# Data Mappings

Richard Veryard, August 2001

Much of this document is a revised version of Chapter 7 of my 1994 book on Information Coordination. I have removed some obsolete references to CASE tools, and added a new introduction.

## Introduction

### Historical Background

Ten years ago, many people believed that one data model should be enough.

It was only a matter of time before proper data administration disciplines and procedures would spread through the software industry, before Information Architects would impose rational data structures across the modern corporation, and all corporate information systems would be brought under this control.

As database technology improved, it would no longer be necessary to have artificial technical differences between the “logical” and “physical” data models. Indeed, operational databases would soon be so large and powerful that it would be unnecessary to pipe data into separate data warehouses for decision-support purposes, and off-line archiving would be a thing of the past.

Furthermore, transition from Old to New – for example during the implementation of new software systems or ways of working – was thought of as a brief period of disruption and interruption to normal service, rather than itself being a normal and indeed permanent state of affairs.

Although many large companies had tried and failed to build and implement a single corporate data model, this was usually attributed to technical shortcomings – which greater dedication and the next wave of tools would surely overcome.

It was against this background that I wrote a book on Information Coordination, which was published in 1994. Instead of a single corporate data model, I offered a vision of a differentiated set of coordinated data models. Since I had been working within the CASE world, the book was primarily addressed to the users and suppliers of CASE tools. At that time, I still saw this community as situated at the leading edge of IT systems development.

But things had already started to change before the book came out. The book was quickly overtaken by technology – especially CORBA and other developments in open distributed processing. By 1994, I was heavily involved in research in this area – working with people who had already started thinking about the business and technological consequences of the World Wide Web and Web Services. The overall vision of the Information Coordination book was, I believed, more valid than ever – but a lot of the technical detail was now obsolete. I wanted to completely rewrite the book – to bring it into line with rapidly changing technology – but the technology was changing too fast for me to write it all down.

In many ways, despite the continued rush of new products, the situation today has somewhat stabilized. The argument for several models is much better understood, and there is growing tool support for the various mappings that are required between them.

Perhaps the time is now ripe for me to produce an entirely new book on Information Coordination. In the meantime, here's an outline of some of the mappings that may be required between models.

### Why do we have several models?

We may need to understand and document the data structure of any of all of these.

- Different stages of the development life-cycle. Conceptual, Logical and Physical. Or perhaps Requirements, Specification and Design.
- Different stages of transformation of data into knowledge. Operational data stores, data warehouses, data marts.
- Division of labour or responsibilities between parallel projects.
- Different platforms.
- Different schemas and notations. Hierarchical, Network, Relational, Object. XML, ebXML and BizTalk schemas.
- Multiple internal sources. For example, a large corporation – perhaps grown through merger and acquisition – with lots of separate legacy systems and data stores, plus various COTS packages.
- Multiple internal destinations. For example, satisfying the information needs of different business processes or user communities within the organization.
- Multiple external sources and destinations.
- Parallel versions and ways of working – in any of the above.

If we can establish data mappings from one model to another, then we can translate data – for example to perform ETL from one data store to another. It also enables us to transform stuff that depends on data structure – in other words, helping us reuse stuff across multiple ontologies.

## Links between models

### Dilemma

Either one large and unmanageable model:

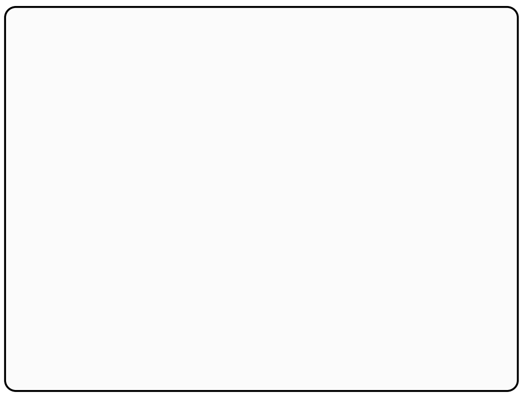

**Figure 1: Single monolithic model**

Or a family of separate models:

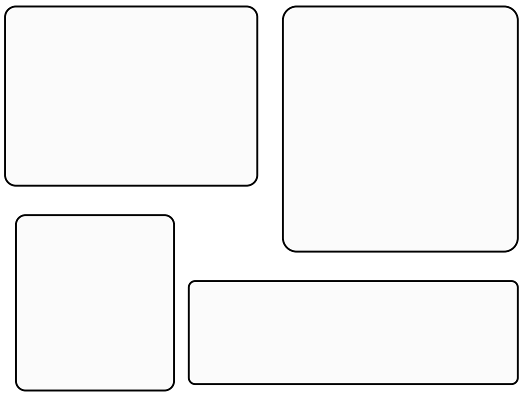

**Figure 2: Unconnected family of models**

## Solution

A family of connected models:

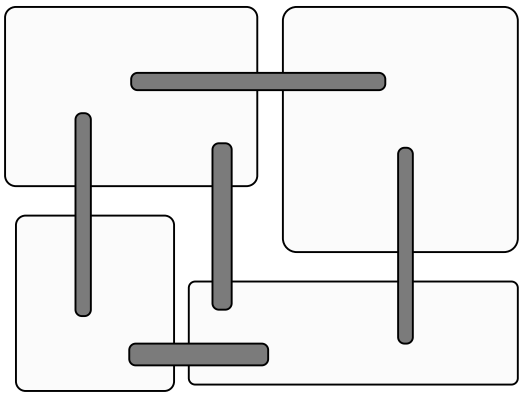

**Figure 3: Linked family of models**

As indicated in Figure 3, we shall try to restrict ourselves to horizontal and vertical links between models. Diagonal links shall be defined as combinations of horizontal and vertical. The aim shall be to connect the family properly, with as few links as possible, since each link entails coordination effort, both to define and to maintain.

## Type of link

There are three types of link we could define between models (or objects in separate models):

| | |
|---|---|
| Identical structure | This is where the structures in the two models are supposed to be (and remain) identical. |
| Equivalent structure | This is where the structures in the two models are supposed to represent exactly the same things, but for different purposes. |
| Alternative structure | This is where the structures in the two models are supposed to represent similar things in different contexts (which may be parallel or completely different purposes). |

Each link has an **extent** (sometimes known as an **aggregate object**). For example, two alternative structures may be centred around different versions of a particular entity type, but may include differences in relationships, attributes, identifiers, data integrity rules, process logic. In the object-oriented world, there may be differences in services or methods.

Two models may have several such links between them. Suppose we have two models A and B, containing aggregate objects $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_m\}$ respectively. Then we could conceivably declare that we want $a_1$, $a_2$ and $a_3$ to be identical to $b_1$, $b_2$ and $b_3$; $a_4$ is to be equivalent to $b_4$, and $a_5$ is to be an alternative to $b_5$. This example is shown in Figure 4.

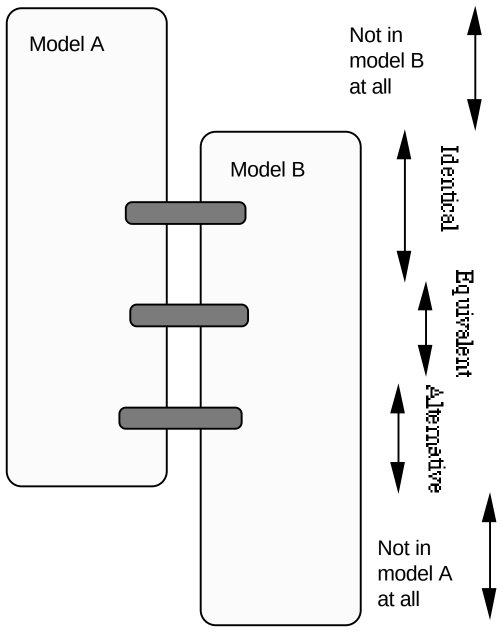

**Figure 4: Several links between two models**

Two models are **consistent** if there are no alternative structures. In other words, all structures are identical or equivalent.

## Identical structures

For structures to be identical, they must not only have the same syntactical structure (relationship cardinality etc.) but the same semantic structure (meaning).

In other words, the occurrences of the entity types should be identical.

The converse of this is that the object descriptions must be sufficiently precise as to prevent different structures masquerading as identical. (If Sandy thinks we have thousands of occurrences of PRODUCT, and Jo thinks we only have one, then they are probably not operating with the same concept of PRODUCT.)

In database design circles, it is often supposed that this is a necessary condition for integration. It turns out that this is too strong a requirement[i]. Integration can be based on weaker conditions of compatibility or equivalence.

## Equivalent structures

There are two main reasons why we may be interested in equivalent structures.

| | | |
|---|---|---|
| **1** | **Different user/application views of the same data.** | Example: the model for Accounting will be at a different level of aggregation to the model(s) for Operations. The former may have a single entity type FIXED ASSET, which acts as a supertype for several separate entity types in the latter model(s) (e.g. BUILDING, VEHICLE, MACHINE TOOL). |
| **2** | **Different development views of the same data requirements (conceptual/ logical/ physical)** | Example: the Design model for Operations divides the data into geographical partitions, whereas the Requirements model ignores this division. |

The crucial point about equivalent structures is that there is an exactly specified transformation from one to the other. (One possible way of specifying these links is through an SQL-like language, similar to the way relational views are specified. Alternatively, some diagrammatic specification may be possible.) Typical transformations are:

i Aggregating several entity types in one model to a single entity type in the other model.

ii Various forms of normalization and denormalization, such as collapsing a relationship with another entity type in one model to a simple attribute in the other model.

iii Objects that are derived in one model may become basic in the other model (because the objects they are derived from are absent from the second model).

Depending on how the link is going to be used, it may be necessary to be able to specify the transformation in both directions, or only in one direction. For example, it may only be necessary to specify a one-way transformation from logical to physical, without specifying the reverse transformation from physical to logical.

## Alternative structures

There are five main reasons why the systems developer may be interested in alternative structures:

| | | |
|---|---|---|
| 1 | Local variants in requirements | Example: European address formats versus US address formats. Different local algorithms for calculation of sales tax or value-added tax. |
| 2 | Local variants in implementation/ technology | Example: physical models tuned for different technical platforms, or for different volumes/usage patterns. |
| 3 | Replacement of old versions with new versions | Example: the business intends to replace historical costing algorithms with standard costing algorithms. This change will need to be rolled out systematically, but not necessarily simultaneously in all areas. |
| 4 | Alternative structures already exist for historical reasons | Example: an organization with 30 existing integrated systems merges with another organization with 20 existing integrated systems. Both organizations have an invoicing system, representing overlapping but different requirements. (Assume for the sake of this example that the two organizations have either used the same development platform, or two compatible platforms.) |
| 5 | Deliberate diversity | Example: in analysing business planning concepts such as COMPETITOR or OPPORTUNITY, it may be dangerous if everyone in the organization adopts an identical viewpoint. |

## Illustrations

These concepts can be applied to the model coordination needs described above. This section explores three: (i) database design (variations between conceptual, logical and physical), (ii) multiple (business) requirements, and (iii) multiple (overlapping) data stores (production models).

### Coordination through development lifecycle

In database design, technical experts define differences between the conceptual, logical and physical models to enable successful technical implementation on the chosen DBMS, and to optimize performance. The simplest way of managing this is through two models: the Requirements Model unmodified by the database designers, and the Design Model with any necessary modifications.

There are eight possible mappings from the Requirements Model to the Design Model:

| | | |
|---|---|---|
| 1 | One-one mapping (same object type) | This is the normal case. A data object in the Requirements Model corresponds exactly to a data object in the Design Model.<br><br>It is to be hoped that at least 95% of the data model can be mapped one-to-one. |
| 2 | One-one mapping (different object type) | A data object in the Design model corresponds to a single data object in the Requirements model, but with a different object type.<br><br>For example, an entity subtype in the Requirements model is promoted to an entity type in the Design Model. Or a relationship in the Requirements model is converted to a foreign key attribute in the Design Model (thus inhibiting any automatic data integrity preservation mechanism). |
| 3 | One-many mapping (same data store) | A data object in the Requirements model corresponds to more than one data object in the Design model. This is usually when an entity type is split into two or more entity types. Either vertically (selection by attribute) or horizontally (selection by occurrence) or both.<br><br>For example, to simplify processing of current transactions, historical data may be represented in separate tables. (This is an example of **horizontal partitioning**.)<br><br>For another example, to enable security software to control data security on database tables, divide entity type PERSON into confidential and non-confidential attributes. (This is an example of **vertical partitioning**.) |
| 4 | One-many mapping (different data store) | A data object in the Requirements model corresponds to data objects in more than one Design model, which will result in duplicate implementation in more than one data store.<br><br>This will be required when several separate data stores are to be implemented, since some cross-reference tables will be needed to link the data together.<br><br>Another common example is the implementation of separate (parallel) data stores for transaction processing and for adhoc enquiry. |
| 5 | One-zero mapping | A data object in the Requirements model is not included in the Design model, because it is not planned to implement this object in the current release. |

| | | |
|---|---|---|
| 6 | Zero-one mapping | A data object is introduced in the Design model that does not correspond to anything in the Requirements model. This may be a design entity type or attribute, or a redundant relationship. For example, date-stamp attributes. For another example, tables of next identifiers. |
| 7 | Many-one mapping | One data object in the Design model corresponds to more than one data object in the Requirements model.<br>For example, several trivial entity types are merged into a single entity type, to simplify access or prevent wastage of disk space. |
| 8 | Many-many mapping | Complex combinations of the above. |

## Coordination between parallel projects

Two projects are exploring and developing requirements for similar or overlapping concepts, which we can think of as multiple requirements on what might appear to be the same entity type. For example, two projects are developing systems for different departments, which have different concepts of GEOGRAPHICAL AREA.

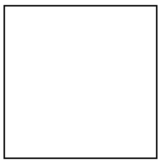

**Figure 5: Is this the same entity type?**

Although the attributes and relationships are similar, the occurrences are different. Marketing divides the USA into ten areas, while Distribution divides the USA into three areas. We may assume this is for excellent business reasons: Marketing would lose focus if it had fewer areas, and Distribution would lose efficiency if it had more areas. Therefore the two projects cannot just get together and unilaterally impose a single concept of GEOGRAPHICAL AREA on both departments.

The required approach is more subtle:

| | |
|---|---|
| 1 | Declare the two versions of GEOG AREA as <u>variant alternatives</u>. This means analysing exactly how far the variation extends. |
| 2 | Analyse the consequences of the variation. Does it matter? Would there be any benefit in reducing the extent of the variation, or eliminating it altogether? |
| Either 3a | Define new versions of each variant that will be closer together. (In other words, with smaller variation extent.) |
| or 3b | Define a consolidated version that will replace both. (This might be identical to the present version of one or other variant, but usually won't be.) |
| Either 4a | Migrate to the new version(s). |

| | |
|---|---|
| or 4b | Declare equivalence between the old version(s) and the new consolidated version. |

## Coordination between data stores

A data store is a collection of data that must be implemented together, on a single machine. In planning we define a logical data store in terms of subject areas and/or entity types. During the requirements definition phase, we refine the scopes of the logical data stores, whose contents are now defined down to the attribute level. In some projects, we may carry out distribution analysis during the detailed analysis phase, to define vertical data partitions, based on geographical or organizational divisions.

During the construction phase of the project, physical data stores are defined from the logical data stores, consisting of tables and indexes. It is usually advisable to manage the data as several physical data stores, for housekeeping and security purposes. (Metaphorically, these are firewalls, designed so that one data integrity problem doesn't bring the whole company to its knees.[1]) There may be a one-to-one mapping from logical data store to physical data store, or then again there may not.

Different data stores may be implemented on different platforms. However, there are restrictions to the distribution of a single data store across platforms.

Inevitably, there will be relationships between entity types in one data store and entity types in another data store. To implement these relationships, there will have to be some duplicate data somewhere, either a table of record identifiers in one or both of the data stores, or a cross-reference table in a third (link) data store, as shown in Figure 6.

[1] This is the proper sense of the word Firewall – despite the fact that the term is now used primarily to refer to an external security mechanism.

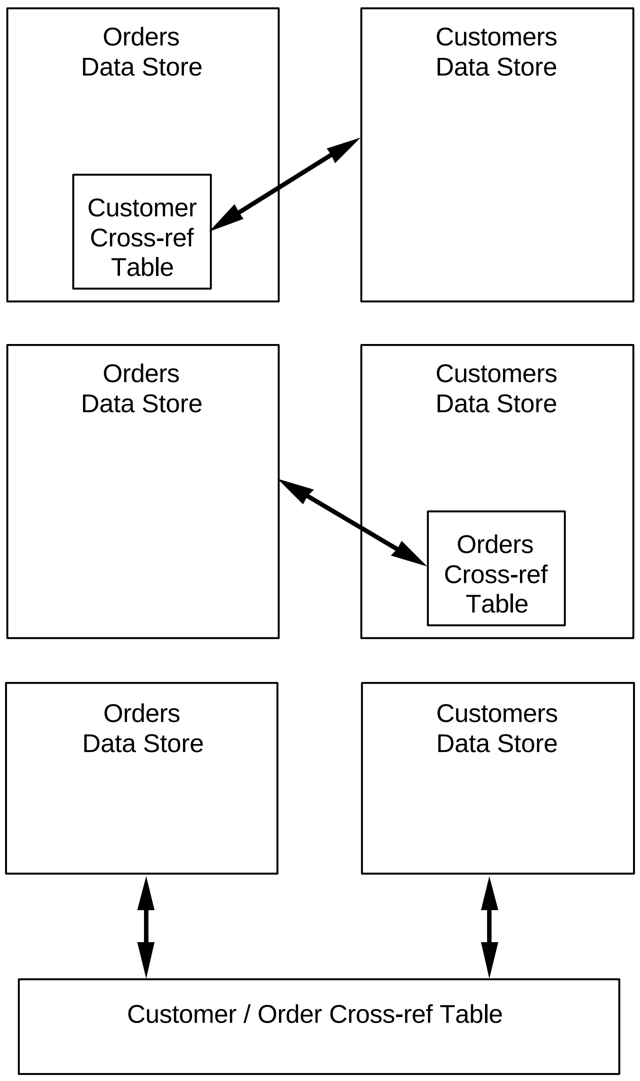

**Figure 6: Relationship between data stores - three possible structures**

In some cases, the desired sequence in which the data stores will be implemented forces the decision about the location of these duplicate data, since it is often impractical to alter the data structure of an existing data store.

Data integrity between the data stores must (at present) be maintained by hand-written code. There are several design alternatives:

1 Real-time update and integrity check

2 Real-time update, off-line integrity check and clean-up

3 Off-line update and integrity check

Obviously alternative 1 is the best, but may be impractical for performance or network reasons. In the past, batch processing has often been necessitated by such reasons. Among designers habit dies hard, and even now some systems are designed for batch operation, when real-time update may be perfectly feasible.

## Possible structure

Figure 7 shows a possible structure for the horizontal and vertical links between models.

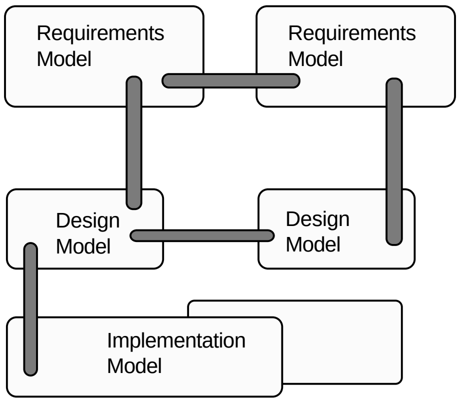

**Figure 7: Possible structure for links between models**

Vertical links must be identical or equivalent structures. Horizontal links may be alternative structures.

If we have hundreds of working models, there will be thousands of links. To reduce the number of links, we can introduce local hubs and define links via these hubs rather than directly between working models. We call these hubs **common object models**.

(One benefit of common object models to development coordination is that they allow asynchronous coordination between projects in parallel. Here's an additional benefit: reducing the complexity of the model network itself.)

## Mechanisms

The following rules need to be implemented, either in procedures or in automated tools, or a combination of the both.

| | |
|---|---|
| Identical structure | Particular objects must remain identical across the two models. One model is defined to be the master source model for these objects. Any changes to the objects must be migrated to the other model. Changes to the other model are prohibited. |

| | |
|---|---|
| Equivalent structure | The extent of the equivalence should be clearly scoped. The equivalence must be precisely specified. The custodian for the equivalence link will usually be the custodian of one of the models. The reason for the difference should be recorded. |
| Alternative structure | The extent of the alternative should be clearly scoped. The reason for the difference should be recorded. Each model will usually have a separate custodian. |

## Manageability

There are practical limits to the number and complexity of different links that can be managed between models. Therefore, there will be some advantage in converting alternative links into equivalence links, and equivalence links into identical links, wherever possible.

However, the provision of automated support for these links should increase the number and complexity that can be managed.

The main complication is probably the overlapping of extents.

## Maintaining data integrity

### Introduction

This section returns to the concept of consistency and compatibility between models. If two objects (or two object versions) are not identical, then in order for them to be compatible, there must be a logical transformation from one object/version to the other. The same transformation must then be obeyed by the respective systems.

### Linked models

The alternative to packing all the analysis into a single model is to spread it across several models. This then raises the question: how do you tie the models back together.

The usual situation is that different areas of the business need to perceive things at different levels of generalization. There is a difference in perspective between the **lumpers**, who concentrate on the similarities, and the **splitters** who concentrate on the differences. When considering assets, for examples, accountants are lumpers ("treat all assets the same"), while production engineers may be splitters ("treat a furnace differently from a numerically controlled cutting tool").

Consider a fictional oil company. Transport strategists (producing long-term plans) regard oil pipelines and oil tankers as interchangeable, while transport schedulers (concerned with day-to-day logistics) regard them as entirely different. Thus we want two models: a 'lumper' model in which pipelines and tankers are occurrences of the same entity type, and a 'splitter' model in which they are occurrences of different entity types.

In my 1992 book it is explained how to establish a compromise between the lumper and the splitter, by using entity subtypes. Thus if it is absolutely necessary (or strategically appropriate) for the transport strategists and the transport schedulers to share the same model, this can be done by defining an entity type TRANSPORT

MEDIUM with two subtypes: VEHICLE and FIXED CHANNEL. Then the strategists can work with the supertype, while the schedulers work with the subtypes.

However, every time this is done, it makes the data model more complicated. We can consolidate two or three sets of requirements in a single model, but if we try to consolidate dozens of different requirements, at different levels of generalization, the model becomes incomprehensible.

So what are the alternatives? If we have a 'lumper' model **L** and a 'splitter' model **S**, how are the two linked together? There are five possibilities:

1. **Linking by partition** - occurrences are added into the model **L** and are instantly available to model **S**.

2. **Linking by aggregation** - occurrences are added into the model **S** and are instantly available to model **L**.

3. **Linking by process (downwards)** - occurrences are added into the model **L** and are subsequently available to model **S**.

4. **Linking by process (upwards)** - occurrences are added into the model **S** and are subsequently available to model **L**.

5. **Complex linking** - some combination of the other four.

## Linking by partition

The two entity types VEHICLE and FIXED CHANNEL in **S** are defined through a partition of the entity type TRANSPORT MEDIUM in **L**. All occurrences of TRANSPORT MEDIUM in **L** (i.e. all occurrences relevant to **S**) automatically become occurrences either of VEHICLE or of FIXED CHANNEL in **S**.

In some cases, the occurrences of the entity type in the 'lumper' model **L** may be distributed among entity types in several 'splitter' models $\mathbf{S}^{i}$, and

Control of the entity type is therefore via the 'lumper' model **L**, and the entity types in **S** are derived from **L**.

## Linking by aggregation

The entity type TRANSPORT MEDIUM in **L** is defined as the aggregation of the two entity types VEHICLE and FIXED CHANNEL in **S**. All occurrences of VEHICLE and all occurrences FIXED CHANNEL in **S** automatically become occurrences of TRANSPORT MEDIUM in **L**.

In this case, control lies with the 'splitter' model **S**, and the entity type in **L** is derived from **S**.

This structure is often useful for modelling accounting and finance, where ledgers contain financial aggregations of non-financial transactions.

## Linking by process

A process is defined to bridge between **L** and **S**. There will be some event causing the release of a given occurrence from one model to the other. This is appropriate where one model needs a longer life-cycle than the other, or when some business control is required. For example, 'what if' products, subjects of research and development projects, may be of interest to the product planning and control function, but should be invisible to the product support function, and therefore absent from the product support model, not only to avoid confusion and clutter, but to prevent products being sold to customers prematurely.

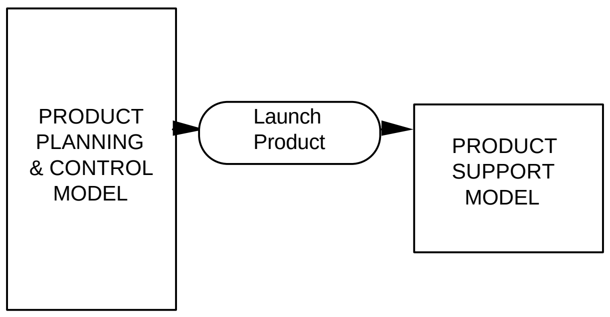

**Figure 8: One model contains more of the entity lifecycle of PRODUCT than the other**

It may be that nobody is interested in the entire lifecycle of an entity type. Each model may only contain a portion - these portions may or may not overlap, as required. Thus for most purposes, the personnel function does not want to be bothered with candidates or pensioners.

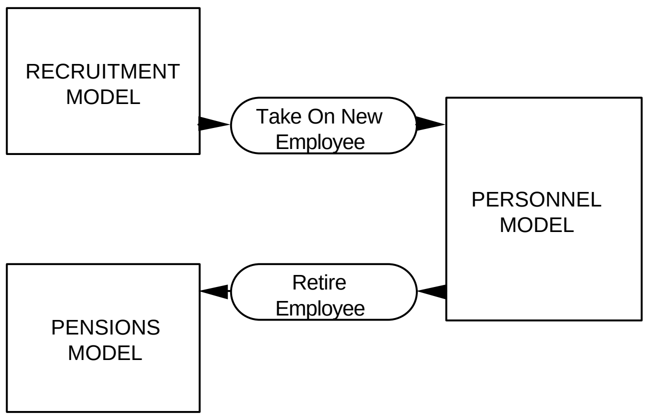

**Figure 9: Each model only contains part of the full entity lifecycle of EMPLOYEE**

## Mapping for change and transition

### Implementation & organizational learning

It is often part of the business strategy to introduce more systematic thinking about some aspect of management. Information systems (IS) development is deliberately used by senior management, to educate middle and junior managers, and to force the organization as a whole to learn new concepts and practices. This has been used to introduce such concepts as project management, quality and cost control.

An information systems project may be used, not just to support changes in the way the organization works, but to encourage these changes.

Now this implies a period of organizational learning, while the business users gradually introduce more sophisticated and accurate measures, e.g. of performance and costs. The new information systems will enable these changes, but they cannot assume the changes have already been made. It would be wrong to make the completion of these changes a precondition for the implementation of the information system.

The information system doesn't have to automate the changes themselves. The change processes may be carried out by manual adhoc procedures. But to design a good information system, that will enable the business to make the necessary or desired changes as smoothly as possible, the change processes should at least be understood, which means they must appear in the business model from which the information system is to be designed.

Stability analysis needs to recognize both user-driven change and IS-driven change.

| **User-driven change** | **IS-driven change** |
|---|---|
| Business uses new concepts | Business uses old concepts |
| Old system uses old concepts | New system introduces new concepts |
| New system catches up with business | But new system must also support old concepts during user transition |
| Users can abandon old concepts quickly | |

## Parallel information structures - options

You can always find some level of abstraction, at which the old and new concepts are equivalent. This follows from the fact that we can, after all, replace one with the other. So there is a temptation to lump the old concept and the new concept into a single entity type or structure.

But this is usually inadvisable. This is like saying "we don't need to build a bridge across this river, because if you go far enough upstream, you can wade across".

If the new concepts differ from the old concepts only in degree, then it may well make sense to have a common information structure. But if the concepts differ in kind, then you risk confusing not only users but also IS staff, if you mix them up.

Example: we already have a concept of STANDARD COST, but it is too broad, it is calculated as an average of too many dissimilar events. We want to replace it with a more focussed concept of STANDARD COST, where we pick a set of similar events and calculate an average. In this example, the concept of STANDARD COST arguably differs only in degree - we want to calculate it more precisely, and associate it with a smaller subclass of events. However, this subclass itself is probably a new concept, and requires a new process to create/ delineate it.

But for a substantial transition period, many of the occurrences of STANDARD COST will still be the old, unfocused values. One option is to leave these values outside the database, to be processed manually. (IS staff like this solution, because it simplifies their job, and they believe it gives the users an incentive to accelerate the transition to the new concepts.) Another option is to capture the unfocused values in the database, and provide some limited support for processing and using them. (User management like this solution, because it at least collects all the data in the same place, and enables the progress to the new concepts to be controlled. There usually needs to be some way of distinguishing the focussed values from the unfocused ones.)

## Model security

### Shared object management

Project team A is given a model to work with, containing objects that are currently being modified by project team B.  There is a plan to merge the two models at a future date, which means either that the versions of the objects may not be allowed to diverge, or that any divergence will have to be reconciled before the merge can be successful.  Assuming that team B has the custodianship of the object, we need **locking protocols**, which define what team A may do with the object, and **refreshing protocols**, which define how team B's work is communicated to team A.

### Locking protocols

There are three main possibilities

| | |
|---|---|
| Strict (hard) locking | Team A cannot make any changes to the object in model A, although they can look at it, and possibly link other objects to it (i.e. create references to it).  If team A requires any changes to the object, these must be carried out on its behalf by team B. |
| Lenient (soft) locking | Team A can make changes to the object in model A, but is warned beforehand, and the change is notified to team B. Team A's changes are temporary and provisional, and must be ratified or authorized by team B, before they can become permanent. |
| Zero locking | Team A can make changes to the object in model A, which must be reconciled with team B's version, by negotiation between team A and team B. |

Hard locking means that if team A want a change, not only the content but also the timing of the change has to be negotiated with team B.  Team A's productivity may be affected, if team B cannot effect the change immediately.  On the other hand, if team B must respond immediately to requests from team A, this affects team B's productivity.

Soft locking allows team A to go ahead with its own changes, as does zero locking. There is of course always a risk that the changes will not be ratified by team B, and the project manager of project A must decide whether this risk is acceptable.

### Refreshing protocols

There are three main possibilities:

| | |
|---|---|
| Instant refresh | When team B makes a change to the object in model B, this change is instantly transmitted to model A, thus becoming available to team A.  If team A has made any changes to the object, team A's version will be archived, and team B's version put in its place.  Team B can access model A, in |

| | |
|---|---|
| | order to view any links that team A has created to the object. Team B can delete the object from model B, and it will instantly and automatically disappear from model A. |
| Delayed refresh | If team B makes changes to the object in model B, these changes are transmitted from model B to model A at a time convenient to both teams. Thus at any given time, team A may be working on an out-of-date version of the object. |
| Zero refresh | Changes are not transmitted from model B to model A. Differences are reconciled when models A and B are merged. |

The 'instant refresh' protocol makes project A highly vulnerable to instability in model B. It forces team A to work with versions of objects that have not yet been p roperly analysed or tested by team B. Model A may suddenly and unexpectedly become incoherent as a result of changes from team B, making it impossible for team A to achieve its project deadlines. No project manager would accept responsibility for project A, without having either the ability to deny or delay changes from model B, or some influence over team B.

The 'delayed refresh' protocol allows team A to postpone receiving changes from model B. This protects team A's ability to meet short-term deadlines, such as important project checkpoints or implementation of subsystems. Of course, there is a risk that work done on out-of-date versions of objects will have to be revised or thrown away later. To reduce this risk, the project manager of project A will prefer, wherever possible, to accept changes from model B as soon as the changes have reached an adequate level of quality and stability. But this should be a decision for the project manager, whose judgement may be backed up by impact analysis and risk analysis.

The 'zero refresh' protocol has the advantage that the teams can (at least in theory) ignore one another during the parallel working stage. The reconciliation is carried out as a single task at the end. If the possible clashes between the two teams are trivial, this may be the most efficient option; however, this option carries the risk that the two models cannot be reconciled without substantial rework, which could then make it the most expensive option.

## Implementing the protocols

Some tools provide partial support for these locking and refreshing protocols. (It is not the purpose of this document to describe the current facilities of such tools.) The facilities of these tools will need to be supplemented by team procedures, including liaison meetings. These procedures will usually need to be administered by a central development coordination group, which will (among other things) monitor the status and consistency of all models, administer the refresh schedule (if a delayed refresh protocol is adopted), carry out impact analysis, and help resolve clashes between teams.

## Political pressures

There are many political strategies or 'games' available to the project manager, to increase the changes of project success. For example, some project managers abuse the soft locking or zero locking protocols, by deliberately building so much work on top of a provisional change to an object, that the team with nominal responsibility for the object will have no choice but to accept the change, since the overall cost of rejecting the change would be unacceptable. One of the main arguments for the hard locking protocol is that it eliminates such political strategies.

Another political strategy is to bargain between project managers, to trade favours.

To be effective, a development coordination group needs to have enough political influence and skill to play the same games. We shall return to these political issues in the final chapter.

## Administration

The management of coordination, whether done by central specialists or by projects themselves, requires a fair amount of accurate information. Who is currently doing what to which objects, where is the latest version of that, etc. Many administrators are still trying to manage this information through complex paper systems, with form-based change control. Such paper systems quickly become unmanageable, and have to be at least semi-automated. In some technical environments, the necessary functionality may be provided by IPSE-like facilities, integrated (or at least interfaced) with the encyclopedia itself. Otherwise, stand-alone systems will have to be acquired or developed, preferably accessing a shared database of project and control information.

Some situations may demand fairly sophisticated audit trails, to track exactly what changes have been made to objects within the encyclopedia by different projects, and to verify that proper management procedures have been followed.

## Summary and Final Remarks

In this document, we have looked at the kinds of link that can be established between models. The management of these links in the models, and in the systems built from the models, provides the technical basis for the coordination of both development and production.

## References and Further Reading

Some of the material in this document was published in Chapter 8 of my 1984 book, and in Chapter 7 of my 1994 book.

Richard Veryard, ***Pragmatic Data Analysis***, Blackwell Scientific Publications, 1984. Richard Veryard, ***Information Modelling: Practical Guidance***. Prentice-Hall, 1992. Richard Veryard, ***Information Coordination: The Management of Information Models, Systems and Organizations***. Prentice-Hall, 1994. All these books are now out of print.

[i] H-J Pels "Decentralized Organizations versus Integrated Information Systems" Proceedings of International Conference on Organization and Information Systems (Bled, September 1989) pp 177-190