Fig. 1

1. Requirements and Validators

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

- When do we write MUST in a standard? And when should we?
- What do validators have to do with that?

A variation on the blog post at
https://blog.g-vo.org/requirements-and-validators.html

(cf. Fig. 1)

2. Take-home Message

A MUST in a standard has significant costs.
Have a good reason for it.

3. Cost?

ObsCore 1.1 (p. 23) says:

Values in the obs_id column must not be NULL.

stilts tries to verify this by running:

```
SELECT TOP 1 obs_id FROM ivoa.ObsCore WHERE obs_id IS NULL
```

That's a sequential scan of the *entire* table if there is no suitable index on obs_id. For services that have no use for obs_id because they are using datalink for what obs_id was designed to do, creating such an index can be rather annoying (obscore can be a view).

Sure enough, my TAP service started to fail with:

```
I-OBS-QSUB-5 Submitting query: SELECT TOP 1 obs_id FROM ivoa.ObsCore WHERE
E-OBS-QERR-1 TAP query failed [Service error: "Field query: Query timed out
```

4. MUST is a Flag for Validators

If people write a validator, they scan the standard text for ocurrences of *must* (in any capitalisation). Every *must* is work for them, and it is prone to blow up as in the obs_id case.

So: If you're writing MUST in a standard, at least hesitate briefly and think if you can do without it. A similar, although less dramatic consideration applies for SHOULD. A validator might still want to check these constraints (and issue warnings if they are not met). But since nothing should break if a SHOULD is not met, validators could take it easy for those.

5. Valid reasons for MUST

Common trait: Something important breaks if the MUST is violated.

- Internal requirements: rules imposed so machines can do their job.
- Functional requirements: rules imposed so results "make sense".
- Hedging against over-clever adoption
- Hedging for the future: rules imposed so a standard has room to grow.

6. Internal Requirements

Example: when asked properly, the vocabulary repository MUST return a json with a certain structure.

If it doesn't the client can't make sense of anything and will not receive the semantic information requests.

Example: ivoid MUST be a primary key in RegTAP's `rr.resource` table.

If it's not, you can't have the foreign keys in the other tables, and you could not obtain more metadata for a resource in those other tables.

Internal requirements are usually uncontroversial: it's hard to argue with code after it has crashed.

7. External Requirements

Example: In LineTAP, the clients should produce labeled markers for spectral lines in a fairly cramped space (the spectrum). Hence, result tables MUST contain at least the location and a suitable marker string. There's a presupposition here that machines cannot come up with reasonable labels themselves; but that presupposition certainly is safe to make for the next few years.

Example: "All concepts MUST have an English-language preferred label" in Vocabularies. If they didn't, clients would show empty labels to users.

These are a lot harder to argue about; you will need well worked-out use cases if you want discussions to go anywhere. In both cases above an advocatus diaboli could find hooks for dissent. For instance, in the example with the label they might say: "What's wrong with just showing the identifier? Or the whole concept URI?"

8. Hedging against Cleverness

Example: SKOS vocabularies can give both wider and narrower properties, but clients MUST NOT rely on the presence of the narrower ones.

This kind of thing typically happens when you allow more than the minimal information set.

In the end, that's more implementation advice than a requirement. Careful readers of the standard would work out themselves that they can only rely on wider and should hence effectively ignore narrower. But since people usually implement against concrete artefacts and not the (pure) spec, they might be tempted to go for narrower anyway. They make get away with that on the one vocabulary they're testing with. And their code will break when they move on to the next.

By the way, if you find yourself hedging against relying on non-mandatory extras, it may be wise to consider outlawing such tempting features; but that needs to be weighed against locking people in too much and blowing compatibility to external standards.

9. Hedging for the future

Example: UCDs in Obscore or SSAP – they have no role in the protocols themselves and just *may* come in useful if non-protocol clients read the tables.

Example: Everything related to actual W3C RDF in Vocabularies – this only becomes relevant when some non-VO client wants to consume our vocabularies (or our clients want to do so using standard RDF tools).

Be very careful with these: These will certainly break without a validator, because for what clients do now, they clearly don't matter. In fact, there are quite a few errata to obscore and SSAP fixing broken or conflicting UCDs – these never registered anywhere because nobody ever needed these UCDs.

In the end, with future-hedging requirements, validators may spend a lot of effort on something nobody ever needs.

On the other hand, it of course sucks if liberties left by standards are used such that they block further standard evolution. Finding the balance between overengineering for futures that never come and underspecifying to the extent that certain implementations prevent further evolution of standards again is an art.

10. And Validators?

If things break when a requirement is not met, then validators checking **all** such requirements are **essential** for correct implementations.

Example: ACPI (in Intel boxes' firmware) – computer vendors only test against a single client (the Windows version they want to sell the box with). In consequence, *all* computers I've ever owned had rather severely broken ACPI.

11. What to validate?

- Internal Requirements: Medium importance. At least in reasonably compact standards, it is likely that clients break when these are not met, so often the implementors will not need the valdiator to tell them things are broken.
- External Requirements: Higher importance. Here, clients will typically fail in non-trivial ways that may only be noticed once you actually try to *use* the data – which implementors rarely do.
- Hedging against cleverness: Most of the time not validatable.
- Hedging for the future: Highest importance. Since nobody uses such things, they will certainly break unless validated. Nobody gets things right on the first attempt.

12. Parting wisdom

The smaller the immediate impact of failing a requirement. . .
. . . the higher the importance of validator.

Corollary: With non-immediate MUSTs you put more load on the validator than with immediate ones.

Hence, think twice: will that requirement make your client writers' (ok, and other implementors') lives that much simpler?

Oh, and please review
https://wiki.ivoa.net/twiki/bin/view/IVOA/ObsCore-1_1-Erratum-3