

Open Geospatial Consortium, Inc.

Date: 2011-12-19

Reference number of this document: OGC 11-107

Version: 1.0

Category: Public Engineering Report

Editors: James Groffen, Cameron Shorter, Rob Atkinson

OGC® OWS-8 Domain Modelling Cookbook

This document is licensed under a Creative Commons Attribution 3.0 Unported License,
<http://creativecommons.org/licenses/by/3.0/>

Warning

This document is not an OGC Standard. This document presents a discussion of technology issues considered in an initiative of the OGC Interoperability Program. This document does not represent an official position of the OGC. It is subject to change without notice and may not be referred to as an OGC Standard. However, the discussions in this document could very well lead to the definition of an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Preface

Use Case modelling has been undertaken to establish the scope and business requirements for system and domain information modelling. 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 modelling undertaken according to *ISO 19103 Conceptual Schema Language* (ISO 1999) and *ISO 19109 Rules for Application Schema* (ISO 2000).

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

| | |
|--------------------|-------------------------------------|
| Document type: | OGC [®] Engineering Report |
| Document subtype: | NA |
| Document stage: | Not approved for public release |
| Document language: | English |

Contents

Page

| | | |
|---------|---|----|
| 1 | Introduction..... | 6 |
| 1.1 | Scope | 6 |
| 1.2 | Document contributor contact points | 7 |
| 1.3 | License..... | 7 |
| 1.4 | Revision history..... | 7 |
| 1.5 | Future work | 8 |
| 1.6 | Forward | 8 |
| 2 | Normative References..... | 9 |
| 3 | Terms and definitions | 10 |
| 4 | Conventions | 12 |
| 4.1 | Abbreviated terms | 12 |
| 4.2 | UML notation | 12 |
| 5 | A Case For Domain Modelling..... | 13 |
| 5.1 | What is a Domain Model?..... | 13 |
| 5.2 | Why Domain Modelling? | 13 |
| 6 | Domain Modelling Principles | 14 |
| 6.1 | Re-use of common concepts..... | 14 |
| 6.2 | Modular design of Application Schemas | 14 |
| 6.3 | Partitioning around Governance Boundaries | 15 |
| 6.4 | Understanding use cases..... | 15 |
| 6.5 | Level of Abstraction..... | 16 |
| 6.6 | Dependency Analysis | 18 |
| 6.7 | Dependency and Version Management..... | 20 |
| 6.8 | Model Registry | 20 |
| 6.9 | Registry Browser..... | 20 |
| 6.10 | Model Hygiene | 22 |
| 6.11 | Conformance checking..... | 22 |
| 6.12 | Attribute Types..... | 23 |
| 6.13 | Definitions | 23 |
| 6.14 | Patterns | 23 |
| 6.14.1 | “Root Class for the Domain” pattern | 23 |
| 6.14.2 | “Implementation Oriented Metadata in a Class” pattern | 25 |
| 6.15 | Sustainable Model Management | 26 |
| 7 | Governance | 27 |
| 8 | Appendix A - Tool Setup..... | 28 |
| 8.1 | An Environment for Modelling..... | 28 |
| 8.1.1 | An Example of Modelling Practices | 29 |
| 8.1.1.1 | Digital NOTAM Event Specification - DNES | 29 |
| 8.2 | Enterprise Architect..... | 30 |

| | | |
|--------|---|----|
| 8.3 | SolidGround | 30 |
| 8.4 | Subversion | 31 |
| 8.5 | HollowWorld..... | 31 |
| 8.6 | EA and subversion Authentication..... | 32 |
| 8.7 | Configuring Enterprise Architect | 33 |
| 8.8 | View the Learning Centre | 36 |
| 8.9 | View Solid Ground Toolbox | 37 |
| 8.10 | Integrating Enterprise Architect with subversion..... | 38 |
| 8.11 | Load HollowWorld..... | 38 |
| 9 | Appendix B - Starting From Scratch | 40 |
| 9.1 | The Agricultural Production Unit (APU) | 40 |
| 9.2 | ProductionUnit Attributes and Inheritance..... | 41 |
| 9.2.1 | Metadata..... | 41 |
| 9.3 | Production Unit Subtypes..... | 43 |
| 9.3.1 | Features | 43 |
| 9.3.2 | Product | 44 |
| 9.3.3 | Activity | 44 |
| 9.4 | Associations between Features..... | 44 |
| 9.4.1 | Association between Production Units | 44 |
| 9.5 | Selecting a level of Abstraction..... | 44 |
| 9.5.1 | Subdomains..... | 44 |
| 9.5.2 | Breakdown Choices | 45 |
| 9.5.3 | Benefits | 46 |
| 9.5.4 | Local Variation and Model Flexibility | 46 |
| 9.6 | Reuse of Existing Conceptual Models | 47 |
| 9.6.1 | Contact Details..... | 47 |
| 9.6.2 | Addresses | 47 |
| 10 | Appendix C - Starting From a Physical Model..... | 48 |
| 10.1 | Physical Model vs. Conceptual Model..... | 48 |
| 10.2 | Supporting Tools | 48 |
| 10.3 | Example: Digital NOTAM Events XSD | 49 |
| 10.4 | Conceptual Classes based on Physical Classes | 50 |
| 10.4.1 | Model Mapping..... | 50 |
| 10.4.2 | Conformance Checking | 51 |
| 10.5 | Documentation | 51 |
| 10.6 | Dealing with Sequence | 51 |
| 10.7 | Dealing with Code Lists | 52 |
| 10.8 | Package Dependency Diagram..... | 52 |
| 10.8.1 | Get Your Stereotypes Right!..... | 52 |

| Figures | Page |
|--|-------------|
| Figure 1 Abstraction | 16 |
| Figure 2 Package dependency diagram – dependencies between Application Schemas..... | 19 |
| Figure 3 Dependencies published in the Registry Browser for the Event Model | 19 |
| Figure 4 Registry Browser with the Event Application Schema selected..... | 21 |
| Figure 5 Error when registering a model - missing dependent package | 21 |
| Figure 6 Example results from a conformance check | 22 |
| Figure 7 Example of defining a root class for the domain | 24 |
| Figure 8 Platform specific details embedded in class..... | 25 |
| Figure 9 Underlying concepts realised by stereotype..... | 26 |
| Figure 10 Solid Ground Tasks in the Learning Centre | 29 |
| Figure 11 Enterprise Architect Model Wizard | 34 |
| Figure 12 Display tagged values pane..... | 35 |
| Figure 13 Ensure duplicate tag values are shown | 36 |
| Figure 14 Solid Ground Learning Center | 36 |
| Figure 16 Version control settings | 37 |
| Figure 15 Solid Ground Toolbox..... | 37 |
| Figure 17 Loading HollowWorld | 39 |
| Figure 18 Import XML Schema..... | 49 |
| Figure 19 Conformance checking | 51 |

OGC[®] OWS-8 Domain Modelling Cookbook

1 Introduction

1.1 Scope

This OGC[™] 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 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.

The recommended approach follows ISO and OGC standards and supports GML application profiles defined using formal UML. The described tools support modelling from conceptual design (UML) to physical implementation (XML Schema).

This document is a deliverable of the OGC Web Services (OWS) Initiative - Phase 8 (OWS-8).

1.2 Document contributor contact points

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

| Name | Organization |
|---|--------------|
| James Groffen (editor) | LISAssoft |
| Rob Atkinson (contributing author) | CSIRO |
| Cameron Shorter (contributing author, reviewer) | LISAssoft |

Australian government organisations contributed to the FarmML domain modelling effort, some of which has been included in this document. Direct contributions from Don Cherry and Susan Robson of DPI Victoria are included.

1.3 License

This Domain Modelling Cookbook, jointly developed by LISAssoft Pty Ltd and the Commonwealth Scientific and Industrial Research Organisation (CSIRO), as a deliverable in the Open Geospatial Consortium OWS-8 activity, is licensed under a Creative Commons Attribution 3.0 Unported License, <http://creativecommons.org/licenses/by/3.0/>



1.4 Revision history

| Date | Release | Editor | Primary clauses modified | Description |
|-------------|---------|--------------|--------------------------|---|
| 21 JUN 2011 | 0.1 | Rob Atkinson | 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 | Jim Groffen | throughout | Added Pre-Deliverable Disclaimer |
| 8 AUG 2011 | 0.4 | Jim Groffen | 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 – 14 Dec 2011 | 0.6- 0.32 | Rob Atkinson, Cameron Shorter | throughout | Provided overview, and simplified language |
| 19 Dec 2011 | 1.0 | Cameron Shorter | throughout | Final review |
| 20 Dec 2011 | 1.0.1 | Jim Groffen | 9 | Added details from the FarmML report. |

1.5 Future work

The first release of this document describes an overview of domain modelling and provides examples for two specific cases of model development. Further model design and implementation patterns may exist and should be incorporated into future versions of this document.

The tools referenced in this Cookbook are under development and tool improvements should be incorporated in future document versions.

1.6 Forward

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.

2 Normative References

Concepts from the following documents are referenced in this document and must be interpreted using this context. 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).
- Atkinson, R., W. Francis, et al. (2010). "Solid Ground Toolset." from <https://wiki.csiro.au/confluence/display/solidground/Solid+Ground+Toolset>.
- Atkinson, R., W. Francis, et al. (2010). Tools and techniques for creation, use and management of information models as metadata for system-of-systems interoperability, CSIRO.
- Cox, S., Atkinson, R. "Configuring your UML tool for HollowWorld." 2010, from <https://www.seegrid.csiro.au/twiki/bin/view/AppSchemas/ConfiguringUMLToolForHollowWorld>.
- INSPIRE, E. (2007). D2.6: Methodology for the development of data specifications, INSPIRE Drafting Team "Data Specifications"
- Proposed methodology for the development of INSPIRE data specifications for the spatial data themes as specified in the Annexes of the INSPIRE Directive
- 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. and T. Duffy (2005). "GeoSciML: Development of a generic GeoScience Markup Language." Comput. Geosci. **31**(9): 1095-1103.

3 Terms and definitions

Abstraction: Deriving a higher concept from more real or concrete concepts. An example would be that a motorway is a special kind of road. The road in this instance is an abstraction of a motorway that has more general application.

Application Schema: Formalizes the information model for an application domain(ISO 2000), usually ISO 19103 Conceptual Schema Language, which is expressed as UML and exchanged in XMI format. A general methodology for the development of an application schema is provided in ISO 19109. A key premise is that communication and primary interoperability concerns centre on a community which shares a model or view of their world, and that the design of an application schema should be scoped to an information community.

Attribute: Synonymous with Property. In UML terms an attribute is a quantifiable aspect of a concept. A road could have an attribute of “maximum speed”. An instance of a road feature could have a maximum speed of 60 km/h.

Conceptual Model: An Application Schema concerned with identification of the concepts used within an application domain. This may be expressed as a set of definitions that are re-used in multiple Implementation Schemas. A conceptual model may be converted to a physical model (e.g. GML or OWL encoding) for exchange of definitions and object identifiers.

Data Product Specification: Detailed description of a dataset or dataset series, together with additional information, that will enable it to be created and distributed to other parties. A data product specification provides a description of the structure, semantics and quality of a data set.

Feature: The starting point for modelling of geographic information. Abstraction of a real world phenomenon. "A digital representation of a real world entity or an abstraction of the real world. It has a spatial domain, a temporal domain, or a spatial/temporal domain as one of its attributes. Examples of features include almost anything that can be placed in time and space, including desks, buildings, cities, trees, forest stands, ecosystems, delivery vehicles, snow removal routes, oil wells, oil pipelines, oil spill, and so on. Features are usually managed in groups as feature collections. The terms feature and object are often used synonymously. The terms feature, feature collection and coverage are defined in line with OpenGIS."

HollowWorld: A suite of configuration tools and resources developed by CSIRO to establish an ISO 19100 and OGC conformant baseline for domain modelling. (Cox)

Implementation Schema: An application schema intended for implementation, for example as a GML application schema or relational database schema. Implementation

Schema are characterized by the addition application specific characteristics as properties of FeatureTypes, and the addition of provenance or lifecycle management metadata.

Normative: prescriptive, (as opposed to informative). Domain models that are used to derive implementation artefacts are normative, since they prescribe the form of these artefacts.

Physical Model: The expression of an application schema in the encoding of a particular implementation technology.

Relationship: In UML terms a relationship is the formal definition of a connection between two concepts. An example could be that a road has a relationship to a council that maintains the road.

SolidGround: A toolkit developed by CSIRO to improve the productivity and quality of modelling using the ISO 19100 framework. (Atkinson, Francis et al. 2010)

Subversion: An open source version control system. Subversion provides a service that can be used for the managed collaboration of files where various versions of the files are tracked.

Enterprise Architect: A proprietary software application with modelling functionality.

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. |
| ISO | International Standards Organisation. |
| GML | Geographic Markup Language – as a modelling language for geographic systems as well as an open interchange format for geographic transactions. |
| OGC | Open Geospatial Consortium – International organization responsible for development of spatial standards. |
| NOTAM | Notice to Airmen – A NOTAM is filed with an aviation authority to alert aircraft pilots of any hazards <i>en route</i> 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 modelling 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 Modelling 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 FullMoon or SolidGround tools.

5 A Case For Domain Modelling

5.1 What is a Domain Model?

Domain modelling extends information modelling by enabling the reuse of concepts, semantics and information organisation (schemas) between related systems.

While information modelling typically refers to modelling just one system, domain modelling involves the practice of creating definitions of concepts which are reused between multiple systems. In the OGC context this is further extended to imply interoperability of models and platform independence.

5.2 Why 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? Why don't you just build an information model for your specific domain?

In practice, integrating data from disparate sources is an ongoing, expensive and time-consuming activity. This is why people develop shared information models, since long experience has shown how expensive it is not to.

The underlying issue with re-use of information models is that similar concepts occur across multiple domains - and if every domain "reinvents the wheel", then competing, overlapping models will arise, making it close to impossible to share information between models.

To achieve interoperability, common concepts need to be jointly modeled and agreed upon. A formal approach should be universally adopted for defining interfaces, data structures, and the content and the semantics of information exchange.

The formal approach that has been established is *ISO 19103 Conceptual Schema Language* and *ISO 19109 Rules for Application Schema* which includes the General Feature Model (GFM).

6 Domain Modelling Principles

In this section, we cover the key principles applicable to successful domain modelling.

6.1 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 to best re-use existing concepts. In the ISO 19100 series of geographic standards, re-use is achieved through a **package import** between **Application Schemas**. Choosing a package to import is becoming more viable now with the publication of model libraries such as the ISO Harmonised Model and OGC Sensor Web Enablement (SWE).

In practice, one can only import modules from a context that is at least as stable, and as recognised as the applicable domain. For example, a national or international context may incorporate elements from the ISO or OGC baselines, but not from an unmaintained academic project, regardless of how excellent the element is.

If a suitable library is unavailable, projects and technology-based models may provide useful clues as to how to structure reusable components. You can usually assume that an emerging candidate concept will become available in the future, so you should partition these into separate modules.

6.2 Modular design of Application Schemas

To achieve reuse, domain models need to be designed in a modular fashion. An Application Schema which describes a domain should be based upon other, more specific Application Schemas which describe each module or base concept. As Application Schemas should be developed for reuse, we need to design them carefully.

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

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

Application Schemas should contain all 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).

A FeatureType describes an information concept that may be realized as an identifiable instance – such as a Building, a Catchment, a Cadastral Parcel, a Unit of Measure, an Organisation etc. It is described as a UML class, with a FeatureType stereotype.

Application schemas should ideally contain around 5 to 10 FeatureTypes or just import a set of application schemas.

6.3 Partitioning around Governance Boundaries

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 chemicals, but won't "own" the chemical concepts, therefore they should be partitioned into a separate package.

It is important to establish which stakeholder will maintain each module, and why. Avoid including any definitions in a package that cannot be maintained by the designated package maintainer – these can be separated into interim packages and imports, pending identification of a suitable candidate.

This modular approach to modelling, based upon importing external modules, has previously had a major practical problem which has stopped people from following these principles in practice. Until recently, it has been very hard with "off the shelf" modelling tools to discover and load modules referenced by a domain model, or to keep them up-to-date (i.e. 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.

6.4 Understanding use cases

Another factor in determining the scope of an application domain is understanding what questions an information model supports, and what knowledge the user must have to invoke such a question. For example, if a system is to be queried by a street address, the exact form of that address, as a set of related elements (house number, street, suburb etc) needs to be modeled, as opposed to an ambiguous form such as address:CharacterString. Alternatively, if a system supports a classification scheme, what type of objects are the

classifications, (Soil particle size? Colour?). Invariably the information model should be as explicit as the behavior of the system demands, without getting overly intricate where no practical use will be made of the additional structure.

Relationships between classes should be “traversable” for the required use cases. Thus, there is a strong relationship between the intended deployment architecture and relationships between FeatureTypes within the information model.

It may be necessary to define specialised FeatureTypes that represent denormalised views of concepts exposed by service interfaces. This in turn raises questions about the level of abstraction of an information model.

6.5 Level of Abstraction

“Level of abstraction” refers to the amount of detail captured in a model, and how specific that detail is to a particular implementation. Models may range in abstraction from definitions of the underlying patterns in modeling, to definitions of concepts, through to platform-specific implementation specifications. Choosing the right level of abstraction for a model is a challenging task. A low level of abstraction, capturing lots of detail, will generate an implementation-specific model at the cost of lower general applicability. A high level of abstraction, used for conceptual models, will allow interoperability at the semantic level with other domains, but may be harder to map to specific terms within these domains.

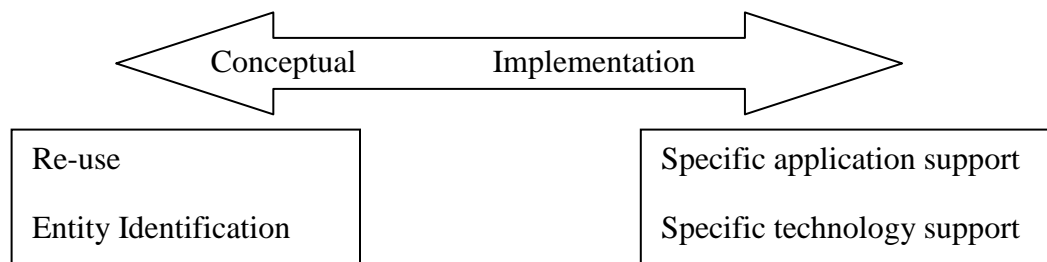


Figure 1 Abstraction

As a rule of thumb, model either specific **data products** (to be persisted or exposed via service interfaces), or **conceptual definitions** to underpin semantic interoperability. Both forms are necessary, but should not be confused with each other.

The difference between Conceptual and Implementation Models is best described by example. Consider a National Roads Authority who is responsible for defining Road Assets. The Authority would be responsible for a **Conceptual Model** which describes the core information about roads. It would include such things as:

- A Road object, which includes:
 - The name of the road;
 - References to the Road Segments which make up the road;

- References to Road Junctions the Road connects to;
- A Road Segment object which includes:
 - An indicative physical location of the Road Segment (the road is a 4-Dimensional object, but its representation as a centerline connecting nodes may be a necessary part of the meta-model.
- A Road Junction object which includes:
 - The type of road junction (round-about, traffic light, etc)
 - Relationships with road junctions
- Etc.

A local council (or possibly another division within the Roads Authority) may be responsible for maintaining assets – which includes maintaining Roads. The council would create an **Implementation Model** for maintenance, which would include such things as:

- An Asset object which includes:
 - An asset type (Road, Park, etc)
 - A reference to the Asset Type (in this case a reference to the Conceptual Road Object)
 - A detailed geometric representation of the road, its border, median barriers etc at a high level of detail.
 - A Maintenance Schedule object, which includes:
 - A Task object which includes:
 - Date
 - Action (eg: Fix pothole)

Similarly, an Implementation Model for Transport and Logistics would reference the Roads Conceptual model, but would be interested in views of the Road Model associated with routing, such as which road connects with which, and the speed limits of each road segment. A real-time traffic model may extend this further.

If the goal is to share information between multiple communities, then a Conceptual Model should be used to define the common identifiers needed to reliably reference the real world object. (Any specific representation, i.e. a data product, is likely to be implemented in a form that may change over time, or multiple forms, and is a poor choice for stable identifiers.

The conceptual, reusable elements of a domain's semantics and object identification needs should be modeled to separate data products. Choose a high-level of abstraction, **including only those attributes and relationships needed to assist in distinguishing between different individual entities.**

For convenience, derivatives of a core model can be extended with additional attributes. Conceptual, identity-oriented models will be highly "object-oriented" and may be mapped to a simpler structure, as in elements from multiple tables mapped to a single view, which assists with specific use cases like data transfer.

Implementation models designed for a specific application are generally hard to re-use across domains because they will include:

1. Only the details and metadata needed for the original application.
2. Simplified copies of concepts that other domains will also need.

Data products, such as a published national road network, should be modeled around familiar and convenient implementations, implementation styles and services.

This separation of concerns between transactional reliability and convenient structure is commonly used within database design, with normalised databases accessed via views (or data warehouses accessed via "data marts").

6.6 Dependency Analysis

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

Circular dependencies, (where a model is dependent on another model, which is ultimately dependent on the original model), are not legal under the ISO rules for Application Schemas. Practically, circular dependencies create significant problems when generating implementations – for example GML schema cannot implement circular dependencies due to XML namespace rules.

Discrete packaging of classes that have well defined, hierarchical package dependencies is encouraged as packages can be safely consumed by external parties. Then tools such as Solid Ground's ModelRegistry interface can be used to formally define and publish a UML package diagram and dependency information. With model dependencies in place, a single click can be used to import a model and all the sub-modules needed to completely load the model into a modelling environment.

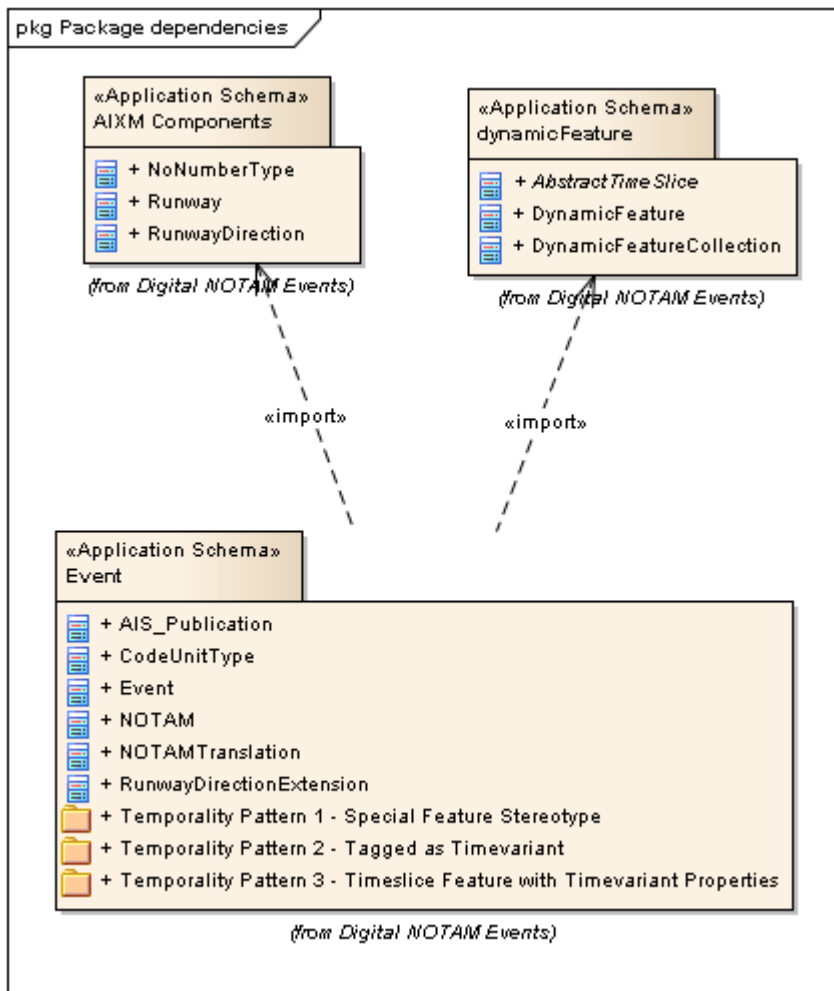


Figure 2 Package dependency diagram – dependencies between Application Schemas

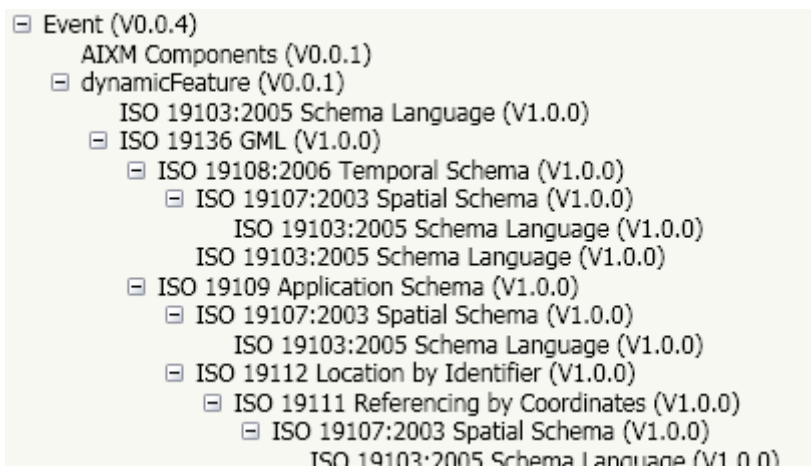


Figure 3 Dependencies published in the Registry Browser for the Event Model

6.7 Dependency and Version Management

Dependencies are a two way street. If someone else is using my model and I want to update it, then these updates may break their use of my model and the user may decide to remain using an older version of my model. They may also ask for changes to be applied to my model to support their use cases.

HollowWorld provides guidance regarding how to configure subversion, however discipline still needs to be applied to recording version numbers in models and derived artefacts.

SolidGround model registration tools provide a helper function for managing version progression which helps enforce best practice for handling change to ensure those dependent on your model can work out what's going on. For more information see (Atkinson, Francis et al. 2010)

6.8 Model Registry

The SolidGround toolset includes a “Model Registry” which is a web service for storing and publishing formal models. The SolidGround Model Registry builds on web registry / catalogue concepts to include the ability to link between modules, including distinguishing between which version of a model should be referenced. While an international standard for the Model Registry interface doesn't exist yet, it is currently being considered for the future.

6.9 Registry Browser

To access the Model Registry, SolidGround provides a “Registry Browser” which interacts directly with the model registry in the following ways:

- Open Model Registry Browser: Connects to a model registry and opens the model registry browser window.
- Update Dependencies Using the Model Registry: Model dependencies are updated from the model registry, picking up any changes that have been submitted to the registry. (Note that a registered version is equivalent to a tagged release in Subversion - not the latest development version.)
- Update a Model in the Model Registry: A model may be added or updated in the Registry Model.

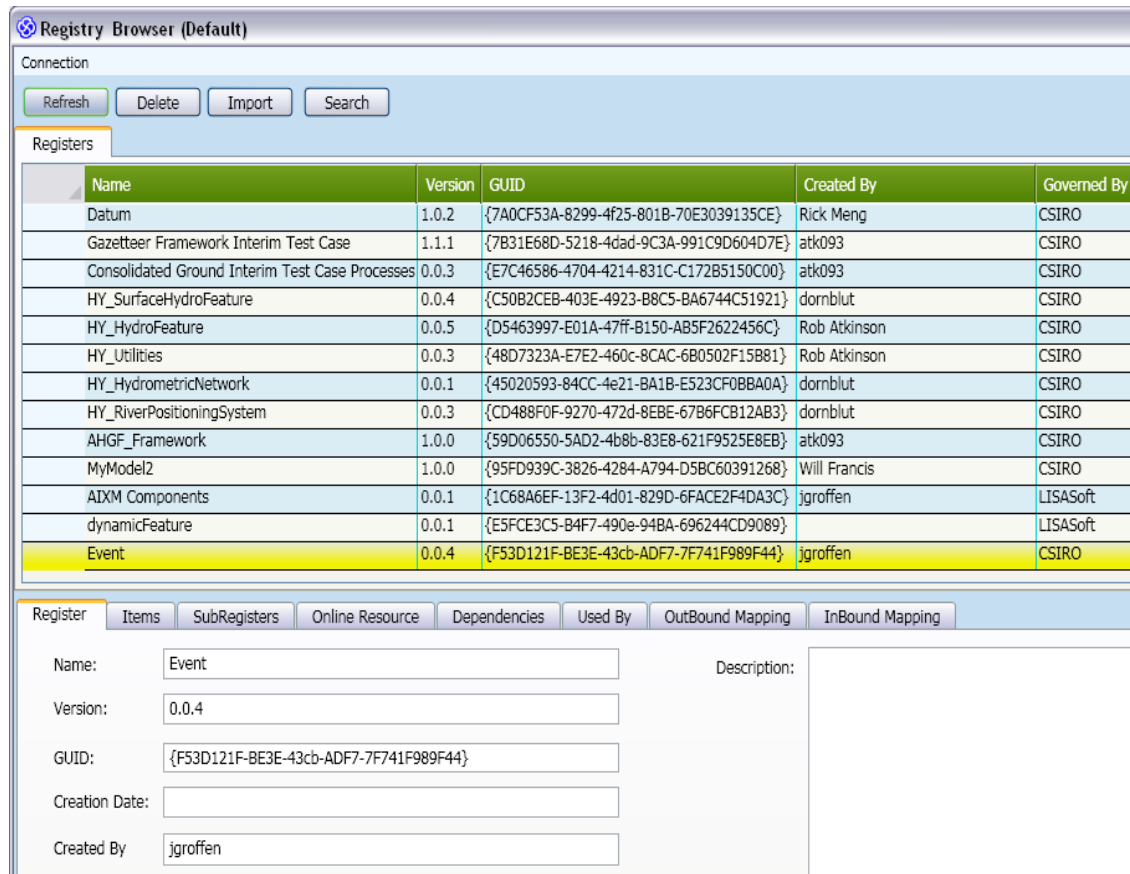


Figure 4 Registry Browser with the Event Application Schema selected

The Registry Browser supports version tracking of models. When a model is registered, the model's Application Schema version number is used to identify the model's version. By convention, model versioning should make use of Major.Minor.Revision version numbers, to indicate to others the nature of the change.

Solid Ground checks dependencies during the model registry process. If there are circular dependencies, or if referenced application schemas are not available, then the registration will fail.

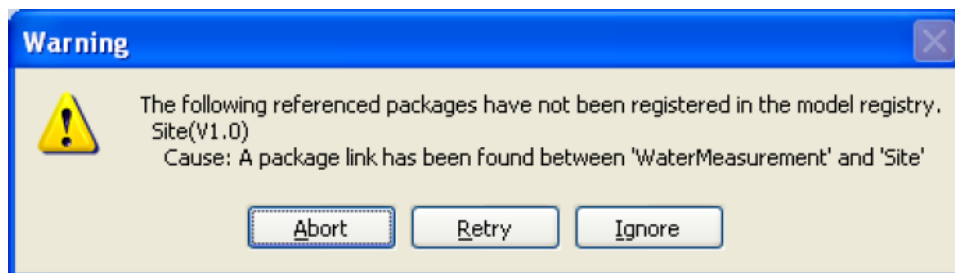


Figure 5 Error when registering a model - missing dependent package

The relationship between version control systems like subversion and the Model Registry is interesting. 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 dependency management and formal release management features.

6.10 Model Hygiene

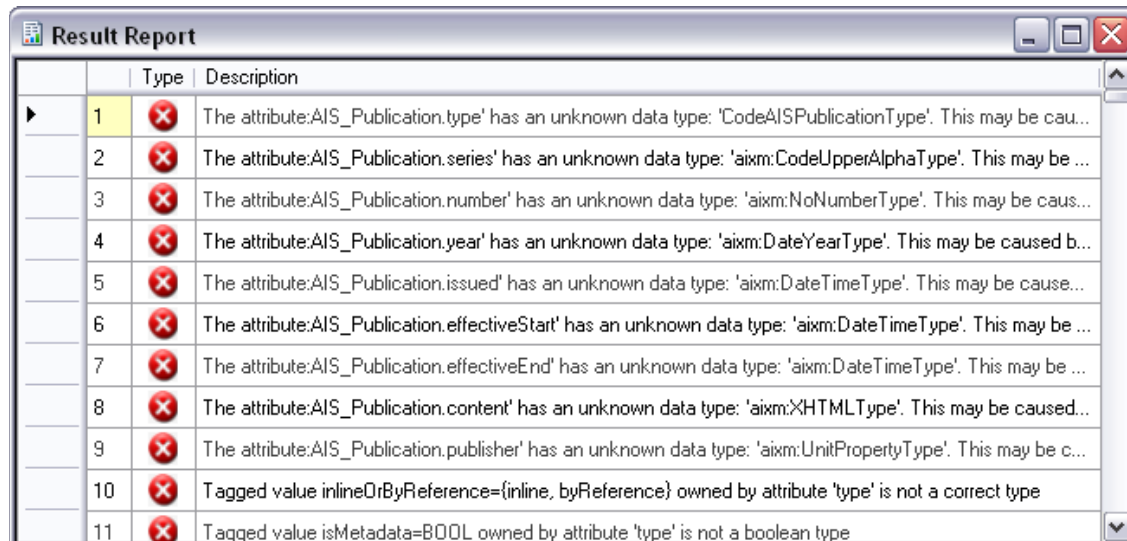
Why does it matter that a model is accurate and only makes use of known terminology? The goal of a model is to document a data exchange and storage agreement, and hence should be designed for processability. They need to be consistently interpretable by tools, and most likely will be transformed into XML or GML schemas. Tools such as SolidGround are emerging for creating ontology viewpoints, database schemas and documentation views based on formal models.

Maintaining the internal integrity of a model is challenging, and tools and the nature of UML make some aspects particularly challenging. UML can provide a class diagram of a model, but the model is **not** the diagram. The diagram is a **view** of the model, but might not show all aspects. Hidden relationships may exist in the model, which change its meaning. For instance, in a UML diagram a relationship between concepts may be included in one diagram but omitted in another. This omission can be interpreted as the relationship isn't there when formally it is.

Some processes for checking and maintaining model integrity can be automated, such as the utilities provided by the SolidGround toolset.

6.11 Conformance checking

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



| | 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:XMLType'. 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 6 Example results from a conformance check

In SolidGround's case, the report can generate errors that indicate an invalid model, or warnings that may indicate breaks from best practice or possible inclusion of values with undefined types.

Entries in the SolidGround conformance checking report provides the context of the element causing the error or warning. Clicking on the row will select the related element in the project browser. This is far more productive than an export->process->review->find->edit cycle with an external tool.

6.12 Attribute Types

In UML modelling tools, one may assign an attribute type a name without linking the name back 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 might not matter for some outputs, such as name matching during GML schema generation, but the model is in fact incorrect. Consequently, automated dependency checking and other ways of processing the model such as creating a Feature Type Catalogue would be compromised.

It is possible to have an attribute type linked to an elements defined in an older version of an imported model– these may be batch updated if required using the SolidGround toolset.

6.13 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 machine readable hyperlinks back to the source element(s). Data dictionaries should be imported if pre-existing, or maintained by an automated export processes from the model as the point of truth. Particular attention should be paid to relationships by putting notes on the target ends of UML associations (the semantics of an association property in the General Feature model).

Note that SolidGround provides tools for import and export of definitions to Excel for convenient editing by domain experts who are not familiar with UML tools.

6.14 Patterns

Two common modelling patterns have profound implications for the potential to re-use domain models reliably. These implementation-oriented patterns are "anti-patterns" for conceptual models.

6.14.1 "Root Class for the Domain" pattern

In the "root class for the domain" pattern all classes in a domain inherit from a single class, which contains common metadata.

An example of this is in the ESRI workspace model, where FeatureClasses are derived from geometry objects (Point).

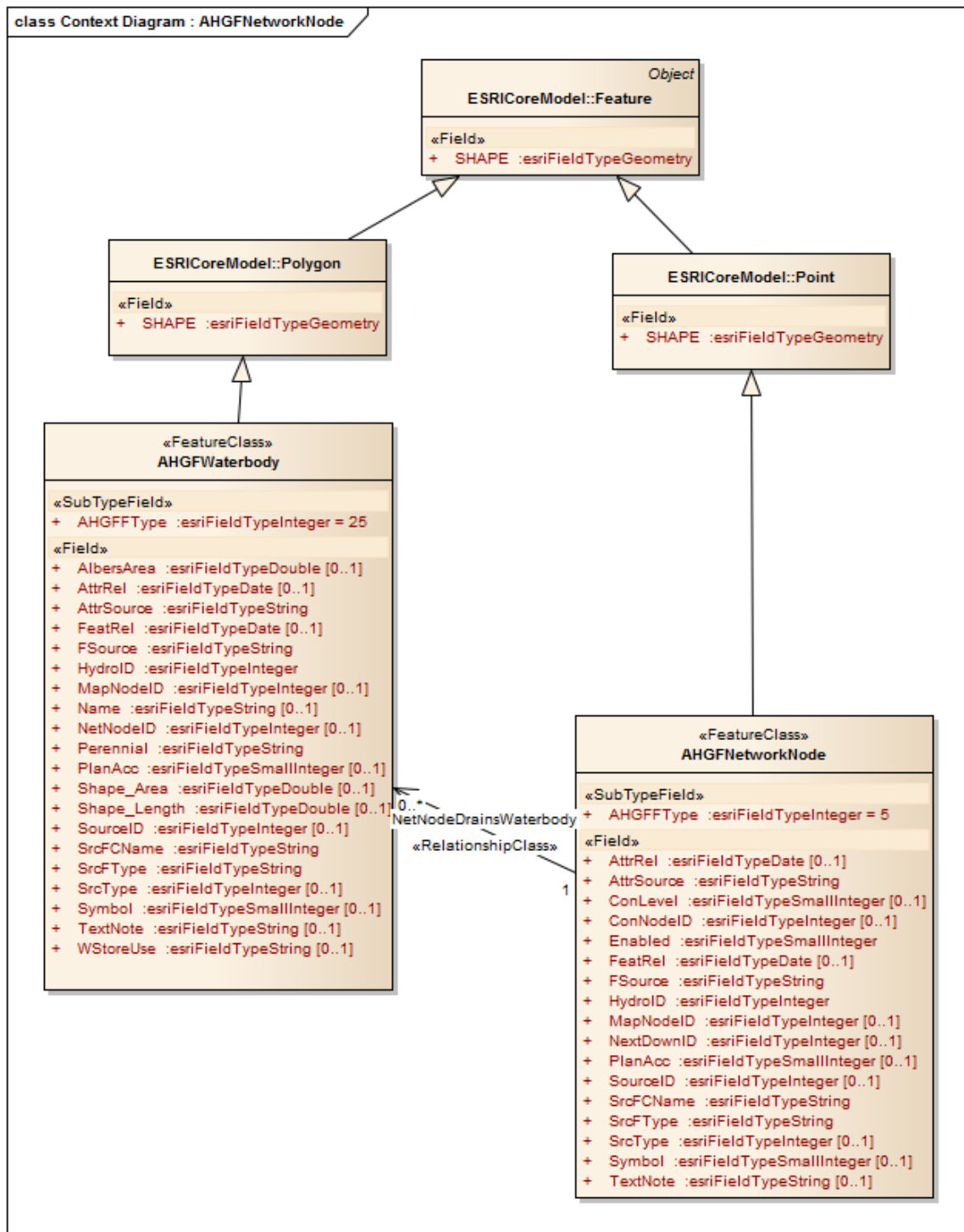


Figure 7 Example of defining a root class for the domain

6.14.2 “Implementation Oriented Metadata in a Class” pattern

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 the base conceptual models, which express the underlying concepts and features in a domain. In a Platform Specific Model, such as an ESRI Geodatabase specific profile, this may be realised by a stereotype rather than direct inheritance, as shown in the following diagram.

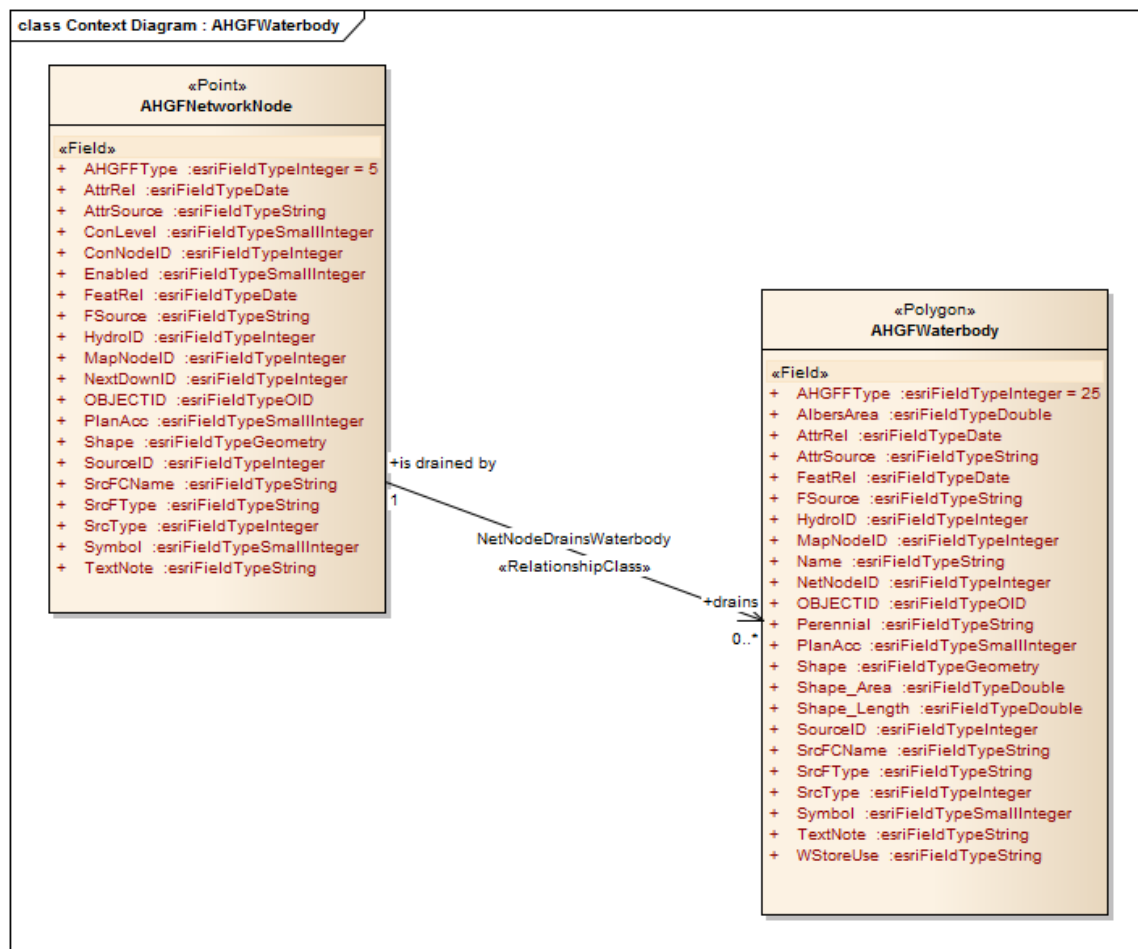


Figure 8 Platform specific details embedded in class

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

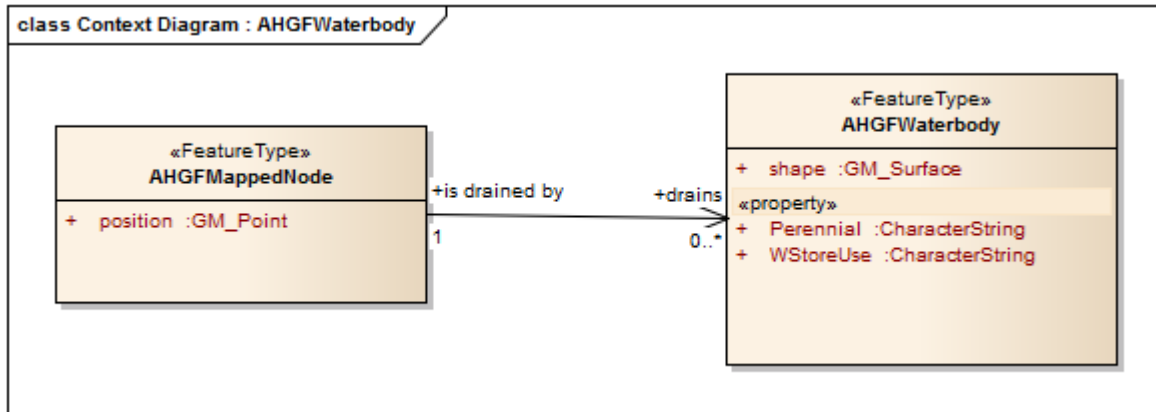


Figure 9 Underlying concepts realised by stereotype

6.15 Sustainable Model Management

Some of the practices presented here seem like a lot of work, and they are, but the trade-off is improved robustness, interoperability, efficiency in creating new systems, and most importantly, the capacity of systems to survive through change.

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 the world is a complex place. However, “complicated” is what we want to reduce. Instead we want to apply a sustainable approach to model management. Here is an outline of the approach:

- Introduce a Model Registry – using 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 model history – using Subversion
 - Any version control system really, but SolidGround has some subversion interaction built-in
 - Version history, concurrent development of models.

The practical steps on how these process are applied are explained in the Appendices.

7 Governance

Governance refers to the allocation of responsibility and decision making processes around the long term maintenance of a Domain Model and the development of an effective governance framework is essential for the long term sustainability of any Domain Model.

Governance is the critical context for any model – since a model represents a semantic agreement, and the governance process and metadata associated with it, defines and documents who the community is, and therefore the scope of the agreement. Without governance, a model has no specific purpose in interoperability, it is merely a perspective on a problem rather than a specification of a solution to a problem.

Documenting Governance best practices is beyond the mandate for this initial release of the Domain Modelling Cookbook. We hope to see this section fleshed out in future releases.

Issues that should be considered when building a governance framework are:

- Who are the stakeholders?
- Which stakeholders should be consulted when making decisions?
- How much influence should each stakeholder have?
- What process should be followed for suggesting, then approving changes?
- Changes in one domain model may impact another domain model, managed by another community. How are these changes managed and coordinated.
- Who will develop and maintain a roadmap?
- How will releases and versions be managed?
- Who is tasked with doing the actual development of the model?
- How are each of the stakeholders and workers funded? How long will this funding last?
- Is the Governance Framework robust enough to continue if funding lapses periodically?

8 Appendix A - Tool Setup

This section describes the suite of tools that have been used and enhanced to create the Domain Modelling processes. While using these tools is not required, using other tools will likely require more manual intervention.

8.1 An Environment for Modelling

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

While domain modelling can be undertaken manually using one of the many UML editing tools, the established modelling processes described in this document is substantially simplified by using the tools specifically developed for Enterprise Architect and Subversion.

Two approaches are suggested. The first is describe in Appendix A - Tool Setup, and is 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” toolset¹.

SolidGround provides a suite of tools for automating various modelling tasks needed for domain modelling, but is essentially a productivity tool that makes a consistent methodology easier to follow.

The tools can be applied independently, though typical modelling methodology patterns and suggest 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.

¹ <https://wiki.csiro.au/confluence/display/solidground/Solid+Ground+Toolset>

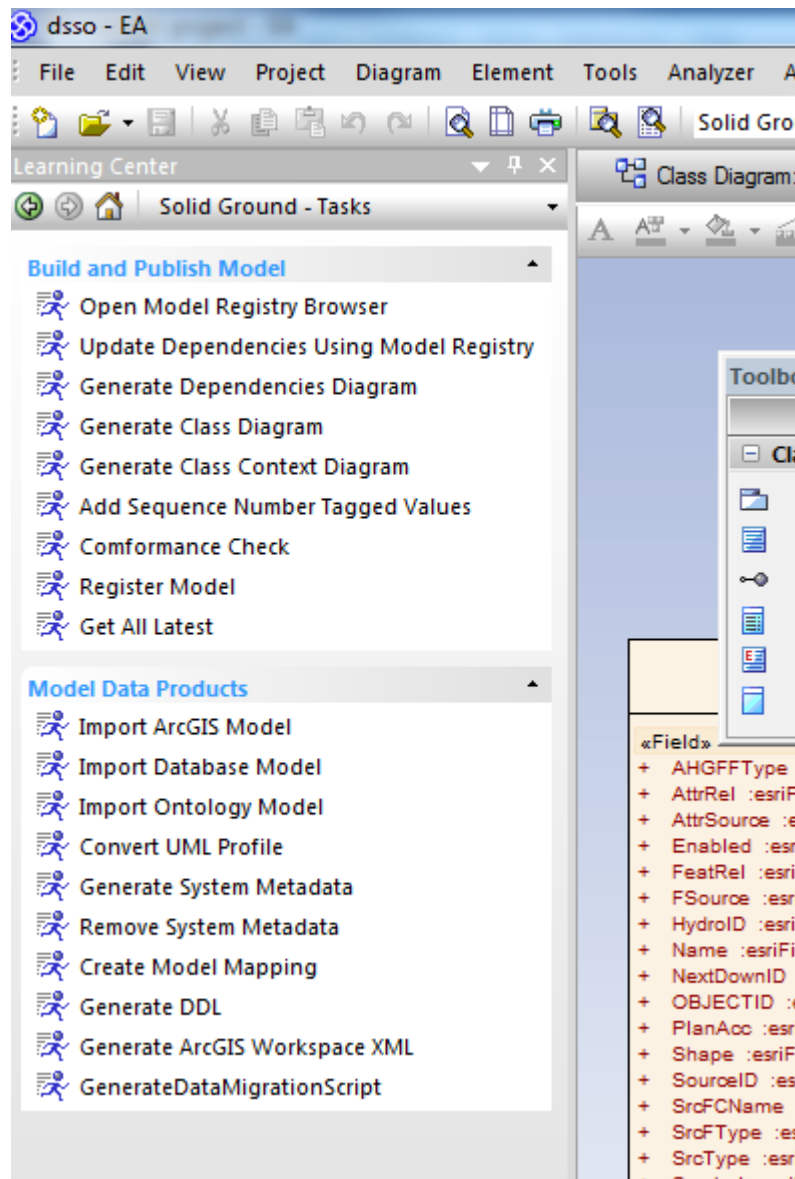


Figure 10 Solid Ground Tasks in the Learning Centre

8.1.1 An Example 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.

8.1.1.1 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."

8.1.1.1.1 Abstraction

Digital NOTAM is unusual in domain modelling in that it 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.

8.1.1.1.2 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.

8.2 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>

8.3 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.

8.4 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.

8.5 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>

8.6 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 Command (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.

8.7 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':

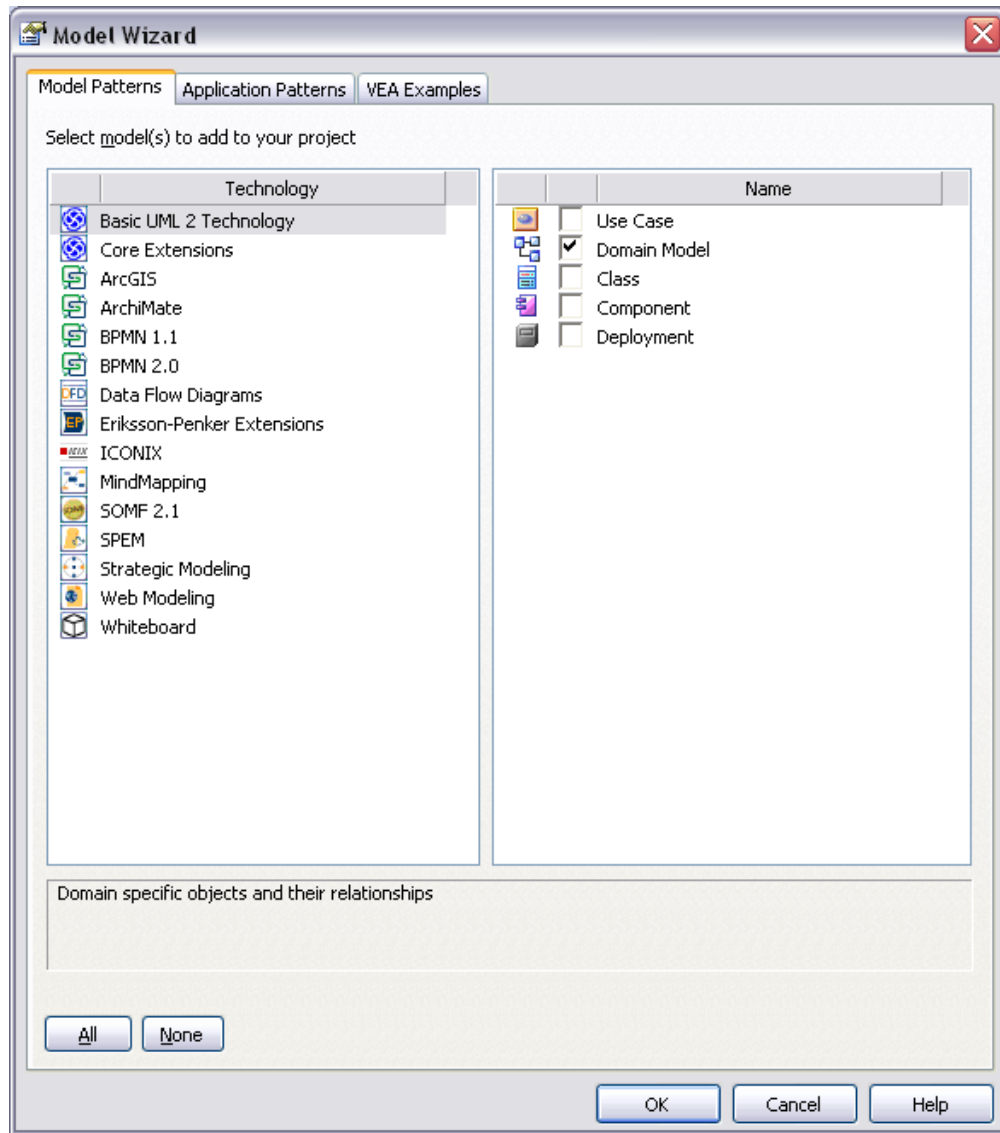


Figure 11 Enterprise Architect Model Wizard

Make sure the Tagged Values pane is open:

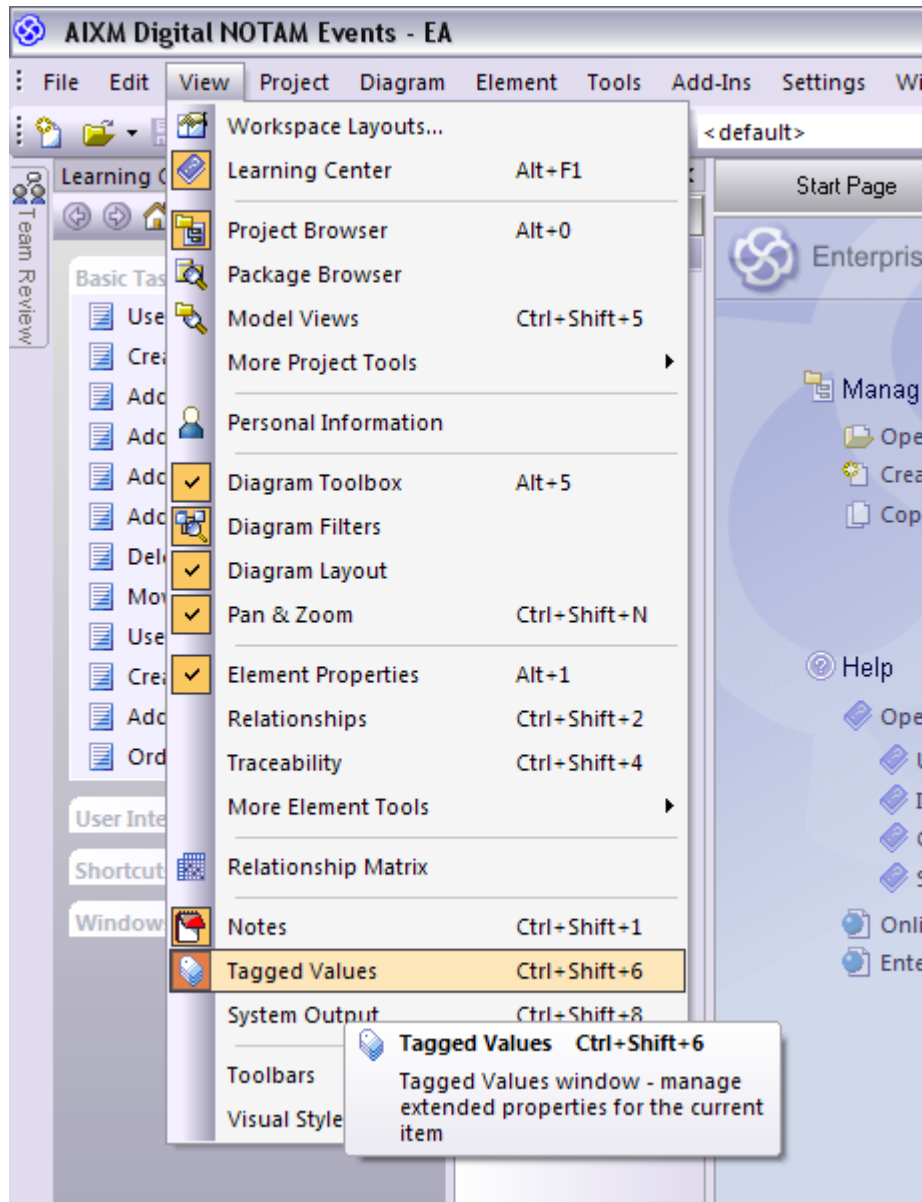


Figure 12 Display tagged values pane

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:

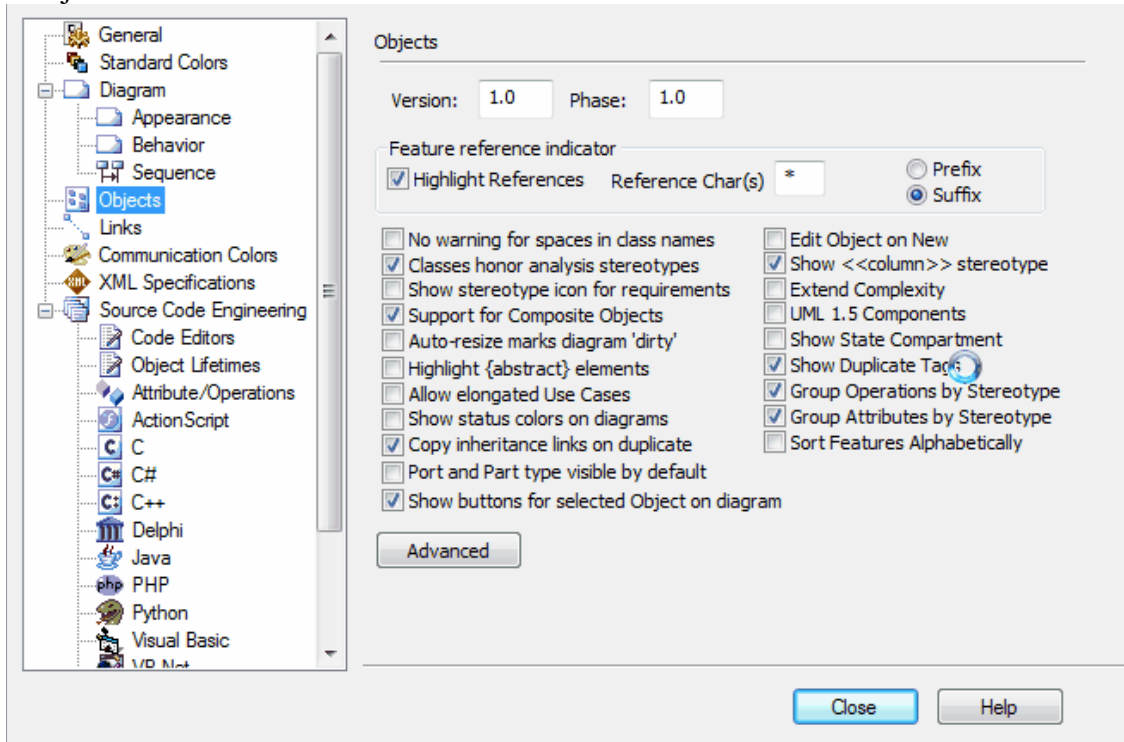


Figure 13 Ensure duplicate tag values are shown

8.8 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.”

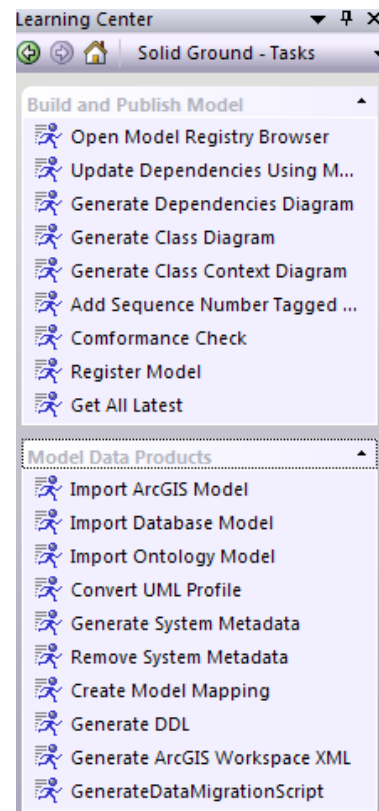


Figure 14 Solid Ground Learning Centre  36

8.9 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.

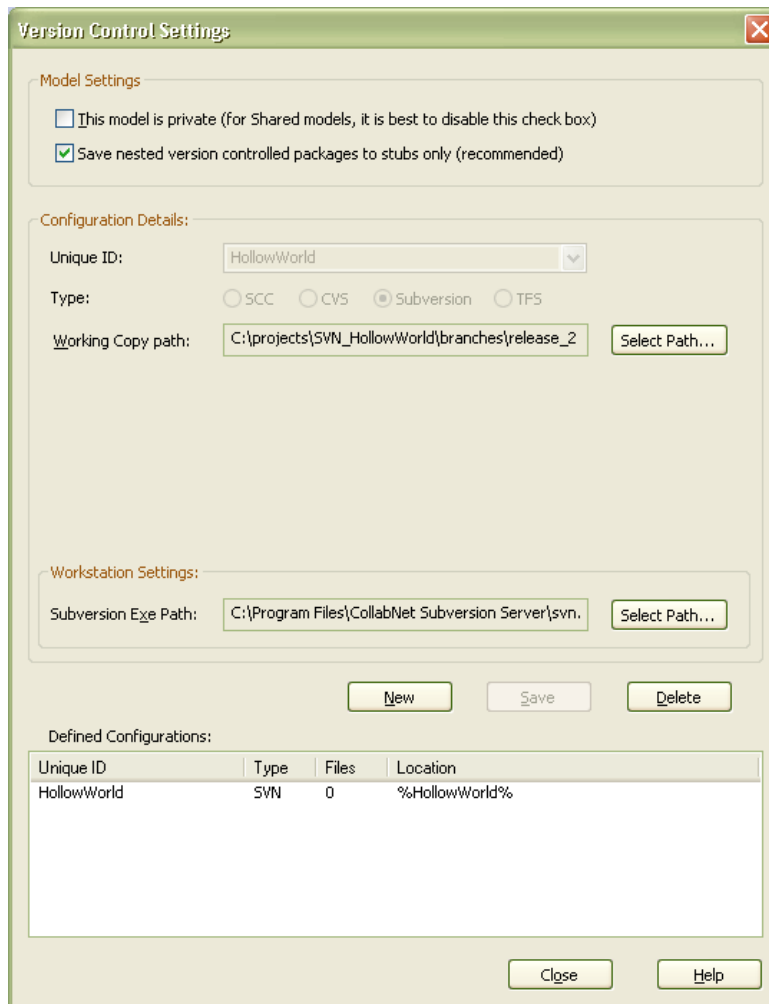


Figure 16 Version control settings

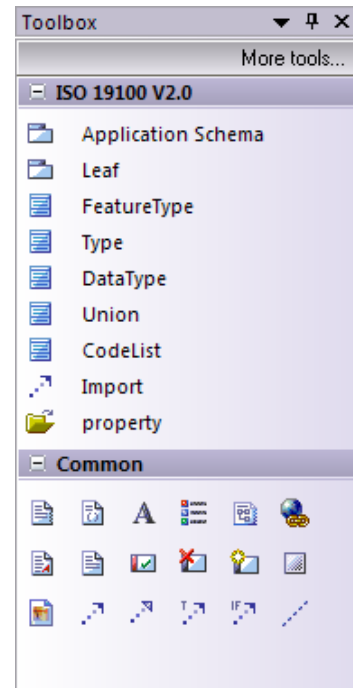


Figure 15 Solid Ground Toolbox

8.10 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

8.11 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)

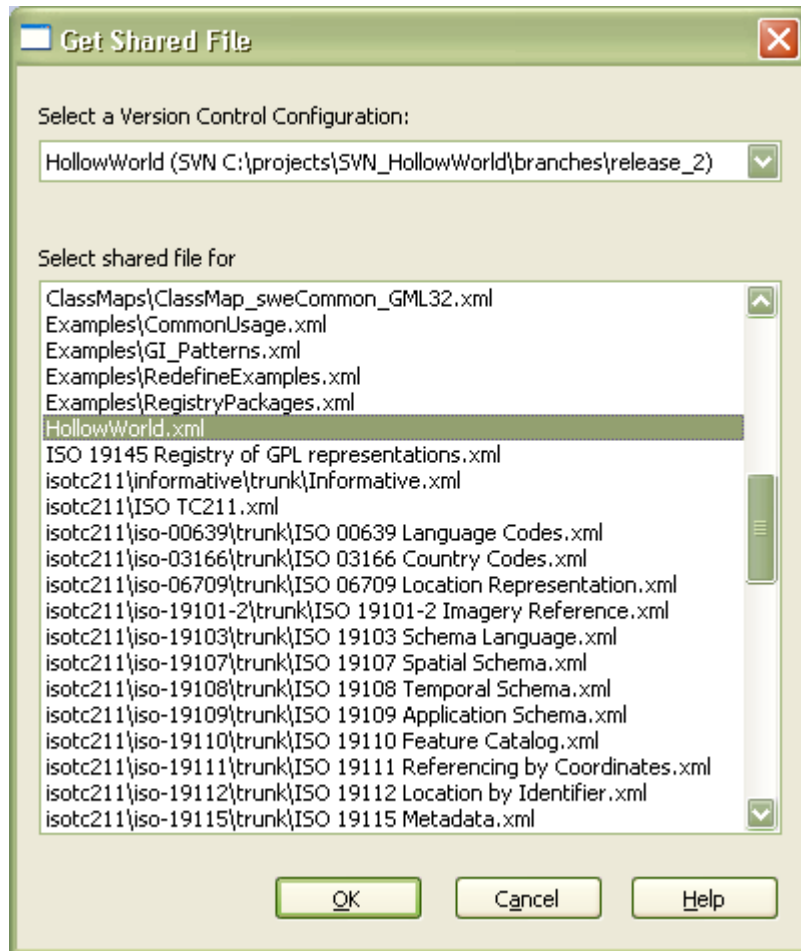


Figure 17 Loading HollowWorld

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.

9 Appendix B - Starting From Scratch

This section follows the development of FarmML which doesn't have any existing modelling work done yet.

The first activity was to describe a set of use cases that are representative of the domain of farming. These are to be used to ensure the appropriateness of the model – to ensure the model is meeting the need it was created for. Once this was done the modelling began in earnest.

It was decided early that the initial modelling activity was to focus on defining the approach taken and the structure of the model. The resulting model for this first phase doesn't describe many concepts in detail, but does organise the concepts and resolves some core concepts for the model, which are discussed next.

9.1 The Agricultural Production Unit (APU)

A simple concept is desired that could apply to many items to describe a farm and the activities performed on a farm. In a similar way to the development of the Observations and Measures standard where an observation and associated results, properties, processes etc can be applied to any feature in a generic fashion; the features that comprise a farm whether they are livestock, crop, tank, shed or paddock all can be described with properties, parameters and processes in a generic way that is applicable to any farming enterprise globally.

The production unit concept in FarmML has similarities to the GeologicUnit in GeoSciML:

Conceptually, may represent a body of material in the Earth whose complete and precise extent is inferred to exist ... or a classifier used to characterize parts of the Earth

Spatial properties are only available through association with a MappedFeature. Includes both formal units (i.e. formally adopted and named in the official lexicon) and informal units (i.e. named but not promoted to the lexicon) and unnamed units (i.e. recognisable and described and delineable in the field but not otherwise formalised).

The concept of a generic unit that applies to a variety of workflows maps very well to the farming domain. Units of production on a farm may be as broad a concept a paddock, or as specific as a single cow. These features are all components involved in the process of farming production. This commonality is conceptually represented in the common base class – ProductionUnit.

While the concept of an Agricultural Production Unit may have been an original concept for the FarmML project, it appears that it is not original in its application. Nash (2009) reports that GLOBALGAP[^] uses the term *Agricultural Production Unit* to describe “a geographic area composed of fields, yards, plots, orchards, greenhouses, livestock building, hatcheries, group of geographic areas of restricted fresh water and/or restricted sea water activities and/or any other area/location/transport used for production of

registered products”. The GLOBALGAP has gone on to continue using the term in it’s Annex of Definitions² but only uses it for defining the geographic area.

The concept as it is applied in FarmML embodies all physical things on a farm that have a role, undergo an activity or produce a product. The farm itself can therefore be assumed to be an aggregated APU which produces something from farm activity. At the simplest level the APU can comprise a single feature (e.g. a cow that can have an individual NLIS* id or milk production recorded against it) or a collection of individual features which make a collective feature such as a herd, shed, orchard, paddock, dam, tank, vineyard or farm as indicated above, which can have a id or geographic location and production recorded against it. Where this differs from the GLOBALGAP concept is the former refers only to the fixed geographic location, while our definition encompasses the fixed areas and those features that may have a temporarily fixed location or be mobile such as a flock of sheep. The concept of an APU also has a time aspect (start, end) along with what ever other parameters describe the feature, then appropriate links to activity and production.

The APU may have an Activity applied to it (e.g. ploughing, shearing, feeding, harvest etc) which has an output or result known as a Product. Each of these concepts can be defined with time, property and other values also. There can also be a feedback loop for such items on a farm which may be products of a farm activity on a particular APU which produces another APU. For example a sheep producing a lamb which forms part of a future flock or wheat harvest of which a portion will go back into the ground to grow the next year’s crop.

When comparing ProductionUnit to GeologicUnit, GeologicUnit documentation currently states that it should eventually be abstract once a complete enough set of specialized subtypes have been defined. In FarmML, ProductionUnit should immediately be abstract as a fixed set of suptypes is known, that are in turn abstract concepts themselves.

9.2 ProductionUnit Attributes and Inheritance

Care must be taken to ensure only attributes appropriate for all FarmML classes considered production units are included at this level. This is a common exercise in inheritance based models. Levels of inheritance in domains such as object oriented programming are cheap to create and maintain but in conceptual modelling they can be expensive and introduce complexity at the physical model level. For this reason levels of inheritance should be kept to the minimum that is logically feasible.

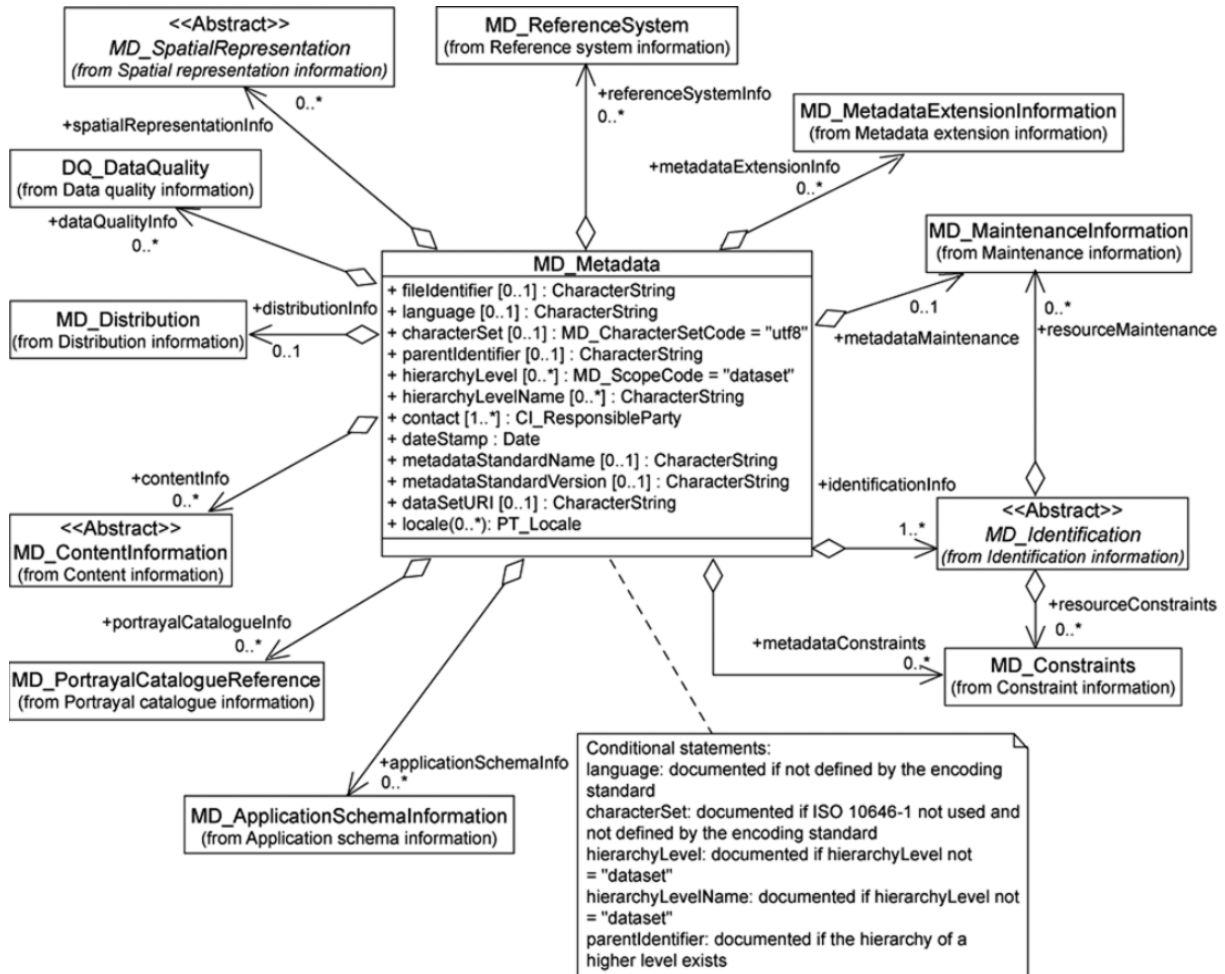
9.2.1 Metadata

The type MD_Metadata from ISO 19115 provides a class that describes metadata conformant to that standard. By including an attribute under the ProductionUnit class,

² [ANNEX I.1 GLOBALGAP \(EUREPGAP\) DEFINITIONS](http://www.globalgap.org/annex11/annex11_definitions.htm) available at www.globalgap.org

this allows any production unit to include appropriate information about that data such as how it was collected, what the quality of the data is etc.

There are a variety of standards for metadata (DC, AGLS, ANZLIC, FGDC, etc) that can be used, and MD_Metadata is designed to introduce a flexible capability to a model so that a variety of standards can be used in a machine-process-able way.



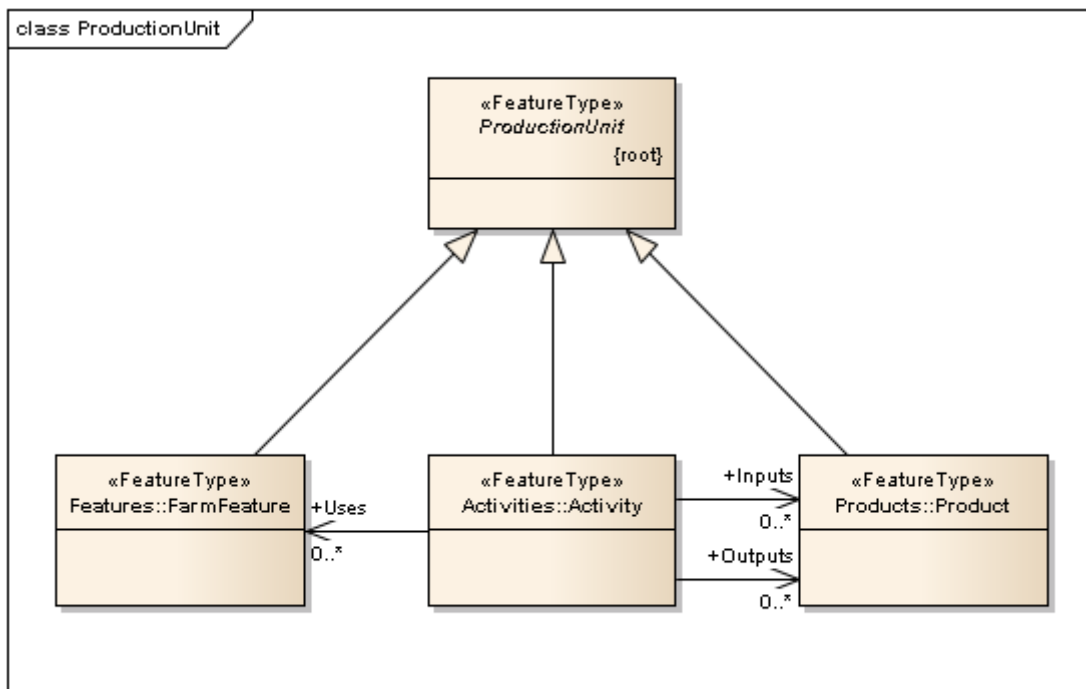
MD_Metadata entity set diagram from ISO 19115:2003: Corrigendum 1:2006

Other classes in this model are appropriate for direct use in FarmML as opposed to indirectly as metadata.

9.3 Production Unit Subtypes

Three major subtypes of a production unit have been identified;

- **Feature** – A feature of a farm, or the farm itself. Farm features usually have associated geometry.
- **Product** – something generated on the farm. May be produced for sale or to support some other farming activity, for instance growing feed for livestock.
- **Activity** – An action taken relating to the farm. Examples include the generation of product or maintenance of farm features.



9.3.1 Features

Farming features are the nouns of a farm – fences, fields, cows, crops. They are the objects of a farm that are used in farming production activities.

In FarmML a FarmFeature is any object that is involved in farming. These are often kept on a farm or used in farming activities. Examples include regions of land (fields, paddocks, water features) or farm assets (fencing, sheds, vehicles, livestock, crops).

Farming features have been subcategorised further:

- **Management Units:** farm boundaries, paddocks, laneways, land classes, soil types.
- **Farm Assets:** sheds, watering points (troughs), electricity infrastructure, irrigation infrastructure, effluent infrastructure, watering infrastructure, electric fence infrastructure, machinery, silos, and yards

- **Natural Resource Assets:** shelter belts, remnant vegetation, pest plants and animals, native animals, land management issues (erosion and salinity).

9.3.2 Product

The main objective of most farms is to generate produce for sale. As the primary purpose that is directly related to the success of the farm is the quality and efficiency of production, farming product has a range of concepts that need representation in FarmML.

9.3.3 Activity

Farms don't run themselves. Any activity performed in the context of farming needs to be able to be described by FarmML. Examples include maintenance and generating produce.

9.4 Associations between Features

Farms have paddocks, paddocks have fences, livestock, crops and so on. These associations are simple relationships between production units that are straight forward to model.

9.4.1 Association between Production Units

There may be a case for supporting generic association between production units at the superclass level. This is because there are situations where one production unit could be associated with many different kinds of other types of production units that may not be easily defined or known in advance.

This approach is not recommended though as it allows for unspecified interpretation of the model. All associations should be explicit and have context. The association should be made at the narrowest possible point of definition allowing for only one possible interpretation of what that association implies.

9.5 Selecting a level of Abstraction

It is an unfeasible expectation that a domain model will completely define a domain at any stage of its development. For the model to be useful to more than a very narrow domain some abstraction is needed to allow the model to be applicable more generally.

9.5.1 Subdomains

While all farming activities share common concepts, farming logically breaks down into a variety of subdomains. To limit the complexity of supporting FarmML it is proposed that a core set of FarmML concepts should be isolated to separately packaged subdomains that can be considered separately.

9.5.2 Breakdown Choices

Breaking farming domains down by category of farming activity seems a logical choice as the activities performed in farming differentiate the various types of farms. An example breakdown could be:

- **FarmML Core:** Concepts that apply to most farming activities. The high level structure of the FarmML model would be contained in this package.
 - **Activities:** Generic farming activities commonly applicable to all farming or abstracted and used as the base concept for more specific farming activities.
 - **FarmFeatures:** Parent concept for management units, farm assets and natural resource assets.
 - **FarmAssets:** Generic farm asset concepts, such as fencing and sheds.
 - **ManagementUnits:** Generic farm management units such as farm boundaries and paddocks.
 - **NaturalResourceAssets:** Generic natural resource assets such as native animals and remnant vegetation.
 - **Products:** Generic farm production concepts.
- **Aquaculture:** offshore longline, rack and caged, and onshore.
- **Dairy:** dairy cattle farming.
- **Fishing, Hunting and Trapping:** e.g. rock lobster and crab, prawn, line fishing, fish trawling, seining and netting as well as general hunting and trapping.
- **Forestry and Logging**
- **Fruit and Tree Nut:** e.g. grape, kiwifruit, berry fruit, apple and pear, stone fruit, citrus fruit, olive growing.
- **Mushroom and Vegetable**
- **Nursery and Floriculture:** under cover, outdoors, turf growing.
- **Other Crop Growing:** e.g. cotton and sugar cane.
- **Other Livestock Farming:** e.g. horse, pig, beekeeping, deer.
- **Poultry:** for meat or eggs
- **Sheep and Beef Cattle:** grain sheep or grain beef cattle farming, rice growing, sheep and beef or specialised farming and feedlots for beef.



FarmFeatures, Activities and Products particular to a farming category will extend the appropriate core FeatureType but be defined in the appropriate subdomain package.

9.5.3 Benefits

This breakdown approach to FarmML improves governance by allowing individual sub-packages to be version controlled separately. Dependencies to external models can also be handled at the subpackage level, meaning a change to Dairy won't affect external parties that are dependant on Husbandry with a whole of FarmML version change.

This approach is also very useful for implementers. Beyond the core package, implementers need only cater for the portions of FarmML that are of interest to their application. It also helps consumers of their applications by allowing for easy advertisement of what parts of FarmML they support.

9.5.4 Local Variation and Model Flexibility

While FarmML is currently driven by the interests of the Department of Primary Industries (DPI) Victoria Australia, a goal has been identified that FarmML needs to be globally applicable; farming is a globally performed activity and collaboration on FarmML from disparate viewpoints only improves FarmML's potential to be interoperable and supported.

To support this while meeting needs particular to DPI, it is recommended that FarmML can also isolate packages based on locality. These packages can be extended at the appropriate level to introduce locally relevant details. An example of this segregation could be:

- **FarmML**– Concepts are applicable to farming in any location globally.
- **FarmML Aus** – Extend concepts with details that are geographically relevant only to farming in Australia.
- **FarmML VicDPI** – Concepts that pertain only to DPI processes.

This separation isolates and removes the concepts not relevant at a more generic level. It is also a logical breakdown of the governance of the domains, allowing for more appropriate custodianship and separate versioning.

The scope of this current effort in modelling FarmML is limited to the generic FarmML Core and potentially some of the subdomains of farming. The local extensions concept is for this effort not a great impact, but how this local variation is to be supported needs to be considered.

It is also anticipated that FarmML will need to support extension for the purposes of specialised farming domains beyond the breakdown choices listed in 9.5.2. It is important that FarmML can cater to these extensions while not restricting use of extended data in a more generic context.

9.6 Reuse of Existing Conceptual Models

If a concept relevant to FarmML has been modelled in another domain it may be appropriate for reuse in FarmML. This saves re-definition of a common concept and also helps promote interoperability between the models. Care must be taken though as introduction of a cross-domain dependency can introduce complexity especially as the models change. CSIRO has taken special care to focus on this problem though, and the use of a domain model registry is one of many strategies in place for accurately managing cross domain dependencies.

This section will list any external concepts considered for reuse in FarmML. While currently only concepts deriving from the ISO 19115 series of standards have been included in FarmML, it is likely that classes from models listed in 3.2 will be used in the future. As each time this occurs introduces a cross-domain dependency, each instance should be carefully considered and discussed.

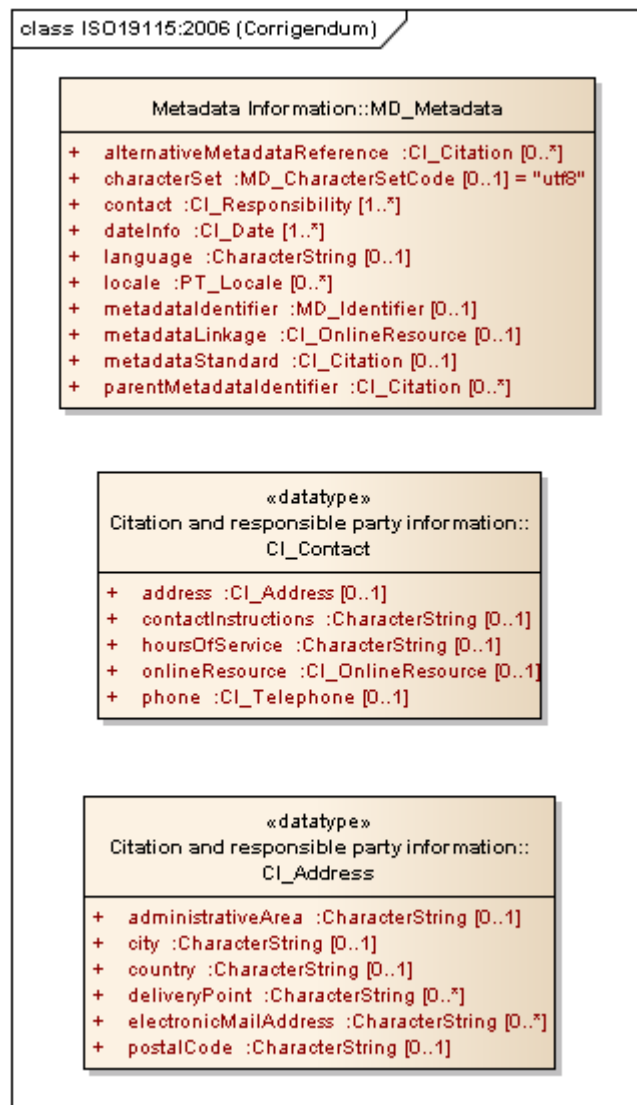
9.6.1 Contact Details

Again ISO 19115 includes types for contact information for data custodians etc. There are appropriate point of contact and contact details that is relevant for FarmML for farms, suppliers and produce purchasers etc. The CI_Contact class provides a model for contact details that can be used in FarmML.

9.6.2 Addresses

The ISO 19115 standard covering metadata is a good choice for a type that defines the structure of address information. As FarmML will already be dependent on this standard there is no new dependency introduced by this cross-domain link.

FarmML needs address information for the farms and possibly other classes. Where CI_Contact is used, CI_Address is included as part of the contact details. There is a likely situation in FarmML where the address class is needed directly without contact information.



10 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.

10.1 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 modeled.

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.

10.2 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.

10.3 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.

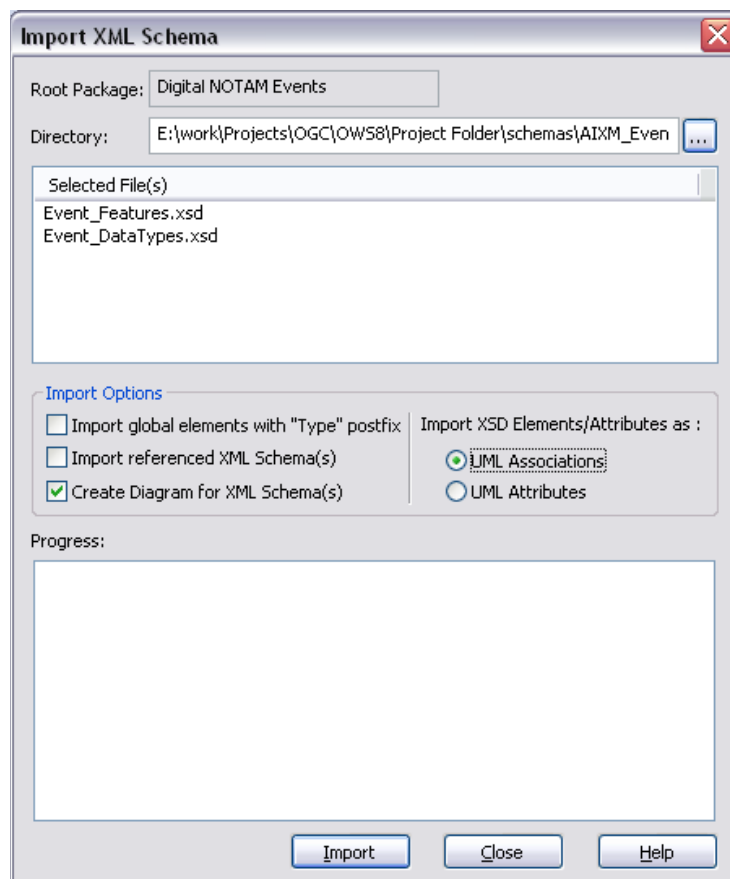


Figure 18 Import XML Schema

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.

10.4 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:

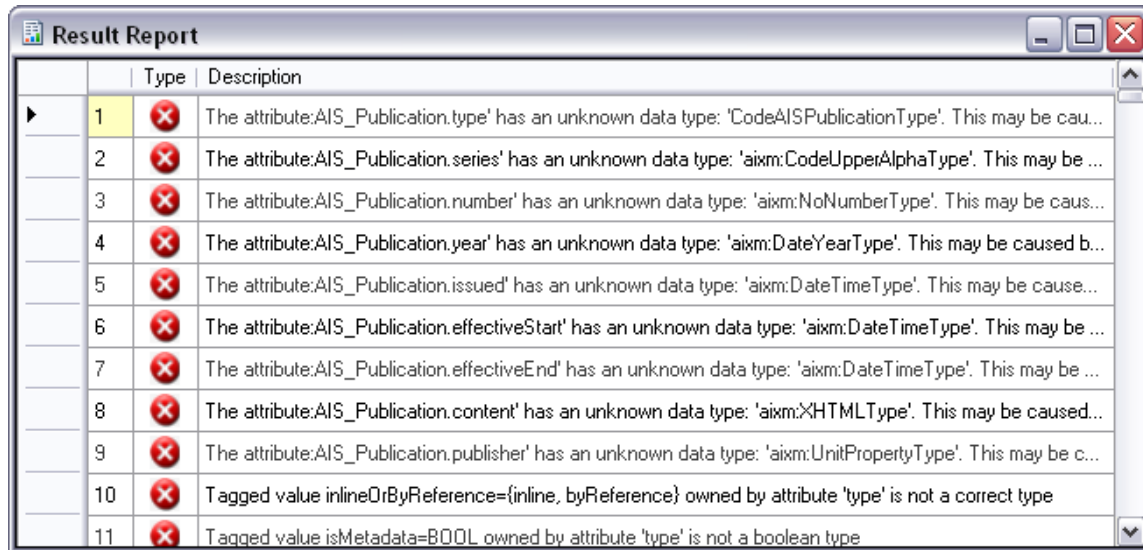
10.4.1 Model Mapping

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.

10.4.2 Conformance Checking

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.



| | | 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 19 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>

10.5 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.

10.6 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.

10.7 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.

10.8 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 modeled 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.

10.8.1 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 toolbox entry (say Application Schema) and select “Synchronise Stereotype” - which will affect all entities for the current model.