A Brief Note on Common Semantic Models, Common Information Models, and Interfaces

William Cox
November 30, 2009

Interfaces

For the Smart Grid we are working to define interoperation between actors, frequently in different domains. Specific domains have developed interoperation on their own—building automation systems can work together through specifications such as oBIX or BACnet that provide an integration layer.

Soup-to-Nuts

Many standards and modeling efforts try to do models of everything—not just what's in the problem domain, but of everything that touches that domain. For example, IEC 61850 has weather and web services definitions.

This generally fails at some point, typically when the domain's definitions conflict with a well defined and evolved other domain.

Deep Interfaces

Within a domain interfaces may be quite deep, including details of a service or devices' interactions that are important to effective interoperation. For example, phase relationships, management of co-generated power from an industrial facility into the distribution and transmission grid, and so forth.

Where each actor in an interaction requires a detailed knowledge of the other side, we say there is a ***deep interface*** or ***deep integration***.

Direct load control, where a utility manages specific devices beyond the premises gateway, is a deep integration—much detailed knowledge is needed to do an effective management job, including knowledge of what *not* to do. Don't cycle the air conditioning compressors too quickly. Don't turn off a compressor that's at a certain stage of operation. Don't turn off the washing machine when there's bleach in a load. And more, limited only by the complexity of what is controlled.

Deep interfaces unfortunately are likely to be ***brittle***—a change in the interface will break everything that uses that interface. This is a key reason that integration and more flexible interfaces are used to solve cross-domain and many within-domain problems.

In enterprise-level software deployment, it is unlikely or impossible that all the software versions in a system of systems can be updated simultaneously. Similarly, in the Smart Grid, we should not believe in simultaneous upgrade.

So interfaces need to have some flexibility—compatible evolution, versioning, supporting multiple versions simultaneously, supporting a simpler interface that evolves more slowly—for use in the Smart Grid.

For each brittle, deep interaction, there may be some specific and limited information that is critically important to what is conveyed—perhaps a signal, a message with only key information. For example, instead of detailed load management information, tell a premises gateway to reduce load due to a grid emergency. The information exchanged is at a ***shallower*** level.

Two Domains

When we support interoperation across two domains, we might take one of three approaches.

(1) Define an interoperation gateway to translate information from one domain to the other
(2) One side or the other says "use mine"—for example when integrating with Wal-Mart or General Motors suppliers used what they were told to use, or alternative suppliers were found.
(3) Create a simplified interaction model to allow simpler interoperation (without transferring unnecessary information)

Three or More Domains

When we have three or more domains the possible solutions are similar:

(1) Use more translation gateways
(2) Integration hub to do the translations because it's now more complex
(3) "Use mine" (doesn't work well with three parties or more)
(4) Create a simplified interaction model

The integration hub failed in the eCommerce world because of the complexities and the ill effects of central points of failure. Of the choices available with two or more domains only the simplified interaction model has the best chance of working long term.

The Internet Terminal

This lesson was learned many years ago in defining the Internet terminal interface. Rather than each system needing to support all possible connected terminals, a single interface was described; then each system could support the terminals it needed, not the entire universe of possible terminal. This avoided a fundamental problem of resource-sharing networks, the issue of how you talk to non-local resources.

This is essentially the same solution as a standardized integration interface—make the interface at a simple enough level and any systems and software can support it. Make it as complex as the least simple system and evolving the whole is very difficult.

Why a Common Model?

A common model, whether it's called a *common information model* or a *common semantic model* for the Smart Grid is sometimes appropriate for development work inside a domain. Need the UML model for your generator? Use the common, possibly standard one. Need to integrate with a switch? Use its standardized model if it exists. And as long as we understand that the model may be too general or too specific we can work with it. (When it's just plain wrong for a new or evolved technology we have a much bigger problem—if a standardized common information model is necessary, how do you opt out or update the standard? And what's the timeframe?)

The term "Common Information Model" is used in many different standards and technology domains. The IEC 61968 Common Information Model is large and complex. As something large and complex, expertise is required, and evolving it is difficult and time consuming as not all interactions are clearly visible.

Often the phrase "Common Model" implies the deep integration we've discussed above. With a layered approach, it is possible to create common models that are flexible and light (such as a model of a well-developed integration interface), but all too easy to include too much detail, or cross too many layers, making the model brittle (and fragile).

A specific case in point is the call for a Smart Grid-wide "Common Semantic Model". To exchange information across an interface there must be some degree of common understanding. But how much is enough? The tendency of we engineers is to include more detail than necessary.

By including more detail than needed, as technology evolves or takes unanticipated directions we discover that we have created brittle interfaces. Avoiding brittleness requires discipline and significant care in design and implementation.

One misconception is that UML models are inherently fragile. They're fragile only if we model at the wrong level, and we unintentionally create exchanges that are too deep. So the problem is not with UML but with applying discipline to use UML effectively.

Another misconception is that we can take a detailed UML model, make changes, and push a button to generate new interface and data exchange code. But this fails because all the independently developed implementations to the original model will break. One cannot control all of the actors in a distributed multi-domain system.

Why Not a Common Model?

We speak of integration interfaces and models at "the appropriate level." But what is that "appropriate level"? How can we know?

Improvements by deeply defining Plug-in Electric Vehicle interactions (which can be defined primarily as a composition of DR loads and DER inputs) don't help solve the DR and price communication aspects of a vehicle—what if those deep soup-to-nuts models define price yet again and DR yet again? And what happens as they evolve?

Trying to integrate two things in different domains, each of which has its respective and relatively inflexible common model, is very difficult. Conformance to two different models defined at a deep level is an over-constrained problem, and is unlikely to have a satisfactory solution. "Use Mine" doesn't help, though it's a certainty that SDOs in the respective domains will say exactly that.

If the two-model problem is not quite impossible, the problem with three or models most likely will be. As we build a generative transactive grid, with applications and solutions we do not envision, we cannot afford the overhead of integrating with multiple deep "common models" at every interface.

Conclusions

If we thoughtlessly leverage what we have without attending to shallow integration, we'll be in a continual series of deep integrations. So-called models of everything don't have a good track record; a detailed semantic or information model of the entire Smart Grid suggests continual deep integration and its attendant problems.

Detailed, deep common models tend to drive stasis. Should we be investing in a fundamental misunderstanding of how to connect software services to each other? Service-oriented architectures and object-oriented programming have succeeded best where the object size is large and the interface complexities are low.

These integration lessons were learned at great expense in the software world. Let's not relearn them at great expense in the Smart Grid.