Enterprise Ontologies – Better Models of Business

Ian Bailey - @MonkeyChap

When I first came here, this was all swamp. Everyone said I was daft to build a castle on a swamp, but I built it all the same, just to show them. It sank into the swamp. So I built a second one. That sank into the swamp. So I built a third. That burned down, fell over, then sank into the swamp. But the fourth one stayed up. And that's what you're going to get, Lad, the strongest castle in all of England.
Monty Python and the Holy Grail

Actually, Major Major had been promoted by an I.B.M Machine with a sense of humor almost as keen as his father's.
Catch 22, Joseph Heller

Introduction – Intelligence-Led Systems Engineering

The title of this book is "Intelligence-Based Systems Engineering". This presents something of a challenge to systems engineers – it implies that there is a practice of systems engineering that *isn't* intelligence-based. Gulp! What should we call this practice? Stupidity-based systems engineering? Suck-it-and-see systems engineering? Guesswork-based systems engineering? Do-it-like-we-did-it-last-time systems engineering?

Despite the well documented benefits(1) of a systems engineering approach, many projects that use the approach fail, and often fail spectacularly. Symptoms are usually one or more of:

- Cost overruns
- Failure to deliver in time
- Failure to meet requirements
- Failure to deliver something useful, even if it does meet requirements

In fact, if you asked someone to think of ten major engineering projects that have failed from the last 30 years, chances are most of them would have used a systems engineering approach. Does this mean that the discipline of systems engineering is flawed, or is that each of those projects were not following the approach properly? If you ask the customers for each of the projects, they would probably all answer that the engineers didn't properly understand their business requirement. If you ask the engineers, they will nearly all answer that the customer failed to describe their requirements adequately, or that the requirements changed mid-way through the project. One of the main tenets of systems engineering is to avoid this sort of conflict through sensible management of requirements. How did it go so badly wrong for all those major projects? There are any number of causes, and it would require more than one book chapter to go into all of them. There are some causes that this chapter will focus on though:

- It is not uncommon to blame the "information revolution" for the rise in the complexity of systems, and the accompanying escalation in systems development costs. The explosion in computing technology has meant that the interface management between sub-systems has become orders of magnitude more complex than when the discipline of systems engineering was first used. Does this mean that systems engineering isn't fit for purpose in the information age? There is clearly some excellent thinking in systems engineering and in general systems theory – common

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

sense tells us that the approaches espoused by systems engineers are good. It's just that information management is a very different discipline to engineering and requires a different set of strategies to achieve success. Information systems development projects tend to focus on the systems, and make the information itself a secondary concern – engineers retreating into their comfort zone and focussing on functional rather than information requirements.

- User (and often systems) requirements are used as a contract between customer and supplier. The customer issues a contract for some requirements to be satisfied, usually for a fixed price. For projects that are going to take many years to deliver, changing circumstances mean that requirements *will* change. If it's an information management system, the requirements can change on a weekly basis. Because a systems engineering approach tends to drive the solution towards minimum complexity, it can also drive towards a solution of minimum flexibility(2). Customers can't understand how a seemingly minor (to them) change in requirements can cause such wailing and gnashing of teeth amongst the engineers. In a rigidly inflexible system, built to meet the requirements at minimum cost, even the smallest change to those requirements can have a catastrophic impact. It's easy to argue that we should never build systems like this, but when requirements form a contract, and the cheapest solution wins, the cheapest solution will nearly always be inflexible. If you bid with a more flexible (assumed to be more expensive) solution, you won't be selected. Furthermore, the customer is unlikely to build flexibility into their requirement specification out of fear of increased cost and the possibility of not getting what they wanted.
- If used unwisely, a systems engineering approach can, and often does, result in project stove-piping. When a systems engineer budgets out requirements, weight, cost, etc. to various sub-systems, the teams responsible for the delivery of those systems are often only able to innovate within their own sub-system boundary. The allocation of budget also means there is little-or-no financial motivation for pan-system innovation. The result of this is often that nobody is designing and innovating at the system level. The consequence is that there is rarely a system-wide approach to information. Information is managed by each system and exchanged on an ad-hoc basis across systems interfaces. It is very rare to find a project which has an overall information model that all sub-systems work from. A project can have superb interface management procedures, but will still fail if there is no overall design for the information

Information management is a key aspect of each of the above points. It is the contention of this paper that a systems engineering approach simply does not work well for systems where information management is a key feature. When the systems engineering approach was first proposed, information was much less of a factor in the systems that were being built. The principles of good engineering – designing-out complexity, working to a clear set of fixed requirements, budgeting problems out into sub-systems – simply don't work for information management problems.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Introduction - Business Ontologies

Having dealt with the title of the book, it's only fair to deal with the title of this chapter. If the term "ontology" is used in an information technology context, it is usually either a reference to data structures defined using semantic web standards, or to data structures used in artificial intelligence projects for purposes of reasoning and inference(3). There is however another use of ontology in IT which is now often overlooked. Several experienced data modellers and business analysts have (often independently) come to the conclusion that the way we currently build information systems is broken(4). To find a better way, they have started to de-construct the current methods, and some have even begun to take a look at what is new in the fields of logic and philosophy(5). They are developing the next generation of information models based on the principles of formal ontology.

Information System Requirements Gathering

Before looking at business ontologies in detail, it is worth examining how information systems are currently built. The process will vary depending on the scale of the project and the methodology the chief architect was educated in. There are usually some common threads though:

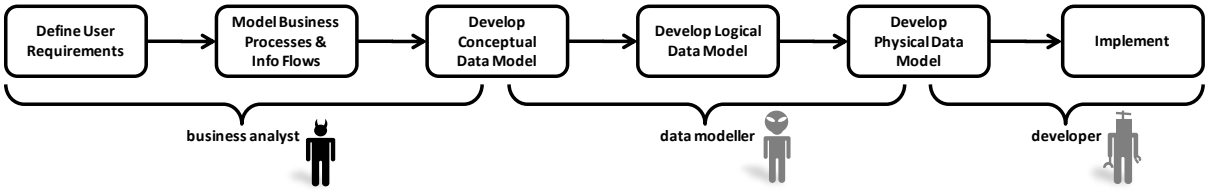

Figure 1 - Typical Development Process for Information Systems

The roles and responsibilities vary from project to project, and there may be more than one person involved at each stage, depending on the size of the project, but this is pretty much the development cycle for the information side of most systems. It seems a fairly sensible approach – separating relatively orthogonal concerns. Someone who understands the business (the business analyst) works with the stakeholders to formalise their functional and non-functional requirements. The functional requirements are (if you're lucky) formalised as a process model, and the non-functional (in part) as a conceptual model. The conceptual model will be built from user terminology, and the information flows between processes. The conceptual model is then used as the basis for a logical model, which is then used as the basis for an implementation model, which is then implemented.

This looks eminently sensible at first glance, but if it's such a good method why do so many large-scale information systems projects fail ? Let's take a closer look at each of the stages and the people involved.

First of all, there is the business analysis. Does the business analyst really understand the business? Chances are the BA is a consultant and has come from an IT background rather than an operational one. Do process models capture an accurate picture of the business? In most cases, the answer is a resounding no. They are usually developed by people who have scant knowledge of the business they are modelling, and they are either based on observation or user interview. Either approach is flawed. If you observe a business, its behaviour will change, simply because it is being observed. Also, you have no idea if the

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

processes you observe are typical, or unique to the time you observe them. If you interview users, they'll emphasise what is on their mind at the time you interview them (the crocodiles closest to the boat) which may not be important at all next week, or in general for the business as a whole.

OK, so we've got off to a bad start. It seems our business models aren't all their supposed to be, but surely our data modeller will save the day. Data modellers have all kinds of tricks up their sleeve, surely? The reality is that data modellers are not well understood – to the rest of the development team, they seem to practice the dark arts. Business analysts don't understand what data modellers do, and neither do the implementers. The data modeller is usually just left to get on with their job with no scrutiny from the business or developer community. There is an industry joke that if you give the same requirements spec or process model to ten different data modellers, you'll get eleven different data models. The route from requirement to data model is not governed by a repeatable, scientific method, it is driven by opinion, experience and shallow pattern recognition.

All is not lost at this stage though. Surely those bright young things in development can paper over the cracks with some clever coding? There definitely are some very bright coders out there, but what gets them excited isn't generally anything to do with keeping the customer happy. They like to use cool new technology. They measure success by who can write the most parsimonious code. They treat logical and set-theoretic constructs such as subtyping as implementation hacks to ensure the right properties and methods are inherited conveniently, rather than as any interpretation of business facts and rules.

All this process amounts to is an elaborate and expensive game of Chinese whispers[1]. At each stage in the process, the deliverable becomes further and further removed from the business requirement. The sheer insanity of this approach is obvious when discussed in this way, but yet it is still the overwhelming choice of method for developing large scale information systems. Let's not forget, it is always the large scale information systems that seem to fail most often. The problem is never fixed, because each actor in the process blames the one next to them – and in large scale systems development projects there are always different actors. The data modeller scoffs at the business analyst for his lack of philosophical thought. The business analyst blames implementation when the software fails user acceptance testing. Everyone involved thinks the data modellers are unnecessary, egotistical and not team players (actually, they may have a point there). Nobody takes a step back and blames the process. Furthermore, in many projects, the business analysts and data modellers (often consultants or contractors) are long gone by the time the problems are found. The result is that development gets the blame, and a great deal of effort has been put into improving development methods, when in fact the problem lies elsewhere.

[1] The British name for a game where a message is whispered along a chain of people to see how distorted the message becomes - other names for the game include "telephone" and "whisper down the lane"

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Driving-Out Complexity

In general, a systems engineer will strive for the least complex system that can satisfy the user requirements. There will always be some level of complexity in systems development – the system often needs to be as complex as the problem it is trying to solve – but generally it is considered a good thing to drive-out complexity. The less complex the system is, the easier it is to predict and control. This means it will generally be safer (or at least simpler to produce its safety case and risk assessment) and more reliable.

When it comes to information, the opposite is true. Information systems are required to store and manipulate information that can be immensely complex. Attempts to simplify the information models used in the system will usually result in a reduction in functionality. It is important to make a distinction between complication and complexity. A complex information model is usually characterised by having a large number of interconnected elements that can be put together in different ways for different purposes. A complicated model generally shares the size characteristic, but usually doesn't offer the possibility for re-use of the same elements for different purposes. Complicated models are easier to understand than complex ones, in that they offer one and only one way to structure information, but their size and the number of attributes and relationships required to make them work means that changes to the requirement have expensive consequences. There are plenty of complicated information models around. In fact cynics would argue that most data modellers cook up the most complicated structures possible as some sort of display of intellectual prowess.

The problem gets worse when the data model goes to implementation. Developers will often fail to recognise the importance of certain design characteristics in the information model and implement the model in a simpler fashion in the mistaken belief that their solution is more elegant or efficient. If an information system is to usefully support real world processes, it must have in-built flexibility, and a degree of extensibility and self-reference(6).

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Stovepipes

We've already joked about ten data modellers producing eleven different data models, but what if they're all working on the same project? On a large project, it's not uncommon for each of the subsystems to have its own data storage capability. Good systems engineering practice should ensure clear interface specifications between the subsystems, but it does not push towards a common information approach across all the sub-systems. In fact, the approach of budgeting out requirements to sub-systems tends to have a negative effect on information coherence.

Sometimes, the same data analysis team will work across the various sub-systems. Even with this approach, it is not uncommon to end up with incoherent data models across the systems. This can be due to a number of issues:

- By far the most common cause is epistemic – when viewing things from different perspectives (e.g. different projects, sub-systems, etc.) data architects fail to recognise when two things are in fact the same. The classic example is customer and supplier. In reality, these are different roles that a person or organisation can play in respect to another person or organisation. However, it is not uncommon (especially in multi-purpose systems such as ERP systems) to see two separate data tables for each of these. When you go and examine the data in them, you will often find the same organisations in both. Elsewhere in the ERP system you may also find a table for organisations, but this will contain another set of organisations – e.g. the departments of the company, regulatory bodies, etc. – as well as some of the organisations in the customer and supplier tables.
- Information modelling is one of the first things to get cut when budgets get tight. Modellers might be asked to do the job as quickly as possible, which prevents any opportunity for reaching consensus across sub-systems. Even worse, the modellers might be cut out of the loop altogether in favour of letting the sub-systems developers do their own data modelling. This is usually fatal - for all their faults, data modellers are infinitely preferable to an enthusiastic end-user with a copy of Visio.
- The tendency to develop data models from process models (as outlined on page 3) means that the scope for the information model is defined by the scope of the processes relevant to the given sub-system – i.e. there is no requirement to look beyond the sub-system boundary.

The resulting information incoherence just causes problems for the interface managers. Data conversion is required at each interface, and uncertainty about whether each system treats a particular term the same as another causes delays and frustration. Many of the issues around semantic mismatch are not actually found until the systems have been running for some time.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

What is Needed for Better Information Systems?

The current approach is clearly broken. Recent history is littered with embarrassing failures of large-scale information systems, and for each one of those failures there are ten that have been declared successful even though they weren't, or simply brushed under the carpet. Very few information systems of an enterprise scale ever deliver on expectations. Read that again. Everyone in the IT industry knows this is true. CIOs know it, and spend most of their working days defending the results rather than working on information strategy. Most CEOs even know it, and regard it as a necessary evil. The problem is recognised, but not the cause. All attempts to fix the problem have centred around software development methods and architectures (e.g. Agile, SOA, etc.) rather than attempting to fix what is really wrong – the inability of the systems to meet the information requirements. If you buy the arguments in the previous sections of this chapter, then it seems what's needed most is:

1) **An accurate, repeatable and defensible way to ascertain the business information requirements**
2) **A way to build flexibility into our information models without making them over-complicated**
3) **A way to ensure that the same concept is treated the same way by different analysts so that when systems are integrated there is less need for triage at the interfaces**

Better Analysis – Getting Your Hands Dirty

The key to achieving point 1) is a reduction in the number of "Chinese whispers" that take place in the analysis stage. Also, the dependence on process analysis needs to be minimised as it is far too open to interpretation. Contrary to popular belief, there are other ways to find out what the business does, but they involve getting your hands dirty. Dirty data is the scourge of IT, and data quality problems are blamed for everything from cost overruns through system crashes to failures in day to day business. This dirty data is an absolute goldmine for the canny information analyst. If analysed intelligently, the data can reveal much more about what the business *actually* does than any process model ever could. What we're interested in here aren't the simple data-entry errors, they're the workarounds instituted by users who needed somewhere to store crucial business information in a system that simply didn't let them do it. These are not failings in data, they are symptoms of a failure to develop an information system that meets the user requirements. Examining this sort of data, with the appropriate analysis techniques, can result in a much better information model than one arrived at through process analysis.

If we can also cut the layers of information / data model from three (conceptual, logical, physical) to one, we stand a much better chance of ensuring that what gets implemented genuinely meets the business requirement. To facilitate this, we need to look at why there ever were three (or sometimes two) layers of data model. In a nutshell, the reason is that database technologies don't store information in the same way that humans tend to think about things in the real world. Hence conceptual models are used to capture concepts and their relations as a first step. The relations can then be rationalised, and attributes added in order to ensure that there is a logically consistent model. Finally, the logical model is bashed and bent to fit whatever underlying storage technology is going to be used. This was a hot

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

topic of research in the 1960s and 70s. Bill Kent (4) highlights the problem: "For some time now my work has concerned the representation of information in computers. The work has involved such things as file organizations, indexes, hierarchical structures, network structures, relational models, and so on. After a while it dawned on me that these are all just maps, being poor artificial approximations of some real underlying terrain." Griethuysen's approach (17) is one of the early 'ostriches', suggesting that there is a simple direct link between the data structures and reality.

What if the underlying storage technology worked the same way as the logical and conceptual models? Then we could have just one model. That, combined with a more rational and defensible analysis technique (based on forensics, not witchcraft), should help ensure we implement something that aligns much more closely with the business.

That last paragraph has left a big question open – data storage technology. Let's return to that point once we've had a look at 2) and 3).

Flexibility – Using the Full Range of Logic

The key to achieving a more flexible information management system is in understanding which relationships in the information model constrain its use, and which give it more axes of movement. This is the essential difference between complexity and complication. Characteristics of a flexible model are:

- Subtypes – i.e. the hierarchy of specialisation of concepts. Flexible models often have the characteristic of extensive subtyping, usually from one top-most class. This allows the frequently used model "infrastructure" to be moved up the specialisation tree and simply inherited by more specialised concepts below. This leads to re-use of implementation patterns resulting in significant savings in code production and maintenance. The problem with this is that if the subtyping isn't based on defensible criteria, the inheritance doesn't work. A new analysis method is required to ensure inheritance works well and accurately mirrors the real world concepts being represented. The classic example discussed by datamodellers is the Circle – Ellipse problem(7). Which is a sub-type of which depends on the definition of inheritance.
- Higher Order – flexible models tend to be able to handle their own classifications. As an example, instead of committing a set of common equipment types as classes (e.g. "pump", "valve", "engine", etc.), a flexible model will allow the users to introduce a class called equipment, and a related class called "equipment type" – though the model may give the users a 'starter pack'. This allows the user to manage the types of equipment at run time as opposed to being fixed in implementation. It sounds obvious for things like equipment, but the requirement for this is less obvious for other data elements such as transactions, and the opportunity to build in flexibility is often missed.
- Names – most systems commit the users to work with only one set of names for things. More often than not, developers will commit this to code by using these names as primary keys in databases. What happens then is that when other communities are offered the chance to use the system, they don't, because the system doesn't use their terminology. The system has effectively stovepiped itself within one business vertical. One could imagine a more flexible system that could

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

cope with multiple communities and multiple names, and occasionally (usually in very specialised situations), these systems do turn up.

These tenets of flexibility run counter to traditional systems engineering and software development. Software developers use subtyping as a convenient way to inherit properties and methods, and pay little or no regard as to the real-world relevance of their assumptions. Higher order systems are rare, and tend only to be used in very specialised circumstances. Systems that can cope with unbounded multiple names are rarer still. Systems that can achieve all three aspects of flexibility are all but impossible to find.

Again, this leaves a dangling question about data storage technology and performance – most developers would shy away from these approaches in the belief that performance and user interfaces would be detrimentally affected. Let's come back to that.

Consistency – Sophisticated, Repeatable Analysis.

Point 3) is the holy grail of information analysis. The goal is that no matter who the analyst is, given the same real world concepts, we will always get the same information model. In practice though, the model is always tainted by the user's view of the world, the analyst's view of the world and the methodology used. What if we had a methodology that allowed us to cut through opinion and only deal with fact? Even if such a methodology could get us to the point where 80% of concepts are dealt with in the same way, it would offer enormous potential to businesses.

Implementation – New Ways of Storing

Now let's get back to implementation – particularly the thorny issue of storage. The relational database rules the roost at the moment, and has done for over 20 years. Its dominance is similar to that of the internal combustion engine – a lot of people think there are probably better technologies out there, but so much has been invested into making the incumbent technology perform that the bar to market entry is very high indeed. In recent years, the relational database has become part of the infrastructure – developers only ever care about physical data models these days if transactional performance is paramount or the queries to be used are complex (e.g. in data warehouses and BI/MI systems). Object-relational projections are now taken for granted in IT. There is no reason why the ideas about accuracy, flexibility and cross-community support outlined above can't also be dealt with in currently available database systems. In fact, in the case of flexibility, a lot of work has already been done. The adaptive object model approach (8)(9) has gone some significant way towards achieving this, but what is often overlooked is the work that went on in Shell in the early nineties on developing flexible data storage systems. This work has now been realised as a successful commercial product – Kalido™ (10)

Furthermore, the *No SQL* movement is producing large numbers of novel data storage approaches, each of which is usually suited to specific types of application. There is great potential for using these technologies to store next generation information structures.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

A New Approach to Information Systems Development

It's all been bad news up to this point. If this chapter has done what it was intended to do, you will have either thrown the book on the fire in disgust (if you're a systems engineer), or be in floods of tears (if you're a customer). The real question is whether you agree with the points made. Few people would argue that there is *something* wrong in information systems development, but do you agree that the problems lay in the analysis approach? The other argument you need to buy is that to fix the problem we need the three points outlined before:

1) **An accurate, repeatable and defensible way to ascertain the business information requirements**
2) **A way to build flexibility into our information models without making them over-complicated**
3) **A way to ensure that the same concept is treated the same way by different analysts so that when systems are integrated there is less need for triage at the interfaces**

If not, then it's probably best you don't read on. If you agree with some or all of the points, then there's a different approach to analysis that you might like to consider. The BORO Method(5) is a forensic approach to re-engineering information systems. It relies on there being legacy data. The older, dirtier and more convoluted this data is, the better. Even so-called green-field IT projects are replacing some form of information system, even if it's paper-based. At the start of any information systems delivery project, there *will* be some legacy data. Traditional approaches tend to ignore this in favour of process modelling – the argument being that the new system has to reflect the new processes. To a certain extent, this is true. The *behaviour* of the system certainly does have to reflect this, but the information is unlikely to have changed much from before.

The premise of BORO is that dirty data is a symptom of an information system that doesn't properly support its users. If you look at the data (not the data model) then you will find out what is really needed to support the business. The challenge is how to get from this data to a specification for a new information system. This is where BORO delivers on point 3). The methodology is designed to be simple, precise and repeatable. Hence, if done properly, two different BORO analyses of the same data (even conducted by different people) should result in broadly similar (sometimes even identical) information models.

The BORO method results in what are called *extensional* ontologies(10). This means that the elements specified in the ontology are not primarily identified by their names, they are identified by their extents. [*Technically, the ontologies have extensional criteria of identity. Often data models have no criteria for identity, so no mechanism for consistently generating agreement about what is being modelled. Clearly criteria of identity are useful – what is more difficult is devising suitable criteria. For those interested in criteria of identity, an extensional criteria approach is one of the few that meets the requirement.*] For individuals, this means their spatio-temporal extent – two things are the same if they occupy the same space for the same time. For types of things (classes), two classes are the same if they have the same members. Although this looks very simplistic, it is reliable and repeatable. It ensures that the same thing doesn't end up in the ontology twice, and that different things don't get mistaken as the same simply because they have the same or similar names. It's also extremely

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

counter-intuitive (especially for computer scientists, it seems), but once you've used the approach long enough, everything else looks primitive, woolly and imprecise.

Once you've identified the things you're dealing with, it is of course useful (and often necessary) to give them a name. The BORO approach separates "name-space" and "object-space". There are things, and there are the names of things. Each name has a context, so there could be a set of names used by one community and another set used by a different community. For a given thing (in object-space) each community might use different names, and BORO allows this. Although the use of multiple names runs counter to the systems engineer's drive for simplicity, it does what the users want. Those communities have probably developed their own terminology for good reason – let them carry on using it. The IT should not dictate terminology to the business.

The ontologies BORO produces allow new classes to be added. This flexibility, combined with the precision of the analysis technique allows applications to be created that are both semantically precise *and* flexible.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Managing Time

As well as the benefits outlined above, BORO also gives the analyst a new way to model time. The handling of temporal information in data models is at best inconsistent and often frankly incoherent. BORO offers a better alternative.

The requirement that BORO places on the analyst to think in terms of extension has a number of interesting consequences. First of all, if you have to consider identity of individual things (buildings, countries, people, etc.) in terms of their physical extent, you are forced to also consider their temporal extent. Two different things may occupy the same space at different times, so the criteria for identity also has to consider the temporal dimension. Philosophers call this "four-dimensionalism". It has its origins in physics, mathematics(11) and philosophy(12) from the early twentieth century, with mainstream philosophy catching up about 90 years later(10)(13)(14). Theoretical and philosophical subjects tend to be ignored by engineers (the author of this chapter included). However it turns out that for information systems, the 4D approach makes life very simple indeed. Data models usually end up having to deal with time in different ways – the time things happen, the duration of things, the time between occurrences, times of the day, days of the week, etc. The 4D approach provides a single, consistent approach to time that covers all the bases a data modeller would ever need – compare this with relatively byzantine structure of, for example, (19). It also provides a very sophisticated way to model how things *change* over time. A whole book could be devoted to the things you can do with a 4D approach – see Partridge(5) and Hawley(14) to get you started – but it's worth spending a bit of time on it here.

If you use a 4D approach, the first thing you have to get used to is that other dimension. Let's get the criticism out of the way first. By now, a few readers are going to be thinking this is all theoretical nonsense and of no practical use. Terms like "extensional ontology" and "four-dimensionalism" are hard to get past the CIO, and will probably send the CTO scuttling back to his cave to read Dr Dobb's journal. The ideas behind it are sound, practical and useable, however. In terms of time, the first thing to get used to is using whole-part relationships – technically called "mereology" (15) – in a temporal setting. We are all familiar with the concept of composition – one thing being part of another. In a 4D ontology though, the individual also has a temporal extent. This allows us to express quite complex situations in a very simple way. If we want to take about cars coming and going in a car park, or in a particular parking place, all we need to do is work out the composition. The example below is a "space-time map"; three dimensions (x,y and z) are compressed and shown on the vertical axis, with time on the horizontal:

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

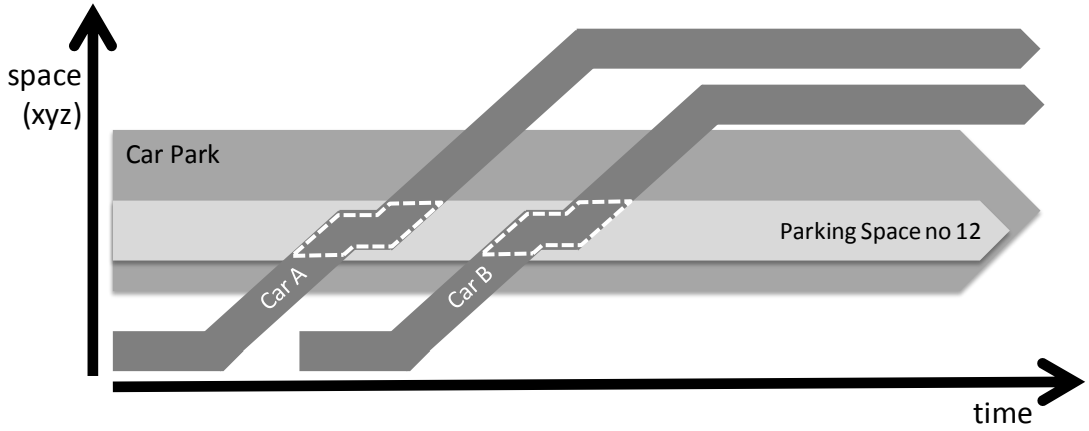

Figure 2 - Space-time map for car parking

Although these diagrams don't allow us to be precise about space, they do help enormously in visualising how things change over time. In the simple example above, we can see immediately that cars A and B come pull into the car-park, park in space number 12 and then leave. In 4D, a (temporal) part of the car (shown by the dotted line) is part of the parking space. The parking space is part of the car park. If we want to be more specific:

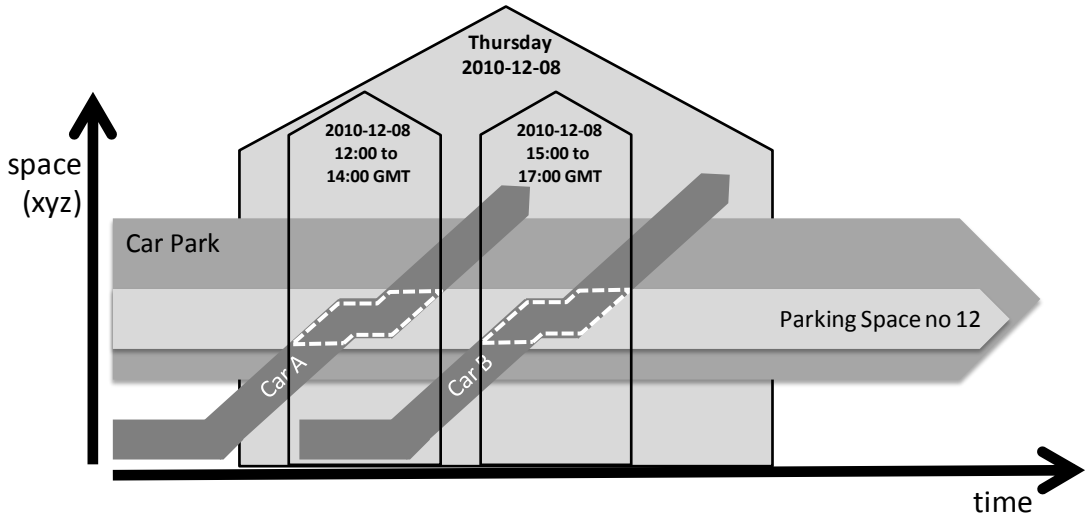

Figure 3 - Space-time map with periods overlaid

In this case, we are stating that the temporal part of Car A while it was in space 12 is part of the period of time 12:00-14:00. Again, all that has been used here is mereology. Although this is a little counter-intuitive, it is consistent. Traditional data models might be able to express what is shown above more concisely – e.g. it might store a set of key pairs for the parking space and the car registration, along with the begin and end times:

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Space	Car	In	Out
12	A123ABC	2010-12-08-12:00	2010-12-08-14:00
12	B234XYZ	2010-12-08-15:00	2010-12-08-17:00
12	etc.		

Figure 4 - Flat-earth data modelling

The problem with this is that it is data *about* the car and car park. The data model for this is not an ontology, because it doesn't model the real world, it models some specific data about the real world. The opportunity for re-use of this data is minimal at best, and the opportunity for re-use of the data model is probably zero. A system built around a model like this can do one thing and one thing only – track parking times. A system built around the more sophisticated 4D ontology understands about cars, car parks, days, times and parking spaces. Furthermore, the simple mereological patterns used to deal with this are also equally applicable to the car-key store in the attendant's hut, the car-park ventilation equipment tag numbers, and the clothes the attendant wears. This, in a nutshell, is why it's worth going to the trouble of doing the analysis properly using precise, repeatable method like BORO.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Against Ontology

The counter-argument to formal ontology is usually that it is complex, hard to implement and will never perform. Let's look at those points one by one.

Firstly, yes a formal ontology is not simple. It is complex, and that is a good thing. However, an ontology need not be *complicated*. There are some horrendously complicated relational data models (ER) out in the wild. They are rarely flexible or extensible, simply because they *are* complicated. They are hard to maintain, and very costly to change once up and running. Ontologies suffer from none of these problems and only seem complex to people who are not familiar with formal logic and set theory. That said, something strange starts to happen when these approaches are adopted. Code starts to shrink. This has been the experience of a number of projects now. People who have used formal patterns such as those defined by the Gang of Four(16) or Martin Fowler(8) have experienced improvements in code acreage. Those that have gone a step further and started to use formal semantics and ontologies are seeing even further gains. Most of these seem to stem from the establishment of even more general patterns – for example the re-use of whole-part composition for temporal matters covered in the previous section.

In terms of being hard to implement, this is an education issue. For a systems engineer who is steeped in the traditional analysis and development methods then yes, ontology is going to be difficult. For organisations that are entrenched(17)(18) in these traditional methods, then it's going to be impossible other than by acquisition of smaller, more agile companies. If you don't carry all that baggage with you, or you're prepared to un-learn what you know then it's actually not that bad. Ontology has been used a lot by the artificial intelligence community, and this has tended to add to its reputation for complexity and opacity. Ontology has utility outside of AI, and implementations (even in relational databases) can be relatively straightforward. Where it *can* get difficult though is in the user interface. When the underlying data model can change without recourse to software re-builds, the user interface needs to be adaptive enough to cope with these changes. This either means building data-driven interfaces, using a service-oriented approach with fine-grain presentation services, or being prepared to change the UI code regularly...none of which are show-stoppers. The same goes for middleware in n-tier implementations.

Performance is another old chestnut. One of the major users of ontologies over the last couple of decades has been the artificial intelligence community. Reasoners and inference engines gained a reputation for poor performance and this seems to have stuck. Although there has been some good work in AI, some of the reasoners can spend days churning through data, only to present an answer that shows all the deductive skills of a labrador puppy. They may well have achieved a great logical feat, but to most IT professionals it is apparent that the same answer could have been arrived at with a couple of queries in a couple of seconds. Ontology doesn't have to be an academic pursuit though. Ontology storage systems can perform well and scale massively. Even triple stores are starting to perform well – a company called Garlik has developed a system called 5Store that they use to manage vast quantities of data with transaction rates of 700,000 triples per second.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

In Conclusion

The theme of this chapter is that traditional analysis techniques rarely go beneath the surface appearance of process models and information flows to find out what is actually going on. This means that the systems that are developed are often not sophisticated enough to cope the problem they were supposed to solve, and are rarely able to extend their functionality or adapt to changing business environments without huge disruption and cost.

If the analysis method used is more rigorous, defensible and repeatable, the information models produced will be more robust. If the models produced are flexible, extensible and closely follow the real world (instead of data about the real world) then the systems produced are more agile. An ontology approach (such as that provided by the BORO method) has the potential to provide the flexibility *and* the accurate real-world modelling.

Literature Search

As you've probably noticed by now, this chapter contains a lot opinion and observation from a practitioner's point of view. It is the result of two decades of dealing with systems engineers who think interoperability is simply a case of having a network. In many cases, the references have been retro-fitted in order to justify some of the more controversial points. But, I would hope that most of the points are *painfully* familiar to anyone who has worked on large-scale systems.

The process of going through these references was a revelation though. To my horror, I discovered that the problems described in this chapter have been around since the 60s (20), and the 4D ontology approach that seems to solve so many problems in information modelling has its roots in early twentieth century mathematics (12) and philosophy (13). None of this is new.

Acknowledgements

Thanks go to Chris Partridge for helping with some of the references – especially the Mealy one (20) – and reviewing the chapter (even though he described my writing style as "shock jock"), and to Dr Graham Bleakley of IBM for reviewing and sanity-checking.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

Bibliography

1. *Understanding the Value of Systems Engineering.* **Honour, Eric C.** Pensacola, FL : INCOSE, 2004.

2. *When Systems Engineering Fails --- Toward Complex Systems Engineering.* **Bar-Yam, Yaneer.** Cambridge, MA : IEEE, 2003.

3. *A translation approach to portable ontologies.* **Gruber, T. R.** 5(2):199-220, 1993, Vol. Knowledge Acquisition.

4. *Data And Reality: Basic Assumptions in Data Processing Reconsidered.* **Kent, William.** 1978.

5. **Partridge, Chris.** *Business Objects: Re-Engineering for Re-Use.* 2nd Edition. London : BORO Centre, 2005. ISBN 0-9550603-0-3.

6. **Barwise, Jon.** *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena.* Cambridge : Cambridge University Press, 1996. 978-1575860084.

7. *From Mechanism to Method: Total Ellipse.* **Henney, Kevlin.** s.l. : Dr Dobbs, 2001.

8. **Fowler, Martin.** *Analysis Patterns, Reusable Object Models.* s.l. : Addison-Wesley, 1997. ISBN 978-0201895421.

9. *Architecture and Design of Adaptive Object Models.* **Joseph W. Yoder, Federico Balaguer, Ralph Johnson.** s.l. : OOPSLA '01, 2001.

10. **Heller, Mark.** *The Ontology of Physical Objects.* Cambridge : Press Syndicate of the University of Cambridge, 1990. 0-521-38544.

11. **Minkowski, Hermann.** Raum & Zeit (Space & Time). [book auth.] A. Einstein, and H. Minkowski H.A. Lorentz. *Das Relativitätsprinzip. Eine.* Leipzig : Teubner-Verlag, 1915.

12. **McTaggart, John M. E.** The Unreality of Time. *Mind.* 1908, Vols. 17, pp. 457-474.

13. **Sider, Theodore.** *Four-Dimensionalism: An Ontology of Persistence and Time.* s.l. : Clarendon Press, 2003. 978-0199263523.

14. **Hawley, Katherine.** *How Things Persist.* s.l. : Clarendon Press, 2004. 978-0199275434.

15. **Simons, Peter.** *Parts: A Study in Ontology.* s.l. : Clarendon Press, 2000. 978-0199241460.

16. **Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides.** *Design patterns : elements of reusable object-oriented software.* s.l. : Addison Wesley, 1994. 978-0201633610.

17. *Generative Entrenchment and the scaffolding of individual development and social institutions.* **Wimsatt, William.** 2005 : International Society for History, Philosophy, and Social Studies of Biology (ISHPSSB).

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3

18. **Wimsatt, William.** *Re-Engineering Philosophy for Limited Beings.* s.l. : Harvard University Press, 978-0674015456. 978-0674015456.

19. *ISO/TC97/SC5/WG3-N695 - Concepts and Terminology for the Conceptual Schema and the Information Base.* **Griethuysen, J.v.** New York, NY : ANSI, 1982.

20. **Date, C. J., Darwen, Hugh and Lorentzos , Nikos .** *Temporal Data & the Relational Model.* s.l. : Morgan Kaufmann, 2002.

21. *Another Look at Data.* **Mealy, G H.** Anaheim, CA : Proceedings of the Fall Joint Computer Conference, Nov 14-16, p565, 1967.

This paper was originally published as a chapter in *Intelligence Based Systems Engineering, Tolk & Jain (editors). The original publication is available at:* http://www.springerlink.com/content/978-3-642-17930-3