

METABULA LIMITED

DataManage Case Study : Layers and Models

Revisions:

1.0	NRB	13/05/2005	First release
1.1	NRB	27/07/2005	Feedback from H. Teiggeler

Table of Contents

- 1. BACKGROUND..... 1
- 2. DATA MODEL OVERVIEW..... 1
 - 2.1 Class Libraries 3
- 3. DATA MODEL IMPLEMENTATION..... 3
- 4. DATA MODEL EXTENSIONS 6
- 5. OVERVIEW OF DATASTORE ARCHITECTURE 7
- 6. ACHIEVING DATA MODEL INDEPENDENCE..... 8

1. Background

ISO 10303 (STEP) and ISO 15926 are two mature standards for data exchange and integration in the Oil and Gas Industry. The STEP Application Protocol (AP) of most immediate interest is AP221. The scope of AP221 extends to Process Plant in general rather than specifically Oil and Gas.

Both of these standards are still evolving and are expected to continue to do so for some time. Several projects have implemented datastore, or datawarehouse solutions 'based upon' different evolving versions of the data models specified by the standards development. Technical or project limitations have constrained a number of these implementations and different interpretations of the use of a generic modelling framework have ensured that no 2 systems are the same.

It is believed to be unrealistic and unwise to assume that there will be complete convergence and consistency of use to a single definitive, unchanging standard across the industry. The 30-40 year asset lifecycle of the industry in question, combined with increasing legislation in operation and the likely IT development over this period indicate that unless a standard can evolve as required then it will fail to deliver its full potential savings.

Furthermore, it is expected that whereas a standard will meet the generalised, common requirements of an industry, each owner-operator will demand additional, more specific compliance to *their* best practice templates and extensions.

2. Data Model Overview

The AP221 STEP (ISO 10303-221 ARM) model and the ISO 15926 data models are built upon the EPISTLE¹ data modelling framework. The two models are often referred to in general terms as "STEP Models". Although not necessarily correct, this has become quite common.

Without going into the full detail of the formal EPISTLE documentation, a number of key distinctions of these models and their use are summarised here.

- EPISTLE models provide a mechanism for specifying data and the context required to describe its meaning.
- The models are defined at a very generic level.
- The models largely remove "real world entities" as terms of reference. i.e. they do not describe things in the way that we would naturally talk about things that we see. The models describe things at a low abstract level, rather than from a particular perspective of use. They revert to the "lowest common denominator" to allow re-use and sharing.

¹ European Process Industries STEP Technical Liaison Executive

METABULA CASE STUDY

- Information is represented as collections of structured data items. The meaning is conveyed by the structure and by the classification of each item.
- Representation and exchange of information in “engineering terms” is reliant upon the use of a library of standard classification types.
- Data is stored without many of the assumptions that are often made:
 - Any individual property of an item, including its name, may have more than one value.
 - Any property may be described in more than one way, e.g. different units, different language, etc.
 - Individual property values may be represented independently as string, integer, real, etc. values, as appropriate.
 - Different representations may store data appropriate to different contexts of reference and use. For example, the same temperature may be described simultaneously as, 200C, 573K, “HOT”, “requiring gloves to touch”, etc.
- A full change history and audit of past values is maintained.
- Lifecycle aspects of data are covered.
- Distinctions can be represented between ‘real’ data and data relating to ‘what if’ scenarios.
- Distinctions between typical and specific things and values are included. This covers actual versus intended and catalogue (= class) versus ‘serial numbered’ (= physical_object) cases.
- Functional and physical items are modelled independently, i.e. a tagged item can be distinguished from the serial numbered part currently installed. This allows the tracking of ‘what was installed where and when’.
- Multiple structures and multiple independent hierarchies of data are supported. This allows different breakdowns to be supported (i.e. system, functional, asset, etc. navigation).

This functionality results in a complex data structure defined in abstract, generic terms. The benefit is high quality, well specified data, stored in a format that does not make assumptions about the ways in which it is, or may be used. The primary motivation to define common standards around high quality data models has been the cost associated with interfacing one system to another, and the costs incurred when assumptions and business rules encoded into the structure of application data models changed. The need for standardisation becomes even more apparent when one considers that it is rarely a case of interfacing one system to another but, rather, of interfacing dozens of systems. Separating applications and data brings many benefits in terms of longevity, maintainability and portability.

This high quality data modelling approach allows a consistency of data as business needs, user requirements and application rules change. It also provides a basis for different application to share common data. This allows single source data entry; eliminates re-keying; avoids transcription errors and provides a basis for managing and propagating changes in data more effectively. It also allows integration of data from disparate application systems such as equipment index lists, line lists, 2D and 3D CAD, datasheets, databases, spreadsheets, etc.

2.1 Class Libraries

The effective integration and exchange of data requires a set of common classifications. These are deliberately removed from the data model. This separates a consistent core data representation from a set of library, or catalogue, classifications. The term “class library” or “reference data library (RDL)” is commonly used here.

The class library provides a hierarchy of classification types. This allows us to say that, for example, that “P-4410” is a CENTRIFUGAL PUMP and that CENTRIFUGAL PUMP is a specialisation of the more general equipment types PUMPS and ROTATING EQUIPMENT. Standardisation of classification types allows us to specify the characteristics and property types required to define our engineering items. Standardisation also provides a common basis for describing, selecting, searching, ordering, etc. this equipment. Specific equipment types may be considered in terms of more generalised descriptions, e.g. CENTRIFUGAL PUMP and RECIPROCATING PUMP are both *specialisations* of the more general class PUMP. Any properties defined for the class PUMP are *inherited* by its *specialisations*.

3. Data Model Implementation

DataManage provides a generic data modelling and management environment. A user defined data model, here POSC/Caesar Snapshot E, is defined using the templates provided in DataManage’ default data model. The data model is loaded directly using an Express loader. This set of classes is defined formally as a Data Model; “ISO/FDIS 15926-002 (lifecycle_integration_schema)”.

The “ISO/FDIS 15926-002” data model (from now on referred to as “LIS”) provides the entity types (classes) that are used to define the Reference Data Library (RDL). The LIS Reference Data Library will be referred to as “LIS RDL” from here onwards. The “LIS Data Model” in DataManage provides the dictionary for the definition of the RDL members.

This means that we can now create an instance of any entity within the “LIS Data Model”. Take the *class_of_inanimate_physical_object* as an example of one of these entities. An example of an instance, or member, of *class_of_inanimate_physical_object* is *PUMP*, *COMPRESSOR* or *BUTTERFLY VALVE*. We now have:

- PUMP is_a class_of_inanimate_physical_object
- COMPRESSOR is_a class_of_inanimate_physical_object
- BUTTERFLY VALVE is_a class_of_inanimate_physical_object

DataManage provides the capability for any object in the database to behave as a class. The RDL members are grouped together into a formal “LIS RDL Data Model” and are given class behaviour.

This is a capability unique to DataManage and forms the basis for a layered data model implementation.

We can now create an instance of any entity within the “LIS RDL Data Model. Take *PUMP* as an example. An example of an instance, or member, of *PUMP* is *P-1234* or *PUMP_10*. We now have:

- P-1234 is_a PUMP

METABULA CASE STUDY

- PUMP_10 is_a PUMP

and

- PUMP is_a class_of_inanimate_physical_object

This process allows applications to “createInstance (of) PUMP” when dealing with the “LIS RDL Data Model” as well as being able to “createInstance (of) physical_object_class” when dealing with the “LIS Data Model”.

The RDL is not, in a native EPISTLE implementation, a Data Model. This is because the models were specified in terms of a structured file format for data exchange. Exchange is defined as off-line sharing between two systems in the form of a file. Here, when the exchange file (a “Part 21 file”) is parsed it is not important that a parsing application has to interpret the meaning of the contents. However, when a data model is implemented as the basis for data sharing (creation, editing, etc.) of data on-line, it is not advisable to store the data in a passive “data bucket” that does not understand the context or meaning of the data it holds. If this were the case, then each application would have to interpret the data model structure and support the functionality required to infer class behaviour, attribute inheritance, etc. This would require a substantial amount of application code to be written directly to the data model structure. It would also be impractical in many cases.

The DataManage layered data model implementation is shown in figure 1. Within a single DataManage datastore, both the Model layer (the “LIS Data Model”) and the RDL layer (the “LIS RDL Data Model”) act as distinct Data Models and can be instantiated independently. Data in each layer (including project or business data, e.g. my_pump) may be accessed using the DataManage API. The definition of independent behaviours for each layer allows underlying layers to be hidden from the end user and user applications. Since the RDL layer insulates the application code from the underlying EPISTLE data model, different class libraries may be used. Logically, these will present the same operations to the application.

Multiple data models within DataManage

All entities are stored within a single DataManage system but can be logically separated into distinct layers that are linked by bespoke class behaviour. This overloaded behaviour is the implementation of a mapping from a model's associations into DataManage relationships.

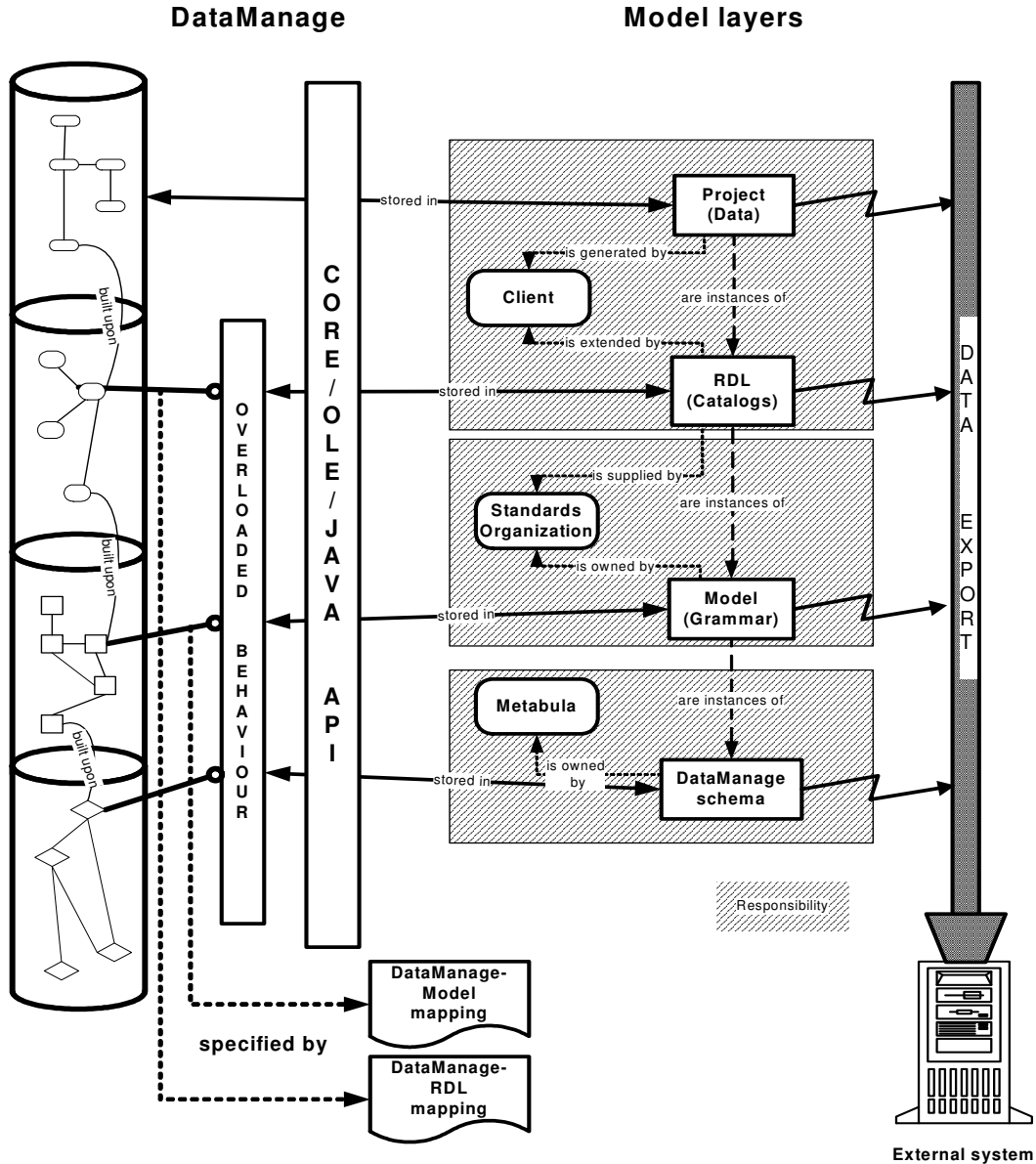


Figure 1

4. Data Model Extensions

A schematic of a layered data model implementation in DataManage has been presented. These layers are derived from a meta-modelling or “class of class” approach.

However, it is not sufficient to consider that a standard data model or RDL set will fully define a layer. In each case, it is anticipated that a set of “Wrapper” classes will be required to define, explicitly, things that may be implicit in the Express model or are otherwise required. This would include the inheritance of class behaviour for the appropriate classes. A set of “Extension” classes is also proposed to manage the addition of a required class not included in an existing model or RDL. The need for extension classes may arise either from incompleteness of the model or RDL, or alternatively from requirements that are outside their intended scope.

It is important that the addition of extensions in this manner should be part of a managed process so that the distinction is maintained between the actual standard (against which conformance can be established) and the project specific extensions over and above this.

Each conceptual layer described previously will actually be implemented as a compound layer, as shown in figure 2.

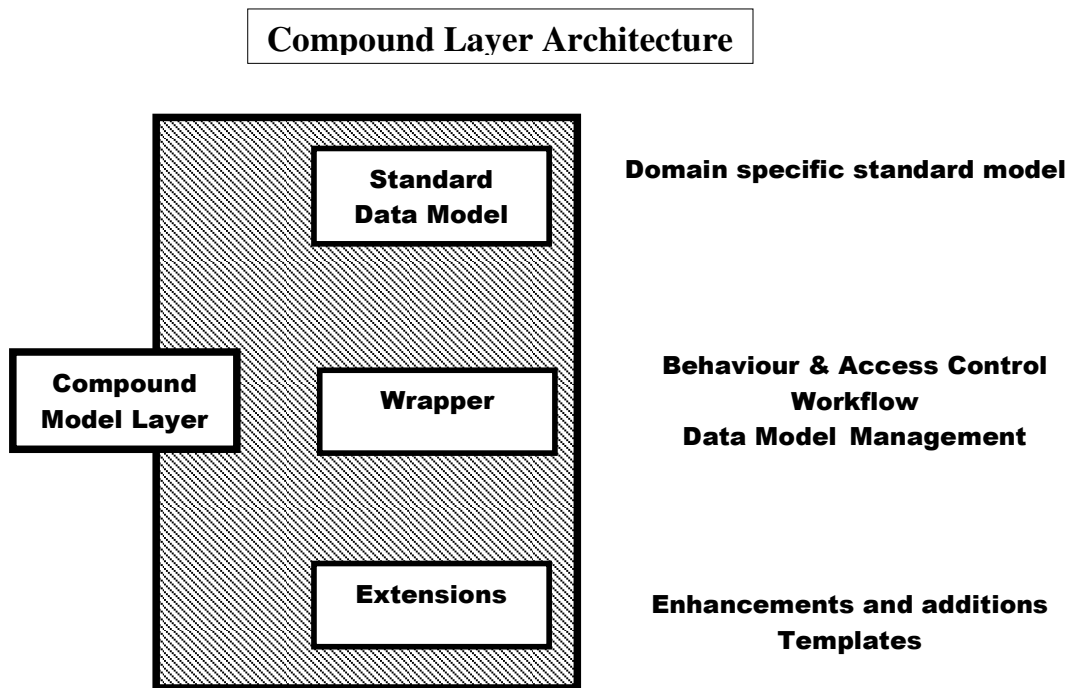


Figure 2.

5. Overview of Datastore Architecture

The representation of data using a STEP or EPISTLE model in DataManage differs from a relational database implementation. The model semantics are not the same because a DataManage datastore can support modelling concepts outside the scope of the relational model.

A typical application has browser logic that is separated from its data access code as shown in figure 3(a). The inclusion of a clearly defined interface between the application and data handling code enables the decomposition of the data access routines into a set of functions. These functions may then be supported independently by a number of distinct *data access modules* as shown in figure 3(b). A given data access module will manage the manipulation of the data and data structures within the relevant underlying repository.

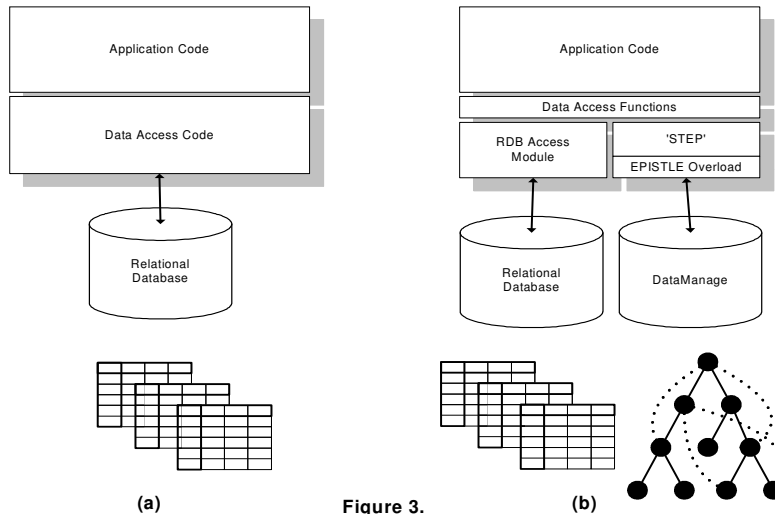


Figure 3.

It is important to note that the two data access modules are not the same internally. They will differ in so far as the logical structure of the data differs. Externally, they will present the same 'signature'.

Metabula provides a set of methods / behaviours to overload the DataManage API to handle an amount of the complexity of the underlying EPISTLE models². The intention here is to hide the physical implementation details and complexity of the model, not hide or overload their common, high quality, data structure. The appropriate level of 'hiding' is envisaged to be at the level of specific, logical data handling requirements:

- CreateObject
- SetValue
- Etc.

² This has otherwise been referred to as the EPISTLE API.

The DataManage data access module may then be written against standard API functions. The overloading of the required EPISTLE behaviour will be hidden.

6. Achieving Data Model Independence

It is believed that application portability across different EPISTLE models may be largely achieved through the redefinition of the EPISTLE overload required for different models (and model versions). This separates (hides) the physical implementation detail from the STEP data access module, the data access module and the application code. This is enabled by, and dependent upon, the conceptual similarity between the EPISTLE family of models, i.e. whatever they do represent and however they choose to do this, they use the same basic logical concepts³. An example here is the difference (or similarity) between the LIS “class_of_inanimate_physical_object” and the AP221 “class_of_material”.

The same isolation of concepts from physical implementation or terms of reference is required between the application code and data access module. It is unsafe to assume, for example, that different models will use the same terms / labels for equivalent things. The previous example illustrates a different naming convention for, arguably, equivalent concepts. Again, note that different models may display different levels of maturity and completeness and that all aspects of one model may not be visible in another. The DataManage “Layered Model Architecture” addresses this and allows commonality to be maintained, where required through a fully managed process of model extension. However, the identification / naming problem may arise at the RDL layer. It should be expected that names for property values may differ from model to model.

An unfortunate example is that a property of “Weight” in one reference library may well be identified as “Weighth” in another. The poor quality of data at the RDL layer is a major problem. Metabula have proposed the introduction of a formal QA process to address this issue. Further support in this area is encouraged. These problems will prove a major barrier to uptake unless they are addressed. This does not currently appear to be a priority for issuing bodies and serves to lessen the credibility of the organisations. Different data models, changes to data models, different class libraries and changes to class libraries can all be managed – poor quality standards are difficult to accommodate.

The switch from one model to another, or one class library to another is handled internally by the DataManage system. This is accommodated by the DataManage dynamic schema. An example of the schema browser is shown in Figure 4 .

³ This is inherited from the application of EPISTLE modelling principles.

METABULA CASE STUDY

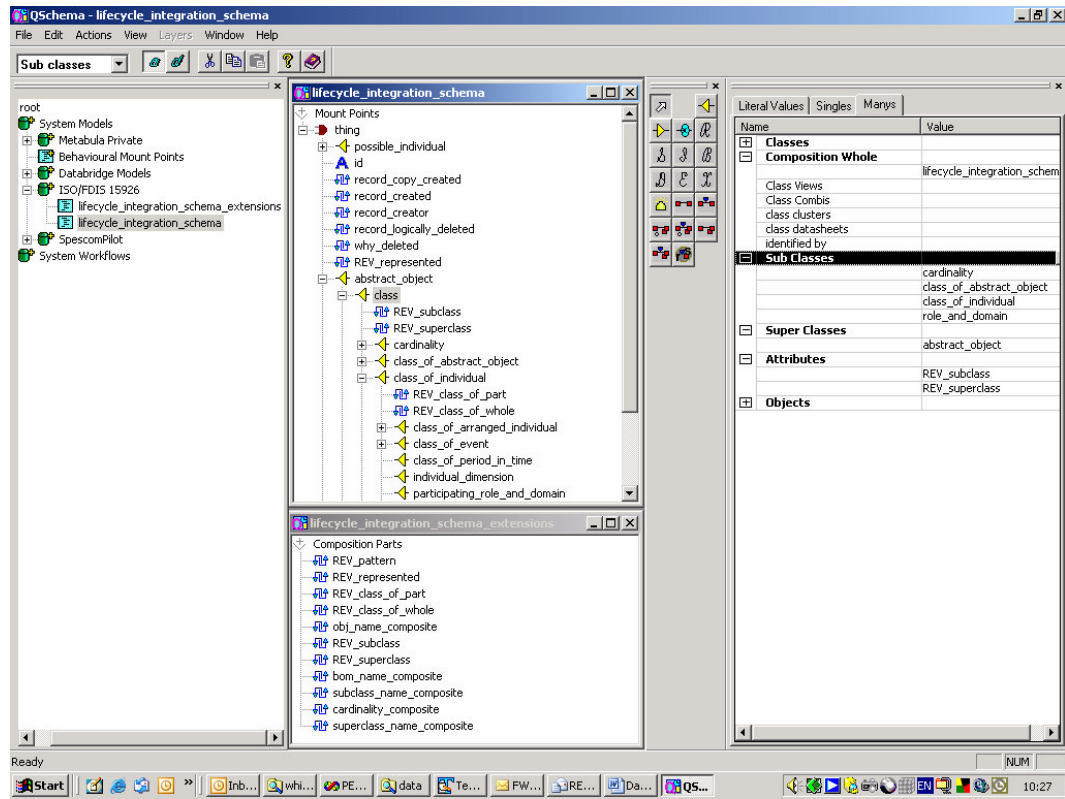


Figure 4