API Specifications and SDKs: The New Standards DevOps Dance

By Dan Pitt, Senior Vice President, and Bithika Khargharia, Director, MEF

MEF has recently begun releasing LSO (Lifecycle Service Orchestration) application programming interfaces (APIs) in the context of its MEF 3.0 framework, designed to enable orchestration of dynamic communication services across a global ecosystem of automated networks.

The way in which the LSO APIs are described and released represents a radical transformation of the standards process to reflect the new reality of telecommunications: that many of the important agreements are implemented purely in software. As we know, software can be modified and updated in minutes, whereas hardware (especially chips) can require many months or sometimes years to update. Interacting parties must still spend time to discuss and agree on objectives, constraints, and mechanisms of standardized solutions, but actual development, and the process to arrive at high-quality agreements, can be much faster when agreements are implemented in software.

In the LSO Reference Architecture (MEF 55), four sets of east-west APIs and three sets of north-south APIs describe reference points between five abstract entities, four vertically arranged within a carrier's domain and one representing the customer, with differences in the east-west APIs between a customer and a carrier and between two carriers. These APIs convey information regarding status or commands between one abstract entity and another in a manner that insulates one entity from the specifics of the other. We document these APIs in three forms: English prose (including use cases and business requirements), formal descriptions (mainly information models), and code (mainly data models, Swagger API definitions, and example software development kits, or SDKs). We bundle the first two into interface profile specifications (IPSs), which are standards documents, and we expect to certify realizations of these by operators and vendors.

The role of the information model is to capture high-level concepts and constructs, like products, services, and resources, and their relationships and interactions, without getting into any protocol-level details. A data model is the protocol-level manifestation of an information model. We typically use YANG for a data model with support for other data models in the future.

A developer implements the data models and not the information models. Some projects use an open-source tool chain to generate data models (with some manual tweaks) from information models and Swagger from data models. We are moving toward a MEF-supported open-source integrated development environment (IDE) that can string together all the open source tools, thereby bringing more automation into the generation of software artifacts from information models. Good tool chains enable the automatic generation of running code, which speeds development, improves quality, and reduces ambiguity. Thus, they assure that models actually describe generalized functions, not merely one-off products, thereby fostering greater interoperability and a more competitive marketplace.

The value of producing both specifications and code is twofold. First, the example code provides experience to developers of the abstract entities and can seed development of commercial products. Second, when the community iterates on the code (in an open-source repository) we learn the good (and bad) features of the API and can improve the IPS before we issue it in final form. Both the development of the code and the feedback loop to the IPS require new ways of working between the traditional standards community and the software community, even within a single organization like MEF (or a single company).

In MEF, our members invest significant resources in committees, which produce the IPSs (the standards), and most projects are initiated in one of these committees in what we call the "standards track." Others increasingly are initiated in a rapid-prototyping track, which means as a MEF implementation project (often in MEFnet, our cloud-based development environment), as a proof of concept using code that can be proprietary or open source, as a temporary project in the office of the CTO, or as a project in one of our semiannual LSO hackathons. When sufficiently mature (meaning the market needs it now), the code is released and the project transfers to the standards track for issuance as an IPS. We have work underway to develop a "software standardization process" in the standards track.

The standardization process today includes standardization of documents (mainly the IPS), but in the future it will evolve to include standardization of associated software artifacts through a similar peer review process for Github content that is followed today for documents but reinforced with prior community buy-in of the code.

Regardless of where the work is initiated, we strive to feed experience back and forth between standards and software, and this we are learning to do as we go. Our goal is to improve the standards with the experience gained from the prototyping exercises and to improve software implementations with the experience gained from writing up the standard. This is a constant and iterative process pre-release and post-release of the artifacts of both types.

Roadmaps and release dates associated with the publication of a standard are indicative of an MVP (minimally viable product) mindset for standards that has typically existed only for software. As much of telecommunication and networking become software-defined, we see this DevOps between standards and software as the path of the future. We are charting that path as we speak, and while we do not claim it to be painless (as is the case for most forms of radical change) we see it as inevitable and highly beneficial to the agility of the industry and the efficacy of the standards community.