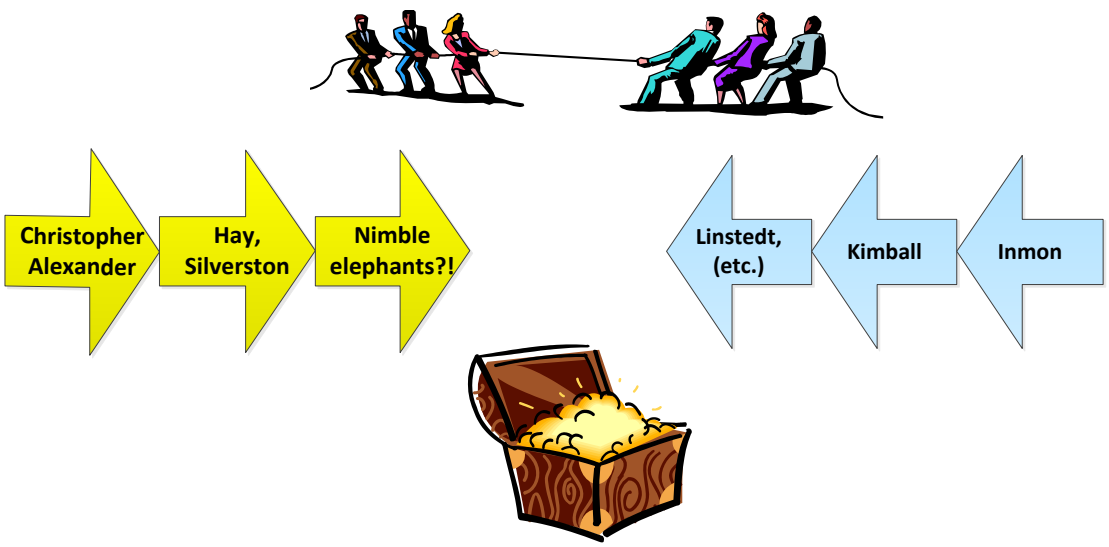
Universal Data Vault:

A case study in combining "Universal" data model patterns with the Data Vault architecture



Country Endeavours Pty Ltd

"Creative solutions for Difficult Problems"

UNIVERSAL DATA VAULT

UNIVERSAL DATA VAULT: A CASE STUDY IN COMBINING "UNIVERSAL" DATA MODEL PATTERNS WITH THE DATA VAULT ARCHITECTURE

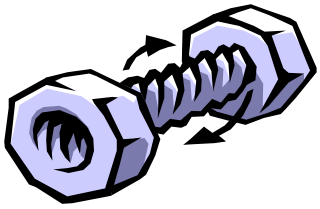
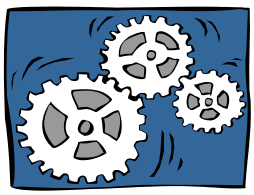
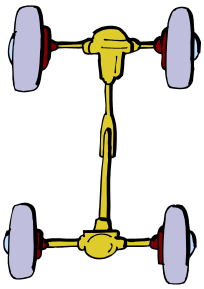
There are plenty of reasons you might want to use proven "universal" data model patterns to improve the outcome of your enterprise initiative or your agile project – the patterns are proven, robust, efficient now, and flexible for the future.

There are also many reasons for considering the Data Vault (DV) architecture for your data warehouse or operational data integration initiative – auditability, scalability, resilience and adaptability.

But if your enterprise wants the benefits of both, does their combination (as a "Universal Data Vault") cancel out the advantages of each? A case study looks at why a business might want to try this approach, followed by a brief technical description of how this might be achieved.

Why would anyone want a "Universal Data Vault"?

Patterns at work

The idea of using proven data model patterns to deliver business benefits isn't new. In his 1995 book, *Data Model Patterns: Conventions of Thought*, David Hay stated, "*Using simpler and more generic models, we will find they stand the test of time better, are cheaper to implement and maintain, and often cater to changes in the business not known about initially.*"

That's the theory, and I'm pleased to say it works. I smile when I remember one interview for a consulting job. The interviewer reminded me that another telecommunications company we both knew had spent years developing an enterprise logical data model (ELDM). He then threw me a challenge, saying, "John, I want you to create the same sort of deliverable, on your own, in two weeks – we've got three projects about to kick off, and we need an ELDM to lay the foundations for data integration across all three."

I cut a deal. I said that if I could use the telecommunications industry data model pattern from one of Len Silverston's data model patterns books, modified minimally based on time-boxed interviews with the company's best and brightest, I could deliver a fit-for-purpose model that could be extended and refined over a few additional weeks. And I did.

I've done this sort of thing many times now. I commonly use "patterns", but it's the *way* they're applied that varies. Sometimes an ELDM is used to shape an Enterprise Data Warehouse (EDW). Other times it is used to define the XML schema for an enterprise service bus (ESB). Or maybe mould a master data management (MDM) strategy, or define the vocabulary for a business rules engine, or provide a benchmark for evaluation of the data architecture of a commercial-off-the-shelf package, or ...

There's a key message here. If you use a pattern-based ELDM for any of these initiatives, the job of information integration across them is much easier, as they are founded on a common information model. This is exactly what an IT manager at another of my clients wanted. But his challenges were daunting.

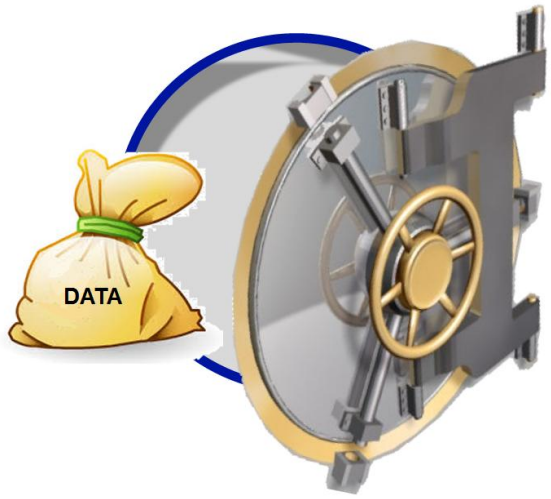
Some of you will recognise the data administration headaches when one company acquires another, and the two IT departments have to merge. It can be painful. Now try to imagine the following. As at June 30th, 2010, there were 83 autonomous but related organisations across Australia, each with something like 5 fundamental IT systems, and each of these systems had maybe 20 central entities. If they were all in

relational databases, that might add up to something like 8,000 core tables. Then on July 1st, under orders of our national government, they become one organisation. How do you merge 8,000 tables?

They had some warning that the organisational merger was going to happen, and they did amazingly well, given the complexity, but after the dust settled, they wanted to clean up one or two (or more!) loose ends.

Firstly, I created an ELDM (pattern-based, of course) as the foundation for data integration. It's a bit of a story in its own right, but I ran a one-day "patterns" course for a combined team of business representatives and IT specialists, and the very next day, I facilitated their creation of a framework for an ELDM. It did take a few weeks for me to flesh out the technical details, but the important outcome was business ownership of the model. It reflected *their* concepts, albeit expressed as specialisations of generic patterns. This agreed expression of core business concepts was vital, as we will shortly see.

A Data Vault solution looks promising



The second major activity was the creation of an IT strategy that considered Master Data Management, Reference Data Management, an Enterprise Service Bus and, of course, Enterprise Data Warehouse components. The major issue identified was the need to consolidate the thousands of entities of historical data, sourced from the 83 disparate organisations. Further, the resulting data store needed to not only support Business Intelligence (BI) reporting, but also operational queries. If a "client" of this new, nationalised organisation had a history recorded in several of the 83 legacy organisations, one consistent "single-view" was required to support day-to-day operations. An EDW with a services layer on top was nominated as part of the solution.

The organisation had started out with a few Kimball-like dimensional databases. They worked well for a while, but then some cross-domain issues started to appear, and work commenced on an Inmon-like EDW. For the strategic extension to the EDW, I suggested a Data Vault (DV) approach. This article isn't intended to provide an

authoritative description of DV – there's a number of great books on the topic (with a new reference book by Dan Linstedt hopefully on its way soon), and the TDAN.com website has lots of great material. It is sufficient to say that the some of the arguments for DV were very attractive. This organisation needed:

- Auditability: If the DV was to contain data from hundreds of IT systems across the 83 legacy organisations, it was vital that the *source* of the EDW's content was known.
- Flexibility/Adaptability to change: The EDW had to be sufficiently flexible to accommodate the multiple data structures of its predecessor organisations. Further, even during the first few years of operation, this new organisation kept having its scope of responsibility extended, resulting in even more demand for change.

I presented a DV design, and I was caught a bit off guard by a comment from one of the senior BI developers who said, "John, what's the catch?" He then reminded me of the saying that if something looks too good to be true, it probably is! He was sceptical of the DV claims. Later, he was delighted to find DV really did live up to its promises.

But there was one more major challenge.

A marriage of convenience? The generalisation/specialisation dilemma

Data model patterns are, by their very nature, generic. Len Silverston produced a series of 3 books on data model patterns (with Paul Agnew joining him for the third). Each volume has the theme of "universal" patterns in their titles. Len argues that you can walk into any organisation in the world, and there is a reasonable chance they will have at least some of the common building blocks such as product, employee, customer, and the like. Because the patterns are deliberately generalised, their core constructs can often be applied, with minor tuning, to many situations. In Volume 3, Len & Paul note that data model patterns can vary from more specialised through to more generalised forms of any one pattern, but that while the more specialised forms can be helpful for communication with non-technical people, the more generalised forms are typically what is implemented in IT solutions. The bottom line? Pattern implementation can be assumed to be quite generalised.

Conversely, DV practitioners often recommend that the DV Hubs, by default, avoid higher levels of generalisation. There are many good reasons for this position. When you talk to people in an enterprise, you will likely discover that their choice of words to describe the business appears on a continuum, from highly generalised to very specialised. As an example, I did some work many years ago with an organisation responsible for emergency response to wildfires, floods, and other natural disasters. Some spoke sweepingly of "deployable resources" – a very generalised concept. Others were a bit more specific, talking of fire trucks and aircraft, for example. Yet

others spoke of "water tankers" (heavy fire trucks) and "slip-ons" (portable pump units temporarily attached to a lighter 4WD vehicle) rather than fire trucks. Likewise, some spoke of "fixed-wing planes" and "helicopters" rather than just referring to aircraft.

In practice, if much of the time an enterprise uses language that reflects just one point on the continuum, life's easier; the DV implementation can pick this sweet spot, and implement these relatively specialised business concepts as Hubs. This may be your reality, too – if so, you can be thankful. But within my recent client's enterprise involving 83 legacy organisations, there was no single "sweet spot"; like with the wildfire example, different groups wanted data represented at ***their*** sweet spot!

Here was the challenge. By default, data model patterns, as implemented in IT solutions, are generalised, while DV Hub designs are, by default, likely to represent a more specialised view of the business world. Yet my client mandated that:

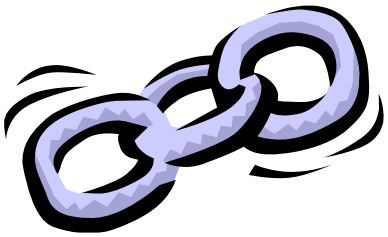
1. The DV implementation, while initially scoped for a small cross-section of the enterprise, had to have the flexibility to progressively accommodate back-loading the history from all 83 legacy organisations. As a principle, any DV implementation should seek to create Hubs that are not simplistic, mindless transformations from the source system's tables. Rather, each Hub should represent a *business* concept, and any one Hub may be sourced from many tables across many systems. So even though the combined source systems contained an estimated 8,000 tables, we did not expect we would need 8,000 Hubs. Nonetheless, if we went for Hubs representing somewhat specialised business concepts, we feared we might still end up with hundreds of Hubs. There was a need for a concise, elegant but flexible data structure capable of accepting back loading of history from disparate systems. I met this need by designing a DV solution where the Hubs were based on "universal" data model patterns.
2. The DV implementation of the new EDW had to align with *other* enterprise initiatives (the enterprise service bus and its XML schema, MDM and RDM strategies, and an intended business rules engine). All of these, including the DV EDW, were mandated to be consistent with the one generalised, pattern-based ELDM. Interestingly, the business had shaped the original ELDM framework, and it was quite generalised, but with a continuum of specialisation as well. I needed to create a DV solution that could reflect not just more generalised entities, but this continuum.

I called the resultant solution a "Universal Data Vault". This name reflects my gratitude to the data model pattern authors such as Len Silverston (with his series on "universal" data model patterns) and David Hay (who started the data model patterns movement, and whose first book had a cheeky "universal data model" based on thing and thing type!), and Dan Linstedt's Data Vault architecture.

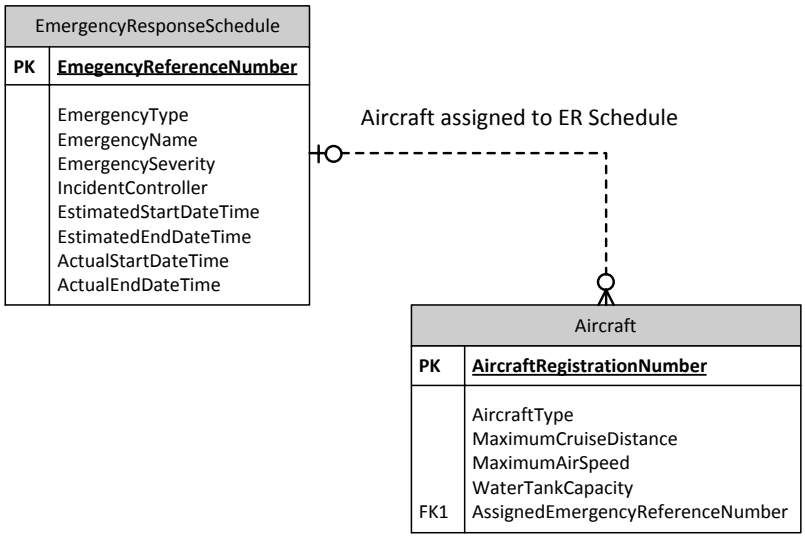
We have now considered ***why*** I combined generalised data model patterns with Data Vault to gain significant business benefits. Next we look at ***how*** we design and build a "Universal Data Vault", from a more technical data model perspective.

The mechanics of a "Universal Data Vault" (UDV)

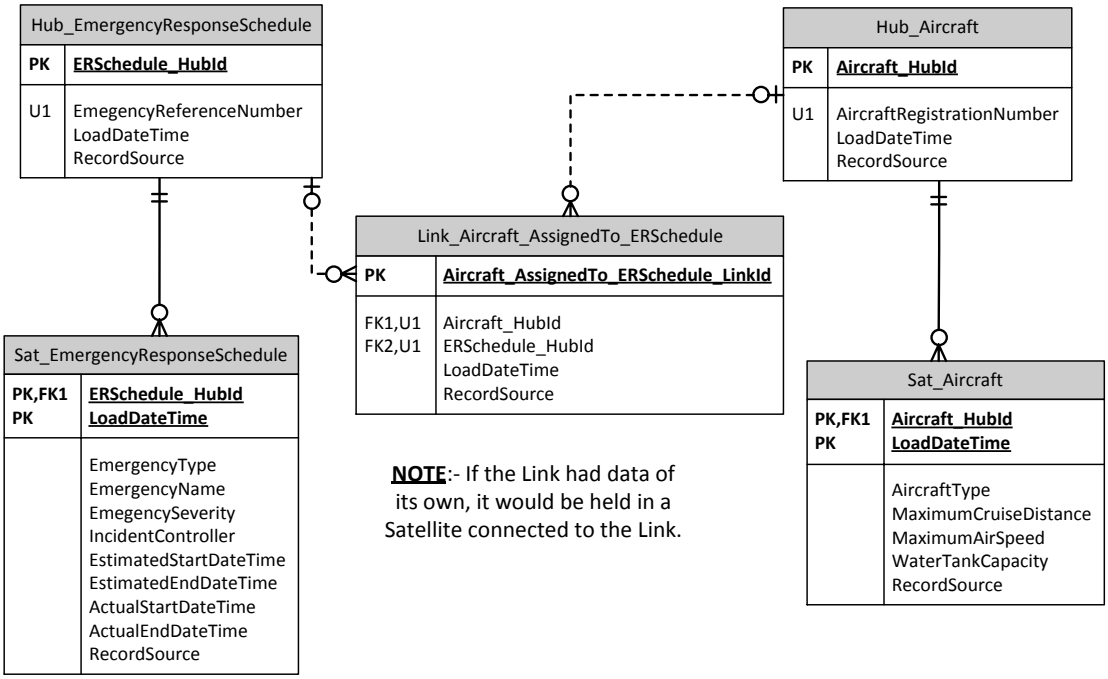
Foundations: A brief review of Data Vault – Hubs, Links and Satellites


If we go back to the emergency response example cited earlier, and prepare a data model for one tiny part of their operational systems, we might end up with a data model something like the following:



Here we see aircraft (helicopters or a fixed wing planes) being assigned to planned schedules-of-action for emergency responses (fires, floods, earthquakes …). This model reflects an operational point-in-time view: at any one point in time, an aircraft can be assigned to only one emergency. Of course, over time, it can be assigned to many emergencies. If we now design a database to hold the history, and if we choose to use a Data Vault (DV) architecture, we could end up with something like this:



Without going into details of the transformation, we can note:

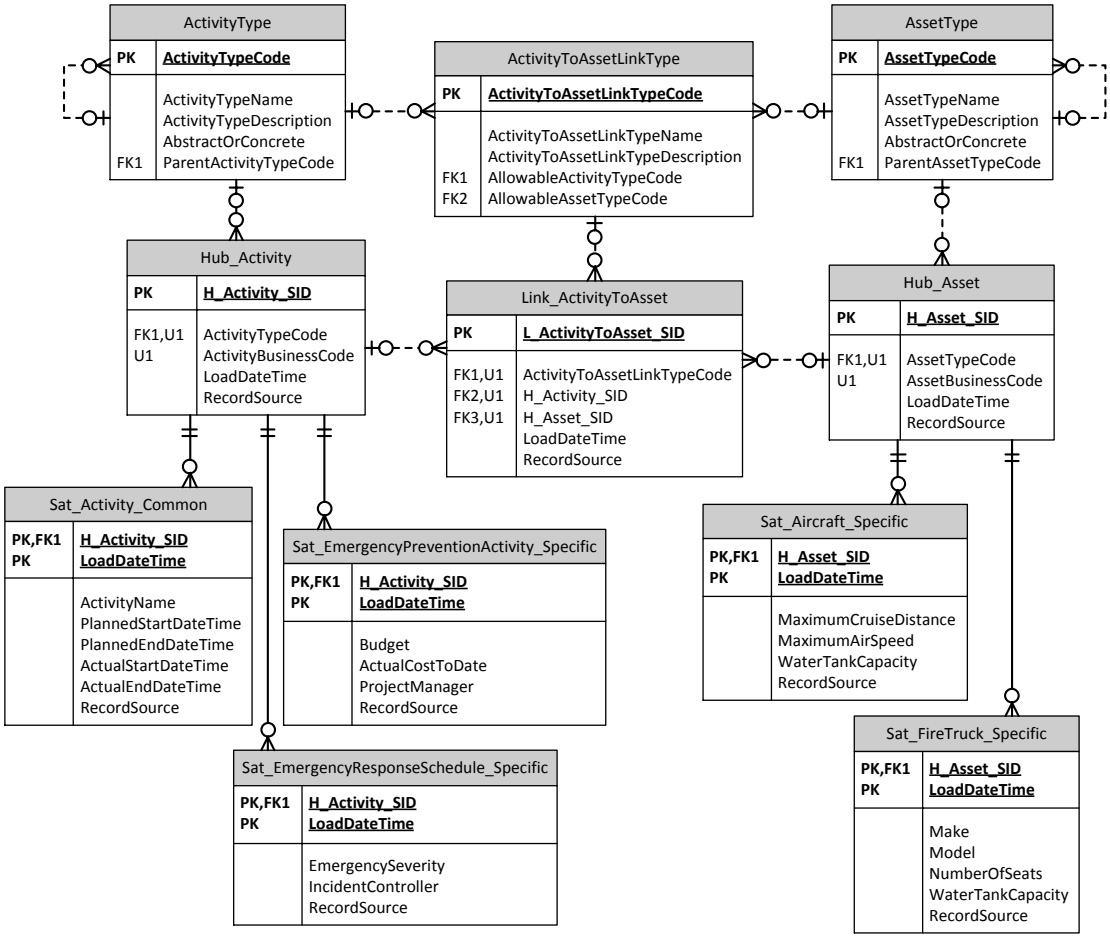
- Each business concept (an aircraft, a schedule …) becomes a Hub, with a business key (e.g. the aircraft's registration number), plus a surrogate key that is totally internal to the DV. Each Hub also records the date & time when the DV first discovered the business instance (e.g. an aircraft registered as ABC-123), and the operational system that sourced the data.
- Each relationship between business concepts (e.g. the assignment of an aircraft to an emergency) becomes a Link, with foreign keys pointing to the participating Hubs. Like a Hub, the Link also has its own internal surrogate key, a date & time noting when the relationship was first visible in the DV, and the source of this data. Note that while an operational system may restrict a relationship to being one-to-many, the Links allow many-to-many. In this case, this is essential to be able to record the history of aircraft assignment to many emergencies over time.
- Hubs (and Links – though not shown here) can have Satellites to record a snapshot of data values at a point in time. For example, the emergency schedule Hub is likely to have changes over time to its severity classification, and likewise, its responsible incident controller.

Fundamentals of the UDV "How-to" solution

As noted in the previous part of this paper, people may challenge the level of generalisation/specialisation of the business concepts that become Hubs. While some may be comfortable with the business concept of an "aircraft", others may want separate, specialised Hubs for Helicopters and Fixed Wing Planes, and yet others may want to go the other way – to combine aircraft with fire trucks and have a generalised Hub for Deployable Resources.

One solution is to have a Hub for the generalised concept of a Deployable Resource, as well has Hubs for Aircraft (and Fixed Wing Plane and Helicopter), Fire Truck (and Water Tanker and Slip-On), Plant (Generator, Water Pump …) and so on. The inheritance hierarchies between the generalised and specialised Hubs can be recorded using "Same-As" Links. This approach is most definitely something you should consider, and is explained in a number of publications on Data Vault. However, for this particular client, there was a concern that this approach may have resulted in hundreds of Hubs.

I created the "Universal Data Vault" (UDV) approach as an alternative. A sample UDV design follows. At first glance, it may appear to be a bit more complicated than the previous diagram. However, the generalised Asset Hub not only represents the incorporation of Aircraft, but additionally it can accommodate a large number of specialised resource Hubs for Fire Trucks, Plant & Equipment and so on. Likewise, the generalised Activity Hub not only handles the Emergency Response Schedule Hub, but also other specialised Hubs such as activities for preventative planned burns.



Let's now look at the individual UDV components.

UDV Hubs

Instead of having (say) a *physical* Hub for Aircraft (plus specialised Hubs for Helicopter and Fixed Wing Plane), and a *physical* Hub for Fire Truck (again with its specialised Hubs for Water Tanker and Slip On), and so on, we can have one generalised Asset Hub to hold all instances of helicopters, water tankers, generators etc. The specialisations are managed as *logical* types of the Asset Hub via a foreign key that links to the subtype definitions in the associated Asset Hub Type table.

The Hub is named according to the chosen selection of generalised data model patterns from the ELDM. They could include, in addition to Asset and Activity as shown above, concepts such as Account, Agreement, Event, Location, Product etc. Another common pattern is the Party and Party Role pattern. This is possibly a

contentious topic that we can leave aside in this article, since alternative modelling choices can easily be accommodated by the UDV structures.

It should be noted that the Asset Hub has a unique index on the combination of Asset Type and Asset Business Code. A regular DV Hub is required to capture the value of a suitable business key that is unique within its domain. For example, aircraft registration numbers can reasonably be expected to be unique across all aircraft, and similarly the registration numbers for fire trucks can be expected to be unique within their context. But when one generalised Hub represents all these things, it might be possible for an aircraft and a truck to have the same registration number. To address this issue we have involved the Hub Type Code as part of the uniqueness constraint.

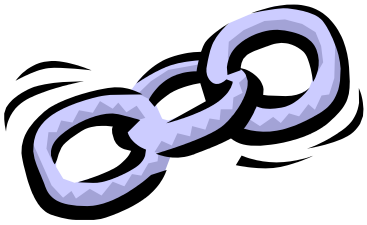
UDV Hub Types

A Hub Type table, such as the Asset Type table, manages the *logical* identification of a supertype/subtype inheritance hierarchy. It is a simple self-referencing "type" table that defines types of (logical) Hubs and their parent/child hierarchy e.g. it might have a row for Fixed-Wing Plane, a row for Helicopter, and a more generalised row for Aircraft as the "parent" of Helicopters and Fixed-Wing Planes.

A UDV Hub Type table is not a regular DV table. It doesn't have a Source column, or a Load Date & Time, as its contents do not come from an operational system. It is simply configuration data, hand-crafted by the design team. This means that if, for example, a new asset type is required (as a specialisation of an existing generalised Hub), you don't have to create a new Hub – just add a row to the Asset Type configuration table, then start populating instances of that type in the Hub table.

There is a column to indicate if a Hub Type is "Abstract or Concrete". This approximates the object-oriented meaning of an abstract or concrete class, but it is not essential to be included in the core UDV architecture. In this sample diagram, an abstract Asset Type would be one that might be used to classify a collection of subtypes, but would not be expected to have any "concrete" instances actually recorded in the associated generalised Hub table.

UDV Links



Just as Hubs can be generalised or specialised, so can Links. For example, if a bank has a generalised Hub for Agreement (Mortgage, Term Deposit, Security etc.) and it also has a generalised Hub for its customers and associated parties, the types of Links between parties and agreements might include signatory-to-agreement, guarantor-for-agreement, witness-to-signature, employee-approval-of-agreement and so on. In UDV, each of these types of Links can be themselves generalised, with a foreign key in each generalised Link that identifies its definition in the associated Link Type table – see below.

UDV Link Types

A UDV Link Type table, such as the Activity-To-Asset Link Type table, contains the *logical* definition for each allowable type of specialised Link between the participating generalised Hubs. As for Hub Types, the UDV Link Type tables:

- Are simple "type" tables that define types of (logical) Links, e.g. with a row for the Aircraft-assigned-to-Emergency Response Schedule Link.
- Are not regular DV tables – they don't have a Source column, or a Load Date & Time column.
- Can have new (logical) types of Links added dynamically where the generalised Link already exists.

UDV Satellites

The Satellites are regular DV Satellites, but they can be hung off Hubs at any level in the (logical) inheritance hierarchy. For example, if all Activities share some common attributes (e.g. activity start and end dates and times), a common Satellite can be defined. Instances of more specialised Hubs can populate the common Satellite as well as their own specific Satellites.

Similarly, Satellites can be hung off Links, and can be generalised or specific to one particular type of Link.

Some optional extras

The standard Data Vault architecture has a beautiful elegance. It has only three core constructs – Hubs, Links and Satellites. Yet within this simplicity, it has the flexibility to accommodate a number of variations. For example:

- Satellites can be split to have multiple Satellites for one Hub (or Link)
- a Hub or Link can have "effectivity" Satellites
- the standard Data Vault can be supplemented with "Point-In-Time" and "Bridge" tables
- … and so on.

These, and many more topics, are best left to the experts who have already published on such matters!

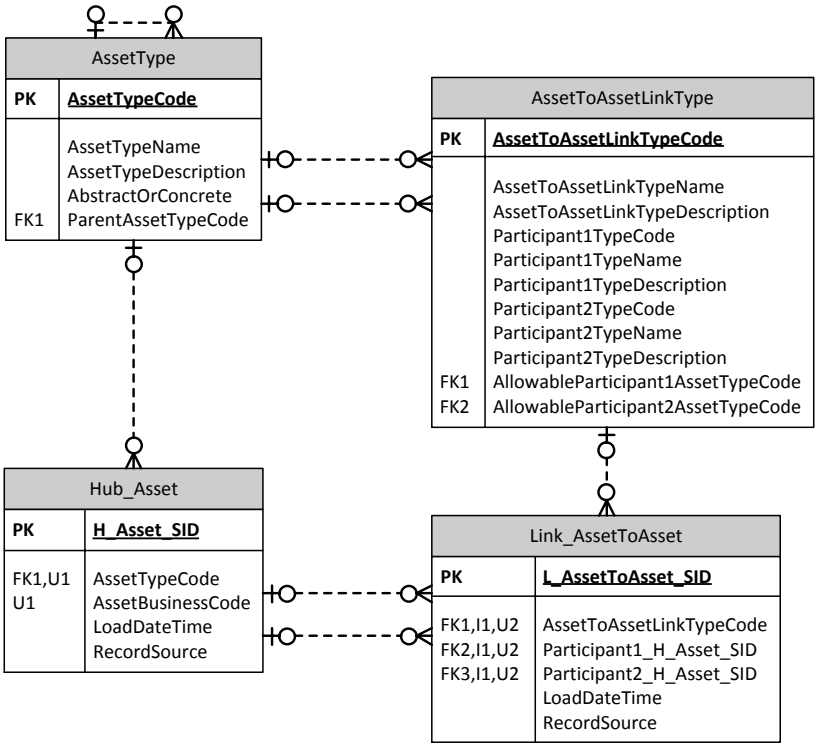
The basic UDV architecture also has both simplicity and extensibility. A few examples are described below.

Self-referencing UDV Link Types

If we have a self-referencing Link named "Asset Contains Asset" that links a pair of assets, we might have to guess which one is the container and which one is the component. Sometimes we might be able to guess correctly. If one item is a car and the other is an engine, it's pretty reasonable to assume the car contains the engine rather than the engine containing the car! But to take a real-life example where the roles of participating Hubs are less obvious, one of our fire fighting agencies sees a fire response unit as *being* a fire truck that *contains* some crew members (i.e. the truck is the container), while another agency sees the grouping of people (the crew) as the container, and the truck as a component.

An amusing twist at another client site relates to office blocks and waste treatment plants. You can have an environmentally responsible office block that contains a small, local waste treatment plant. Conversely, you can have a large regional waste treatment plant that contains its own office block!

In such cases, rather than *implying* the role of the participants, the UDV Link Type may need to *explicitly* define the roles. An example of such a structure follows, where the UDV Link Type table has columns to define the role for each participant:



Rules

Data Vault has a principle of "All the data, all the time". You don't filter out data during the load process just because it breaks some perceived business rule. You want to capture the actual data in the operational systems, whether it is "right" or "wrong". After all, maybe it's the rule that's wrong!

The UDV has the potential to define a number of rules in its metadata, and it can easily be extended. For example, the UDV Link Type tables can be extended to hold expected multiplicity (optionality and cardinality). But the key message is that any such rules are to be used to report *apparent* breaches of rules against the data as loaded, rather than excluding data from the load.

Visualisation

If we had a physical Hub for Aircraft and another for Emergency Response Schedules, plus a physical Assignment Link between them, we can inspect the database and "see" the structures. There may be hundreds of Hubs, Links and Satellites, but they are somewhat visible. Conversely, if all this design is held as data in UDV Hub Type and UDV Link Type tables, the visibility of the structure is different. We can still find it by looking at the rows in these "type" tables, but some data people may be less comfortable.

One solution is to read the metadata in the "type" tables (plus the column definitions in the Satellite tables) and create XML Metadata Interchange (XMI) files which in turn can be imported to an XMI-compliant tool and visualised as a UML class diagram.

Reflections

Pattern-based Enterprise Logical Data Models (ELDMs)

There can be passionate debates about top-down (generalised) versus bottom-up (specialised) modelling. I care less about where people *start*, but I do recommend that before anyone claims to have reached the *end*, they consider both aspects.

Graham Witt is a fellow Australian that I respect highly. He came across a story of what I will call a "bottom-up" designer. The database had to include student records for a school. The designer had noted that in a given sample of data, all the students had two parents that shared the same surname and same address as the student. Of course, once you encounter real-life data, this may not work! The mistake on the surname was embarrassing, but little more. The mistake regarding a common address was far more serious. A mother was living in a women's refuge due to domestic violence issues, and this modelling "mistake" was the catalyst for having her address made known to her abusive husband.

There may have been many issues behind this story, but I am guessing that at least two were in play.

Firstly, I am suggesting that if generalised data model patterns were taken as a starting point, the additional flexibility of these patterns, such as each person having different names (and possibly multiple names), could be reviewed, and unnecessary "over-engineered" features consciously removed. This would have been far less painful than discovery of missed features.

Secondly, the developer in this real-life example was presumably a bit of a novice. I highly recommend anyone considering a pattern-based ELDM (or its logical extension, namely a UDV) purchase the books on data model patterns by Len Silverston and David Hay. In my book, *The Nimble Elephant*, I also provide insight into how these patterns can be effectively applied. But reading the books may not be

enough. A bit of hard-won experience in the use of these patterns may prove to be valuable.

Universal Data Vault: a Battle or Bonanza

There is a saying that necessity is the mother of invention. At some of my client sites, I felt the need to see if I could combine the benefits of data model patterns with those of Data Vault architectures: and it worked, and worked well!

I remember a colleague who challenged the suitability of the *generalised* data model patterns. He wanted to examine its ability to address the *specific* needs of the client. He took the most complex set of sample data he could get his hands on, and we mapped it to the patterns. I think he was pleasantly surprised when over 90% fitted. Being a cheeky sort of a bloke, I teased him by saying that if we had also incorporated the "survey" pattern in the ELDM, we would have got very close to 100%.

There are some drawbacks with UDV. One already mentioned is the lack of visibility of the data structure, at least in a form that is familiar for many data modellers. Another is that finding suitable business keys is hard for regular DV, but even harder for the more generalised Hubs of the UDV architecture.

Yet in spite of these challenges, the first UDV project was such a success that one of the employees at this client site subsequently approached me to "do the same thing" at his new site after he had left and joined another organisation.

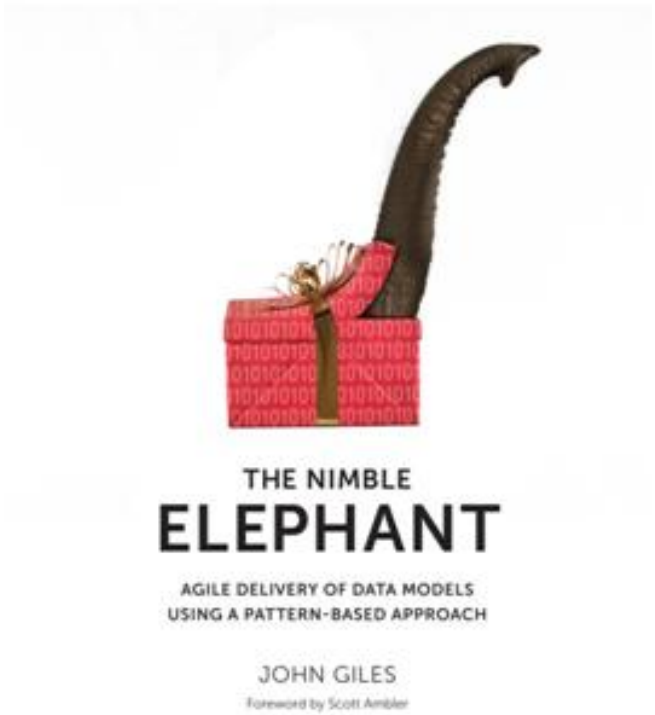
In summary:

- Data model patterns are generic in nature, and may well offer benefits for you.
- Data Vault architectures also offer much:
    - If your enterprise can find one "sweet spot" on their generalisation/specialisation continuum, standard DV is great
    - If your enterprise can't define a single level of specialisation but has a limited number of focal points on the continuum, explicit Hubs for each point, supported by "same-as" Links, should be seriously considered.
    - If your organisation sees value in having a multi-level generalisation/specialisation hierarchy, especially if based on data model patterns, you may find the Universal Data Vault approach will demonstrate that by combining universal data model patterns with the Data Vault architecture the "whole is greater than the sum of the parts".

About the author

John Giles is an independent consultant, focussing on information architecture, but with a passion for seeing ideas taken to fruition. He has worked in IT since the late 1960s, across many industries. He is a Fellow in the Australian Computer Society, and completed a Master's degree at RMIT University, with a minor thesis comparing computational rules implementations using traditional and object-oriented platforms.

He is the author of "*The Nimble Elephant: Agile Delivery of Data Models Using a Pattern-based Approach*".



Further details about the author are available at www.countrye.com.au