Ch 3 Mezzanine - Information - Stories about Processes

3.1 What is Information?

There is no such *thing* as information. It is not a *thing* you can get hold of, but a mapping - a relationship - between data and knowledge. The drawing of my family tree includes relationship lines connecting us, but if I sit down with my sister, there is no stick labelled "sister" pointing from me to her - that relationship is an abstraction from our family history.

To get a handle on information, consider the case where we have a numbered list of a hundred messages. I can communicate by simply sending the number of the message and then receiver can use the number to look up the message in their copy of list. Since we need to represent a number in the range one to a hundred, we need a binary number of at least seven bits (a range 0 to 127). That is, the information content of the message is seven bits, whether the message referenced says simply "No" or contains the entire text of *War and Peace*. And then we can change the message list, so now the entry number will refer to a message in a different set of knowledge. There is no "information" that persists, only the message number and the list - the data and the knowledge.

Things look a little messier when our message sends actual text. As a communications channel, the information content of text refers to the relationship between the encoding of each letter and the knowledge to decode at the other end. Suppose we represent A by the number 1, B by 2, etc, and then send each letter using the number that represents it - the knowledge we are mapping to is simply the letters of the alphabet. With the shift key, there are about hundred different letters and characters, so we need seven bits per letter, and hence the message "Seventy-Seven" requires 13 letters or 91 bits. However, if the number "Seventy-Seven" is the index to the list of 100 messages, then the information content is still only seven bits - it's just that we are using the communications channel inefficiently.

The concept of information was developed to provide answers to hard-headed questions like "what is the capacity of this communication channel", and "how efficient is our encoding", not woolly thinking like "a way of communicating concepts".

3.2 What is Information? - Try again.

Information is the mapping between data and knowledge. In the last chapter, data types were firmly anchored to the states of a machine. However Knowledge, the other end of the mapping, will prove elusive. Will this uncertainty about knowledge make it difficult to pin down Information? Obviously yes. However, we will not be able to firm up ideas about Knowledge without a better grasp of Information. Think of this as the first turn in a spiral development cycle [3.1], in which rough ideas are sketched out, to be firmed up later.

In practice, we can start the spiral by treating Information as Data with a layer of Meaning slimed over the top. When we anchor data in a very specific context - *this red light on this fuel gauge* - then information emerges as a level above how data is represented: this binary digit (displayed as a light) maps to the message "the fuel is running low". The difficulties emerge when we move from a very narrow context to a much broader one. For example, does the fuel warning light in a car mean the same thing as one in a jet fighter? I show I understand the first light by stopping at the next motorway service station, but locating an airfield and getting permission to land is much more complex - I show I understand the second situation by panicking and trying to revive the pilot.

And yet increasingly, the "solution" is to move "information" between systems regardless of context: "the information is on the Web so use it!" However, in practice, what we see is a proliferation of tightly bound, very limited contexts: for example, a driver may need to use six different Apps to find a charging point for an electric car, even though every App ostensibly solves the same problem, sending the same data types across the same infrastructure. But the knowledge needed to decode the message differs between every app - it is private to the developers of each app, Consequently, the apps are information incompatible.

The leap between information and meaning is a big one, and it will take several smaller jumps to get there. But first we need to find a way of talking about information. We can start with our naive understanding - that words "name a concept that is out there" - and to imagine that we share a common culture and defer the problem of meaning until later. But we still need move from vaguely waving towards "information" - to work down to specific elements of information that we can anchor to data. Welcome to the world of Information Modelling.

I should start with a few warnings. Firstly, an information model is not a data model - or rather, a data model is what is left when you strip an information model of its context and all the knowledge that binds the model to that context. I spent a couple of decades working with the STEP series of information models [3.2], so I will frequently refer to STEP for examples. STEP messages are used to share complex product information such as CAD files, Electrical Circuits and Wiring, or complete Maintenance Procedures. They don't do transaction processing protocols such as Internet shopping. In fact, there is an international committee to mediate the scope of various standards so they don't duplicate work done elsewhere, except that when I last looked, W3C didn't join in.

STEP messages are packaged in Application Protocols (AP). Each AP comes as a high level process model to set the context, a set of data model diagrams for the computer programmers and a huge wad of text explaining what the data means in the context of the process - ten times as many pages as the data model.

The second warning is that there are lots of conventions for modelling data, although they all end up more or less mumbling the same things. See next section.

Thirdly, I am going to change the focus from "data representing a system state vector" to "information models telling a story about a business process". The ancient reader will be reminded of when the System Analyst was king, but the focus here is *outcome*, not *modelling methodologies* such as SSADM [3.3]. The information model is bound to a business process by the process story, thereby exhibiting the meaning of information through the behaviour of the actors in the process.

And finally, this is not "information modelling 101". I should admit that my account is likely to be flamed by some conventional data modellers - this chapter is both a sketch of information modelling and a critical hack at the conceptual weeds that have grown up round dodgy foundations. You will discover that what seems a nice clear diagram is a cheat - extra dimensions have been squashed in, layers of interpretation hidden in dark corners and large chunks of what constitutes the model are not included. Working round historical simplifications has made information modelling far too complicated, but I'm not here to fix information modelling - that is another book, and hopefully someone else's. Rather, this is a landing between stairways from data up to the knowledge floor.

3.3 What goes into an Information Model?

An Information Model is used to communicate between the end users of a (computer) system and the people who implement it (computer programmers). However, since neither group is likely to fully understand the other, it is ideally mediated by a specialist in both information modelling and in the domain in question - someone with enough knowledge of the users' domain to ask them the right questions, and enough knowledge of computing to produce a data model for the programmers. And, most importantly, they document how the data represents the user's knowledge, because the information model is also vital for maintaining the system - users change their processes asynchronously with software maintenance schedules and then mutter at the costs and delays in keeping the software up-to-date. Useful documentation saves the programmers having to start from scratch when they learn what the software needs to do. Although writing useful software documentation is an art - I have seen whole systems of documentation that were best binned before reading - but that is also another lecture.

Of course, in order to write down the information model, we must have writing conventions - have ways of condensing the information fog into distinct data droplets that have a definite usage. The term "reify" means to "make a thing of". The most radical step in writing an information model is "making a thing" of each concept - deciding the nucleation points about which the cloud of information condenses - identifying the topics for the conversations between users and programmers. These are radical abstractions of the user world, designed to fit in with the limited tools of software. It may seem strange, but the fleshy reality of "Person" and social abstraction of "Role" have often been reified in the same way, and sometimes into the same thing. A handy rule of thumb is that if you can change something about one concept independently of another, they should be reified as separate entities: if we have a *Person* with a *Person.name*="John",this tells us nothing about the *Qualification.name* = "Fire Awareness" needed to take the *Role.name* = "Fire Warden". Conversely, making *Fire_Warden* a thing means that when a person stops being a fire warden, all the other information accumulated about them must be copied to a new entity, such as managing director or cleaner.

In information modelling terms, an *Entity* is a reification of a autonomous concept: the entity *Person* reifies what we we want to say about a person. Individual details such as height or name are reified as *attributes*, while the relationship of one entity to another entity is reified as a *relationship*. This may be recorded either using a textual convention such as EXPRESS [3.4] or as a diagram, such as the EXPRESS-G diagram [3.4] in figure 3.1.

```
ENTITY Person
     name: STRING;
     height: NUMBER;
     position: Job;          /** a relationship referencing another entity
END_ENTITY;
```

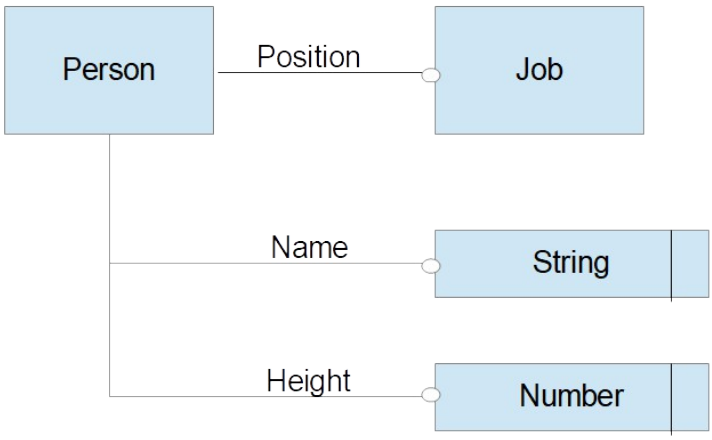


Figure 3.1 EXPRESS-G version of *Person*

In practice, things get messy: in the EXPRESS modelling language [3.4], a simple attribute is represented by a data type, but a relationship also written as an attribute but one that points to another entity. Things are a little clearer in EXPRESS-G, the graphical form of EXPRESS, where an entity is a plain box, a data type shows up as a decorated box, and a relationship as a connecting line to another entity box. You can also decorate a relationship line to show its start and end, and add a minimum of annotation, but you can't directly add attributes to a line. Hence, if the relationship has a property that needs recording, the relationship itself must instead be reified as an entity (a box). That is, the limitations of old graphics tools have ended up rewriting the modelling formalism to suit their limitations and so obscure the formalism.

When discussing modelling issues with users, they are most happy at the level of particular facts, but if asked will be given to vague conceptualisations - "on the form, the box 'name' is filled in with the *name* of a *Person".* The information model formalises these concepts into binary distinctions - *Name* is the given name for a *Real_person* on standard identity documents, and excludes nicknames, aliases and so on. The information model uses modelling conventions so that it can be unambiguously communicated, particularly with other information modellers (an argumentative bunch). Information modelling formalisms are often based on data modelling formalisms, which provide a combination of graphical and textual representations. Such formalisms include IDEF1X, EXPRESS, UML and OWL [3.5].

3.4 Modelling Conventions

There are conceptually four levels to an information model:

1. Instances - examples of particular user statements that illustrate and explain the model;
2. The Information Model, describing the user domain using a modelling formalism;
3. The Formalism itself - how the general concepts are defined and expressed;
4. Conceptual (or meta-formalism) - most useful when comparing different formalisms.

For example:

1. "John works for department X" is an instance level statement;

2. Which is modelled using the abstractions *Person*, *Department, works_for* and *name;*
3. In EXPRESS, *Person* and *Department* are entities, *works_for* is a relationship;
4. The same concepts *entity, relationship* etc. occur in other formalisms under different names.

To expand on the last point, there are four basic concepts that drive information modelling:

1. The domain of discourse - i.e. the tiny universe that is being modelled, for example, a company in which *name* is unique over all *persons* in the scope of domain
2. "Thingies" that exist without reference to anything else - variously called entities, objects, classes, etc.;
3. Relationships between *entities*;
4. What needs to be said about entities or relationships - variously attributes, properties, etc.

One might wonder whether other conceptualisation might be possible, but I have not come across one, and these ways of talking are so widespread that using an alternative conceptualisation is to forget the primary reason for modelling - communicating with other people.

Although information models use the same basic concepts - universe, entity, relationship, attribute - their translation into the modelling formalisms is rather hit-and-miss. Table 3.1 gives a comparison of the *vocabulary* of various modelling methods, although how precisely one uses the vocabulary may vary.

IDEF1X	Entity	Relationship	Attribute
EXPRESS	Entity	Relationship	Attribute
UML	Object	Relationship - in 6 classes	Attribute
OWL	Thing	Property	Property

Table 3.1 Typical Modelling Vocabularies

A recurring problem is that formalisms often reflect how a data model will be realised in software as much as they reflect the information to be represented. For example, EXPRESS represents a simple one-to-one relationship by a pointer from one entity type to another, but a many-to-many relationship can only be reified by creating an extra relationship entity (sic) with the many-to-many converted to a pair of one-to-many relationships which point out from the relationship entity to the two entities that are related: in database terms, this avoids the need to have records with loads of spare entries "in case they are needed". Consequently, "entity" is used in multiple ways to represent "thingies", relationships, and even compound and complex data types such as dates (confusing).

There is also a human factors aspect to modelling: matching the model to the attention scope of the humans who work with it. Even leaving out the details, the diagram for a small industrial information model can cover a couple of square metres. What is needed - and what few formalisms provide - is the ability to focus on different levels of detail. For example, in aircraft design, each team may design on a particular part - say flaps or ailerons on a wing - while different specialists will focus on the shape, or performing a stress analysis or planning how to manufacture it. Object Oriented Design manages this type of hierarchical structure reasonably well, but other modelling methods just demand one big piece of paper.

3.5 Information Rights - What Does an Information Model Look Like?

3.5.1 Information Rights - A Process View

The scenario: a car manufacturer has asked a couple of subcontractors to tender for the rear light cluster for a new model. To enable co-operative design, the car manufacturer will provide access to their design environment, but the subcontractors will see only the section of the body shell where the lights will go - car shape is often classified until the design is ready to manufacture. Moreover, since each subcontractors designs will be stored in the manufacturer's design environment, they must also be prevented from seeing each others.

A process is one thing after another - actions in a temporal sequence. It is typically modelled by a box for each action, with inputs, outputs, controls and resources - see Figure 3.2 [3.6] - although the interpretation of these boxes and lines can vary. Since here we are looking at the way a process informs an information model, I will be using a cut down model, ignoring resources, controls and details of timings.

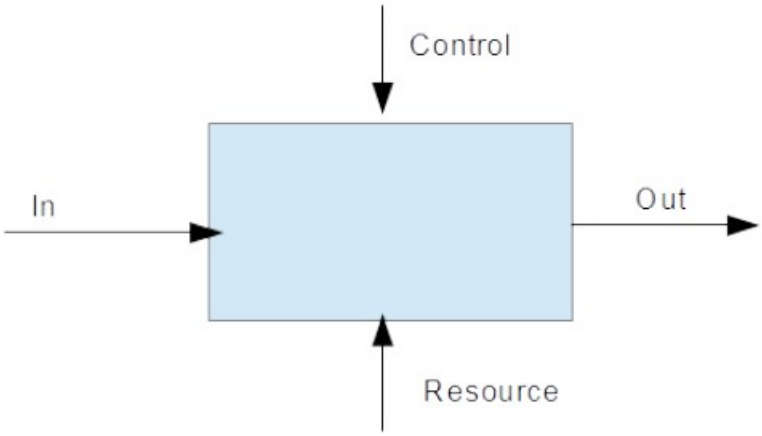
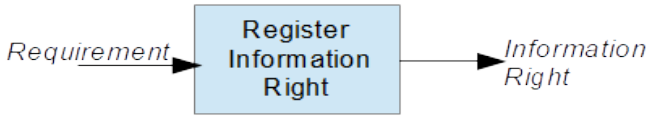


Figure 3.2 A Process Model Template



3.3a Register Information Right Process

3.3b Apply Information Right Process

3.3c Control Access Process

Figure 3.3 Information Rights Process

Information rights runs through three related processes:

- The legal team defines an *Information _right* to meet the requirements of the project;
- The contracts team sets the terms for access to the design environment;
- An administrator checks individual access requests to see if they are permitted.

The cut-down models of the processes for information rights are shown above (figure 3.3).
One could link these three processes together, but that might mislead one into thinking the process steps run one after another. For example, the legal department may generate a catalogue of *Information_rights* applicable to multiple projects, and then, some months later, the contracts team go to the catalogue to look for one that fits the current project. Moreover, as the design progresses, the contracts department will need to add new *Product_Versions* so that the subcontractors can keep up with design changes.

The assumption here is that, if an access request is made by a subcontractor, an administrator will read the *Information_right* description and grant the right if appropriate. To automate this step, one could encode the right in a policy-based access control system such as XACML/SAML [3.7]. However, last time I was involved in such a project, the "user friendly interface" needed to define the policy ended up requiring the user to learn logic - exactly the part of the process that the interface was there to simplify. Nor do many information management products provide policy-based access controls to a common standard. And goodness knows what the legal team would make of a request to write logic. So for the present, we leave granting access rights to an administrator.

One should also note that a peculiarity of process modelling is that it does not explicitly include "background knowledge" as an input. For example, it is assumed the legal team know how to frame legal rights so that they can be upheld in a court of law - that is, the law degrees of the legal department do not count as inputs. And we also assume that the legal department also know how to reframe legalese in such a way that the users - the contracts and design departments - understand them. At least the administrator's understanding reduces to a simple yes/no response to each access request.

3.5.2 The Information_Rights Information Model

The basic information model is a collection of entities and their relationships. This is an irreducibly 2D structure, and best presented graphically, often by the data type "Directed Graph" [3.8]. A directed graph has nodes shown as boxes and shows relationships as lines pointing from one node to another. Languages are essentially linear (one word after another), so - except in the simplest of cases - any textual presentation cannot describe the 2-D structure of boxes except by creating text segments corresponding to each box and finding a way of referring to each segment individually. Then, since everything in the textual presentation is text, the two dimensional structure of the model is obscure compared to a diagram.

However, older computer systems worked from text rather than diagrams, so 2-D models were developed as text. Tools then converted the text back to a diagram, however the graphics they produced were often less than helpful. Modelling diagrams, like graphic novels, come with reading conventions such as "start in the top left corner". A good information modeller will structure the diagram so that it tells a story - is easy to read. In my original version of the Information Rights model for ISO 10303 (part 1249), I made the mistake of putting the entity *Information_right* in the bottom right corner, rather than - as the key element in the story - the top left, causing a German colleague to read it in the wrong order - whence he complained it didn't make sense.

Figure 3.4 shows a much simplified Information Rights model. The graphic conventions are my own, using EXPRESS and UML as a starting point. The outer boundary denotes what ever local universe the model is to be used in, such as a company, a collaboration or a project. This is not something that is usually referenced explicitly in an information model, however integration projects are essentially about expanding the local universe, and therefore need to know how other universes impact the one the model starts from. For example, if identifiers are specified as being unique within a company, and the company merges with another, there is an obligation to check that identifiers remain unique and modify the data model (and the data) if they don't.

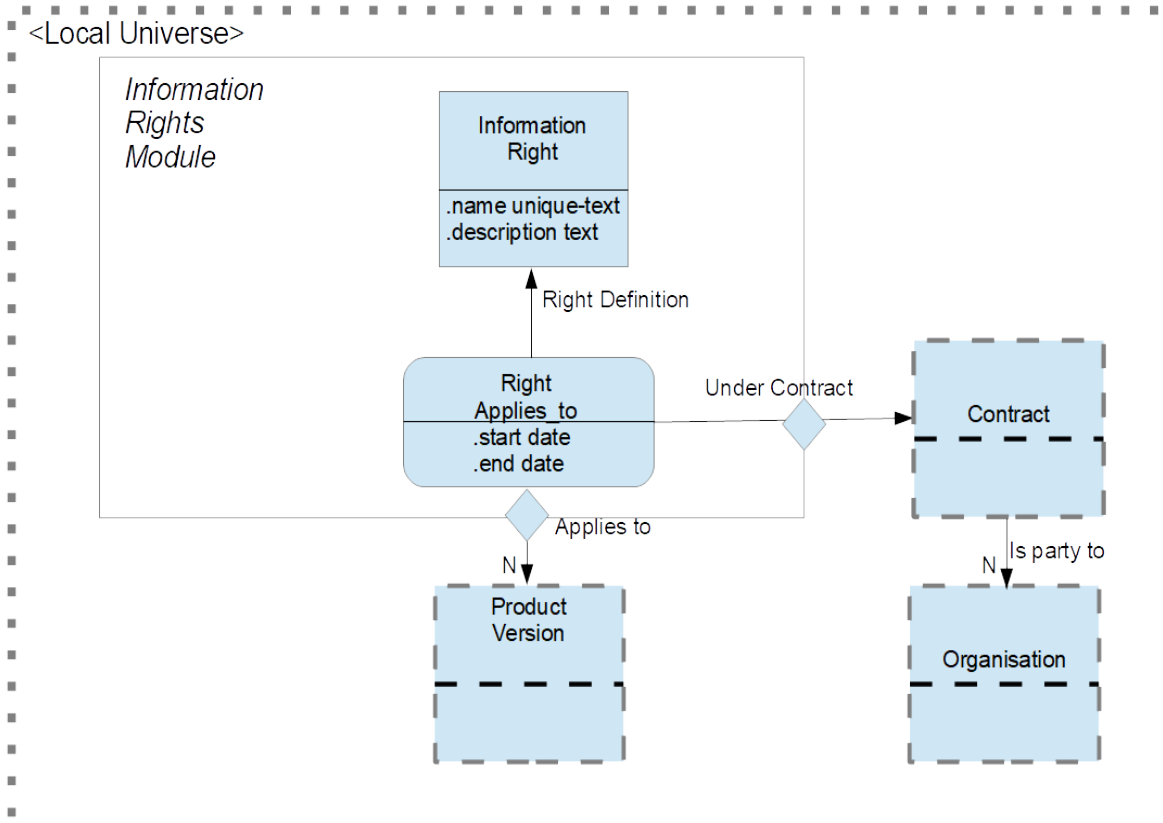


Figure 3.4 Information Rights Information Model

The Information Rights module contains an entity *Information_Right* (square) and a relationship *Right_applies_to* (rounded box). In information modelling, relationships are generally explicitly dyadic (connecting two boxes), linking the two entities which have there relationship, but are implicitly connected to the context in which the relationship applies. Often this context is the Local Universe, although in the case of an *Information_right* the context is a *Contract. Contract* is entity within the local universe but not part of the *Information_rights* module - and this context creates a triadic relationship (connecting three boxes) linking *Information_right, Contract* and *Product_version* - *Product_version* rather than *Product*, as the design changes over time.

The entity *Information_Right* is modelled with two attributes: a unique *Name*, used to reference the right, and *Description,* a general textual description of the right. A "unique" constraint implicitly references the local universe, as in "the name must be unique across all information rights used by the company". In information terms, the *name* is a human readable text, though most modellers will be more used to seeing the data type *String*. This is a legacy of the dual role many models play in

being both a data model and an information model. However, the data type *text* may have other representations than a simple string. For example, *multi-lingual_string* provides the text in two or more languages - with such a representation, the end-user is presented with text in their own language. Key point: Information is not Data, and there are often multiple choices about how represent information as data.

The relationship *Right_applies_to* asserts that the particular right applies - in the context of a particular *Contract* - to a particular *Product_version* (typically the current version of the product). The diamonds indicate that these are referenced through the module interface. It is impossible to reference an *Information_right* except via a *Right_applies_to,* and the structure of the model enforces the constraint that the right can only be assigned in the context of a *contract* - the spy cannot give someone access to company data without alerting the Contracts department.

The story of the *Information_rights* module is that an *Information_right* can be assigned to a *Product_version* in the context of a *Contract*. The ISO-10303 version of this story is rather longer - still only a single diagram, but with some twenty to thirty pages of supporting text, much of which is tied up with the technicalities of implementing information exchange interfaces.

Basics covered, now for some of the complications.

3.6 Supertypes/subtypes

My collection of oddments includes a Chaos Space Marine. Not familiar? A figure from War Hammer 40,000, a table-top war game using models to represent troops. "Ah" you say, "the town centre shop - mostly younger teenage boys". One can see subtyping as a means of knowledge compression - once you have learned the characteristics of the supertype (War Hammer 40,000) then you can apply those characteristics to the subtype (Chaos Space Marine). Conversely, you need only learn the detailed characteristics of the subtype you are interested in, ignoring all others (Space Marines, Tyranids, etc). Or you might stop at "war game" and decide you know as much as you care to know.

Subtyping in information modelling is precisely and vaguely defined. Precisely - every member of a subtype inherits the properties (attributes) of its supertype. In OO this extends to the object methods. However, if you want to use a different processing path for two subtypes, then you need a property that differentiates the subtypes according to which processing path they follow - and this is the bit of the definition that is vague. Processing path is not an explicit element of an information model, but it is key to understanding subtyping. If you are not going to do something different with a subtype, why differentiate it?

OWL takes subtyping to a logical extreme - everything is a subclass of *Thing*. The classes that are directly derived from *Thing* are a philosophical statement of how the little universe of the model is to be separated into distinct classes - a sort of ontology, if you like - the computing use of ontology is there to confuse those brought up with the philosophy definition. Entity-Relation and OO methodologies implicitly assume that entities or objects are some sort of Thingie at the highest level, and so can treat them in the same way. However, below the top level, things get messy.

The sanest way to expand the discussion of subtypes is to start with a partition, in which the set of subtypes are each distinct and every individual entity of the supertype appears in exactly one of the subtypes - like cutting up a pizza quattro stagioni so that each quarter has ingredients represent spring or summer, etc.

A partition is not the only options, and it is not necessary that every entity appears in one of the subtypes. In "not a partition", one may say no more about some entity than it is in the supertype without specifying a subtype - like putting an egg in the centre of a four seasons pizza. It is also legitimate to have overlapping subtypes, though this is where I start to doubt the sanity of the modeller. Overlapping subtypes implies that an entity in the overlap can be subject to two processing chains, and that therefore each processing chain must take count of the other. It might be better to admit that there are three processing chains: A, B and (A and B), and create a partition of three separate subtypes. And if you are using subtypes to optimise aspects like code or storage, do that at the data model level and don't confuse the end user with computing technicalities.

Conversely, similarity of processing can create unlikely subtype bedfellows. Sequential development of ISO 10303 has resulted in the Entity *Product* undergoing a semantic drift from "a thing that was sold" to a more generic concept of something that has a *product structure.* Its subtypes include *Part*, *Part_occurrence* and *Slot*. In the design world, a *Part* is a unique design; in production, "Part" is actually an individual *Part_occurrences* manufactured from a *Part* (the design); and in the support world *Slot* is a position in which a *Part_occurrence* is installed. In a Support product structure, an assembly (a *Product* with a product structure of further *Products*) may include both *Part_occurrences* and multiple *Slots* - that is, it consists of parts that are there and parts that aren't. Moreover, the actual words used by the people doing the job take their meaning from their context - they know what they are talking about - you just have to remember their context when talking to them. It's when computers get involved that the modellers need to design a uniform information model, and map the terms used locally to that model - words mean what the business decides they mean, not what they mean in the modeller's world.

In terms of modelling, type hierarchies add a third dimension to an information model, with each subtype adding a layer up the z-axis. This is not a concept that has mapped into modelling software, and subtyping is usually kludged into the entity/relation plane by using a thicker relationship line or decorating the line differently - that is, the graphical information model shows both the information relationships between entities and the subtype structure of entities.

After this comes multiple inheritance and relationship subtyping and things get too complicated for a short book.

3.7 Reference Data and Attribute Subtypes

While entity subtyping adds another dimension to the diagrams, reference data hides away in dark corners. In reference data, an attribute value is encoded by a reference data element taken from a predefined reference data library (RDL). For example, a *Measurement* may be modelled as a number plus a *Measurement.Unit_of_measure* which takes the values *kilogram, metre, second* etc. - the names of the units are the reference data. Early uses of reference data were scornfully referred to as "magic strings" written into the definition of the entity - the RDL was embedded in the model. However, by creating an external RDL, the range of values could be agreed separately to the information model, an advantage when internationally defined standards may take years to update.

At the extreme - here I am thinking of ISO 15926 [3.9] - much of the model consists of templates which are then populated at runtime with reference data, which makes the model very flexible - and very interoperable - at the expense of having to understand the model at two different levels: the template and the reference data. Taking this to ultimate extreme, one could create a model with only two entities and one relationship covering everything. Tact is needed to encode enough of the

process story in the model structure to make it comprehensible, while leaving the details which require flexibility to reference data.

Hierarchical reference adds a further complication by arranging the reference data itself into a taxonomy tree. For example, emergency response vehicles may include:

V (vehicle)	P (police service)	patrol (car)	
		riot (armoured bus)	
		prisoner (transport)	
	F (fire & rescue service)	fire (tender)	Sam 1
			Sam 2
			Elvis
		rescue (vehicle)	thunderbird 1
			thunderbird 3
	A (ambulance service)	emergency (ambulance)	

Table 3.2 Hierarchical Reference Data

If you are in the fire service going to an incident, you may see that the vehicles attending include a "V.F.fire.Sam_1" and know that that appliance comes with a 10m ladder and 5,000 litres of water, and that there is also a "V.F.rescue.thunderbird_3" which has cutting equipment - you know what to expect when you arrive. A policeman may have no idea what a "Sam 1" contains, but is reassured that it is a fire tender. The ambulance driver cares only that the Fire and Rescue Service are attending. As with a Chaos Space Marine, the "ah-ha" moment comes at different levels in the taxonomy, depending on the knowledge of the reader, and conversely, the reader needs only understand the taxonomy at a level relevant to themselves. This is a reminder that the "meaning" of an information model is found in the knowledge that the data represents.

For an integration environment, hierarchical reference data is used to focus down to the knowledge that the recipient has available. Once the upper levels of the hierarchy are agreed, the lower levels can be extended by an individual organisation to communicate across that organisation. This does not need the detailed agreement of the other actors, who can simply ignore the extensions to finer detail - at least until they update their systems with knowledge of what the extensions mean. However, the other actors must be warned of the extension so that they do not themselves extend the taxonomy tree ways that are incompatible.

A modelling language such as OWL can be used to manage reference data - this was done in ISO 10303-239 - although, with OWL, because the class name does not contain the class hierarchy, one has to access the OWL repository to work back up through the hierarchy. This also illustrates how different modelling conventions can get kludged together to make up for their individual limitations.

One of those limitations may be the absence of constructs for attribute subtyping. For example, in an Entity-relationship method, to represent a data type such as date, with separate day, month and year components, one has to create an entity *date,* with *day, month* and *year* attributes. However, *date* as a data type then gets confused a *date* used as an entity: for example,

Project.start_planned_date is a reference point for project planning, and changing it ensures that knock-on effects such resource deliveries get re-scheduled. However, *Project.start_actual_date* is an historic record, and it makes no sense to set it before the actual date or to change it after the date has occurred. That is, the two subtypes have different behaviours and should be differentiated at the information model level, however the differences in information model behaviour are not reflected in any difference in the behaviour of the data type.

The moral of this particular story is not to confuse data level arguments with information level arguments, otherwise the arguments will run and run. Data and information operate on quite different conceptual levels.

3.8 The Product Model

It would be unfair to give the impression that the complexity of information modelling is confined to a few subtle distinctions. A minimal information model for product development is shown in figure 3.5. A design for a commercial or military jet aircraft can take thousands of engineers anything from five to thirty years to complete, and, unsurprisingly, the process is book-length complex [3.10] Consequently the formal information model is shown here is just a skeleton that needs fleshing out. Or rather, by examining what else one needs to say, one can see better the limitations of information modelling.

Firstly, the model (Figure 3.5) runs down two spines - the synchronic and the diachronic. The synchronic view provides a snapshot at a particular point in the process. It shows that a *Product* - and by implication, a *System, Part* or other *Product* subtype - is built up from a number of different views, such as the systems view, the mechanical view, or the support view, depending on the product and business process. Each view is composed of a number of different properties. For example, a simple mechanical part will have a model for its shape, a weights model, a stress analysis, a manufacturing process plan, a numerical machining part program, and so on. By the end of the process, the set of properties should be complete and consistent - for example, the estimated weight should be calculated from the current shape model and material. The design as a whole will be signed of as meeting its functional requirements, as being complete and consistent, and as meeting a range of "-ilities" [3.11] such as manufacturability or recycleability, Additionally in aerospace, it will be approved as Airworthy - safe to fly.

The diachronic view shows the control structures that track the incremental development of the design - the *Product* goes through a series of *Product_versions*, each *Product_view* in the *Product_version* goes through a sequence of "formations" (I ran out of suitable names) and each *Property* through a sequence of *Property_iterations*. Inevitably, as one goes down the chain, the speed of iteration increases, so a new *Product_version* may take a month, during which time there may be daily iterations of a *Property.* And, BTW, in the aerospace sector, because the time taken to complete a new *Product_version* can be longer than it takes to manufacture an individual aircraft, the versions are also mapped to production schedule through "effectivity", the complications of which take us into the realm of "teach yourself brain damage". [3.12]

These two spines illustrate that the model represents two entirely different stories - the first that a *Product* has a single, conceptual structure made of a large number of properties, and second that product development has a complex dynamic which needs tracking as the process proceeds. Neither of these concepts are part of information modelling, they are concepts laid over the model. Moreover, a vital third story, the production schedule, is never made explicit, although effectivities

turn up wherever one sees an *Is_part_of* relationship.

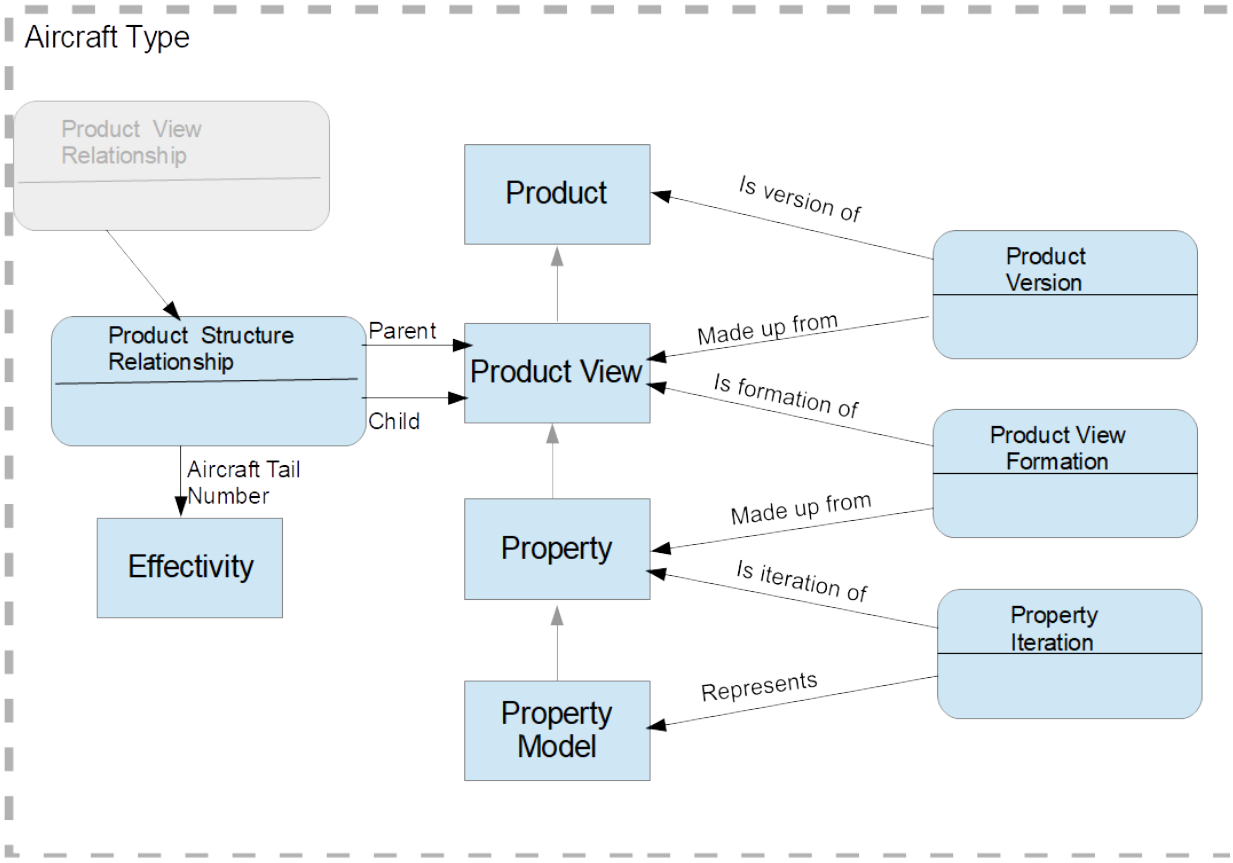


Figure 3.5 A Minimal Product Model

A second genre of stories is the way configurations boundaries cross the two spines. Configuration control is a major process in aerospace, and it is there to ensure that an aircraft is assembled from the right set of parts, and all the parts are airworthy. A configuration is set when a *Product_version,* a view formation or a *Property_iteration* is approved as fit for purpose. This approval freezes the version and all its components. Any further iterations at a lower level must not be used until they are approved to be incorporated into the next version of the level above. However, in aerospace, approval of a new configuration is enormously expensive. Consequently, if a new manufacturing tool is introduced, one must check the tool does not affect the manufactured part, but this does not affect the design the part is made from. This puts an invisible boundary through the model delimiting what needs to change if any component changes. However, this boundary is only evident in a configuration manager's view of the information model - except that information models do not come with such "views".

On a personal note, this analysis of the Product Model is based on years of work within the aircraft industry. Being embedded within the industry, I not only learned about how an aircraft was designed, but also the values that provide the rationale for the design processes. That is, the knowledge needed to create the information models was broader than simply that needed to model a particular situation.

3.9 Reversing the Mapping

So far we have looked at creating a model of information used by a process or a system - the mapping from knowledge down to data, defining the role of data as representing elements of the system or process. But if we look out from the information model in isolation, what do we get?

Start with entities. In the Information Rights model, the only entity is abstraction *Information_right*, defined by the text string *description.* However, if we were to read a legal document that described the right, one would expect legal clauses and subclauses galore; a setting out the legal context, means of redress and so on. Or if we go back to the example of a petrol tank, all we would know from the information model is that it has a state element "fuel level" - not that it is a physical structure and is part of a car. And in fact fuel level is not a feature of the petrol tank, it is the result of multiplying a sensor reading by a scale factor so that the readings run from empty to full. That is, viewing the state though a state viewing matrix conceals the detail of the mechanism - as is intended.

The entities in an information model are connected by relationships, but the term "relationship" means little more than one entity is connected to another. The nature of the relationship is usually hinted at by the relationship name: *isPartOf, isVersionOf, isRepresentedBy, isOwnerOf, ...* However, the exact meaning of a relationship is determined by the knowledge of the situation discussed, and one cannot simply rely on similarity of name when mapping between one company's model and that of another.

Information models don't exist, and they don't exist in the same way that doorways don't exist. Doorways are clearly shown on architectural plans, but they are a gap in a wall, used to go from one room to the next. Information models can run to thousands of pages, but the point is to go through them from the the kitchen where data is prepared, to the dining area where the business issues are discussed. Consequently, if all you have is an information model, it won't tell you how to run a business or construct a system from it.

It turns out that what we have in an information model is a template based on a recurring business process, where the boxes in the template show where to put particular instances of knowledge about that process. Those boxes, and the data they contain, are used to run a model - an abstraction - of the business process, making it relevant to a particular situation. With Information Rights, the process is designed to grant access to a particular set of data brought together under a *Product_version.* If someone has misused the information - say subcontract shared it with a rival prime - then the data might be used in evidence, but that same data tells us nothing about the court proceedings. That is, the information only directly makes sense in the context it is designed for, and use in any other context can only follow from an intelligent review of the knowledge that bounds and defines that context. The "tank low" is flashing on my car - very exciting when I am driving down a motorway and there are service stations in 2 and 50 miles, but no interest what so ever to anyone else on the motorway, or even me 10 minutes later when I have just filled up. An information model tells us about a particular here and now, provided we can put it in that particular context.

3.10 To Sum Up - Where Does This Take Us?

The aim of this chapter has been to develop a model of information that will allow me to make definite statements about knowledge and its relation to data. How have I done? Three out of ten?

Firstly, the model of information - entities, relationships and attributes - still looks like a data model with a moss of "information" growing on top of it. On the one hand, this makes describing the mapping between data and knowledge more straightforward, but on the other it still leaves what is "information" rather fuzzy. I have given a preliminary sketch of how "information" relates to business process in the discussion of Information Rights but have not provided a rigorous methodology to show process elements mapping to information elements. However, another dozen pages on process models would not add much to our sketch.

Secondly, information models contain a lot of complexity that a simple 2-D diagram - or 1-D text - obscures. For example, the use of subtyping adds layers to a model which need to be squashed flat in a diagram. Hierarchical reference data is entirely omitted from the diagrams, yet it can add a structural complexity similar to that of subtypes. Synchronic and diachronic perspectives are outside the model structures of mainstream modelling methods, and are added via an additional layer of interpretation. That is, the information model presents less than half the story.

With a data model, the model is itself a data type - a subtype of *directed graph* - and this means it can be mechanically translated into data structures or a database schema. The additional complexity of an information model - the hidden details of the process - is hand coded by computer programmers into the software they produce. The information models presented above make explicit some of the complexity hidden by data models. For example, I treat relationships as fundamentally different to entities and so avoid the reification of complex relationships into "things that exist independently".

The treatment above also explicitly includes the context, which allows constraints such "names must be unique" to be put into that context, and thereby create a route to merging contexts. For example, within an aircraft factory, every part has a unique number, but in a broader context, airlines operate aircraft from multiple suppliers, so aircraft parts are marked with a supplier identifier as well as the part number. And just to be clear, they are also marked with the reference organization that allocates supplier identifiers, and the names of those reference organizations are defined (as reference data) in the aircraft part marking standard [see previous chapter]. The standards of the aircraft industry are quite exacting.

Thirdly, the chapter makes the point that an information model is there to communicate between humans. The warning is that humans are very flexible in their approach to language, and will use terms in a broadly analogous way - a relationship gets reified as an entity, and so on - terms become ambiguous. Unfortunately, computers do not do ambiguity. They might be trained to be very precise in recognising known ambiguity, but will not instinctively recognise a new ambiguity.

The next step is to use this model of information model to say something less vague about knowledge - onwards and upwards to the next chapter.

Notes and References

3.1 See, for example, Wikipedia "Spiral Model" https://en.wikipedia.org/wiki/Spiral_model Accessed 15/2/23

3.2 STEP stands for "Standard for the Exchange of Product model data". It is published as ISO

10303 "Automation systems and integration — Product data representation and exchange." It is a very big, very complex standard - start with an introductory text, but even Wikipedia is quite heavy going (brown banana).

3.3 SSADM - start at Wikipedia "Structured systems analysis and design method", https://en.wikipedia.org/wiki/Structured_systems_analysis_and_design_method Accessed 29/03/24

3.4 EXPRESS is part of the STEP series of standards: ISO 10303-11:2004 Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual. Phil, who used to edit it, didn't think there would be a new edition when I spoke to him about it c. 2010.

3.5 All have entries on Wikipedia - for original (brown banana) sources:

- IDEF1X - https://www.idef.com/idef1x-data-modeling-method/ accessed 15/2/23
- EXPRESS - see 3.4 above. See also STEP Tools Inc "Information Modeling with STEP and EXPRESS" https://www.steptools.com/training/express.html Accessed 15/2/23
- UML - "Unified Modelling Language 2" http://www.uml.org/ Accessed 15/2/23
- OWL - W3C Semantic Web "Web Ontology Language (OWL) https://www.w3.org/OWL/ accessed 15/2/23

3.6 IDEF "IDEF0" https://www.idef.com/idefo-function_modeling_method/ Accessed 15/2/23

3.7 See Wikipedia for a yellow banana description or for brown banana OASIS-open "XACML SAML Profile Version 2.I0" http://docs.oasis-open.org/xacml/xacml-saml-profile/v2.0/xacml-saml-profile-v2.0.html Accessed 15/2/23

3.8 Wikipedia "Directed Graph" https://en.wikipedia.org/wiki/Directed_graph Accessed 15.2.23 Note particularly Rooted Trees and State Machines

3.9 15926.org "home" https://15926.org/home/ accessed 15/2/23

3.10 Barker S, "Aircraft as a System of Systems: A Business Process Perspective" SAE International, Warrendale 2019 ISBN-Print 978-0-7680-9402-2

3.11 Wikipedia "Non-Functional Requirements" https://en.wikipedia.org/wiki/Non-functional_requirement Accessed 15/2/23

3.12 As an aside, my colleagues used to mutter about my using complex words such as "synchronic" and "diachronic". While each term can be simply explained - e.g. "synchronic" as a snapshot - in more complex sentences, it is much easier on the brain to treat "synchronic" as a single, unexpanded concept - using bigger words allows you to think bigger thoughts. Although, conversely, "higher level" views should only be used if you know how to cash them in low level detail, otherwise there is a risk of "reality detachment". My book "Aircraft as a System of Systems" [3.10] was written in response to senior engineers complaining that "top level management" thought they took a high level view, but failed to understand why product design is different to manufacturing nuts and bolts.