# Modelling the Enterprise Data Architecture



Unlike the simplistic models in books and training courses, a real enterprise has a very complicated data architecture. Most of the data will be held in large legacy or package systems, for which the details of data structure may be unknown. Other data will be held in spreadsheets and personal databases (such as Microsoft Access), and may be invisible to the IT department or senior business data administrators. Some key data may reside in external systems maintained by service providers or business partners. As you explore your own complex data architecture, you come to accept two realities:

1. You have little control over the way high-level business data concepts are realised. Data is likely to be highly dispersed, often without adequate controls on quality.
2. Most data is duplicated across a number of systems, with significant variations in quality, format, and meaning. Some of the copies, maintained by Enterprise Application Integration technology (EAI) or careful business processes, may be good (but probably not perfect). Most are very poor, maintained only by occasional batch transfers and stressed or broken manual processes. Organisational and business process conflicts, or simple failures of trust, may get in the way of common sense improvements.

The poor copies may be causing business problems. Furthermore, initiatives such as Customer Relationship Management (CRM) and Business Intelligence will need to merge data from various sources. Some organizations work to harness various legacy systems in end-to-end processes. Either the business or IT may be driving changes to simplify business processes, streamline data flows, and reduce duplication. Modelling can be of great benefit in meeting these challenges. But most traditional modelling approaches don't address these requirements.

They produce models which are either too detailed to be of use, or not detailed enough, and typically fail to focus on the difficult issues of the enterprise data architecture and the integration of its various components.

We believe it is important to create powerful, simple, but effective models of the data structure from an enterprise viewpoint -- a set of models known as the "Enterprise Data Architecture." This article describes a new approach, based on UML, which we believe meets the real requirements of modelling the Enterprise Data Architecture.

Note: some of the later steps of this approach introduce techniques which may at first seem a little complicated. Don't worry! You don't need to use all the techniques every time, and the earlier stages deliver benefit in their own right. The important thing is to develop models which help to answer *your* problems.

## What *is* the Data Architecture?

An enterprise's information systems architecture has many interrelated aspects, including applications, hardware, networks, business processes, technology choices, and data. As shown in Figure 1, the data architecture is a layered set of models which provides a solid foundation for strategic initiatives such as:

- A Data Strategy, outlining the business's aims and objectives for improved collection and use of data,

- Business process improvements,
- Decisions on the future of new and changed systems,
- Integration, data warehousing, and reporting initiatives.

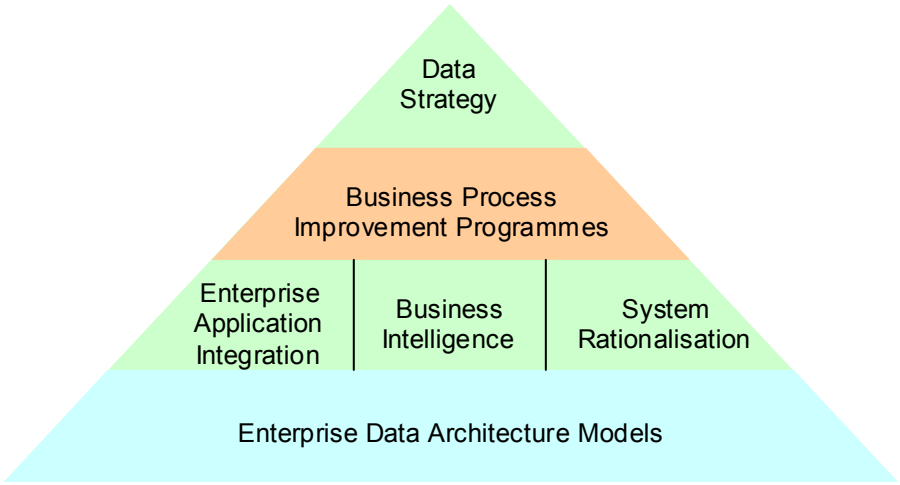


Figure 1: Enterprise data architecture models support a variety of common IT and business improvement initiatives.

Before describing what a data architecture is, it is helpful to consider first what it is *not*. As shown in Figure 2, the data architecture is not the set of detailed models of individual systems, because they cannot convey the "big picture" information required to meet the above needs. And it is not just the top-level models of business processes and system scopes, since they don't include enough detail to answer the real questions.

Figure 2 is a "data architecture map", which shows the scope and context of the data architecture. The idea is to map the major data areas in the enterprise on one axis, and the various types of models on the other axis, ranging from highly business-focused models to very detailed system structures. The scope of a complete data architecture is shown as a band across the middle of the chart:

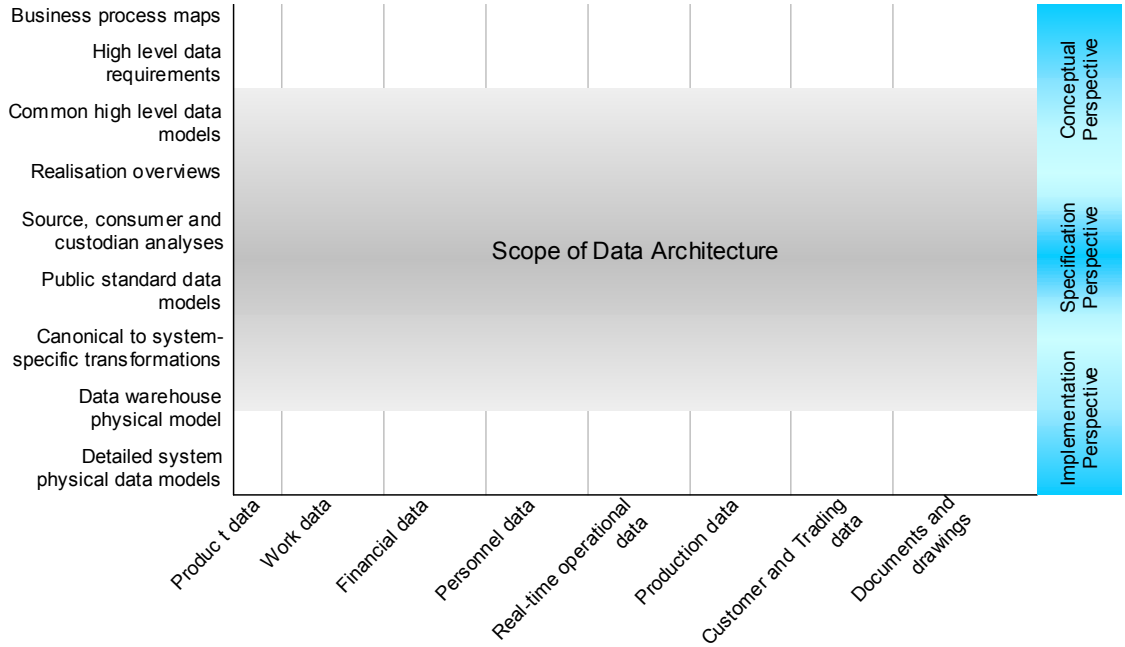


Figure 2: The data architecture map shows which models exist for which major data areas in the enterprise. A complete data architecture is a band across the middle.

The models which comprise the data architecture are described in more detail in the following sections. The groupings on the horizontal access will vary from enterprise to enterprise, but those above represent a typical set. The bands on the right-hand edge are not really part of the "map," but show how the models map onto the standard three-level perspective of UML-based methods such as RUP.

As well as explaining the scope of data architecture work, you can use this model to build a map of the current state of knowledge, and the scope of ongoing or planned activities. Simply plot existing or planned modelling efforts at the appropriate intersection. You can also use colour to indicate the status or validity of a model, which may be useful.

The data architecture map describes "what" comprises the data architecture. The Data Strategy and initiatives supporting it explain "why". The individual models describe what the data is, where it is held, how, when and by whom it is changed.

## Which Models Constitute the Data Architecture?

The data architecture is defined primarily by models at four levels, described in the following sections. As a general rule the high level data model will only change when there is a significant change in business processes, but the other models will exist in various versions representing the "as is" structure and one or more "to be" evolutions.

### High-Level Data Models

The top level is a group of high-level data models describing the business data from a conceptual viewpoint, independent of any current realisation by actual systems. Each high-level data model (HLDM) comprises:

- A common (canonical) UML class model of the main data items (the business entities) and their relationships,

- A superset of business attributes, including descriptions of their meaning (semantics), standardised formatting (syntax), and universal constraints.

Since these are *data* models, they will typically exclude class methods, although it may be appropriate to summarise where one business object has responsibility to manage the structure of others.

The model should include all attributes of business significance, and any which define the data structure (for example, inputs to a business rule which controls multiplicity).

Consider a hypothetical car rental company. The following is part of an example HLDM, showing how the business entity "vehicle" has two variants: cars and vans, and how any vehicle may be the subject of one or more rentals:

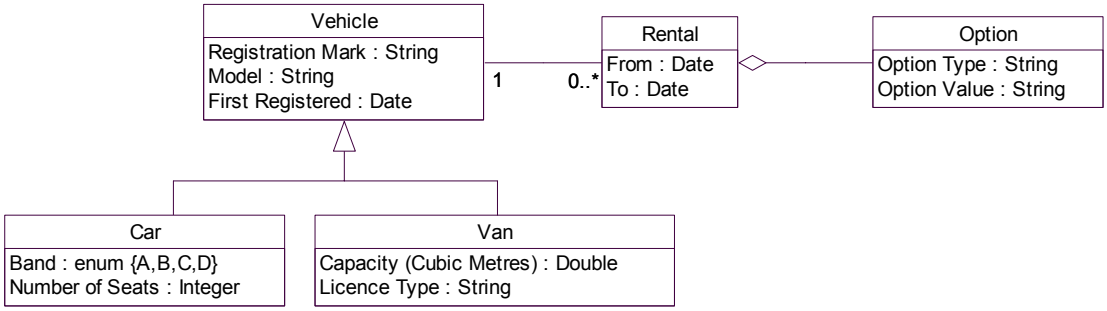


Figure 3: Partial high-level data model for a hypothetical car rental business.

For the purposes of this article, our examples have been dramatically simplified, but we believe you can see how the techniques could apply to examples with real-world complexity. We have also relaxed UML conventions on naming classes and attributes to aid in readability -- e.g. "Registration Mark" includes a space.

## Realisation Overviews

The next step is to model the relationships between the conceptual entities of the HLDM and the real key data objects of current or planned systems, showing how the conceptual entities are *realised* by the real ones. Relationships between different realisations of the same data item, and how changes are propagated across the various systems, are modelled at a later stage.

The key here is to focus on the "visible" data structure of the systems, i.e., the data structure exposed by the user interface, reports, and data interfaces. This may not be the same as the physical data structure, but that is unimportant. Highly-customisable packages may have a complex meta-model internally, but you are interested in its instantiation in terms of your business. An ancient legacy system may have an arcane physical structure for historical reasons, and the implementation details of an external service may be completely hidden behind an interface façade, but in both cases your focus will be on the visible structure -- the *logical system entities* and their attributes.

The following example shows how our simple car rental HLDM is realised by three systems: CarFleet (an in-house fleet management system), VanCare (an external system used to support outsourced maintenance of the van fleet) and RentalSystem (the main rental control system):

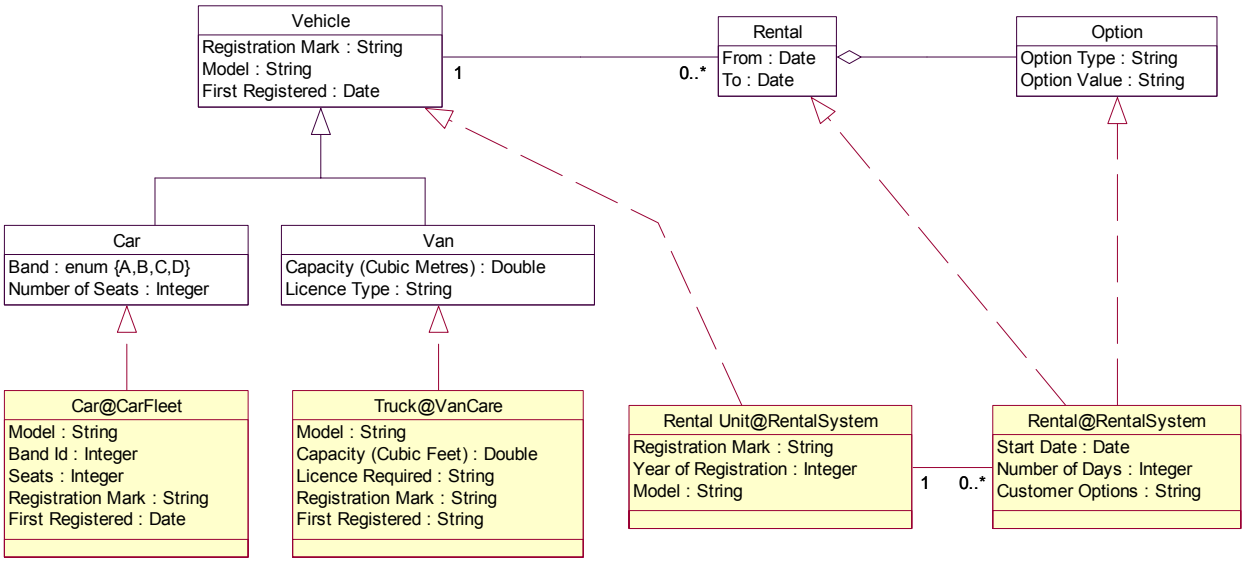


Figure 4: Partial realisation model shows how conceptual entities from the High Level Data Model are realised by the key data objects in three systems, shown in yellow.

UML realisation relationships are key to this model. Colour and physical layout can be used to good effect, and a consistent naming scheme such as the one shown should identify both the logical system entities and their host systems.

Where the conceptual and real entities have a different structure or meaning, then generalisation and aggregation relationships are used to break down the class structures until the realisations can be mapped directly, as shown in Figure 4. This approach can be used even where the HLDM is a meta-model and implementation models are concrete, or vice-versa.

## Source and Consumer Models

The next layer of models show the relationships between different realisations of the same data item, how changes are propagated across the various systems, and the organisational custodians of different data elements.

The models are similar to the realisation overviews, except that the focus is on identifying the role, provenance, and evolution of each data item, using the following stereotypes:

- <<Master>> identifies an agreed master source of data
- <<Use in place>> and <<Update in place>> identify where one system is able to use another system's data directly via existing interfaces. Notes should explain how this works.
- <<Copy>> and <<Updates Copy>> identify where one system takes a regular or irregular copy of another system's data (or list of updates), and whether this copy is used unmodified, or modified by the receiving system. Notes should describe timing and similar issues.
- <<Independent master>> identifies where a system is not the master, and should theoretically have a copy of the master data but where the processes are insufficiently established and as a result the second data set has diverged.
- <<Custodian>> identifies a custodial relationship between a data item and an organisation or role (shown as a Business Actor, with dependency relationships to appropriate data classes).

- <<Uses>> identifies a significant cross-organisational usage of data.

Where different attributes are handled in different ways (for example one realisation is master for some attributes of a class, and another realisation is master for others), the high-level data model should model those attributes using two or more separate classes. The Source and Consumer model can then clearly show the different responsibilities.

The following diagram extends our example to show as-is provenance and responsibilities:

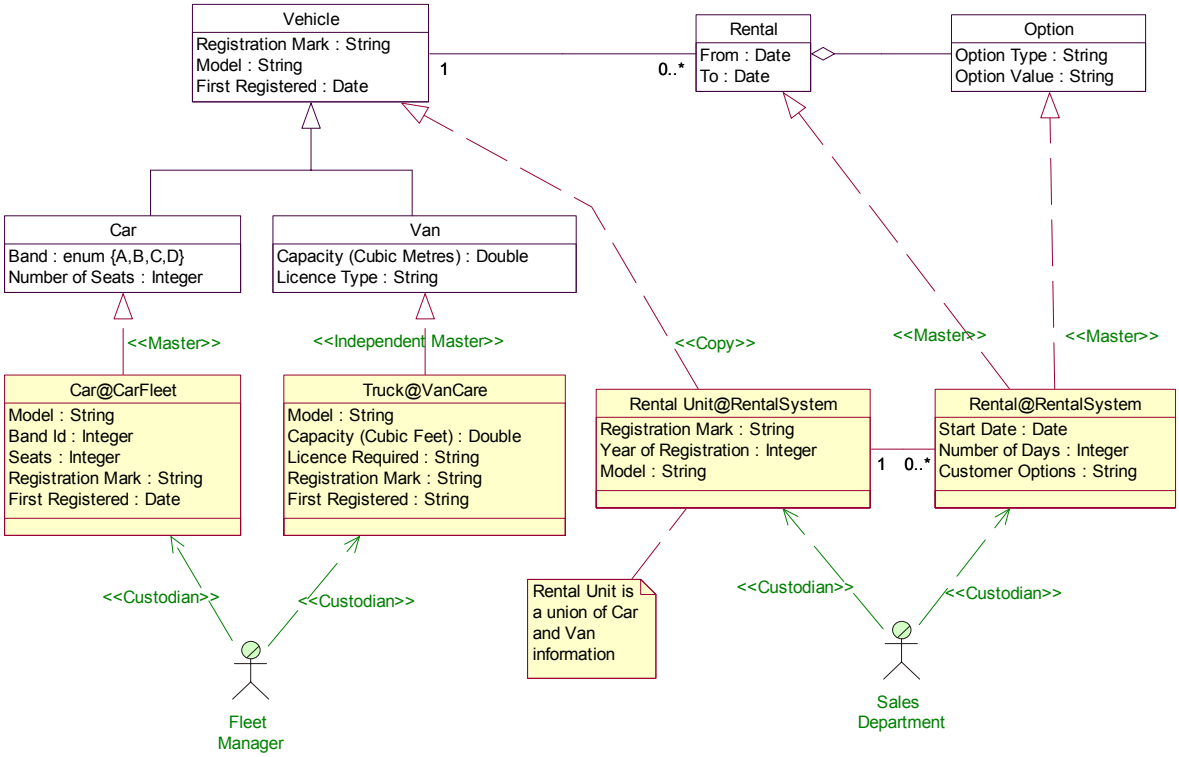


Figure 5: The source and consumer model adds information (in green) which describes how different realisations are related, and how they relate to different organisational roles.

## Transportation and Transformation Models

The last layer of models describes how the data in implementation systems is transformed as it moves between systems. They comprise:

- Physical class and attribute structure of system interfaces (which will equate to database structures where direct data access is the best or only option). This model will also show realisation of the HLDM within interface mechanisms such as an EAI hub or backbone,
- Realisation relationships between the different physical data structures,
- Transformation rules at the attribute level, documented using Object Constraint Language (OCL),
- Interface driver, constraint and timing rules, modelled using interaction or sequence diagrams.

If this looks a bit complex remember that you don't have to use this technique all the time, and you can use simple textual notes rather than OCL if you prefer.

Extending our car rental example, we want to use EAI to keep the Hire Unit list in RentalSystem up to date, extracting, merging and transforming the two source lists. The following "to be" model describes the physical interfaces and transformation rules required, including the canonical structure of data in the EAI messages:

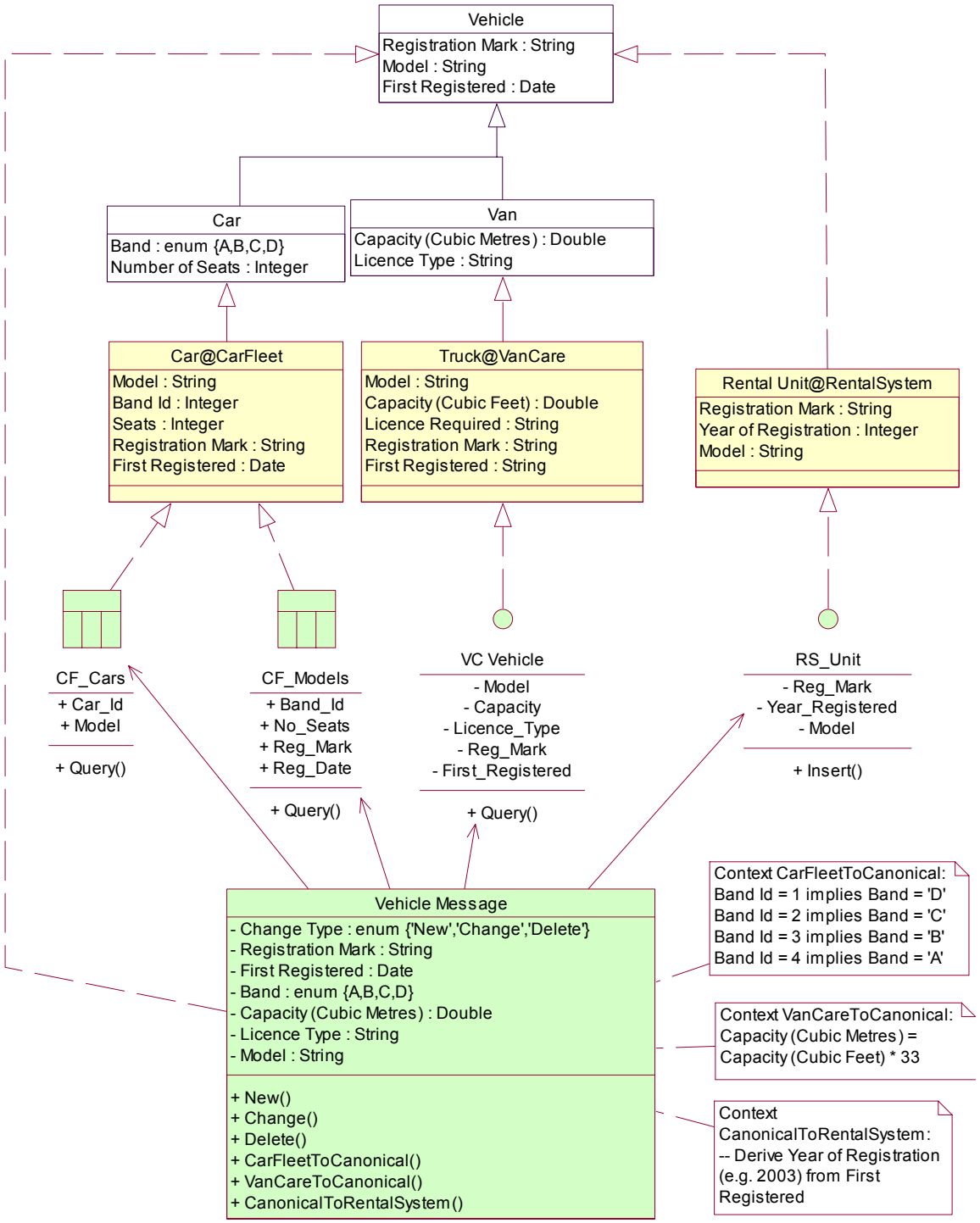


Figure 6: A transformation model adds detail showing how data is transformed as it moves between systems.

CarFleet has a data-based interface consisting of two main tables, Vancare has a programmatic interface (for example an object model or web service), as does Rental System which includes an Insert() function to receive the updates.

### Public Standards

Public or industry standards may have two roles:

- They may form the basis for either the HLDM, or realisation of the HLDM within the EAI backbone and external interfaces,
- They may determine the data structure of external interfaces or some physical systems, and therefore represent physical data structures to be transformed within interfaces.

### Model Meta-Model

To sum up, the meta-model below shows how the various models in our scheme and their components relate to one another:

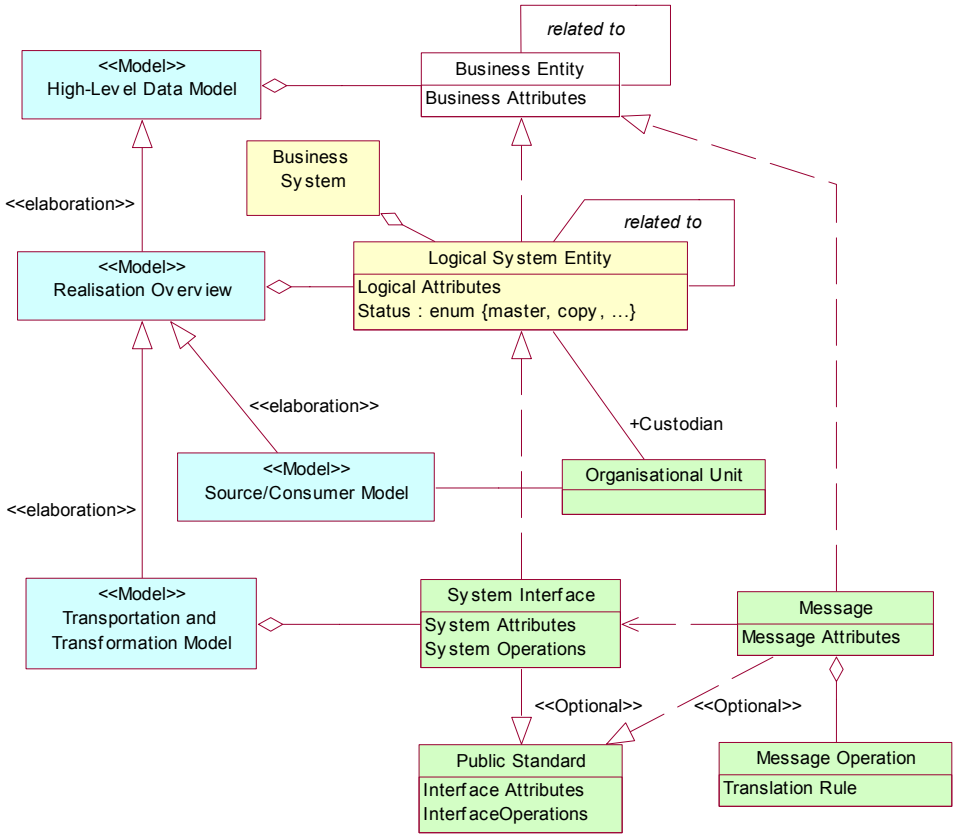


Figure 7: The meta model shows how the various models and model elements in the data architecture relate to one another.

## Using and Developing the Data Architecture

The data architecture has many uses. It helps you to get a handle on data as it is really used by the business, and is a key artefact if you want to develop and implement governance supporting a Data Strategy. It should also be used to guide cross-system developments such as Enterprise Application Integration (EAI), common reporting, and data warehousing initiatives.

Although our explanation proceeded from the top down, the data architecture is usually developed from the "middle-out," working from the data requirements of specific system interfaces and rationalisation exercises, *not* based on an exhaustive top-down process and

information requirement analysis. This allows it to develop to address specific tactical and strategic requirements without unmanageable dependencies, and provides a cross-check to data analysis originated on the basis of separate top-down and bottom-up modelling exercises.

The data architecture may never be "complete" for the whole enterprise. Even so, it provides a consistent approach and context for modelling activities. However, as the data architecture matures it may be appropriate to undertake some work to "fill in the gaps".

The models, in particular the Source and Consumer models, will support validation of target business processes, by identifying whether target data is contained within a single system, maintained by well-defined interfaces and processes, or spread across several (potentially inconsistent) sources.[1]

## Improving the Data Architecture – A Data Strategy

Modelling the "as is" data architecture can be extremely useful, and it can certainly show where things are sub-optimal. However, if you want to make future improvements, you will need more than just good models. Most of the issues around improving data collection, usage, and governance are non-technical. You will need to develop several things, working between IS and the business:

1. Principles establishing how the enterprise aims to collect, manage and use data,
2. The data architecture including both "as is" and "to be" models,
3. Governance rules and change control processes for the data architecture, managed jointly by IS and appropriate business representatives,
4. Policies for data management in each business area:
    - What data is stored,
    - Who is responsible for its collection and quality,
    - Who controls and who administers it,
    - How long it must be stored for, and how it is disposed of or archived afterwards,
    - Who may have access to it, and how it should be disclosed to parties outside the normal user groups.
5. A scheme for classifying information and associated risks so that appropriate security measures can be defined.

You may also need to help improve and document business processes where these need to change to improve data management.

This Data Strategy needs to be founded on clear, agreed principles, such as the following:

- Wherever possible, data must be simple to enter, accurately reflect the situation, and be in a useful, usable form for both input and output.
- Data should only be collected where it has known and documented use(s) and value.

---

[1] We plan a future article to discuss how the various models relate to the process of establishing an integration "backbone" or "hub" between systems, and using this to create interfaces and populate a data warehouse.

- Data should be readily available to those with a legitimate business need.
- Processes for data capture, validation and processing should be automated wherever possible. Data should only be entered once.
- Processes which update a given data item should be standard across the enterprise.
- Data should be recorded as accurately and completely as possible, by the most informed source, as close as possible to its point of creation, in an electronic form at the earliest opportunity, and in an auditable and traceable manner.
- The cost of data collection and sharing should be minimised.
- The enterprise, rather than any individual or business unit, owns all data.
- Every data source must have a defined custodian (a business role) responsible for the accuracy, integrity and security of that data.
- Data must be protected from unauthorised access and modification.
- Data should not be duplicated, except where essential for practical reasons. If data must be duplicated one source must be clearly identified as the master, there must be a robust process to keep the copies in step, and copies must not be modified.
- Data structures must be under strict change control, so that the various business and system implications of any change can be properly managed.
- Where possible, adopt international, national or industry standards for common data models. Where this is not possible, develop organisational standards instead.

## Conclusion

A documented understanding of the Enterprise data architecture is an essential pre-requisite to many common IS and business improvement initiatives. The appropriate models are quite distinct from both detailed system models and high-level business models. This paper outlines a set of UML models and techniques which should help to meet these needs.

## Further Reading

The use of UML for enterprise modelling is an emerging field. The techniques described here are new, and this is the first time they have been described publicly. However, we have found the following very useful introductions to the wider problem of modelling with UML at an architectural or business level:

1. *Business Modeling with UML – Business Patterns at Work*, by Hans-Erik Eriksson and Magnus Penker, pub. Wiley 2000.
2. *Enterprise Modeling with UML – Designing Successful Software Through Business Analysis*, by Chris Marshall, pub. Addison Wesley 2000.
3. *Realizing e-Business with Components,* by Paul Allen, pub. Addison Wesley 2001.

## About the Authors

Andrew Johnston is an independent consultant with two decades experience of most parts of the development life-cycle, and a variety of business sectors in the UK. In recent years he has focussed mainly on the Enterprise Architecture space, in the gap between IT Strategy, application design and technology choice. The second edition of his book on software project management and design, *A Hacker's Guide to Project Management*, has just been published by Butterworth Heinemann. For more details you can contact Andrew and view more material at www.andrewj.com.

Richard Wiggins has over 20 years experience in the IT industry, and has worked as an independent consultant for some of the top listed UK companies. Richard specialises in introducing or extending the use of UML for client organisations, in particular to address strategic and architectural perspectives. For more details, please contact Richard via e-mail at rw@patouche.co.uk.