WISHI

2019-11-07 call:

- IETF 106 hackathon planning
- Data model versioning
- Continue OneDM discussion from the Kista work meeting

— Notes: https://hackmd.io/AOp8fkqjQJ2vYygIPOBLKQ?edit

— WebEx: https://ietf.webex.com/ietf/j.php?MTID=mcd31ffd95bba01534e8cf88aefa529f4

Versioning

Solving all the problems — badly

(and creating persistent terminology confusion)

Software versions

Software gets improved over time, new releases installed

— Which release do I have?

➔ Version number!

Fiction: Development runs in a linear progression

Library versions

Version numbers are not just for humans

— describe an **interface** (obvious in libraries)

Independent evolution of interdependent entities

Kinds of changes

— Additions/Extensions

— Reinterpreting existing interactions

Backward compatibility
— Interoperability with legacy [systems]
— may need to replicate known flaws (bug compatibility)

Forward compatibility
— Tolerate evolved input
➔ enables extensibility

Interface vs. Format compatibility

Interfaces: Set of interactions, a protocol
Formats: A reduced interface with one interaction (convey)

Indicated vs. Negotiated Evolution

— Indicated: producer declares, consumer has no say

— Negotiated: Both sides have a say;
producer adapts to consumer

Versioning

Originally: project evolution to a linear number space

Purpose in format evolution:
prevent false interoperation

Semantic Versioning (semver)

From library versioning:

— Major: Prevent false interoperation
— Minor: Add features (backwards compatibility)
— Patch: Should be inconsequential

Expresses intent (but note that there are bugs!)

Versions vs. features

— Backwards compatible features (ignore unknown)

(Minor semver = roll-up of such features;
works in single-implementation world)

— Must understand features

(Major semver = roll-up)

Version: Single number (or github commit?)
Features: Set of identifiers

Example: HTTP

1.0 was baseline for the 1990s

— Lots of features added as header fields
— ignore-unknown
essentially attained feature-set of 1.1, but cruft accumulated

1.1 was a major (!) revision

— reinterpreted some of the header fields, roll-up of features

2 was a complete replacement of a layer (3 will be, again)

Example: HTML

Very different kind of ecosystem: oligopoly (especially since the 2000s)

Backward compatibility (new browsers can show old sites) is must-have

Forward compatibility (old browsers can show new sites in degraded form) crucial in short term
— lack of such needed as browser upgrade motivation

Guidelines for format evolution

Background: multiple implementations ➔
different features are introduced in different timescales

➔ Use features (ignore-unknown) instead of versions

➔ Use must understand features for moving forward

Strong disincentive to roll-up feature bundles into new versions: Creates flag day between producers and consumers
➔ May be desirable eventually, as a coordinated clean-up

Model vs. format evolution

Similar considerations apply

May want to expose feature set in self-description
Version as a roll-up may be attractive sometimes

Versioning vs. must-understand to break false interoperation
Composition, inheritance, enhancement…

Indicate evolution of components?

Evolve model vs. evolving instance (e.g., typo fixes)

Profiles

roll-up sets of features for a subset of users
(which subsets might overlap!
➔ need conflict-free profiles)

might in turn have a feature name (composition)
levels vs. profiles; strictly nested

ZigBee alliance: Profiles do overlap; get rid of profiles

Deprecation

explicit deprecation/"replaces" relationship
instead of:
implicit deprecation in linear version progression
(may want to indicate edge on both vertices)