

The Role of Enterprise and System Object Models in Information Systems Development and Integration

Matthew West - Shell Services International

Matthew.r.west@is.shell.com

1. Introduction

The objective of this paper is:

- to identify the distinctive roles of enterprise level and system level data/object models,
- to understand how the models can be used together to achieve integration across an enterprise, and
- to identify some practical implementation architectures for enterprise integration.

2. The Role of System and Enterprise Object Models

2.1 System models

System models are usually about supporting some specific task, which often involves the creation of data. As a result, the models used tend to focus on the roles that things play in the system. So it is not unusual to find object classes, or entity types that reflect the roles played in the system, rather than their underlying nature, such as *employee*, or *student*, rather than *person*. This is understandable because the system focuses on the behaviour in that role and for the purpose of the system, rather than in the whole of what a person is.

2.2 Enterprise Models

Enterprise models are primarily about enterprise integration. They need to identify what things are in terms of their underlying nature, so that the same thing can be recognised as such wherever it turns up. If your enterprise deals with both *customers* and *suppliers* it might be nice to know that the *supplier* you are just paying and invoice for is a *customer* who owes you ten times what you owe them. You can only do this if you recognise generic object classes that are not role based like *person* and *organisation* that might play the role of *customer* or *supplier* in some transactions.

There are two aspects to the use of enterprise models in business integration. The first is to ensure that the same thing is identified the same way throughout the enterprise and to provide information across the business about those things. In particular things like products, processes, organisations, locations, assets, and properties. It is surprising how often different systems result in the creation of different code for things like products. The second is to bring data together from different systems, to provide management information across the business.

2.3 Component and Data Reuse

Whilst what I have said so far is in the context of systems and enterprise models, there are wider implications for components and data in the enterprise. Components are essentially pieces of behaviour concerned with doing business, whilst data should represent the facts about an enterprise. Component reuse is about recognising that the same tasks need to be carried out within some different process. The things playing the roles in the task may be different in the different contexts, as represented by the data. The connection is made by aggregating the tasks to perform some useful purpose, and recognising what

plays the roles in the task for the different circumstances. Data reuse on the other hand happens through the same data being involved in different tasks, playing different roles.

2.4 Using System and Enterprise Object Models Together

From what I have said above, it turns out that when used properly, system and enterprise models complement, rather than conflict with each other. Each has its distinctive role to play. The key is in understanding that enterprise models are interested in the things in the business independent of the role played in a particular system, system models are interested in the role played in that system, and that these two views need to be linked together, by understanding what things play what roles.

3. Practical Architectures for Enterprise Integration

3.1 The nature of enterprise information

Enterprise information comes in a number of broad types. These are illustrated in Figure 1 below.

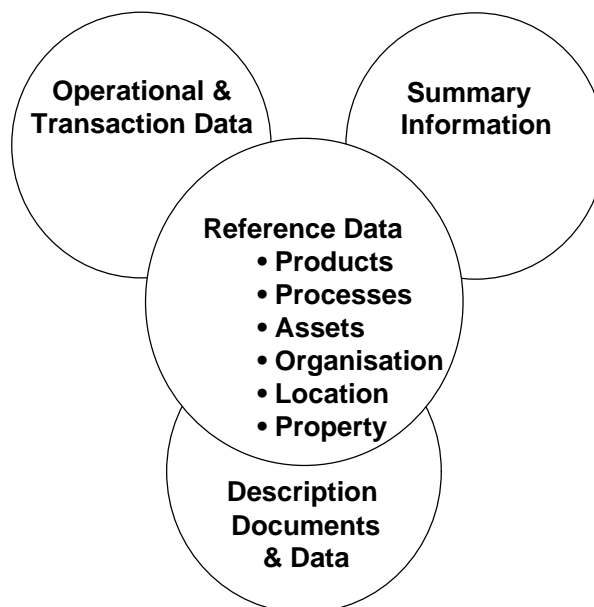


Figure 1: Enterprise information

Operational and transaction data, is the data that arises from doing business, whether its selling products, manufacturing the products, maintaining your plant, or providing services. Summary information is the summarisation of the operational and transaction data to provide oversight of the business, often in the form of key performance indicators, or in management information systems where the data can be sliced and diced to give different pictures of the business. Reference data is what is common and relatively static across the business. They provide the way transactions are categorised, and are the basis for slicing and dicing your summary information. Description documents and data describe mostly the reference data, providing the specifications for products and process, the drawings and data sheets for assets, etc.

Unfortunately, our systems architectures do not really reflect this. Below I identify some of the many implementation architectures that can be employed, followed by a practical architecture that can be implemented with today's technology.

3.2 Some Approaches to Integration

To illustrate different approaches, it is desirable to have a notation so different approaches can be compared. Figure 2 defines the notation I have used here. The notation is designed to make explicit the tasks that have to be performed to achieve integration.

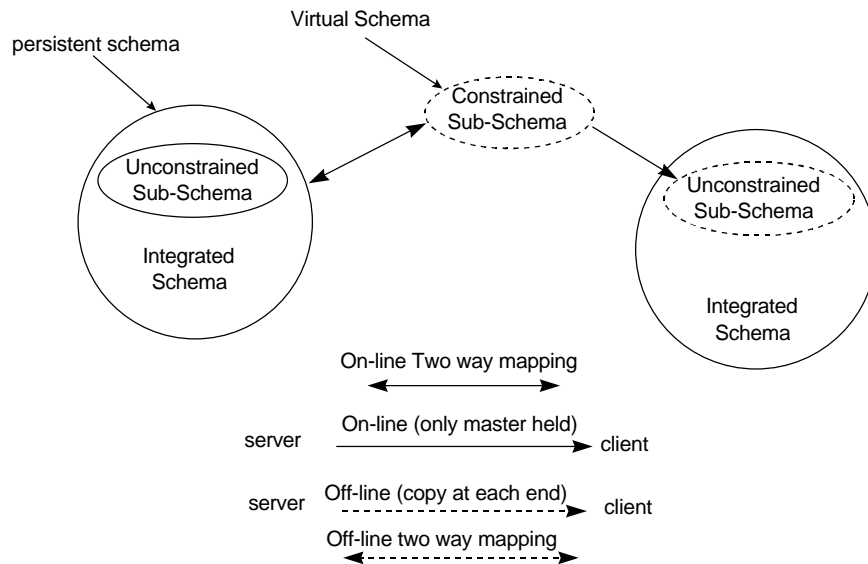


Figure 2: A notation for systems integration architectures

Circles indicate a complete independent schema, ellipses indicate a sub schema. Where the schema is shown as a solid line, then the schema is used for the persistent storage of data, when it is shown with a dashed line, this indicates that the schema is only used for transient storage of data.

A double headed arrow indicates that a mapping is necessary between the two models because they do not have the same structure. A single headed arrow indicates the master schema for the data. Where the lines are solid, this means an online mapping or transfer of data. A dashed line indicates an off-line mapping or exchange. For an online exchange, the data is persistently held in the master schema, whilst for data exchange the data is held at both ends.

Using this notation, I will now identify a number of different approaches to integration.

3.3 Full Integration

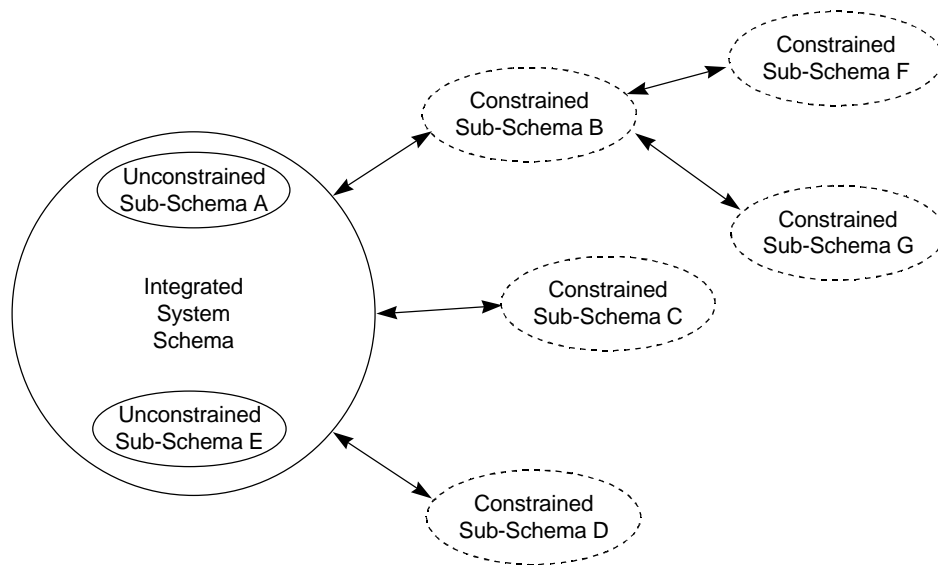


Figure 3: Full integration

Figure 3 above shows a fully integrated set of systems. There is only one persistent database, and all systems models are represented as views on that database, which would use the enterprise model. In principle this is the ideal architecture, because integration is explicit. However, there may be a number of reasons why this is not practical with today's technology. Physical distribution of data may be one reason, and security (all your eggs in one basket) might be another. However, the general trend over time in this direction seems inexorable, although it seems to be linked to buying all your software from one source, rather than in being able to integrate software from different sources.

3.4 On-line Messaging

One alternative to full integration, on-line messaging is shown below in Figure 4.

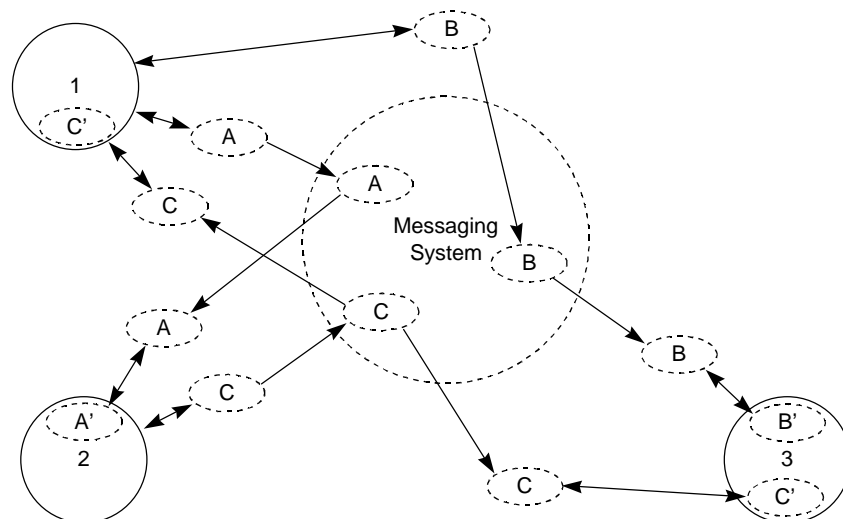


Figure 4: On-line Messaging

Here all the data is managed in separate systems, and a messaging system is used to make data available where it is required. The messaging system uses the enterprise model whilst the systems 1, 2, and 3 use their own system model. For any piece of shared data one system is declared the master, where that data is kept (though it may be updated from other systems). When system 3 needs some data that is held in sub-schema B', it makes a request to the messaging system for that data. The messaging

system knows which system is the master for that data, and transfers that request to the master system, in this case system 1. System 1 then maps the data into the structure of the enterprise model and returns it to the messaging system. The messaging system passes the data to the requesting system, 3, which then maps the data from the enterprise model into its own format.

Superficially, this may look more complicated than point to point interfaces, but when the number of systems is relatively large, the benefit for each system is that there is only one model you have to map into and out of, and you are effectively isolated from changes to the other systems.

This approach is particularly relevant when inter-application communication is required where the data requested is relatively likely to change. The disadvantage is that one system may suffer limited functionality because of the unavailability of another. It is also implicit that there is one system that is able to support the needs of all other systems for the data about some thing. This is usually not true. Most systems use an extended subset of the information about something required by another system.

This approach is sometimes known as wrapping, when applied to making legacy systems access their reference data from another system.

3.5 Off-line Messaging

A variant of the messaging approach is off-line messaging. This is illustrated in figure 5 below.

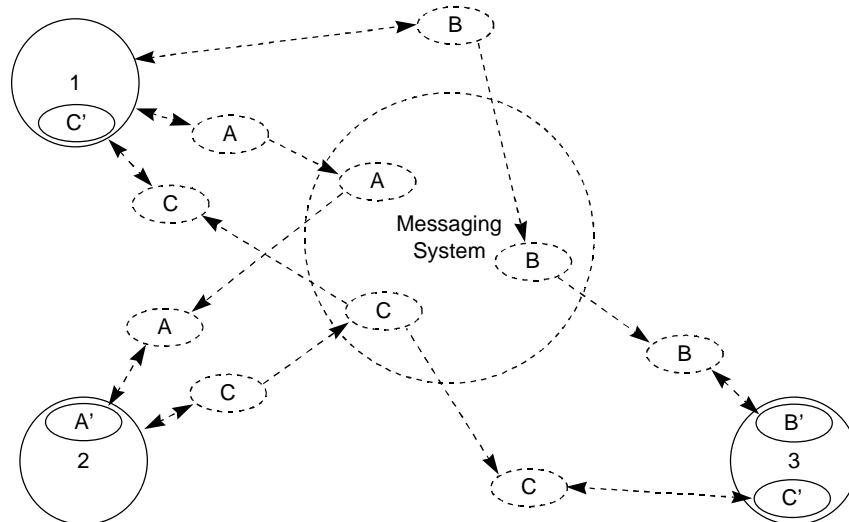


Figure 5: Off-line Messaging

Off-line messaging is also known as publish and subscribe. Here, when data that is declared to the messaging system is changed or added to, then the changes are mapped to the enterprise model, and sent to the messaging system. The messaging system looks up which other systems have subscribed to that data, and forwards the data to them. The subscribing systems map the data into the local system model and store it. The data that is mastered from another system must be locked to prevent update locally.

This approach can be useful where master data does not change rapidly, and any delays in propagation are acceptable. It overcomes the dependence of one system on another, so it is potentially more resilient. However, there is still the issue of identifying a system that is truly able to be the master for the data about some thing, relative to the needs of all other systems, which is more of a problem for reference data than for transaction data.

Both this approach and the one above are sometimes referred to as federated systems.

3.6 On-line Shared Database

One approach to overcoming the problem of sharing data between different systems is to have a shared database. Figure 6 below shows this with on-line sharing.

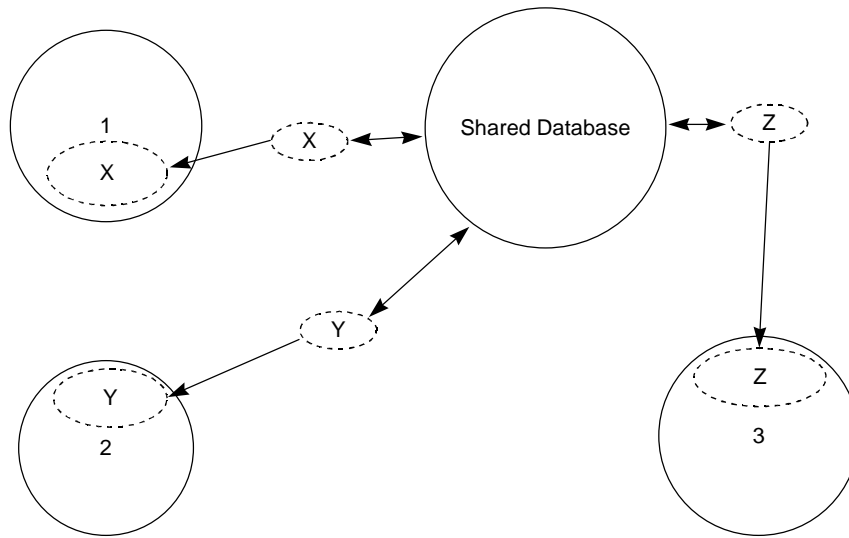


Figure 6: Shared Database

In this example, the systems that want to access the shared data get it on-line from a shared database. The shared database presents a view that matches the requirement of the application. On the request of the using system, the shared database translates the data from the enterprise model to the view of the using system, and transfers the data to the using system. Any changes made by the using system are transferred back, and mapped into the shared database.

This is similar to the on-line messaging approach, except that persistence is managed centrally. This means that for any request for data, that data has only to be mapped once, rather than twice, before it gets to the using system.

This approach can be useful for making reference data available to a number of systems, and where that reference data is relatively liable to change. It will also be favoured when none of the using systems holds a superset of the needs of other systems for some particular type of reference data.

The disadvantages of this approach are those of dependency between systems. All systems are dependent on the shared database.

3.7 Off-line Shared Database

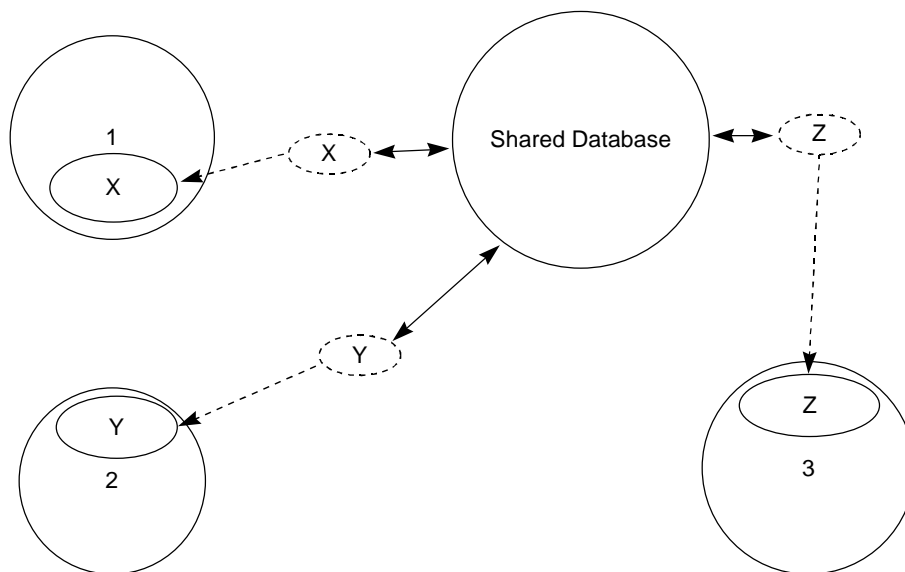


Figure 7: Loosely integrating database

As with the on-line messaging, it is possible to use a shared database in a data exchange mode. This is illustrated in Figure 7 above. Here, shared data is held in a share database, and when updates are made it is mapped into a suitable view for the using systems and sent off-line to them. There it is consolidated with their other system data and held for use locally. The shared database needs to know which systems require which data. It also need to provide mechanisms for entering and updating the data. The local copy of the data needs to be locked to prevent local update.

This approach can be useful for making reference data available to a number of systems, especially when that data does not change rapidly, and when none of the systems is able to act as a master for data of a particular type for all other systems. This approach also means that the different systems are not highly dependent on each other.

This method of integration can also be reversed, to provide a database of transaction data from many systems, so that comparisons and summaries can be made across systems.

3.8 A practical enterprise integration architecture for today

All of the approaches above are appropriate for some circumstances, and there are many occasions where it may be appropriate to combine them. However, in general shared databases lead to less complex solutions, based on readily available technology.

Figure 8 below shows a systems architecture to support enterprise integration, based on using shared databases for reference data, and management information.

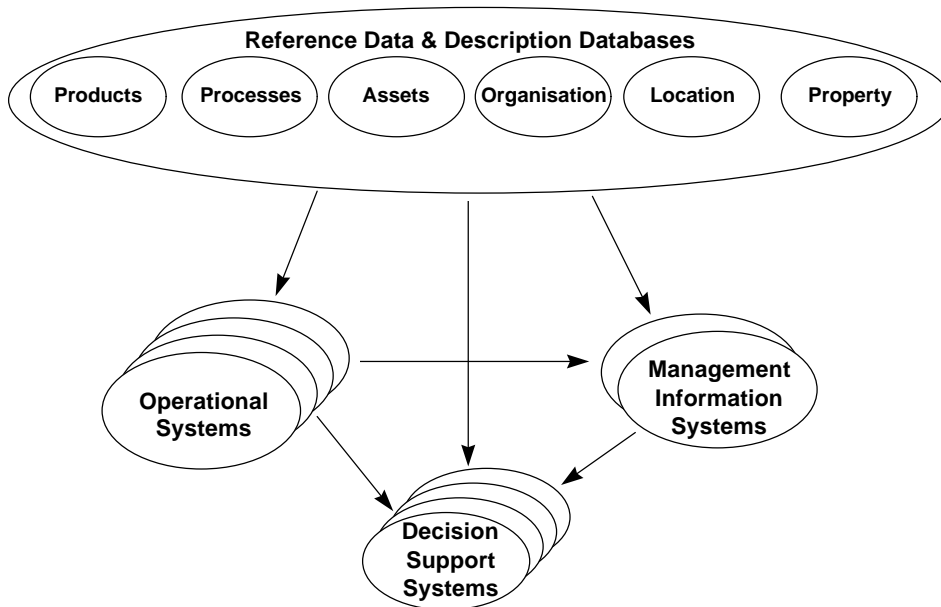


Figure 8: A Practical Systems Architecture

Consistency of reference data is managed by keeping this critical data in one place (virtually at least) from where it is downloaded to other systems that use it. Similarly, management information systems use a database that is supplied information from a number of transaction processing systems. Decision support systems (often spreadsheets) will be able to acquire consistent data from a number of sources.

4. Conclusions

1. Systems and enterprise models play different roles
2. The purpose of an enterprise model is business integration
3. Currently there is no “one size fits all” solution to systems integration, however,
4. There are a number of practical approaches that can be applied to achieving systems integration.