

**IN MODERN DEVELOPMENT ENVIRONMENTS, INNOVATION MUST BE BUILT IN EXPLICITLY AND NOT EXTERNALLY IMPOSED.**

Successful cloud computing methods are characterized by their intrinsic utility, capacity for highly scaled implementation, and ability to adapt to rapid change. These methods can be classified for practical purposes in terms of APIs, protocols, languages, and tools.

Classic approaches to producing standards are as archaic when applied to cloud computing as last century's lighting, transportation, and communication systems are to the rest of our society's infrastructure. For standards to work and be suitable in this new setting, we need an approach that promotes rapid feedback and simultaneous or near-simultaneous development and implementation.

In earlier columns, I've explored the role of communities in developing cloud standards and laid out the landscape of the organizations operating in this space. I've also argued that standards are part of a continuous spectrum of development that ranges from purely practical to purely theoretical end points, and that a standard can be defined lightly as "anything agreed to by more than one party." More formal definitions are certainly possible, and I've also discussed the various types of standards organizations and the importance of defining our terminology precisely to understand this spectrum of development.

This time, I'll compare and contrast the different types of cloud software components, and discuss the pros and cons of taking a combined development plus operations ("DevOps") approach to accelerate progress on software and standards. I'll focus on practical ways in which standards fit into familiar categories used by programmers on a day-to-day basis, and on how rapid feedback can improve them for use in these settings.

**Cloud Development Categories**

For convenience, we can organize the components of cloud software and associated methods into broad categories. I present one such classification here. It isn't the only possible scheme, and might not satisfy architecture purists, but I've simplified the discussion to focus on current cloud computing trends and needs. The point of this classification is to expose

# Cloud Standards and the Spectrum of Development

features that relate directly to current innovation opportunities and to discuss the consequent need for standards development methods that can keep pace with rapid software progress.

**Application Programming Interfaces**

APIs have emerged as a key feature of the new cloud ecosystem. They've become so popular that they're sometimes the only components of cloud software design that beginning programmers encounter, and such beginners can be forgiven for thinking that these are the only components of

**ALAN SILL**

Texas Tech University,  
[alan.sill@standards-now.org](mailto:alan.sill@standards-now.org)

cloud software that matter. APIs dominate current discussion to the point that they've developed their own conferences, trends, and place in the economic landscape. Nonetheless, they aren't the entire picture and can't stand on their own. We'll have to delve deeper to understand their history and relation to other important cloud functionality and features.

APIs represent boundary-level conditions needed to transfer information into and out of cloud software environments. Classically, these environments

As the popularity of cloud computing grew, the API paradigm became so useful that almost all cloud software developed APIs, even if they weren't interfaces to "applications" in the classical sense. As this occurred, an evolution took place in the design of APIs and their use. Historically, APIs were often specific to the actual programming languages used and weren't generally interchangeable between different language calls to the methods. In cloud computing, the dominance of Web-based models and of their formal

entirely from a compact description of the interface, the Web Services Description Language (WSDL), without reference to any other knowledge of the interface's characteristics.

This important development played a crucial role in getting programmers to think of APIs as potentially language-independent constructs that could be useful by themselves. Despite the obvious value of the language independence afforded by WSDL and SOAP, programmers eventually rebelled at the XML-only basis and prescriptiveness of these methods. Although they're still in use in a variety of Web services and enjoy a strong following for certain types of programming, many of the new features of cloud methods have transitioned to the REST paradigm.

This style change has been driven partly by the desire to be able to refactor services in different ways to span smaller or larger portions of the problems to be solved, and partly by the need for control to define the boundaries of the portion of the system exposed through an API. One of the defining characteristics of cloud computing is flexibility to draw this control boundary in ways that sometimes cross the conventional norms of service-oriented architecture.

Modern API design for cloud computing often uses design principles in ways that are beginning to resemble formal guidelines that lead in the direction of standards, or even to be expressed formally as standards. Associated tools are emerging to allow APIs to express discoverability of functional features and to build in self-description of their characteristics and methods of use.

Examples of API format description and definition tools that include open formal specifications in addition to related implementation software in-

were executable applications that were incapable of exposing their internal processes or parameters to the outside world for alteration or external consumption, hence the need to pass input, output, and control features through a defined interface. The other major category of boundary-level interfaces used in computing in general is often referred to as *application binary interfaces* (ABIs). I'll reserve discussion of ABIs to a later column.

In general, an interface defines features such as syntax, semantics, and optional versus required components of the information to be presented. In general use, APIs and ABIs also describe characteristics of the programming call sequences, such as classes and details of the methods to be used and objects or information to be exchanged.

service-oriented architecture underpinnings produced allowed cloud APIs to be used across several different language implementations.

APIs based on the Representational State Transfer (REST) design pattern now dominate designs currently employed in new cloud software. It's worth noting, however, that earlier progress in decoupling APIs from dependence on specific language call interfaces and methods was driven by the previously dominant method in service-oriented architecture design, which is that of the pattern introduced in the late 1990s as the Simple Object Access Protocol (SOAP).

Web services based on SOAP and other closely related methods used XML to define interfaces so formally that code could actually be generated

Successful cloud computing methods are characterized by their intrinsic utility, capacity for highly scaled implementation, and ability to adapt to rapid change.

clude Swagger (<http://swagger.io>), API Blueprint (<http://apiblueprint.org>), and the RESTful API Markup Language (RAML; <http://raml.org>). These approaches can be used to encourage or enforce API self-documentation and structure. In addition, a wide variety of related open source and commercial software has emerged to provide manageability, analytics, and other features, sometimes provided externally by third parties as filters or add-ons to existing APIs.

## Protocols

Protocols are another important component of cloud methods. No matter how well described, the interface (API or ABI) can't exercise all of the functionality needed to control and interact with running processes alone. A complete approach to online communication also needs protocols to define and describe the sequence of operations, format, and sequence of bits "on the wire" and characteristics such as timing, content, or other design principles that govern the information to be passed through it.

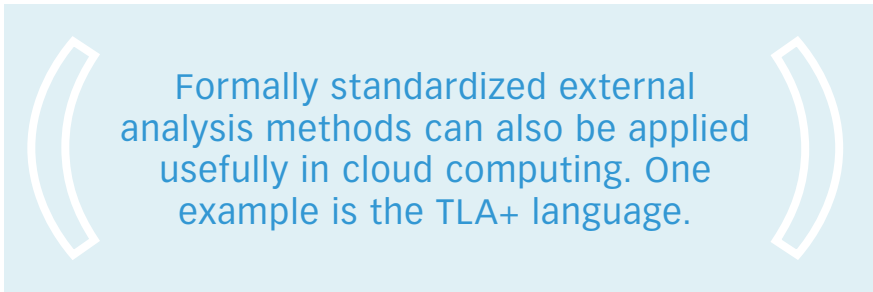
Protocols can be distinguished from APIs by the degree to which they specify interrelationships between different aspects of the information to be presented, and often the time sequence, content, and/or ordering of data and operations. Protocols also cover address and data formats, and the mappings that are needed to interrelate them. They can express subtle characteristics of sequence and flow in ways that are difficult to express purely within the context of an API. TCP/IP, which governs most operations on the Internet, is a good basic example.

Unlike APIs, protocols were originally designed to be as independent of language implementations as possible. Because APIs have recently become in-

creasingly language independent, protocols and APIs are now often designed in tandem, and many current cloud standards are written with both protocol and API components. The Open Cloud Computing Interface (OCCI), for example, defines a boundary-level API and protocol for RESTful control of cloud computing components existing within the boundary of the system to be controlled. Other standards sometimes concentrate on one or the other of these aspects, or on specific details of control and communications.

## Languages, Tools, and the Overall Development Environment

Cloud implementations are written in a variety of programming languages with methods that are supplemented by an even wider variety of tools. It's easy to ignore that these languages and tools are themselves often organized and defined by standards and divided into releases and versions. Use of languages and tools follows a pattern with wide variation in terms of size and type of supporting organization, and solutions with a single person underlying the ap-



Formally standardized external analysis methods can also be applied usefully in cloud computing. One example is the TLA+ language.

REST-based standards and models that use HTTP as their transport protocol can be further distinguished from each other by their use of hypermedia, which is an essential feature of modern cloud API usage that depends on the detailed nature of HTTP.

Protocols are generally used in organized versions, so are best developed as standards. This aspect of cloud development is easy to miss, because it's essentially taken for granted that good protocols will be used to organize the communications handled by our APIs. Organizations that develop protocols include all of the major standardization bodies, such as the World-Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF), and essentially all of the organizations covered in previous columns.

proach aren't unusual. There is only space in this column to touch lightly on this topic.

The concepts of interoperability and scalability have made the distinction between different types of languages and tools largely irrelevant by design as a deliberately targeted feature of cloud solution deployment. It's taken for granted that a successful cloud infrastructure won't depend unduly on features of the programming methods used to create and implement it. This aspect is almost a design requirement for modern cloud development.

Formally standardized external analysis methods can also be applied usefully in cloud computing. One example is the TLA+ language, specification, and tools developed by Leslie Lamport (see <http://research.microsoft>

.com/en-us/um/people/lamport/tla/tla.html). Using mathematical set theory and predicates, the TLA+ approach describes the legal behaviors of a system. Amazon has recently used TLA+ along with similar methods to find and eliminate problems due to time sequencing, dependencies, and design flaws within its software and infrastructure.<sup>1</sup>

Because of the importance of networking to the successful deployment of cloud-based systems, a wide variety of work is ongoing to develop new stan-

machine and prebuilt custom container images. These are clearly not the only methods of software development and distribution, but they've gained importance over the last several years.

## A DevOps Approach for Standards

Cloud computing emerged specifically to deliver methods for providing services that are easy to factorize and deploy, and can be implemented rapidly at greatly variable scales. Such a setting

improvement in any area, and is especially needed in cloud computing. The quicker the feedback, the quicker we can expect progress.

Unfortunately, earlier standards development practices were based on slower and more formal communication patterns that don't lend themselves to today's rapid progress and rapid cycling between conceiving new ideas and testing them in the field. To alleviate this shortcoming, we need to take a DevOps approach to bridge the gaps more quickly between formal ideas and practical implementation. In doing so, we also need to scale the communication patterns horizontally to involve more opinions and feedback for the betterment of the field.

One reason that OpenStack (<http://specs.openstack.org>) is making such progress now, for example, is that it has exposed its specification-writing process to community input and formalized the process of pulling resulting improvements into the project's core development, selection, and verification procedures. Other similar software projects, such as CloudStack (<https://cwiki.apache.org/confluence/display/cloudstack/design>) and OpenNebula (<http://community.opennebula.org/interoperability>), are also providing such information. This approach will be strongest if mutual engagement occurs between standardization communities and software developers in each project.

Engagement of this type is beginning to happen, and open source implementations of OCCI, Topology and Orchestration Specification for Cloud Applications (TOSCA), Cloud Data Management Interface (CDMI), Cloud Infrastructure Management Interface (CIMI), and other emerging cloud standards are now available in each of the above software efforts, as well as in general-purpose software libraries suitable

dards and languages that can be used to express the features of networks in cloud settings. I will explore this topic further in a future column.

The environment in which cloud methods are developed and used is as important to their success as the interface, protocols, languages and tools used to implement them. Because the cloud environment emphasizes advantages in scalability, on-demand deployment, flexibility in interoperation, and bridging between multiple levels of information, people working in this area prefer tools with the same flexible characteristics.

Open project and code repositories and software distributions laid the groundwork for the cloud environment, and lately this approach has been extended to include methods for public sharing of libraries of entire virtual

requires procedures that allow quick cycling and continuous integration between the development and operational deployment of cloud services. The industry has therefore adopted the widely popular DevOps strategy, which combines aspects of development and operations to speed implementation and testing of new solutions and application of new methods.

Although earlier computing models could have used this approach, cloud computing has several characteristics, such as ease of simultaneous side-by-side comparisons of performance and factorization of services, that make the DevOps approach particularly attractive.

This approach can apply equally well to standards. Identifying methods to feed input from real-world experience back to standards developing bodies is an important and necessary step toward

Cloud computing emerged specifically to deliver methods for providing services that are easy to factorize and deploy, and can be implemented rapidly.

for use in other settings. A quick search in the GitHub repositories will yield several relevant projects.

Where closed-source development still occurs, it also needs to be pursued in a way that encourages rapid cycling between ideas and implementations for standards to be effective in these settings. Some degree of interoperability can be extended to otherwise nonstandardized commercial products through associated open source projects. The Eutester project (<https://github.com/eucalyptus/eutester>), for example, can be used to automate tests of a Eucalyptus or Amazon cloud. Although no formal open consensus-based standards exist to provide the community underpinnings for Amazon-compatible products, projects such as Eutester can partially fill the gap between product features and their user communities.

### Consensus Versus Speed, and Rapid Testing as a Cure

The downside of taking a rapid-cycling approach aimed only at functionality is that it can place a great deal of pressure on the methods commonly used to develop consensus within open standards communities. Standards work best if they can be used to bridge the differences between projects to provide the basis for interoperation. Developing tools that can be adopted effectively in multiple software projects and in commercial products taxes our collective ability to coordinate and test new features in different settings and build the consensus needed to create effective open standards.

Such consensus is one of the five core principles recently enumerated by the OpenStand effort. Several major standards developing bodies, including the Internet Society, the IETF, the Internet Architecture Board, the W3C, IEEE, and the Open Grid Forum, have

endorsed its joint statement of affirmation (see <http://open-stand.org/about-us/affirmation>).

Despite this strong endorsement, other standardization organizations have departed from or haven't yet endorsed the OpenStand principles. Among these is the Web Hypertext Application Technology Working Group (WHATWG), an organization that formed a decade ago to pursue evolution of hypertext-related specifications and explicitly includes a non-consensus-based membership steering group partly justified by the professed need for speed in development. Consensus unfortunately takes time and can produce slow and variable results.

One way to mitigate these problems is to encourage rapid testing against major implementations, which can sometimes squeeze out opinions not backed by large-scale organizational participants. The divergence between WHATWG and W3C specifications for HTML is an example of the potential pitfalls in this area. Cloud computing needs processes to create open active communication between development of software and standards without encountering such difficulties.

**FUTURE INSTANCES OF THIS COLUMN WILL LOOK AT INDIVIDUAL STANDARDS IN TERMS OF CONCEPTUAL FUNCTIONS THEY CAN BE USED TO PERFORM, SUCH AS IMAGE PORTABILITY, JOB PROVISIONING, AND TASK ORCHESTRATION.** Meanwhile, the information presented in this column should help illustrate the use of standards in necessary components of day-to-day software development.

Please respond with your opinions on this or previous columns, especially if you disagree with me, and include any

news you think the community should know. You can reach me at [alan.sill@standards-now.org](mailto:alan.sill@standards-now.org). ●●●

### References

1. C. Newcombe et al., "Use of Formal Methods at Amazon Web Services," online publication, 29 Sept. 2011; <http://research.microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf>.

---

**ALAN SILL** directs the US National Science Foundation Center for Cloud and Autonomic Computing at Texas Tech University, where he is also a senior scientist at the High Performance Computing Center and adjunct professor of physics. He serves as vice president of standards for the Open Grid Forum and co-chairs the US National Institute of Standards and Technology's "Standards Acceleration to Jumpstart Adoption of Cloud Computing" working group. Sill holds a PhD in particle physics from American University. He's an active member of IEEE, the Distributed Management Task Force, TM Forum, and other cloud standards working groups, and has served either directly or as liaison for the Open Grid Forum on several national and international standards roadmap committees. For further details, visit <http://cac.ttu.edu> or contact him at [alan.sill@standards-now.org](mailto:alan.sill@standards-now.org).

