Meta-Model Design Guidelines

Avoid redundancy

Redundant Property Types

Because the Design View is so easy to use, I have often seen experimental work that has been left in a production model. For example, I once saw the Process Object Type with 80 Properties (it has 30 by default).

This untidiness can confuse the users and make the product appears much more complicated than it is. For example, a typical business analyst opens the Process Properties dialog and sees all these properties. "What are they for? What must I fill in? Why is there no guidance?" It's enough to make you think "Visio was so much easier!"

It also generates scope for error – users filling in the wrong property (e.g. at a recent client, there were about 4 properties that all seemed to be doing the same thing and an association that also seemed to be capturing the same data; I had been tasked with rationalizing the model so deleted all properties and relied on the association).

It also confuses matters when creating outputs (published, c4w), which depend on a thorough understanding of what each property is for.

It will also slow down publishing as the whole model must be analyzed before publishing starts.

There is scope for improvement here (and potential consulting work):

- Need for a proper meta-model (properties being equal to entity attributes) along with meta-model documentation (which can be produced with CP, but not yet with c4w)
- Need for training in data modeling \ meta-model creation
- Need for training in import\export (e.g. many clients do not realize that import will often bring in unwanted property types – see also note below).
- Need for modeling standards that specify which properties must be populated (and which are optional)

Note

There appears to be a bug in import whereby property types are imported without showing up on the Import review screen, so one activity of the Model manager is to be vigilant, to constantly conduct meta-model housekeeping, deleting redundant meta-model elements.

Redundant Association Types

This is similar to redundant use of Property Types – and is usually the result of early experimentation and prototyping. On the other hand, a rich set of associations may well be required. For example, one client I worked with wanted to associate Processes to Document Objects (integrating their Process Model and Document management System). Easy to do.

However, the feedback from the published output was negative: as a regulatory body, documentation was central to everything they did and they had 13 different types (category) of Document ranging from "Memoranda of Understanding" to "Emergency Bulletin" to "Work Instruction" to "Guidance". A process object could be associated to all 13 types of document and what end-uses wanted was to see these associations under separate sub-headings, not one big list.

The solution, therefore, was to set up 13 Association Types between the Process and Document objects. Another relevant situation here is RACI modeling: do you have one association type with RACI as properties of the association (like CRUD modeling) or do you have 4 separate associations?

Again, there is no right or wrong answer here; it depends on what the client wants to do and how complex (more maintenance) they want their model. More Association Types mean increased data maintenance and slower publishing.

Object Types

When creating a user-defined Object Type, you need to give it both as plural and singular name. There are two reasons for this:

- In Model Explorer, the right-hand List Pane shows the singular version, while the left-hand Model Tree shows the plural version (this has confused some users in the past)
- In published output, the plural name appears as Index page headings, while the singular name appears as prefix to objects (when use of categories is switched off).

It is not mandatory to have both singular and plural names and sometimes, I have seen absurdities – a previous Casewise extension model, for example, used to have something called "Documentation" "Documentations".

It goes without saying that user-defined Object Types should have clear names. However, this in part depends on understanding the principles of data modeling and taxonomy. For example, an Object Type is like a generic "class of thing" and so should be clearly understood as relating to something in the real world. Use of categories provides sub-classes. So in the business world, POLDAT provides a clear taxonomy.

However, I have seen some meta-models that are quite simply a mess with no apparent understanding of how to categorize the real world into useful "things". At one level, it can be far too complicated with separate object types for what should be modeled as categories.

Other times, it is just obscure and perhaps confusing when you try to populate its properties or publish it. For example, is something like "Communication" really an Object Type? Communication can be a process, perhaps the output from a process (as in an objective realized); it may be the purpose of a Document, or it may be a Channel through which communication takes place. I would doubt, however, if it is a "thing" that can be modeled.

Association Types

Making sure Associations add value

There is clearly no right or wrong answer here. You need to explore the issue with the client. For example, after publishing, you can show the client just what a page looks like and ask the question: is this useful and if so, who will find it so?

Several times, I have seen clients decide to stop making so many associations and to concentrate more on genuine impacts and interdependencies, not just make associations all over the place.

Another "value check" is the consistency of associations. For example, Application objects associated to process objects at different levels (an alternative option might be to say that the application is never associated to a process at levels 0 to 3).

Association Types Naming Conventions

When you look at Model Explorer's Design View, you will see lots of Object Types that do not appear in the Object View. These are Association Types. If they have not been named using a clear convention, then it can be really confusing when trying to maintain the meta-model.

For example, the Casewise TOGAF meta-model uses this convention:

> *lnksourceobjectytpe*2*targetobjectytpe*

This is at least consistent, but why use this kind of condensed form? I tend to use this convention:

> *Association: Source Object Type – Target Object Type*

I have often seen meta-models where it seems no thought whatsoever has been given to this with the Design View full of obscure object types. It is often difficult to work out which two object types they associate.

It also causes problems if you need to:

- Add a User-defined property Type to an Association
- Enable Interface Associations

More seriously perhaps is that you must know the name of an Association Type if you want to

- Maintain their properties using Auto Modeler
- Use associations in c4w documents

There is scope for improvement here (and potential consulting work):

- Change the Association Type names (part of a wider meta-model improvement exercise)
- Proper meta-model documentation
- Default Names as a special case (e.g. Reason for Involvement, Data Usage)

By way of illustration, short (1-3 hour) workshops could also be done on the role of Association Properties (practical usages), covering one or all of the following:

- How they are displayed in CP
- How they have powerful capabilities in c4w output
- How you can populate Association properties using Auto Modeler

Association Descriptors

Association Descriptors are the words you see in brackets in published output (an Association descriptor can be left blank). For example, on a Process page, you might see an association to an Application that has been set up like this:

> **Applications (uses)**
>
> Here there is a list of associated Applications.

The word "uses" is the "descriptor". My own view is that such one word descriptors are meaningless to an end user – the question begged above is "what uses what?" For It people, a "process" is something a CPU does, so they may see the Applications as using the process you are looking at.

Therefore, from a publishing perspective, I try to adopt a convention that spells out unambiguously and in plain English exactly what the relationship is, such as:

> **Applications (that are used by this Process)**

Then, on the Application page, you see:

> **Processes (that use this Application)**

Now a user, who is at this point reading about a process or an application, can clearly see the context (and the dependency a process has on applications).

Some clients, who are IT-oriented, can be resistant to this. The TOGAF meta-mode for example uses these one-word descriptors and when I wanted to change them to plain English, I was prevented because they would then make the model "less than a pure TOGAF model"!

However, if one of the model's purposes is as a knowledge base to be used by a wider employee community, I consider this be absolutely crucial ("information for consumers rather than producers").

Another practical issue is that (I find, at least) most associations are made in batch via Auto Modeler. These one-word descriptors mean that I often find that users load them the wrong way round. So, though this issue may seem somewhat obscure, I consider it one of the most important things in a meta-model that aids usability (of the product) and communication (of the output).

Associations between the same Object Type

This area is problematic. For example, I have seen models which have tried to set up a kind of "See Also" association between instances of the same Object Type. For example:

> **Processes (see also)**

There is no direction (next/previous, up/down, upstream/downstream, parent/child, depends on/dependent on,) in this association. This causes problems with the published output, where you will see two identical headings:

> **Processes (see also)**
>
> *List of processes*
>
> **Processes (see also)**
>
> *Another lists of processes*

However, to complicate matters, you will not always see two headings; it depends on whether the process you are looking at has associations where it is both source and target object. For end users of published output, it can be confusing; it certainly looks bad and unprofessional. (It was a problem in the old Compliance Cartography model that tried to map objects of the same type between different Standards so we had an ISO Control mapped to a SOC control through these See Also associations.)

My advice here is to never use them unless you can identify a direction or hierarchy (e.g. using the above Compliance example, could we say that "SOX" was "downstream" of "ISO"? Probably not; but another option here is to simply use the parent model's name, so we could set up this association as follows:

> **Controls (that support SOX)**
>
> List of associated SOX Controls
>
> **Controls (that support ISO)**
>
> List of associated ISO Controls

A similar issue plagued one client I worked with regarding the use of the Organization object. They had around 7 categories and 7 association types that modeled things like:

- Roles who participate in a Team
- Manager *who manages a* Team
- Teams *in a* Department
- Roles *in a* Working Group
- And so on

The problem was that, in the published output, they were getting what appeared to be meaningless results – things were associated to the wrong thing (e.g. Manager being managed by a Working Group).

Essentially, my analysis was that here they really should have used separate object types (not necessarily 7, but trying to overload the Organization object like this was not really useful – also, a Role and a Working Group, being inherently changeable, could well have been separate object types, say Role and Actor – to use TOGAF concepts).

What had happened is that because no real attention had been made to directional association descriptors, many associations had been made using the wrong one.

As that most of these Organization objects had been diagrammed, so it was not practical to re-design the meta-model.

The solution, therefore, was to put the effort into making the descriptors as clear as possible; some examples:

- Organizations (Roles that make up this Team) – Organizations (Teams that this Role participates in)
- Organizations (Manager for this Team) - Organizations (People that report to this Manager)

And so on. We then deleted all existing associations and used Auto Modeler (whereby the column headings must match exactly these descriptors) to implement the new associations. It also made the Organization's Associations menu almost impossible to use (it now had 14 associations to itself!) Therefore, as a means of working, it was crucial that these associations were maintained only through Auto Modeler.

It also followed from this that the client needed Modeling Standards that explained that Organizations of a specific category could only use specific Associations (unfortunately, this cannot be enforced in the software).

Categories

Supplied categories vs user-defined categories

If a client has started modeling using the supplied Casewise models (e.g. The Casewise Framework or The Standard Model), they will have - literally - 100s of irrelevant and meaningless Categories at both the Diagram and Object Level.

This is based on several (incomplete) initiatives from the past. In 2000, Casewise produced an "e-commerce" model. As this was before user-defined Objects, the Process object was used to represent "web pages" – hence the load of Process categories, which refer to web page types, but which simply confuse people. The first thing to do therefore is delete them.

Secondly, in 2002, Casewise created the "Casewise Framework" which aimed to combine the old "Catalyst" concepts (Business Dynamics Model, System Dynamics Model, etc) with the Zachman Framework. This is the basis for the collection of Diagram categories you see in the "Casewise Framework". These may be useful, but most are probably not.

The point I often try to get over to clients is that the use of categories is under their control: they do not have to learn what all these "methodology artefacts" mean or how to apply them. Simply, if they do not mean anything to them or their business, then delete them and create a set that are meaningful (whether that be "Process Flow Diagram", "End-to-End Process Diagram", or, as in BPMN, just "Business Process Diagram".

This applies equally to the legacy Catalyst concepts regarding Processes. How useful are the concepts of "Derived Logical Process" or "Elementary Business Process"? The key point here is that the use of categories is a good way of communicating business concepts to end users using terminology they can identify with.

Using categories to indicate "levels" within a model

Another thing to consider is the use "levels" in categories. Many people get hung up on whether something is a level 1 or level 2 Process or diagram, so it is often better to avoid them. However, levels do work if there if a rigorously applied hierarchical decomposition, but they become misleading if the model has more of a web-like structure, with parent objects (i.e. those that have child diagrams) appearing on many diagrams at different levels in the hierarchy.

With hierarchical models, it can be useful in published output: being told you are on a level 5 diagram is a useful orientation cue when navigating the model. One thing to avoid is a situation whereby a Level 3 diagram (say) has a Level 2 as a child diagram (which is surprisingly common).

Another problem is whereby an object has multiple parents: e.g. in a "Business Capability" model I worked with once, a "Capability X" was re-used as child of both a level 3 and a level 5 parent Capability. As such, it could not be both Level 4 and 6, causing some inconsistency in the data. The options I gave to the client were:

- Identify its true parent and stick with that
- Consider whether it really should be reused and create two different child Capabilities
- Give it a new category that is generic rather than related to a level
- Do nothing on the basis that it’s too minor to be an issue

Again, we can see that there is no right or wrong answer here; the objective is alter the user and get them to take ownership of the quality of their own data.