Information Notions

Richard Veryard, August 2001

This document contains a brief explanation and commentary of some of the common and technical terms of data and information modelling.

Abstraction

Abstraction makes a model more powerful and broad by distancing it a little from the specific business situation we started with.

Abstraction clears away some of the specifics, and allows us to see the structure. If abstraction is taken to the extreme, no specifics are left at all. Except for skilled mathematicians, who are trained to understand highly abstract structures with no direct relationship to the real world, most people find such a model incomprehensible. Thus abstraction should be practised in moderation, leaving a sufficient amount of specifics for the model to remain meaningful.

> *"The relationship between the ease with which a model can be understood and its abstraction level does not appear to be a simple one. As one attempts to be more all-embracing on a given abstraction level, the ease of understanding ... will surely deteriorate.*
>
> *"As the abstraction level increases from the lowest possible level, the comprehensibility will initially increase because of the reduction in the number of concepts ... to assimilate. As the abstraction level is raised further and further, the sheer abstractness counterbalances the gains achievable by abstracting."1*

There are three methods of abstraction: aggregation, classification and generalization.

Aggregation

Aggregation is the putting together of different things, to form a coherent whole. Thus, instead of talking about BUILDING and STREET and TOWN and COUNTY and POSTCODE, these may all be lumped together as ADDRESS. Or instead of talking about a CPU and a keyboard and a disk drive and a monitor, these may be bundled together into COMPUTER.

People may be aggregated into teams or departments; products and services may be aggregated into compound products (thus, for example, when you buy a hifi, a one-year guarantee and repair service may be bundled in as part of the product price).

This form of abstraction is useful when decisions are made at the level of the aggregate or compound. However, sometimes what is required is the opposite of aggregation: the analysis of information down to **data atoms**. A data atom is the smallest unit of information, free of interpretation or ambiguity, that cannot be derived from any other information.

[1] William Olle et al., Information Systems Methodologies: A framework for understanding (Addison-Wesley, 1988)

Information needs are usually compound rather than atomic. In Chapter 5, we will show how complex information can be built up from data atoms, and conversely how compound information can be decomposed into data atoms.

It is often asserted that a computerized information system should capture information at the atomic level, and then provide various levels of summary and aggregation, depending on the level of interested management, or on the purpose for which the information is required. This is indeed an attractive approach, because the structure of the atomic data is likely to be more stable than the structure of the day-to-day information needs compounded from it, but it is not always practicable. Sometimes the information does not exist in atomic form, and it would be a burden to the business to create or collect it.

Consider the Post Office. The atomic entity is a single letter being posted. It is almost certainly impractical to capture data on each individual letter. However, information is required about the throughput and bottlenecks of letter handling. There are various ways of providing this information. Perhaps the letters are batched into bundles, and data are captured for each bundle. Or perhaps instead of tracking every single letter, a random sample of letters is selected and tracked in detail. Thus the simple and atomic entity type LETTER (with millions of occurrences per day) is not modelled; instead the model includes complex entity types such as LETTER BUNDLE or LETTER TRACKING SAMPLE (with far fewer occurrences per day, allowing monitoring and control processes to be carried out effectively).

The trouble with such aggregated or sampled entity types is that they are arbitrary. Whereas the entity type LETTER would provide stability to the information model, because it is fundamental to the business of the Post Office, the compound entity types are not stable. The Post Office could want to change its bundling mechanism, or its sampling mechanism, and thereby invalidate the definition of the entity type in question. Therefore a system designed on the basis of LETTER BUNDLE or LETTER TRACKING SAMPLE is less flexible, more vulnerable to changes in the business procedures, than a system designed on the basis of LETTER.

An alternative is to design a computerized information system to break the data down into data atoms. This is already done with some marketing systems, where you start with the total sales figures, and then use statistics from market research to break these figures down. Clearly such a breakdown will be an approximation, but perhaps other situations could be conceived that would be wholly accurate.

Archive

IT practitioners have a specialized meaning of archive. It's what we do to data that nobody ought to want any more, but we aren't allowed to delete it altogether, so we wipe it off the main database and stuff it onto tapes in a vault and God help anyone who wants to read it.

But archive also represents organizational or cultural memory. Derrida draws attention to the fact that the prefix ***arche*** (found in both archive and architecture) represents a starting point or founding act in two senses – where things begin from, and where things derive their authority from.

Atomic Data

See **Aggregation**

See **Grain, Granularity**

Attribute

An item of information that describes an entity. For example, the attributes of the entity type PERSON may possess the attributes: NAME, ADDRESS, HOME PHONE NUMBER, An attribute is in fact a type of information, general to the occurrences of an entity type. Each occurrence of the entity type may have a **value** for the attribute. Thus for the entity type WAREHOUSE, three different occurrences may have attribute values as shown in the three columns that follow:

attribute	values		
LOCATION	Newton	Milton	Crichton
CAPACITY (sq metres)	20,000	5,000	12,000
FRIDGE CAPACITY ?	Yes	No	Yes
TELEX NO	457246	568358	578396

Bandwidth

The quantity and complexity of interactions across an interface or business relationship. The capacity of an interface or relationship, in terms of its ability to handle quantity and complexity of interactions.

Cardinality

A rule governing the number of pairings across a relationship that a single occurrence can participate in. Thus the number of instances that appear at each end of a relationship.

In data modelling, cardinalities are usually defined as ***Zero, One or Many***. UML allows more complicated rules to be defined – allowing a range of numbers, or a set of ranges. UML calls this **multiplicity** instead of **cardinality** – and this is therefore the term used in most of the OO world.

Rumbaugh, one of the main authors of UML, has argued that it is strictly incorrect to use the word **cardinality** at all.

Characteristic Feature

See **Classification**

Class

A set of identifiable objects. A class must have a membership rule – which tells us whether an object belong to the class or not – and an identity rule – which tells us whether two members of the class are the same. See **Identity**.

For implementation purposes, a class must be finite – in other words, only allow for a finite number of members. Thus the following specifies a type, but not a class: “A string of characters, of arbitrary length.” If you wanted to implement that as a class, you’d need to find a way of cutting it down to size.

Classification

One of the ways we simplify and make sense of the world is by dividing people and things into classes. This reduces the amount of information we have to collect,

maintain and consider. If a teacher assumes that all eight-year-old boys are the same, if a recruitment officer assumes that all black women engineering graduates are the same, or if an advertising draftsman assumes that all consumers of chocolate are the same, this saves the trouble of considering each individual separately.

Classification of some sort is a necessary fact of life. We want to be able to discriminate between capable and incapable, safe and dangerous, polite and rude, even perhaps good and evil. But sometimes classification is arbitrary; and there may be as many classifications as there are interested parties.

The very word 'discrimination' is often used to denote unjust or unfavourable treatment of an identifiable group of people. But it is a fallacy to think that classification itself, or discrimination are themselves inherently undesirable. After all, unfair discrimination can only be recognized (let alone corrected) by a similar (but not necessarily equally unfair) discrimination: if a black woman engineer wishes to prove that she has been unfairly discriminated against because of her colour or sex, she must herself classify herself in this way.

Classification of people is not just a necessary evil, but is (most of the time) a useful and acceptable procedure. Consider schools, for example, where at first sight the word 'class' appears to have a rather different meaning. However, schoolchildren are divided into classes by some classification, based on age (usually), sex (often), ability (possibly), mother tongue (perhaps), or some other characteristics. (For the rich, for the physically disabled or musically talented, and for religious minorities, there may even be separate schools.) Too great a diversity of children within a class makes it impossible for the teacher to communicate effectively with the whole class. Educationalists may argue at great length exactly which characteristics should be used, and exactly how much diversity or uniformity is desirable, but few of them would expect a 16-year-old bookworm, who spoke three languages but whose English was rudimentary, to be forced to learn alongside a 9-year-old who couldn't yet read, and only spoke English.

Classification of physical objects causes much less concern. British Rail may classify its buildings into Stations, Offices, Workshops, and so on. This could result in a classifying entity type BUILDING TYPE, related to the entity type BUILDING. This would for example enable policy decisions (such as frequency of repainting) to be made once for each type of building, instead of once for each building, thus reducing the number of decisions that have to be made.

A good way to discover such classifying entity types is to examine the attributes of an entity type, and ask: why does a given occurrence of the entity type have a particular value.

For example, in a building supplies wholesaler, the entity type PRODUCT has an attribute UNITS OF MEASURE, as shown in the following table.

Product Name	Units of Measure	Product Type?
Wallpaper pattern 4711	Metres	Wallpaper
Wallpaper paste (domestic)	Litres	Paste
Wallpaper paste (industrial)	Litres	Paste
Wallpaper pattern 4712	Metres	Wallpaper
Crimson Gloss Paint	Litres	Paint
Paintbrush 6 inches	Each	Tool
Wallpaper brush (9 inches)	Each	Tool

If all patterns and textures of wallpaper have the same units of measure (i.e. metres), and all colours and consistencies of paint have the same units of measure (i.e. litres), then it may be worth introducing a second entity type (as implied by the third column in the table), and making UNITS OF MEASURE an attribute of PRODUCT TYPE instead of PRODUCT.

This kind of classification can sometimes help increase the flexibility and reduce the redundancy of a model, and is often worth considering.

Some useful thoughts on membership rules can be obtained from Wittgenstein, whose followers distinguish between **characteristic features**, which are likely to belong to an object of a given class, and **specific features**, which are common to all members of a given class.

Some more jargon has been introduced by anthropologists, which can borrowed here. **Monothetic classification** defines a class in terms of specific features. **Polythetic classification** defines a class in terms of characteristic features.

Information Scientists have usually assumed class membership can be defined monothetically, despite Wittgenstein's famous counter-example, based on his definition of the class GAME.

Whereas operational entity types (such as EMPLOYEE) can usually be defined monothetically, strategic entity types (such as COMPETITIVE THREAT) often cannot. The natural definitions of such entity types may include words like 'typically'. To avoid this, the model may fall back on definitions that make the adhoc judgement explicit, by specifying a judge, or a judging process. ("A competitive threat is anything identified by the strategic planning director as a competitive threat.") However, it is still useful to document the characteristic features.

The behaviour of an entity is usually a characteristic feature, rather than a specific feature. Thus it is usually inadequate to define an entity type merely in terms of what it does (or what its occurrences do). This can be like defining a dog as something that eats dogfood. A good definition of an entity type specifies what it is (or what its occurrences are).

A definition of an entity type in terms of what the occurrence might do (as with COMPETITOR or DANGEROUS DOG) is even more difficult to treat objectively. What the marketing department (or the dog-catcher) needs is a way of recognizing members of the entity type before they display the potential behaviour. Thus DANGEROUS DOG may have to be defined by specific characteristics, such as breed or size. COMPETITOR may have to be defined in terms of those characteristics that make an organization capable of mounting a competitive threat, rather than in terms of competing products already on the market.

Component

A growing number of CBSE experts steer away from the simple term component – speaking more precisely of component specification, component implementation, component object, and so on. The concept of component has apparently vanished – distributed somehow between several more precise and narrowly defined concepts.

M y view is that the original concept of component survives – sustained by the **articulation** between these more precise concepts.

Thus a component takes the form of an association between a service and a software device. The device implements the service, the service specifies the device. The component is neither the service alone, nor the device alone – it is the relationship that is established between them.

A component is therefore not a fixed thing. It is this declared (and possibly dynamic) linkage between a parcel of capability and a parcel of service.

The focus is not on the identity of the component, but on the act of componenting: the (always provisional) declaration that a given lump of business capability, or a given lump of software, suitably wrapped, shall match the demands of a given service – until something better or cheaper comes along.

For many purposes, however, it is useful to fall in with conventional idiom: to talk about components as if they were objects. But beware – there are two rival conventions to contend with.

Some people have an inside-out definition – a component is essentially a lump of software with certain properties. And some people have an outside-in definition – a component is essentially a set of services accessed through a specified interface whose implementation satisfies certain properties.

Business components also suffer from the same ambiguity, although this is not formally developed to the same degree as in the software industry. Some people will use an inside-out definition, resting on some notion of capability, while others will use an outside-in definition, resting on some notion of service.

Obviously if you show the same configuration to these people and ask how many components can you see, how much usage or reuse has been achieved, you will get quite different answers.

These different notions of component can only be reconciled, not at an abstract theoretical level, but through practical engagement with the business and technological drivers of a specific project or situation.

The component relationship is a many-to-many one. The device implements the service, the service specifies the device. One service may be implemented several different ways, by different devices. One device may satisfy many different specifications, describing different services, accessed via one or many interfaces.

In practice, components often fall short of this ideal definition. It may be more accurate to say that the device claims to implement the service, while the service tries to specify the device.

Conceptual Model

A model supposed to represent the business requirements, or the business domain.

Countable

The term ‘countable’ has a practical sense, namely the practical feasibility of counting. (Countability is a consequence of identifiability - if you want to know how many entities of a particular type you have got, you have to be able to avoid counting one entity twice.)

This is distinct from the mathematical sense of a (possibly infinite) mapping to the natural numbers.

CRUD

Create/Read/Update/Delete. See **Data**.

Data

Data means: that which is given (to an organization or community) (from the past).

Among the data there will be records of recent transactions and decisions, results of surveys and analyses, mixed up with a lot of much older stuff. But in order for an organization to assimilate these various data, the data must themselves be organized. And for learning to take place, data must be reorganized.

And this is where we slip into infinite loops. The organization that organizes and reorganizes its own data, its own memory, its own archive, is thereby organizing and reorganizing itself.

The traditional IT view of data is of something that can be Created, Read, Updated and Deleted - sometimes known as CRUD. This appears to be true of "physical" data storage, where the binary patterns on a magnetic disk can be rearranged or erased (although techniques exist for reading supposedly erased data - see Palimpsest). But from a "logical" business viewpoint there are only two meaningful operations: Read and Write.

Data Dictionary

See **Repository.**

Data Warehouse

A data warehouse can be described from the outside as a very large component offering a range of data services, through a number of well-defined interfaces. Internally, the data warehouse may be implemented through multiple heterogeneous mechanisms, and may even be distributed across multiple platforms / locations, which suggests a component-based internal architecture, but this complexity can be hidden from data users.

This approach separates two aspects of Data Warehousing:

- how you specify and introduce (gradually, or all at once) a set of data services that link to some notion of business process and business value
- how you support these data services with an (evolving) assembly of new pieces and legacy pieces

Designation

A term introduced by Michael Jackson in an attempt to clarify what models mean.[2]

Briefly, a designation is a link between a sign (such as "customer") and what the sign designates or signifies. Jackson considers that what is essentially signified by "customer" is the recognition rule – how you know that something is a customer.

Drill-Down

Progressive decomposition or decapsulation – getting more and more detail and complexity on demand.

Encyclopaedia

See **Repository.**

Entity

An entity is any object of interest within the area being modelled, about which information may be collected, manipulated or stored. Entities can be people, material things, events, locations, or more abstract concepts and groupings.

Entities are classified into types: an **entity type** is a class of similar entities. For example, in the Payroll subject area, the relevant entity types could perhaps include EMPLOYEE, SALARY PAYMENT, TAX CATEGORY and EXPENSE CLAIM.

A particular employee – Jacob Zlàddyr, for example – is said to be an **occurrence** of the entity type EMPLOYEE.

When talking informally and imprecisely, people (including myself) often use the term **entity** to mean either **entity type** or **entity occurrence**. However, it is important to keep a clear distinction between these two notions in formal specifications.

Entity–Relationship (ER) diagram

A box-and-line notation for data and information models. The boxes represent entity types, and the lines between the boxes represent binary relationships between entity types.

Epistemology

A fancy word for what we know and how we know it.

Fetish

The tendency to describe a property as an attribute of a single entity, when it should properly be described as a relationship between two entities, is known as **fetishism**. (Marx identified economic forms of fetishism; Freud identified psychological forms; both follow the same logical structure.)

---

[2] Michael Jackson, Software Requirements and Specifications – A Lexicon of Practice, Principles and Prejudices. Addison-Wesley, 1995.

A good example of this is where an attribute represents a subjective assessment of the entity type. For example, in the EMPLOYEE entity type, there might be an attribute called PROMOTION PROSPECTS. This is potentially misleading, because it hides the source of the assessment. Or in the PROJECT PROPOSAL entity type, there might be an attribute called ESTIMATED BENEFIT. This is also potentially misleading, because it hides the source of the estimate.

Foreign Key

A mechanism for storing relationships between entities, whereby the primary key of one entity is stored as an attribute (the so-called Foreign Key attribute) in the other entity.

Generalization

Generalization is the putting together of similar things, by selectively ignoring their differences. For example, photocopiers are not the same as computers, but a model might usefully lump them together as OFFICE EQUIPMENT ITEM.

Generalization is a useful way of reducing the number of entity types in a model. Generalization is unavoidable in building an information model, since without any generalization at all, each entity type would only have one occurrence.

The key question is not whether to generalize at all, but how much to generalize, and where to stop generalizing.

Grain, Granularity

Grain is a term that originates in photography. It refers to the degree of detail and precision contained in an image - the pixels or dots per inch - or preserved and communicated by a given medium, such as a film or screen.

An image is **grainy** if the imprecision is visible - in other words, even if you can't see the individual dots, you can see that the image is composed of dots.

In information management, granularity refers to the degree of detail or precision contained in data.

Multiplication	Where lots of measurements are taken, granularity refers to the intervals (in space or time) between the measurements.
Division	Where entities are being sorted into categories, granularity refers to the choice between a large number of narrow categories or a smaller number of broad categories. Where a group (such as a community or market) is being divided into subgroups, granularity refers to the number of subgroups. Where a space (such as a network of requirements or solutions or activities) is being carved into manageable chunks, granularity refers to the size of the chunks. (If a clustering approach is used, then the desired granularity will typically be one of the input parameters of the clustering algorithm.)

In modelling, granularity refers to the degree of detail and precision contained in a model.

In some domains, there may be a maximum granularity - in other words, perfect precision. Data with maximum granularity is known as **atomic data**.

When you are modelling something from a single perspective, granularity is often not very important. A single data item is modelled as a single attribute, with a defined granularity.

Granularity becomes an important issue for data modelling when you are trying to map or merge information across multiple systems or data stores – because the likelihood is that the granularity doesn't match. It is an issue for the flexibility of the data model and artefacts designed from it.

Granularity is also a problem with distributed systems, especially where web services are involved, since it may affect the number of service calls across a network, perhaps by an order of magnitude. It may also affect the burstiness of the distribution of service.

And when you are trying to merge data from several sources into a single data warehouse, there are significant technical performance implications of the granularity decision. Some data warehouse experts recommend storing everything into the data warehouse as atomic data – on the grounds that the atomic level is the most stable level, and also represents the highest common factor – but this approach is problematic in some domains. In any case, it places a great burden on the conceptual data modelling phase, to ensure that the atomic level has been correctly identified.

Simplistic data modelling assumes that there is a clear distinction between atomic data and derived (molecular) data – but it doesn't work out as clearly as this in practice, and this issue may have sweeping implications for system architecture and design.

Homonym

A word that is used for two or more different things, in different contexts. For example, in a purchasing application, ORDER means PURCHASE ORDER; while in a sales application, ORDER means SALES ORDER.

Identifier

A mechanism within a data model for establishing identity.

Entity occurrences are identified using keys. A simple key is a single attribute that uniquely identifies the occurrence. A compound key is a combination of identifying attributes and relationships that jointly identify the occurrence.

Most database management systems (DBMS) require that one unique and unchangeable identifier is selected as the primary key. Among other things, this key is used internally by the DBMS for maintaining data structure and integrity

Identity

Identity means - how do we recognize or refer to something as the same again. The ancients knew two stars, the morning star and the evening star - but these turned out to be different manifestations of the same planet (Venus). The customer who has just bought a sack of potatoes happens to be the same person who bought a chip pan from the hardware store next door (which we might discover if we had access to his credit card information).

For many purposes, two things are the same if we cannot tell them apart. Identity amounts to a lack of difference. And we can define information as "a difference that makes a difference".

In conceptual data modelling, each entity type has a requirement for identity. This entails the ability to recognize an entity as **the same again**. In customer relationship management, for example, considerable effort is devoted to establishing a stable identity for customers – sometimes against the wishes or interests of the customers themselves. As a private customer, I don't necessarily want a supermarket to record and analyse details of my purchases, and I may only be persuaded to grant access to my identity in return for a discount scheme implemented using a so-called loyalty card. As a car driver, I may want to obtain and compare several different quotes for car insurance – and I specifically don't want the insurance industry to recognize me as the same again. Meanwhile, the insurance industry very much does want to recognize me as the same again, because it sees the tactic of hidden identity as a cover for fraud.

Questions of identity are also critical for business performance measurement. For example, if we want to measure the proportion of sales enquiries that are converted into sales, we have know whether the phone call from Mrs Smith counts as a follow-up to the phone call from Mr Smith, or whether it is an entirely separate enquiry. In other words, what is the business identifier for the entity type SALES ENQUIRY.

Identity is therefore a very important business issue, and a key element of the requirements for a business information system.

Information

When people are asked to provide a definition of "Information", they often spout a piece of communication theory by Shannon and Weaver that few people properly understand. Or they say that **information equals data plus meaning**. Some people even use both definitions in turn – although it's not obvious how they can be integrated.

In practice, though, people use the term "Information" in a way that barely resembles the definitions they cite.

Implicit Definition	Implications
“A by-product of a business transaction.”	Each insurance policy has a different “customer”. Each “customer” has a separate name and address. These “customers” may coincide - one real person may appear many times in our files. Difficult to consolidate data across products to produce such a unified view. Some business strategies demand a “unified” view of “customer”. Therefore this view of information is inadequate - typically inhibits business strategy.
“A fact about a real-world object.”	“Customers” in our database represent “real” people in the “real world”. Since there is only one “real” person, there should be only one database record. Similarly, each database record should represent a single entity or object in the “real world”. Our knowledge of the real world is typically incomplete and imperfect. Customer identity may not be straightforward. ° married couples ° small businessman ° proxies & agents
“A difference that makes a difference.” (Bateson)	Databases and systems should support any distinctions that business strategy demands. Databases and systems should support any connections that business strategy demands. Business rules and regulations demand correctness and consistency.

The equation **information equals data plus meaning** remains important, but it is not universally true, and needs to be demonstrated for particular circumstances.

If **meaning** is the result of an interpretation, then the process of interpretation needs to be visible. If we accept Wittgenstein's slogan: **meaning is use**, then we often want to think of information as data in the context of some human (or at least conscious, intentional) **use/purpose**.

Inheritance

See **Subtype**.

Instance

A member of a **type** or **class**. Also called **occurrence**.

Interface

It is a common pattern of technical materials, to provide a definition of a notion taken from a well-known dictionary, as if a technically adequate definition could be produced by refinement from the everyday usage of the word. If you go to the dictionary expecting to find a clear and simple definition of a complex notion, you

will usually be disappointed. But if you go prepared to find several diverse definitions, you will usually be rewarded.

The ***Oxford English Dictionary*** offers three different notions of interface. It is not obvious at first reading whether these three notions can be combined into a single complex notion, or whether it would even make sense to try.

- An interface may be a surface or boundary between two portions of matter – a spatial separation.
- An interface may be a meeting point between two systems – a bundle of interactions, liaison or dialogue.
- An interface may be a device connecting two or more other devices.

These three definitions represent three different perspectives on interfaces. Each of these perspectives is useful.

Interference

A complex phenomenon that occurs when you try to put two or more items of information together. This is one of the reasons why information doesn't obey simple arithmetic. *see also Management Interference*

Like many other things, LIGHT has changed in our understanding, from being particles to being waves to being something else.

This 'progress' can be regarded as an increase in complexity or an increase in simplicity - or both.

Now perhaps it is the turn of INFORMATION to be rethought.

Recall the surprise when, in high-school physics, you put two pinholes in a card and found out that this didn't always mean you got twice as much light through.

Information also suffers interference patterns, which have not (as far as I am aware) been properly studied.

If I buy two newspapers, does that mean I get twice as much information? Of course not. And if an organization receives two different messages on the same subject from different sources, they may sometimes reinforce one another, sometimes cancel one another out, among other possibilities.

Traditional notions of information (as particles of meaning) fail to account for these and other important phenomena.

Involuted Relationships

A relationship associating occurrences of a single entity type with other occurrences of the same entity type. Appears as a loop or dog's ear on the ER diagram.

An involuted relationship can be read iteratively or recursively. Thus if you have the ***manager of*** relationship linking employees, this implies a tree structure that can be read upwards or downwards – to retrieve the ***manager of manager of*** and ***manager of manager of manager of*** and so on. Reading the ***manager of*** relationship downwards, we get the set of people managed by a given employee – and then the set of people managed by them, and so on. The term **recursive closure** denotes the set of occurrences that can be reached by reading the involuted relationship any number of times.

For this reason, involuted relationships are sometimes incorrectly called recursive relationships. But recursion strictly refers to a style of processing, not to a data structure that may support such processing. In any case, recursive processing can be supported by more complex data structures, involving more than one entity type, such as Bill of Materials.

Most involuted relationships, have a no-return rule – in other words, you can never get back to the occurrence you started with. You cannot be your own ancestor. However, this no-return rule is not always valid. Consider the relationship ***doctor of***. There is no reason to exclude the following loop: Dr Smith has Dr Patel as a patient, Dr Patel has Dr Chan as a patient, and Dr Chan has Dr Smith as a patient. Indeed, if we assume that everyone – including doctors – needs to be registered with a doctor – then there must be at least one return loop somewhere.

Key

See **Identifier**

Leak, Leakage

The flow of information across controlled and supposedly secure boundaries - usually from protected to unprotected, or from private to public.

Logical Model

A model supposed to represent the externally accessible structure of an interface or device.

Message

A package of data passed from one application to another, usually asynchronously.

Some communication technologies describe themselves message-oriented – in other words, they specialize in passing messages between applications.

Meta Data

Metadata are data about data – or information about information – describing the structure, physical distribution or logical ownership, and other such characteristics.

Meta Model

Strictly speaking, a metamodel is a model that expresses the syntax or semantics of a modelling language. Provides a formal structural description of a modelling tool or repository.

In normal parlance, the term metamodel is also used to refer to a structured collection of metadata.

Model

A model is a representation of the structure of information within an organization. It may represent the present structure of business and computer systems – actual or

imagined – or a desired future structure. A description of an area as a set of constructs, which are named, defined and interrelated in a standard way.

A model has a **scope**, whose exact boundaries are often difficult to establish. It also has a **perspective** and **purpose**.

Modelling

The creation of a model – often called **analysis**. This activity is often seen as a process of discovery of pre-existing facts, but it is more useful to see it as a process of negotiation – developing a consensus between different stakeholders in the area. It is therefore important to be aware of the **perspective** of the model – i.e. which stakeholders are included, in which roles.

Negation

A complex phenomenon that occurs when you try to subtract (or erase) information. This is another one of the reasons why information doesn't obey simple arithmetic.

Stubborn information - refuses to be erased, resists eradication. (Try to get rid of a bad credit report, a hostile press statement, or even a completely groundless rumour.)

Information goes underground - it's still there somewhere, even though it's not visible any more - and may come back to haunt you. (Something may seem to be forgotten, but it suddenly pops up again - and always at the most inconvenient time.)

Negative Pattern

In a design inspection, the reviewers are looking for opportunities to improve an artefact – such as a model. This is essentially a pattern-matching exercise, in which each reviewer searches for familiar ways in which the artefact may be flawed – yielding error, inefficiency, inflexibility or some other negative quality. These negative patterns can be characterized not only by a familiar structure, but by a common source or motivation – the designer was trying to do something else, but it turned out like this. These are sometimes also called AntiPatterns or Bad Smells.

Normalization

A procedure invented by Ed Codd, supposed to remove various forms of inefficiency, inflexibility and redundancy from a relational data model. After normalization, the model should be in some normal form – typically third normal form – where all the data in each row is dependent on “The Key, The Whole Key and Nothing But the Key, so help me Codd.”

Notation

A formal style in which information models may be presented. Popular notations for data models include ER models and IDEF1X models. The class model notation in UML is roughly equivalent.

Although some notations are more expressive or elegant than others, the choice of notation is largely a matter of taste.

Object

Leibniz called them monads, and thought of them as two-storey houses. Downstairs is the public area, where you entertain guests. Upstairs is the private area, reserved for sleep, sex, and other spiritual matters.

The OO view of object is remarkably similar. They have a public reception area, known as the interface, and a private area, known as the internals. The world consists of interactions between these objects.

Occurrence

A member of a **type** or **class**. Also called **instance**.

Ontology

A fancy word for what there is – what objects exist. A theory – either explicit or implicit – about what objects exist.

The term has become fashionable of late, because of the growing recognition that different people, different organizations possess different ontologies. Effective communication therefore requires **either** establishing shared ontologies **or** establishing reliable translations between different ontologies.

Establishing the ontology of a strange tribe – such as "users" or "customers" – is fraught with error, misunderstanding and misinterpretation. The American philosopher WVO Quine argued that we can never be entirely sure we've fully grasped someone else's ontology – and that translation can therefore never be proved correct. However, with considerable effort, we can often get close enough for most practical purposes.

Palimpsest

A memory device which retains traces of previous data or versions. For example, magnetic media that allow erased data to be restored (using special techniques), humans that retain unconscious memories or habits from the past, organizations that preserve traces of previous structures or processes. Detectives and archaeologists can reconstruct the past, can follow inexpertly doctored audit trails, can deconstruct legacy.

Partition

See **Subtype**.

Pattern

A popular misunderstanding about patterns is that they are merely abstract components – prefabricated chunks of analysis or design that can be instantly assembled into a solution – the engineering equivalent of convenience foods.

Anyone who reads Christopher Alexander's original work on patterns cannot fail to sense an entirely different passion and purpose for patterns – the need for quality in the finished product – a quality that Alexander calls The Quality Without a Name.

On this view, patterns are not intended to make the engineering process faster or more efficient – or even more reliable. The purpose of patterns is to impart **character** to an artefact. Pattern languages are intended to help engineers communicate an **awareness** of the character and quality of design – with one another and with their clients.

Patterns are often described as solutions in context. This description can be traced back to Christopher Alexander, sometimes seen as the father of patterns, who is very keen to emphasize the grounded, context-dependent nature of true patterns.

Within the software engineering world, in contrast, there has been a strong emphasis on reuse. In order to reuse something, it has to be taken away from its original context – at least to some extent – and generalized so that it can be plugged into a range of contexts. This means that a pattern has to be partially abstracted from its context.

Good pattern work has always maintained tension between two opposite positions – the immanent and the transcendental – between solutions that are grounded in the specifics of a given situation, and solutions that are timeless and universal. Alexander expresses this tension by insisting that you can use a pattern a million times over, without ever doing it the same way twice. In a manner of speaking, this is repetition without repetition.

A pattern has the structure of a judgement – it involves an **appreciation** of a situation, leading to an intelligent response or **action**. This leads me to define a pattern as a judgement partially abstracted from its context.

See also **Negative Pattern**.

Persistence

The continued existence of some information after an activity has finished.

Perspective

The perspective of a model indicates one or more roles whose views are represented or incorporated in the model.

Physical Model

A model supposed to represent the internal structure of an interface or device. There are various reasons why this may differ from the logical model

- Quality of service, including performance and security requirements.
- Current technical features of the platform.
- Legacy – in other words, traces of past complications.

Primary Key

See **Identifier**

Punctuation

Punctuation refers to the way that a complex communication process – perhaps a complex negotiation or exchange of information – is broken down into discrete messages or speech acts.

Where feedback loops or learning loops are involved, punctuation also refers to the start point – where to break into the loop.

Punctuation is often subjective or arbitrary. Different analysts may punctuate a communication process in radically different ways. This critically affects the perceived structure of the situation. Is A helping B, or is B coaching A? Is C commanding D, or is D controlling C? Who is controlling whom, who is proactive and who reactive?

Recursion

See **Involuted Relationship**

Redundancy

In the context of information modelling, redundancy means that a fact is represented in the model twice. Since facts are represented by objects, or combinations of objects, this is equivalent to saying that one object in the model can be derived from other objects in the model.[3]

There are two reasons for removing redundancy: to make the model simpler (and thus easier to understand and use), and to make the ensuing system more efficient. During analysis, we should only remove redundant objects for the first of these two reasons, since the second reason is a matter for systems design, but we need to identify all aspects of redundancy at this stage.

Storing a fact more than once opens the door to inconsistency, if the two versions of the same fact are incompatible. Thus non-redundancy makes it easier to ensure consistency.

However, some situations demand a higher level of control, where consistency needs to be actively checked, rather than merely automatically ensured. Two or more versions of a fact are deliberately captured, so that they may be compared, and discrepancies highlighted. The classic example of this is double-entry book-keeping, which indicates its redundancy by its very name.

Reification

It would be pleasant to imagine that somewhere in the world – perhaps in the East, wherever that is – people are more focused on relationships than things. In the West, we seem to be obsessed by things.

Materialism is not just a matter of wanting to possess things, although that's certainly part of it. It's a matter of perceiving the world as if it were composed of things. Children are taught this from an early age: most of the available books for toddlers have one word on each page, and the word is a noun: ball, bear, banana.

---

[3] Devotees of the relational model often use a different definition of redundancy, tied to the specifics of nth normal form. Note that some intelligent translation of such concepts is required between the relational model and the entity-relationship model.

> Do you think that anyone has made a conscious decision that toddlers should start their acquisition of language by learning the names of things, or is it just something that happens by default? What is the alternative?

If you really make an effort, you can find books showing activities (bathing, building, blushing) or spatial relationships (inside, outside, upside down) or even feelings (happy, sad, tired). But it's still difficult for us adults to escape from the materialist mindset, or to avoid transmitting it to the next generation. After all, materialism is embedded in the structure of the child's book (one page, one picture, one word), together with the implicit notion that the child's task is to accumulate vocabulary, one word at a time.

That's why it's so difficult to see the world other than as objects, and why the object-oriented paradigm is so attractive, especially when there are excellent techniques for creating objects out of processes, out of relationships, or perhaps even out of nothing. Philosophers and software engineers have a word for this; they call it **reification**.

When relationships are regarded as things, this usually focuses attention either on the bridging mechanism, or on a static snapshot of the relationship, as for example represented by a legal contract. When processes or services are regarded as things, this usually focuses attention on the deliverable or end-result, as shown in Table 1.

Planning as Process Making scheduling and resourcing decisions in response to changing events	→	**Plan as Record** A consistent set of schedule items and resource assignments.
Negotiation as Process Ongoing negotiation and development of the terms of business.	→	**Contract as Record** A legally binding description of the relationship between two companies at a particular time.
Information as Process Selection, interpretation and dissemination of relevant business data.	→	**Information as Document** Formal results of the selection and interpretation of data.

Table 1 Regarding processes as things

The object-oriented way of describing components is extremely useful, especially for designing and managing components. It is also useful for describing the behaviour of components, and their performance in complex environments. But there are limitations to an object-oriented view of systems and components.

Relational Model

A view of data structure as a set of tables, with rows and columns. Each row represents an occurrence of the underlying entity type, while each column represents an attribute.

This view corresponds fairly closely to the logical structure of a relational database.

Relationships between entity types are represented in the relational model by cross-reference attributes, known as **Foreign Keys**.

Relationship

A relationship is a pattern of binary association between entities – it associates pairs of entities together. Usually the entities belong to two different entity types. Some relationships, however, associate pairs of entities belonging to the same entity type: these are known as **involuted relationships**.

Relationship Pairing

A relationship pairing is an instance of a relationship. It associates one entity occurrence with another entity occurrence, according to a specific relationship.

Repository

A store for models and related information. May be provided as part of a modelling tool, or may be a separate artefact. Sometimes known as a **data dictionary** or **encyclopedia**.

Science

People frequently complain that modelling "is not scientific". So what is science anyway?

Science involves making and testing hypotheses. A model is a hypothesis, a theory – which is tested in various ways – by expert examination, by test cases (including monsters). Science operates by trial and error.

Screen

The computer offers information as services through a screen. The screen is both literal and metaphorical. It is a surface on which the data are presented, and also a filter that controls what the user sees. To screen something implies both show and hide.

Secondary Key

See **Identifier**

Semantics

Strictly speaking, the **semantics** of a model (or modelling language) denotes a relationship between the model and something else, which gives the model its meaning. Semantics is therefore the study of meaning.

However, many people in IS use the term **semantics** to refer to the formal structural constraints of a modelling language. This should more accurately be called Syntax.

Specific Feature

See **Classification**

Subject Area

A broad field of information, usually involving many interconnected entity types.

Subtype

An identifiable subset of a type with special and relevant attributes, relationships, states or responsibilities.

The subtype usually **inherits** all the properties of the supertype, including the identity rule, and has some additional distinguishing properties.

A type may be **partitioned** into several non-overlapping subtypes, using a partitioning rule, which allows us to determine for each occurrence which subtype, if any, it belongs to.

Synonym

A **synonym** is an alternative name for the same thing. Thus CUSTOMER and CLIENT may be synonyms, if they mean exactly the same thing.

The trouble is that near synonyms are more common than exact synonyms – and you can get into trouble if you incorrectly assume that the synonymy is exact. For example, SUPPLIER is not an exact equivalent of CREDITOR if the latter includes the taxman and the former does not.

Exact synonymy means not only that the two terms have exactly the same set of occurrences today, but that they always will have. An infant school may assume that PRIMARY CARER is equivalent to MOTHER because at present all the primary carers happen to be mothers.

Syntax

The formal structure (or “grammar”) of a model or modelling language.

Third Normal Form

see **Normalization**.

Two-Faced

A **two-faced** entity type is one that serves more than one purpose. This is found particularly with abstract entity types such as MARKET or ACCOUNT. The principle of data sharing seems to urge that each entity type serve as many purposes as possible, but there are situations where a single term hides a multiplicity of purposes, and must be pulled apart.

A good example of a two-faced entity type occurring in many models is PRODUCT. This has two aspects: what is bought by the customer, and what is delivered to the customer. Superficially these appear equivalent, but they often turn out not to be. Consider tinned peaches. The consumer selects a brand, and perhaps doesn’t care where the peaches are grown. Indeed, the same brand of tinned peaches may contain Californian peaches at one time of year, and South African peaches at another time of year. Furthermore, peaches from the same source may be tinned under several different brand names (including supermarkets' own brand labels).

Thus we have many facts about a tin of peaches, including its brand name and retail price, as well as the source of the peaches inside. Confusion arises if we try to model all these facts in a single entity type called PRODUCT. Instead, it is usually a good idea to distinguish two entity types: PRODUCT and BRAND, with a many-to-many relationship between them.

Some companies are accustomed to making a conceptual and management separation between PRODUCT and BRAND. The same goods and services are presented to the customer under multiple brand identities. For example, a joint venture between a supermarket and a bank may result in the bank's financial products being sold to customers under the supermarket's brand name.

Or conversely, the same brand identity may be used for more than one product. For example, in the oil industry, the same brand name may be used for a range of slightly different products. The exact petrochemical mixture pumped into cars varies according to the time of year, the state of the oil market, and numerous other factors. Several other industries, including brewing, have different recipes for summer and winter. The customer is not expected to notice the difference.

But some companies have traditionally maintained a one-to-one correspondence between PRODUCT and BRAND. This can turn out to be a significant constraint, especially in IT systems. For example, a financial services company wanted to license its products to a sister company within the same conglomerate. In other words, the sister company would market the products to its own customer base under its own brand name. But the IT systems supporting the products couldn't accommodate this flexibility. Among other things, the billing systems were hard-coded to print a specific brand name at the top of each communication to the customer.

And the constraints were not just from the IT systems. For example, the customer support staff answering the phones didn't know what brand the customer had bought the product under.

From a business perspective, there were three possible outcomes of this lack of articulation:

1. Confusion and dilution of brand identity (loss of clarity).

2. Loss of brand identity – anonymous and unbranded information & services (loss of engagement).

3. Separate capability for each brand – thus restoring one-to-one relationship between BRAND and PRODUCT (loss of intelligence).

Type

A specification or rule, to which any number of objects may conform. A type may be empty or infinite.

A given object can conform to arbitrarily many specifications – and can therefore belong to as many different types as you choose to define.

A type may be implemented as a class in OO, or as a logical entity type.