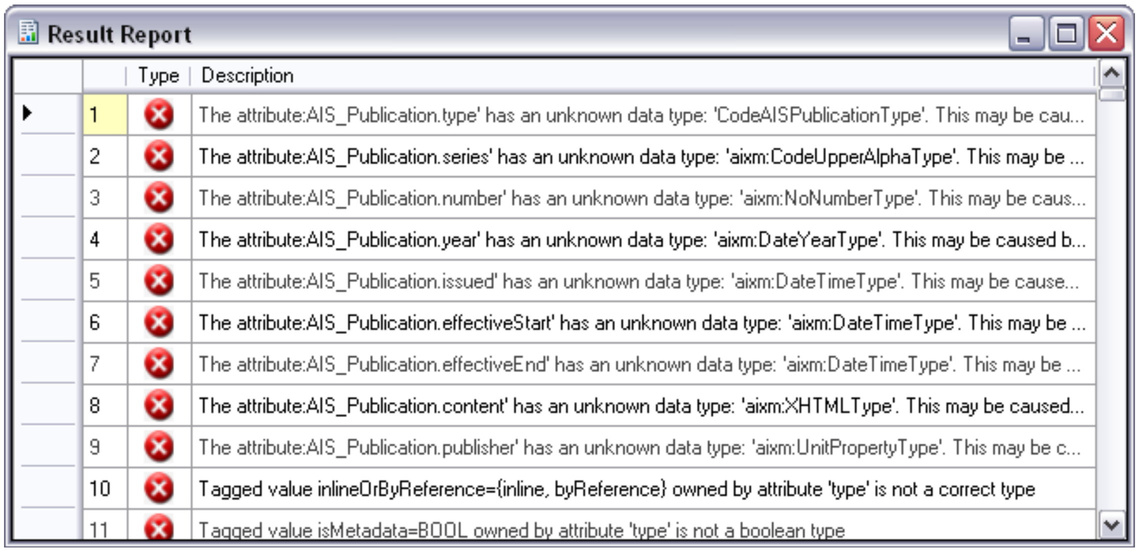
Maintaining the internal integrity of a model is challenging, and tools and the nature of UML make some aspects particularly challenging.

The model is **not** the diagram – the diagram is a **view** of the model, but may not show all aspects. Hidden relationships may exist in the model, which change its meaning. Names may appear, for instance on attribute types, that are not connected by id to the definitions they are intended to.

Certain processes can be automated, such as the utilities provided by the SolidGround toolset.

7.1 Conformance checking

SolidGround and Fullmoon provide conformance checking functions that will catch most technical issues. An example of a report generated by SolidGround conformance checking is below:



Result Report

	Type	Description
1	❌	The attribute:AIS_Publication.type' has an unknown data type: 'CodeAISPublicationType'. This may be cau...
2	❌	The attribute:AIS_Publication.series' has an unknown data type: 'aixm:CodeUpperAlphaType'. This may be ...
3	❌	The attribute:AIS_Publication.number' has an unknown data type: 'aixm:NoNumberType'. This may be caus...
4	❌	The attribute:AIS_Publication.year' has an unknown data type: 'aixm:DateYearType'. This may be caused b...
5	❌	The attribute:AIS_Publication.issued' has an unknown data type: 'aixm:DateTimeType'. This may be cause...
6	❌	The attribute:AIS_Publication.effectiveStart' has an unknown data type: 'aixm:DateTimeType'. This may be ...
7	❌	The attribute:AIS_Publication.effectiveEnd' has an unknown data type: 'aixm:DateTimeType'. This may be ...
8	❌	The attribute:AIS_Publication.content' has an unknown data type: 'aixm:XHTMLType'. This may be caused...
9	❌	The attribute:AIS_Publication.publisher' has an unknown data type: 'aixm:UnitPropertyType'. This may be c...
10	❌	Tagged value inlineOrByReference={inline, byReference} owned by attribute 'type' is not a correct type
11	❌	Tagged value isMetadata=BOOL owned by attribute 'type' is not a boolean type

Figure 5 – Example results from a conformance check

In SolidGround's case, the report can generate errors that indicate an invalid model, or warnings that indicate breaks from best practice or potentially idiosyncratic model details.

Entries in the SolidGround report have context of the model element that is the cause of the error or warning. Clicking on the row will select the related element in the project browser.

7.2 Dependency Analysis

Identifying dependencies can capture potential problems such as mutual dependencies – these may be simple errors or logic problems that will make a model impossible to maintain.

Dependency cycles are not legal under the ISO rules for Application Schema. They also create significant problems generating implementations – for example GML schema.

Discrete packaging of classes that have well defined package dependencies is encouraged. These package dependencies are automatically modelled by the SolidGround tool, which will generate a UML Package diagram that includes dependency information that can be interpreted as normative.

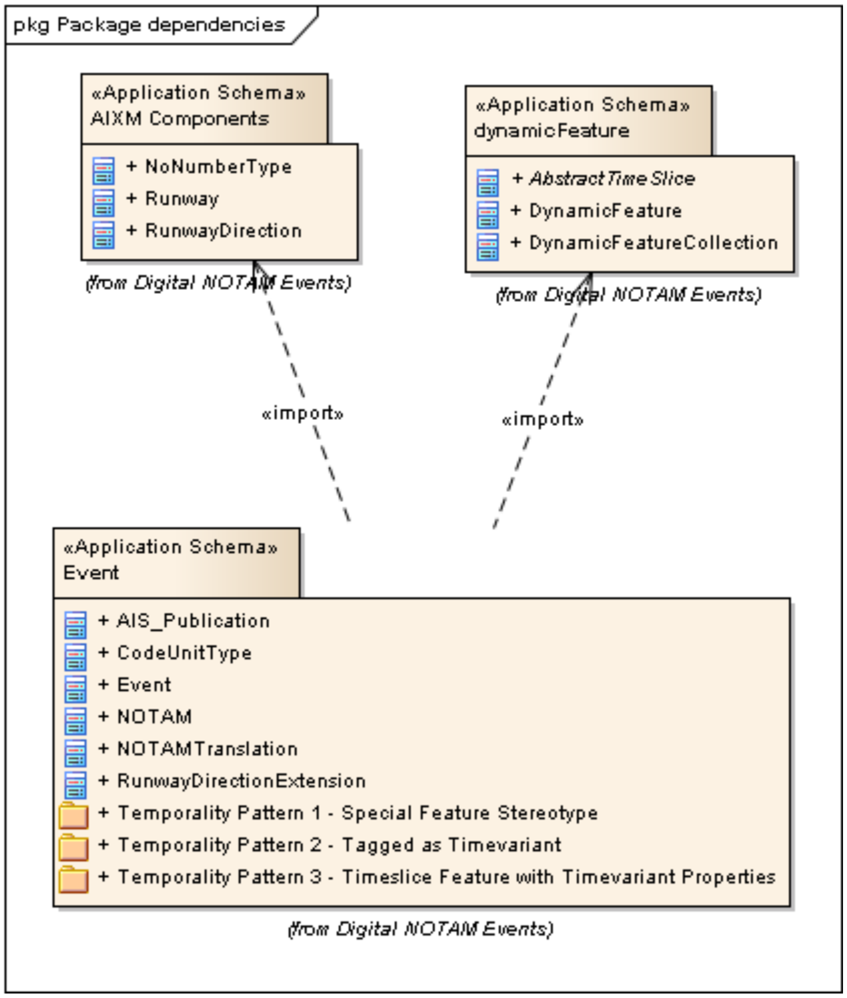
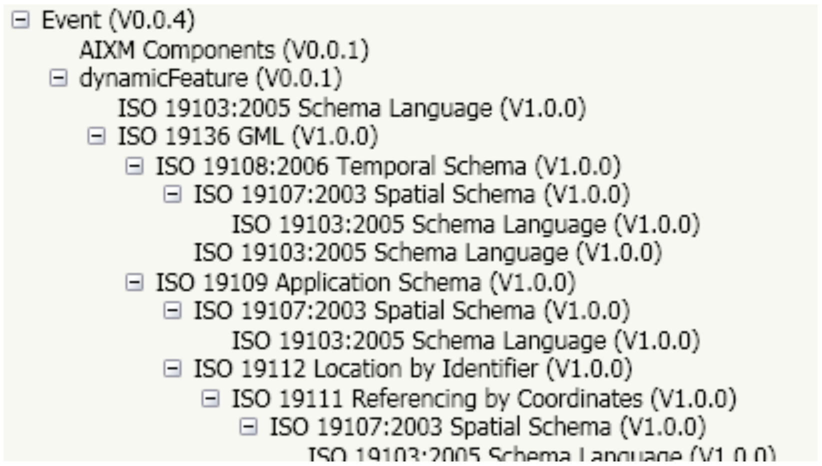

pkg Package dependencies
«Application Schema»
AIXM Components
+ NoNumberType
+ Runway
+ RunwayDirection
(from Digital NOTAM Events)
«Application Schema»
dynamicFeature
+ AbstractTimeSlice
+ DynamicFeature
+ DynamicFeatureCollection
(from Digital NOTAM Events)
«import»
«import»
«Application Schema»
Event
+ AIS_Publication
+ CodeUnitType
+ Event
+ NOTAM
+ NOTAMTranslation
+ RunwayDirectionExtension
+ Temporality Pattern 1 - Special Feature Stereotype
+ Temporality Pattern 2 - Tagged as Timevariant
+ Temporality Pattern 3 - Timeslice Feature with Timevariant Properties
(from Digital NOTAM Events)

Figure 6 – Package dependency diagram – dependencies between Application Schemas

By enforcing no circular references between packages they can be safely, individually consumed by external parties. Tools such as the Model Registry Browser can use formally defined dependency information to publish the dependency information.



```
⊟ Event (V0.0.4)
    AIXM Components (V0.0.1)
  ⊟ dynamicFeature (V0.0.1)
      ISO 19103:2005 Schema Language (V1.0.0)
    ⊟ ISO 19136 GML (V1.0.0)
      ⊟ ISO 19108:2006 Temporal Schema (V1.0.0)
        ⊟ ISO 19107:2003 Spatial Schema (V1.0.0)
            ISO 19103:2005 Schema Language (V1.0.0)
          ISO 19103:2005 Schema Language (V1.0.0)
      ⊟ ISO 19109 Application Schema (V1.0.0)
        ⊟ ISO 19107:2003 Spatial Schema (V1.0.0)
            ISO 19103:2005 Schema Language (V1.0.0)
        ⊟ ISO 19112 Location by Identifier (V1.0.0)
          ⊟ ISO 19111 Referencing by Coordinates (V1.0.0)
            ⊟ ISO 19107:2003 Spatial Schema (V1.0.0)
                ISO 19103:2005 Schema Language (V1.0.0)
```

Figure 7 – Dependencies published in the Registry Browser for the Event Model

This dependency information can be taken advantage of when importing models, in this case all dependent models can be imported from the registry.

7.3 Attribute Types

Names may be misleading, and a named attribute type may not actually refer to a base class (for example typing "GM_Point" into a datatype field is not the same as selecting it from the ISO 19107 base model).

This may not matter in some instances, such as name matching during GML schema generation, but the model is in fact incorrect, and automated dependency checking and other ways of processing the model (for example to create a Feature Type Catalogue) would be compromised.

It is also possible to have links to older versions – these may be batch updated if required using the SolidGround toolset.

8 Sustainable Model Management

Some of the practices presented here seem like a lot of work, and they are, but the trade-off is the potential failure of systems to survive through change without this process in place. Improved robustness, interoperability and efficiency of creating new systems.

Making models is comparatively cheap in comparison to maintaining models in the long term, especially as more complexity is introduced. Complexity is inherent because we are modelling the world and it's a complex place. Complicated is what we want to avoid.

One of the aims of the CSIRO effort is to introduce a sustainable approach to model management. Here is an outline of that approach:

- Introducing a Model Registry - Solid Ground
  - For managing libraries and getting the work of others
  - Benefits from Registry concepts such as automatic harvesting between registries.
  - Interpret the XML for the users
  - Keep the model up to date
  - Share a model within an organisation or between organisations
- Managing My Models - Subversion
  - Any version control system really, but SolidGround has some subversion interaction built-in
  - Version history, concurrent development of models.

8.1 Managing Dependency Associations

Dependencies are a two way street. Someone else is using my model and I want to update it. These updates may break their use of my model. They may also ask for changes to my model to support their use of it.

HollowWorld has a helper for managing version progression which helps enforcement of best practice for handling change to ensure those dependent on your model can work out what's going on.

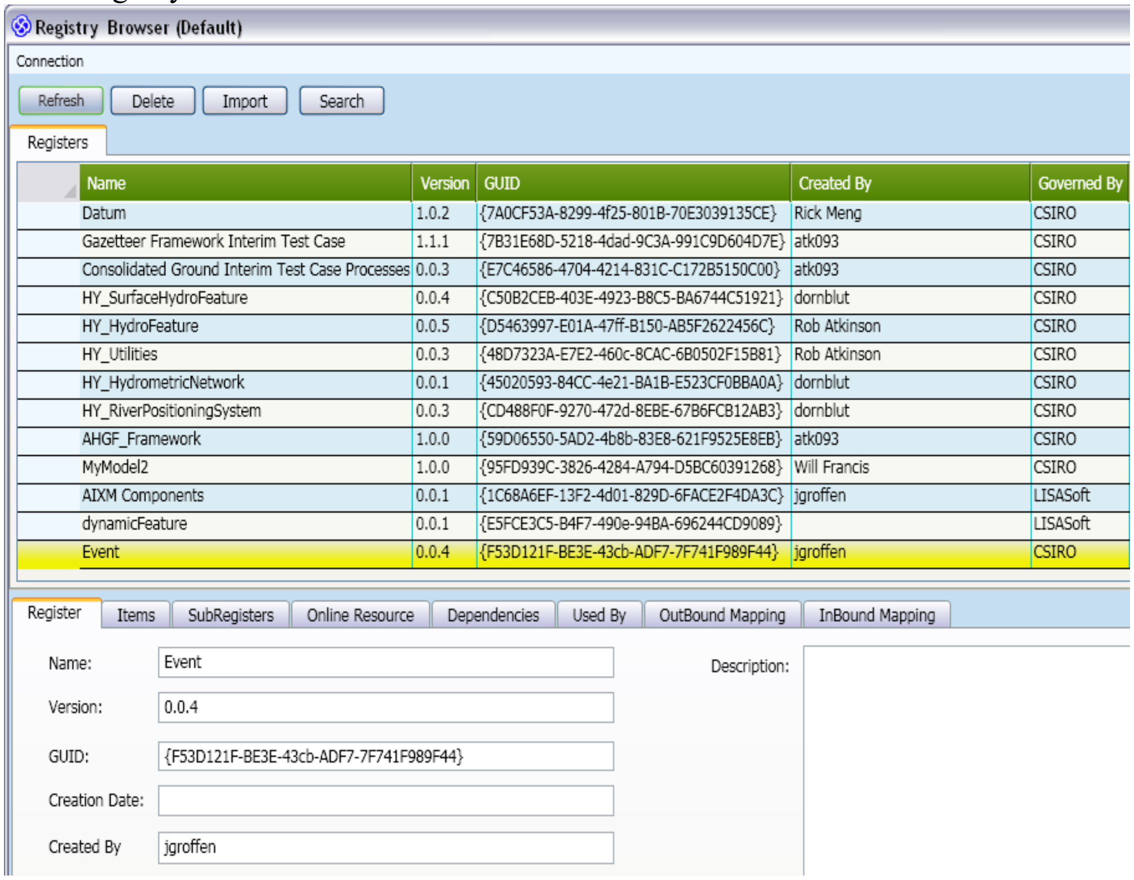
(Rob has a good set of slides on this topic)

8.2 Model Registry

By formalising the model it can participate in a broader community of domain models. Extensive work on the Registry Browser has been performed to provide a register of domain models that can be imported, extended and provide publishing services for models.

The image to the right is of the tasks available in the Solid Ground plugin for Enterprise Architect. This plugin interacts directly with the model registry in a few ways;

- Open Model Registry Browser: authenticates to and opens the model registry browser window.
- Update Dependencies Using Model Registry: All packages the selected package depends on will be updated from the model registry, picking up any changes that have been submitted to the registry.
- Register Model: initial registration or update of the selected model to the model registry.



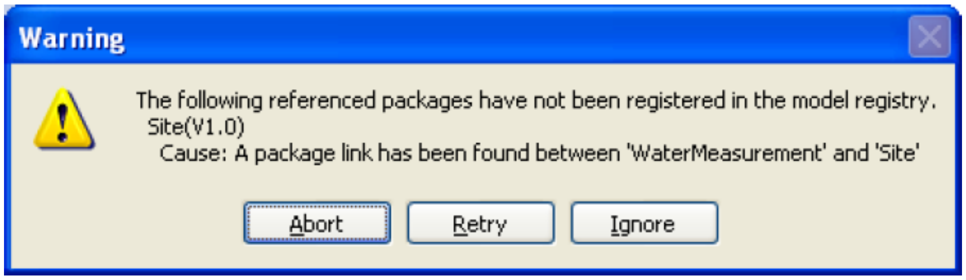
Name	Version	GUID	Created By	Governed By
Datum	1.0.2	{7A0CF53A-8299-4f25-801B-70E3039135CE}	Rick Meng	CSIRO
Gazetteer Framework Interim Test Case	1.1.1	{7B31E68D-5218-4dad-9C3A-991C9D604D7E}	atk093	CSIRO
Consolidated Ground Interim Test Case Processes	0.0.3	{E7C46586-4704-4214-831C-C172B5150C00}	atk093	CSIRO
HY_SurfaceHydroFeature	0.0.4	{C50B2CEB-403E-4923-B8C5-BA6744C51921}	dornblut	CSIRO
HY_HydroFeature	0.0.5	{D5463997-E01A-47ff-B150-AB5F2622456C}	Rob Atkinson	CSIRO
HY_Utilities	0.0.3	{48D7323A-E7E2-460c-8CAC-6B0502F15B81}	Rob Atkinson	CSIRO
HY_HydrometricNetwork	0.0.1	{45020593-84CC-4e21-BA1B-E523CF0BBA0A}	dornblut	CSIRO
HY_RiverPositioningSystem	0.0.3	{CD488F0F-9270-472d-8EBE-67B6FCB12AB3}	dornblut	CSIRO
AHGF_Framework	1.0.0	{59D06550-5AD2-4b8b-83E8-621F9525E8EB}	atk093	CSIRO
MyModel2	1.0.0	{95FD939C-3826-4284-A794-D5BC60391268}	Will Francis	CSIRO
AIXM Components	0.0.1	{1C68A6EF-13F2-4d01-829D-6FACE2F4DA3C}	jgroffen	LISASoft
dynamicFeature	0.0.1	{E5FCE3C5-B4F7-490e-94BA-696244CD9089}		LISASoft
Event	0.0.4	{F53D121F-BE3E-43cb-ADF7-7F741F989F44}	jgroffen	CSIRO

Name:	Event
Version:	0.0.4
GUID:	{F53D121F-BE3E-43cb-ADF7-7F741F989F44}
Creation Date:	
Created By	jgroffen
Description:	

Figure 8 – Registry Browser with the Event Application Schema selected

The registry browser supports version tracking features. When a model is registered the version number recorded in the Application Schema package of the model being registered is used to identify a model version. This is used to notify custodians of models that are dependant of your of the change, including the nature of the change by following a common Major.Minor.Revision approach.

Solid Ground will check dependencies when registering models. If the application schema you are registering references other application schemas that are not available in the model registry then the model register will fail. Also, circular dependencies will be discovered during this process, which will also cause the model registration to fail.



The following referenced packages have not been registered in the model registry.
Site(V1.0)
Cause: A package link has been found between 'WaterMeasurement' and 'Site'

Figure 9 – Error when registering a model - missing dependent package

The relationship between version control systems like subversion and the Model Registry is an interesting one. The model registry manages releases of a model that may be referenced by external parties whereas subversion or similar provides concurrent development and version control over development of the model.

While both can be used to manage and distribute domain models, the model registry is preferred for distribution because of the formal release management and dependency management features.

9 Appendix A - Tool Setup

9.1 Install Software

While using these tools is not required, this document will provide practical descriptions of using these tools. The modelling practices covered can be applied using any tools, though most likely with more manual intervention.

9.1.1 Enterprise Architect

Install Enterprise Architect (EA), preferably the latest version (9, build 908), but at least v7.5, build 850. A free trial version is available at:

- http://www.sparxsystems.com/products/ea/downloads.html

9.1.2 SolidGround

CSIRO have developed an EA plugin that automates some of the routine tasks, such as

- assigning sequenceNumber tagged values
- generating a context diagram for every class
- generating the package dependency diagram

This installer is available directly from the CSIRO at the moment by contacting:

- solidground-support@csiro.au

SolidGround should be installed at this point.

9.1.3 SubVersion

Install the latest SubVersion client, currently available from:

- http://subversion.apache.org

We will configure Enterprise Architect to link directly to packages managed in subversion.

9.1.4 HollowWorld

HollowWorld is a template for building GML Application Schemas. Using HollowWorld to build a domain model ensures the models will be easily transformed into a GML-conformant XML Schema which specifies the document format for transfer of domain data as a standard XML document, compatible with OGC WFS.

HollowWorld is a UML template that includes consistently pre-loaded a variety of geospatial standards including the ISO 19100 framework, which in turn are primarily from the ISO/TC 211 Harmonized Model.

To add HollowWorld to your domain modelling development environment, use SubVersion to checkout a local copy; either the trunk:

- https://www.seegrid.csiro.au/subversion/HollowWorld/trunk/

… or one of the stable branches:

- https://www.seegrid.csiro.au/subversion/HollowWorld/branches/release_1/
  2006 version of ISO/TC 211 Harmonized model
- https://www.seegrid.csiro.au/subversion/HollowWorld/branches/release_2/
  2009 version of ISO/TC 211 Harmonized model
- https://www.seegrid.csiro.au/subversion/HollowWorld/branches/release_3/
  2009 version of ISO/TC 211 Harmonized model + OGC SWE v1.0

To do this, run the following commands from a command prompt, replacing "your_workspace" with the location on disk where you want to store your local files, and the trunk or branch of your choice:

1. cd c:\your_workspace
2. mkdir hollowworld
3. cd hollowworld
4. svn co https://www.seegrid.csiro.au/subversion/HollowWorld/branches/release_3/

A copy of the ISO Harmonized Model will also be needed:

- https://www.seegrid.csiro.au/mirrors/iso-harmonized-model/isotc211/ISO%20TC211.xml

The full repository path which can be used to check out all trunk and branches is:

- https://www.seegrid.csiro.au/mirrors/iso-harmonized-model

To do this, run the following commands from a command prompt, replacing "your_workspace" with the location on disk where you want to store your local files:

1. cd c:\your_workspace
2. mkdir ISO Harmonized Model
3. cd ISO Harmonized Model
4. svn co https://www.seegrid.csiro.au/mirrors/iso-harmonized-model isotc211

There may be other models you will want to reference in your model, such as GeoSciML, EarthResourceML, MOLES, GWML. CSIRO make a variety of these available:

- https://www.seegrid.csiro.au/wiki/AppSchemas/AvailableModels

9.1.5 EA and SubVersion Authentication

Enterprise Architect piggy-backs on your local machine's authentication arrangements. You can accomplish this by following this procedure:

1. open a CMD window,
2. cd to the directory containing your local copy of a subversion you intend to access from Enterprise Architect
3. run `svn update`
4. when it asks, accept the cert (p)ermanently.
5. repeat for every other model

For every repository you intend to commit changes to, you will need to also perform the following:

1. open a CMD window,
2. cd to the directory containing your local copy of a subversion you intend to access from Enterprise Architect
3. run `svn --username fred.bloggs lock file.ext` (use your own uid and touch a file that actually exists)
    a. when it asks, enter your password
4. run `svn unlock file.ext` to undo what you just did

Note that you can instead manually maintain your models and update models you are using directly with SubVersion. In this case you won't need to link Enterprise Architect with subversion.

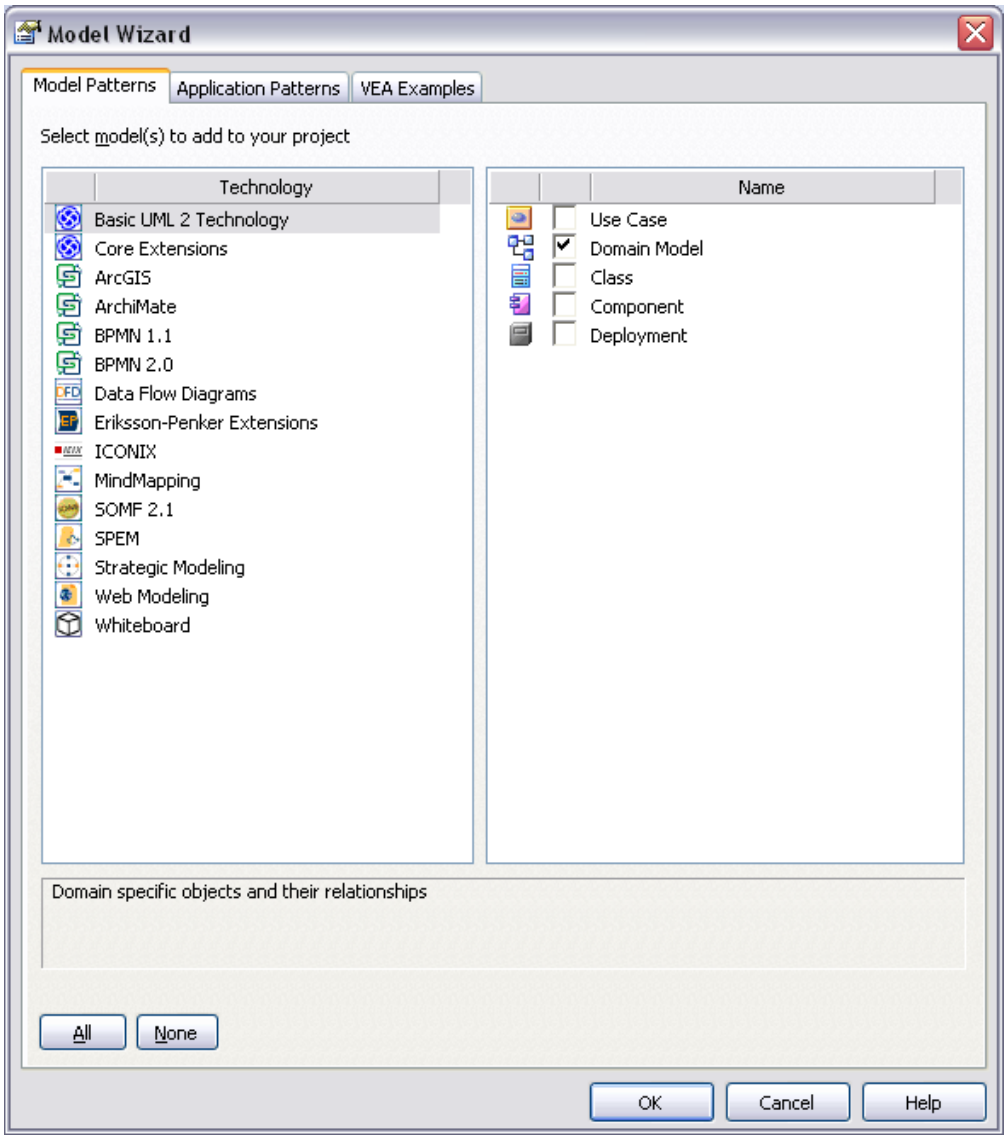
9.1.6 Configuring Enterprise Architect

For the rest of this document we will set a couple of parameters:

- the path where the local copy of your selected version of HollowWorld is stored will be designated $HollowWorldLocal
- the path where your local copy of the ISO Harmonized Model is stored will be designated $ISOLocal
- the path where the local copy of our new project will be designated $NewProject. This folder should not clash with either $HollowWorldLocal or $ISOLocal

Start Enterprise Architect and create a new project, choosing a name related to your application. We will choose ‘AIXM Digital NOTAM Events’. Create this new project in the $NewProject folder.

During project creation you will be asked what mode you want to be in, select ‘Domain Model’:



Select model(s) to add to your project

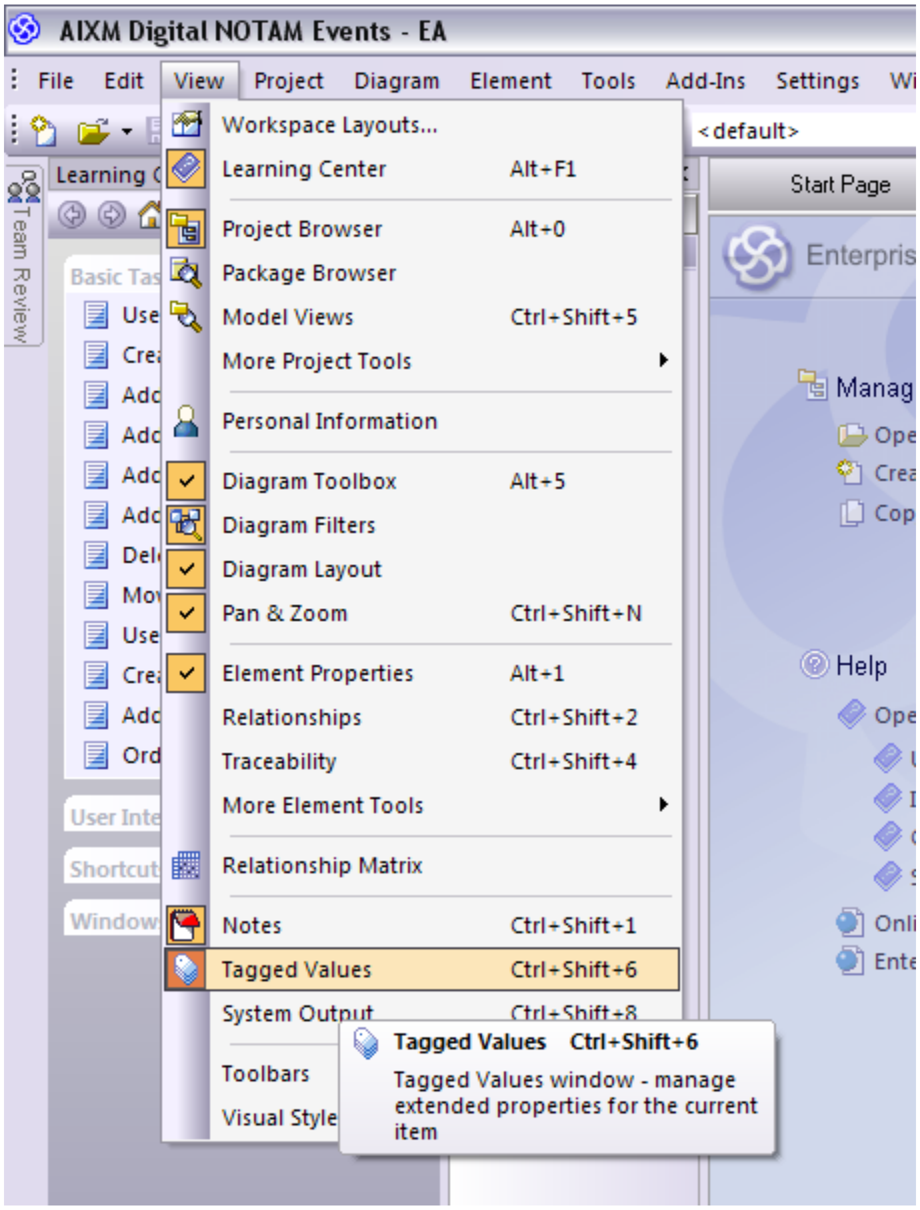
Technology
Basic UML 2 Technology
Core Extensions
ArcGIS
ArchiMate
BPMN 1.1
BPMN 2.0
Data Flow Diagrams
Eriksson-Penker Extensions
ICONIX
MindMapping
SOMF 2.1
SPEM
Strategic Modeling
Web Modeling
Whiteboard

	Name
☐	Use Case
✓	Domain Model
☐	Class
☐	Component
☐	Deployment

Domain specific objects and their relationships

Figure 10 – Enterprise Architect Model Wizard

Make sure the Tagged Values pane is open:

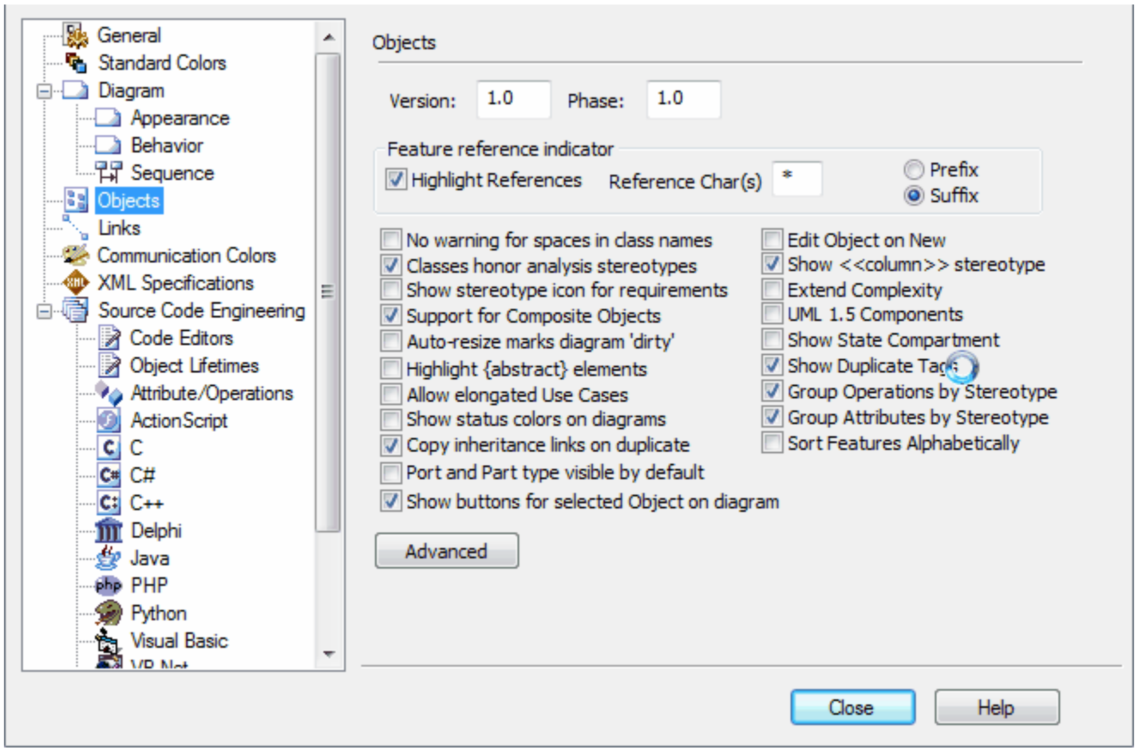


Tagged Values Ctrl+Shift+6

Tagged Values window - manage extended properties for the current item

Figure 11 – Display tagged values pane

Make sure that you see duplicate tagged values: use the 'tagged value options' button at the top of the tagged value pane, or from the menu select: Tools->Options->Objects:



Objects

Version: 1.0 Phase: 1.0

Feature reference indicator
[✓] Highlight References Reference Char(s) * ( ) Prefix [✓ Suffix]

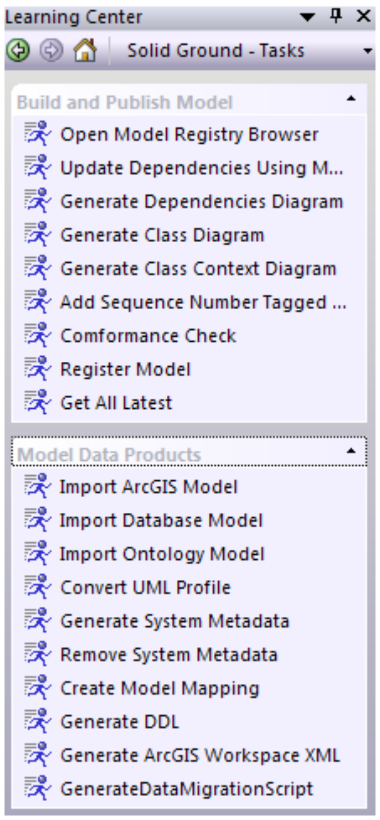
[ ] No warning for spaces in class names
[✓] Classes honor analysis stereotypes
[ ] Show stereotype icon for requirements
[✓] Support for Composite Objects
[ ] Auto-resize marks diagram 'dirty'
[ ] Highlight {abstract} elements
[ ] Allow elongated Use Cases
[ ] Show status colors on diagrams
[✓] Copy inheritance links on duplicate
[ ] Port and Part type visible by default
[✓] Show buttons for selected Object on diagram
[ ] Edit Object on New
[✓] Show <<column>> stereotype
[ ] Extend Complexity
[ ] UML 1.5 Components
[ ] Show State Compartment
[✓] Show Duplicate Tags
[✓] Group Operations by Stereotype
[✓] Group Attributes by Stereotype
[ ] Sort Features Alphabetically

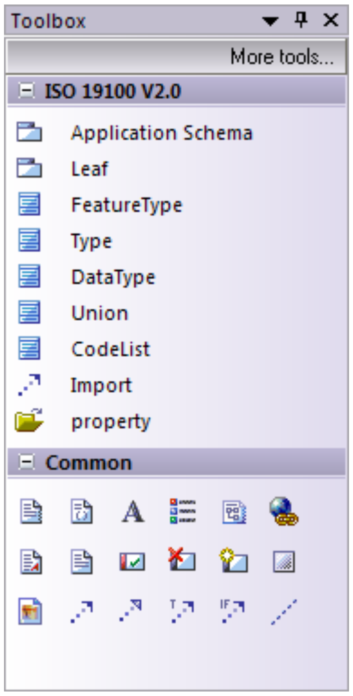
Figure 12 – Ensure duplicate tag values are shown

9.1.7 View the Learning Centre

Solid Ground adds tasks to the learning centre in Enterprise Architect that can be more convenient to use over the right-click add-ins menu. To display these tasks in Enterprise Architect do the following:

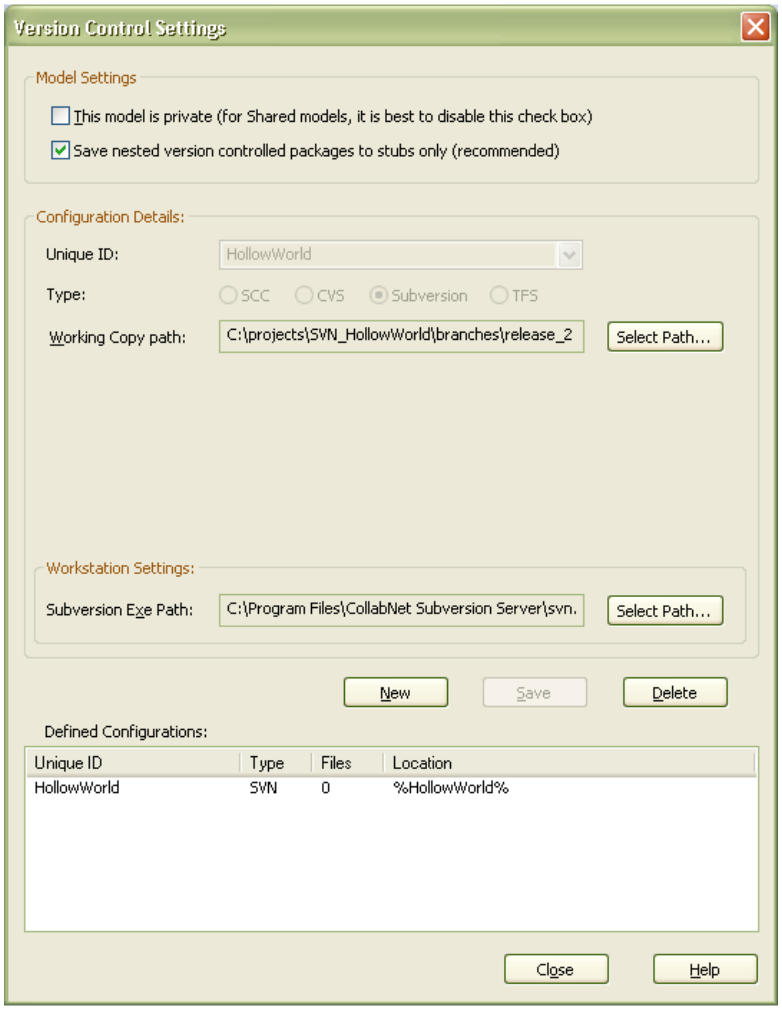
- Ensure “Learning Centre” is visible. If it is not, use the View menu and click on Learning Centre, or use the shortcut ALT+F1.
- At the top of the learning centre dialog is a dropdown of the task groups that are available. Select Solid Ground - Tasks from the list.
- Two tasks groups should appear that you can expand. One called “Build and Publish Model”, and another called “Model Data Products.”





9.1.8 View Solid Ground Toolbox

Solid Ground also provides toolbox that provide diagram drawing tools to support model development. To display the diagram toolbox, use the view menu or press ALT+5. Once the toolbox is visible you can show one of the Solid Ground toolboxes with it, for instance selecting the ISO 19100 tools.



[ ] This model is private (for Shared models, it is best to disable this check box)

[✓] Save nested version controlled packages to stubs only (recommended)

Unique ID: HollowWorld

Type: SCC CVS [✓ Subversion] TFS

Working Copy path: C:\projects\SVN_HollowWorld\branches\release_2

Subversion Exe Path: C:\Program Files\CollabNet Subversion Server\svn.

Defined Configurations:

Unique ID	Type	Files	Location
HollowWorld	SVN	0	%HollowWorld%

Figure 13 – Version control settings

9.1.9 Integrating Enterprise Architect with subversion

Next we will configure the Version Control Settings in the Project Browser, right-click on *Model*->Package Control->Version Control Settings

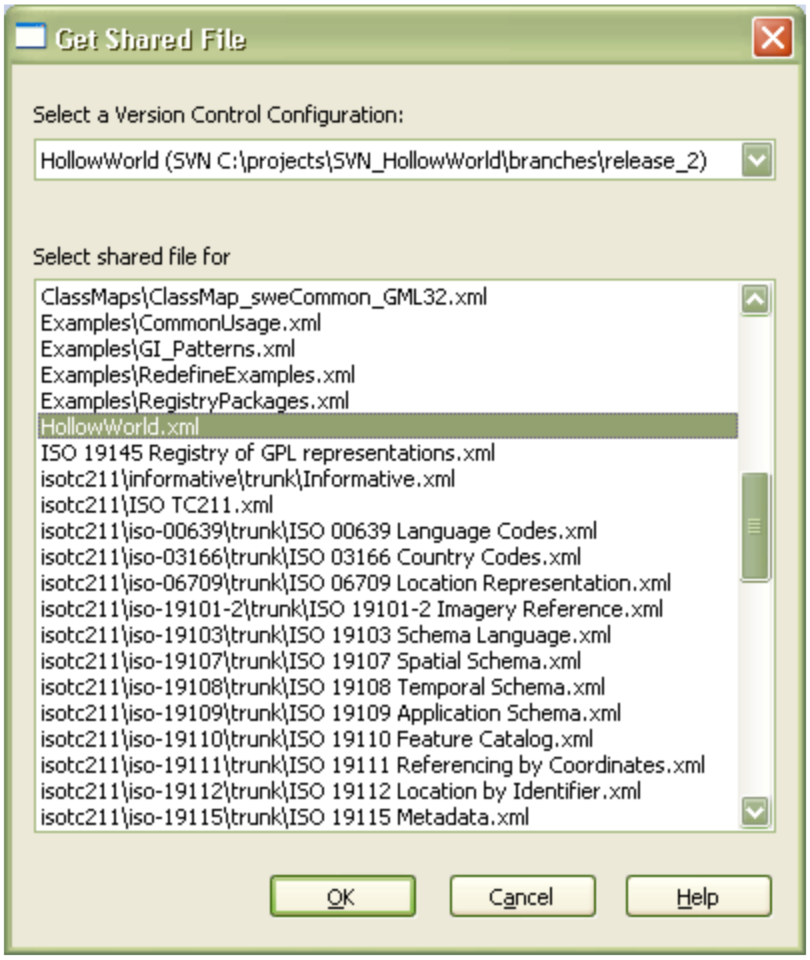
We will configure subversion for the HollowWorld and ISO Harmonized models separately, as they link to separate subversion projects:

1. HollowWorld model
    - Set “Save nested version controlled packages to stubs only” to true
    - Unique ID: HollowWorld **it is important that this is set to exactly this (case-sensitive) value**
    - Type: Subversion
    - Working Copy Path: $HollowWorldLocal
    - Subversion Exe Path: EA may find this automatically, but check it
    - Save
    - Close
2. ISO Harmonized model
    - Save nested version controlled packages to stubs only - true
    - Unique ID: isotc211 **it is important that this is set to exactly this (case-sensitive) value**
    - Type: Subversion
    - Working Copy Path: $ISOLocal
    - Subversion Exe Path: *set path*
    - Save
    - Close

9.1.10 Load HollowWorld

In this step we will load the HollowWorld in the Project Browser. To do this:

- right-click on *Model* and select Package Control->Get Package
- Select the "HollowWorld" Version Control Configuration
- Select the shared file "HollowWorld.xml" (you may have to scroll down the list a way)



Select a Version Control Configuration:
HollowWorld (SVN C:\projects\SVN_HollowWorld\branches\release_2)

Select shared file for

```
ClassMaps\ClassMap_sweCommon_GML32.xml
Examples\CommonUsage.xml
Examples\GI_Patterns.xml
Examples\RedefineExamples.xml
Examples\RegistryPackages.xml
HollowWorld.xml
ISO 19145 Registry of GPL representations.xml
isotc211\informative\trunk\Informative.xml
isotc211\ISO TC211.xml
isotc211\iso-00639\trunk\ISO 00639 Language Codes.xml
isotc211\iso-03166\trunk\ISO 03166 Country Codes.xml
isotc211\iso-06709\trunk\ISO 06709 Location Representation.xml
isotc211\iso-19101-2\trunk\ISO 19101-2 Imagery Reference.xml
isotc211\iso-19103\trunk\ISO 19103 Schema Language.xml
isotc211\iso-19107\trunk\ISO 19107 Spatial Schema.xml
isotc211\iso-19108\trunk\ISO 19108 Temporal Schema.xml
isotc211\iso-19109\trunk\ISO 19109 Application Schema.xml
isotc211\iso-19110\trunk\ISO 19110 Feature Catalog.xml
isotc211\iso-19111\trunk\ISO 19111 Referencing by Coordinates.xml
isotc211\iso-19112\trunk\ISO 19112 Location by Identifier.xml
isotc211\iso-19115\trunk\ISO 19115 Metadata.xml
```

Figure 14 – Loading HollowWorld

Make sure we have the latest copy of the files from the SubVersion repository by:

- In the Project Browser, right-click on *Model* and select Package Control->Get All Latest. If "Get All Latest" is greyed out, you will need to check the "This model is private..." tick box under Version Control Settings. It should then become active.
- Select "Import Changed Files Only" and then click "OK".

At this point we are now ready to do some modelling.

10 Appendix B - Starting From Scratch

TBD - This section will follow the development of FarmML which doesn’t have any existing modelling work done yet.

TBD - Use Case approach, look at existing datasets.

11 Appendix C - Starting From a Physical Model

It is often the case that a physical model exists for a conceptual model you are working on. This is true in the case of Digital NOTAM Events - actually there is a conceptual model as well but we will start from the physical model here.

Both EA and SolidGround provide tools for starting from a physical model and ending up with a conceptual model that is compliant to the ISO TC211 harmonized model. This section will step through the process of doing so.

11.1 Physical Model vs. Conceptual Model

While the conceptual model represents concepts and the relationships between them, a physical model is an application of those concepts in some tangible form; for example a database structure or an XML Schema.

Many systems often start with a physical model and the conceptual model only appears on paper or even in the system designer’s head. This is especially true when a single practical goal is being met by the physical model - there may be little justification for the added design overhead of generating accurate and complete conceptual models.

There are many reasons already discussed why at some point it is appropriate to do the work of modelling the domain concepts themselves, and having one or more physical models to start from provides direct feedback on the practical application of the concepts to be modelled.

Physical models will often have plenty of implementation baggage not relevant to the underlying concepts. While important to the physical model for the system to function, they should never appear in a conceptual model.

Conceptual models should never include implementation details.

Examples of implementation details include; primary and foreign keys, encoding information, platform or language specific details such as data types or structures.

It is common for systems to work with two or more conceptual domains. These cross-domain features in physical models can lead to fuzzy boundaries between conceptual models if not managed appropriately.

11.2 Supporting Tools

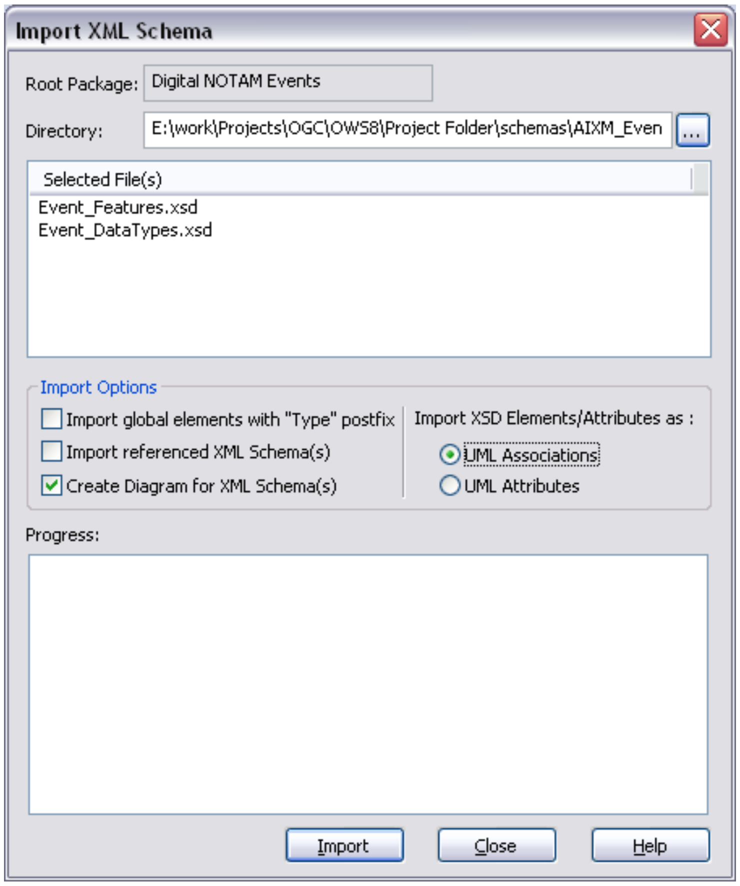
Enterprise Architect includes a Code Engineering feature in non-desktop versions that support import from as well as export to a variety of sources; ODBC databases, mainly object oriented programming languages, XML schema and web services (WSDL).

Once imported the physical model will be represented in UML in Enterprise Architect. To get this into the conceptual model needed there will be a few steps involved. Some of these will need to be manually performed but SolidGround provides a variety of automated tools to improve this process.

11.3 Example: Digital NOTAM Events XSD

Next, start the Digital NOTAM refactor work by using the XSD files as a starting point to generate the conceptual model.

1. In our Digital NOTAM Events project, create a new package called Event with a stereotype of Application Schema.
2. Right click on the *«Application Schema» Event* package and select *Code Engineering->Import XML Schema...*
3. Browse to your AIXM Event schema files and select both Event_Features.xsd and Event_DataTypes.xsd.
4. We do not want to import referenced XML Schemas or add a Type postfix to global elements, but we do want to create the diagram.
5. Import XSD Elements/Attributes as UML Associations, not UML attributes.
6. Check your settings against the image below, then press Import. It will take a while.


Import XML Schema

Root Package: Digital NOTAM Events

Directory: E:\work\Projects\OGC\OWS8\Project Folder\schemas\AIXM_Even

Selected File(s)
Event_Features.xsd
Event_DataTypes.xsd

Import Options
- [ ] Import global elements with "Type" postfix
- [ ] Import referenced XML Schema(s)
- [✓] Create Diagram for XML Schema(s)

Import XSD Elements/Attributes as : [✓ UML Associations] UML Attributes

Progress:

Figure 15 – Import XML Schema

Once complete, take a look at the created diagrams.

Before we get into the manual work of removing implementation specifics we will use the SolidGround tools to do some more of the work for us.

11.4 Conceptual Classes based on Physical Classes

The first thing we want to do is create a new package which will be the application schema for our conceptual model. We will keep the imported physical model isolated and in its original form as a reference.

Copy a class from the physical model - EventPropertyGroup to a new diagram called "Event". The same class now appears in two diagrams, change one and the other will update. We want to change the new class. Right click on the copied class and select "Advanced->Convert linked copy to local copy" - this makes this copy a separate class to the original. Now rename the class to Event and see that the original doesn't change. This is how a conceptual model based on physical model is created while keeping the original physical model intact.

You can also simply drag classes into your diagram from the project browser, which gives a variety of options on how to treat the inclusion of the class in the diagram. The point being made here is that the diagram is NOT the model, only a view of the model - the same class can appear in many diagrams, but we actually want a new class based on an existing one.

We repeat this process for select classes of interest to get the structure of our conceptual model. For each copied model we will do some cleaning up:

- Rename the class - from a name that is an implementation detail of XSD to the appropriate concept.
- Remove the XSD based stereotype
- Recreate links between classes. Especially in an XSD physical model the existing links are riddled with implementation detail. Both the classes and linkages will be much simpler. You'll need to have a good comprehension of the different kinds of UML relationships there are and their application. Make sure you set the multiplicity of the relationship based on the physical model you imported.
- Remove implementation detail from the model. This is in the form of tagged values that are not part of the ISO approach.
- Ensure all model details are properly documented.

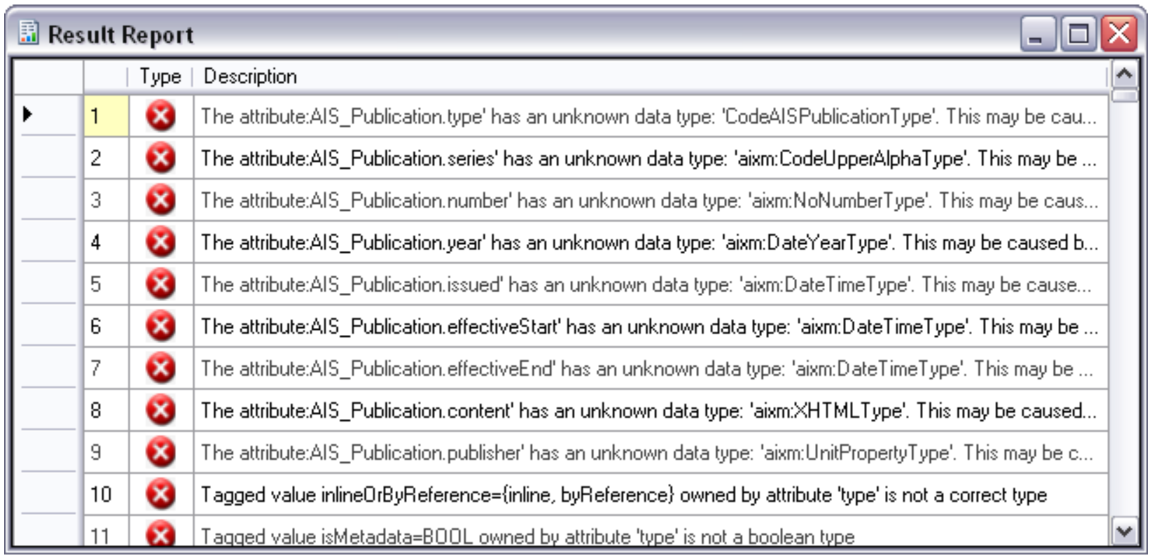
We can use the Solid Ground tool to help with many of these tasks in a couple of ways:

11.4.1 Model Mapping

Solid Ground supports a feature that will convert model details based on an XML configuration file. Files exist that can work with a model that has been imported from

Oracle and convert details of this model over to the ISO Harmonized model and the formal model definition needs. Examples include detecting basic or more complicated types and converting them over to ISO types or removing superfluous tagged values or implementation specific attributes.

11.4.2 Conformance Checking



Result Report

	Type	Description
1	✖	The attribute:AIS_Publication.type' has an unknown data type: 'CodeAISPublicationType'. This may be cau...
2	✖	The attribute:AIS_Publication.series' has an unknown data type: 'aixm:CodeUpperAlphaType'. This may be ...
3	✖	The attribute:AIS_Publication.number' has an unknown data type: 'aixm:NoNumberType'. This may be caus...
4	✖	The attribute:AIS_Publication.year' has an unknown data type: 'aixm:DateYearType'. This may be caused b...
5	✖	The attribute:AIS_Publication.issued' has an unknown data type: 'aixm:DateTimeType'. This may be cause...
6	✖	The attribute:AIS_Publication.effectiveStart' has an unknown data type: 'aixm:DateTimeType'. This may be ...
7	✖	The attribute:AIS_Publication.effectiveEnd' has an unknown data type: 'aixm:DateTimeType'. This may be ...
8	✖	The attribute:AIS_Publication.content' has an unknown data type: 'aixm:XHTMLType'. This may be caused...
9	✖	The attribute:AIS_Publication.publisher' has an unknown data type: 'aixm:UnitPropertyType'. This may be c...
10	✖	Tagged value inlineOrByReference={inline, byReference} owned by attribute 'type' is not a correct type
11	✖	Tagged value isMetadata=BOOL owned by attribute 'type' is not a boolean type

The conformance checker in Solid Ground will report any details of a model that are not consistent with the formal modelling approach. To generate the report, select "Conformance Check" from the learning centre, or right click on the application schema to check and use the add ins->Solid Ground context menu.

Figure 16 – Conformance checking

The technical report for Solid Ground has extensive information on what conformance checking Solid Ground does as well as how to interpret the report entries. This document is available at:

https://wiki.csiro.au/confluence/display/solidground/Documents

11.5 Documentation

It is best practice to ensure that every model detail includes notes that document the model. Solid Ground has a tool that allows export / import of model details into an Excel spreadsheet to allow for easy documentation of model elements. This spreadsheet view of the model is useful to check documentation coverage as well.

11.6 Dealing with Sequence

Attribute order is not important to UML but is very important to GML. Controlling order in UML is achieved by applying a sequenceNumber tagged value to each attribute and association connection role.

Solid Ground will automatically apply this tag for your application schema using the “Add Sequence Number Tagged Value” task. This ensures generated GML encodings are consistent.

11.7 Dealing with Code Lists

Code lists often have a need to be consumed or extended by others. A good practice is to contain code lists into their own package within your application schema to allow for easier extension and version management.

11.8 Package Dependency Diagram

Solid Ground can automatically generate a package diagram that documents the dependencies between the packages in your model. This information is very useful for governance and model sharing purposes.

When making use of an existing model, the level of granularity for dependency purposes is the application schema, which is modelled as a package. Within an application schema you have the classes which can internally have mutual dependencies (e.g. a leaf package can have mutual dependencies) but you can’t have mutual dependencies across application schemas.

The application schema is the point of governance whereas the class is the point of reference - what you will make use of. When you reference a class though you import it’s containing application schema. This is synonymous with XSD in that you can import from another namespace, or include from within a single namespace.

Generating the package diagram will report on irresolvable referenced entities in your model, though It doesn’t remove dependencies that are no longer in use.

11.8.1 Get Your Stereotypes Right!

It is important that when you select stereotypes you are correctly referencing the appropriate UML profile. Don’t just type in the stereotype name without ensuring you connect that name to the correct element or it will create a duplicate and confuse EA. Make sure you push the ellipses button next to the stereotype field and select the stereotype from the correct package.

You can also use the Solid Ground Toolbox to create a new class with the correct stereotype (and tagged values), but if you didn’t do that and instead manually created a new class you can right click on the relevant toolboox entry (say Application Schema) and select “Synchronise Stereotype” - which will affect all entities for the current model.