Programmatic Interfaces for Web Applications

Tomas Vitvar
Oracle

Steve Vinoski
Basho Technologies

Cesare Pautasso
University of Lugano, Switzerland

eBay's launch of its API in November 2000 marked the beginning of an era in which Web applications offer services for third-party application integration. The rapid growth of programmatic interfaces for Web applications has recently revolutionized online content integration and created new opportunities for vendors to build developer ecosystems. According to ProgrammableWeb, a leading service and mashup directory (www.programmableweb.com), the number of open Web APIs has steadily increased since 2008 (see Figure 1). Although it took eight years to reach 1,000 APIs in 2008, and two years to reach 3,000 in 2010, it took only 10 months to reach 5,000 by the end of 2011 (http://vitvar.com/events/aaai-ss12/slides/jmusser-keynote.pdf).

Today, programmatic Web interfaces have become a core feature that developers expect from any Web application. Services let application vendors track who is using applications, as well as how and why, allowing them to more effectively foster affiliates and drive traffic back to applications. By offering services, vendors provide application functionality and start charging for it. A small fee per service call is acceptable for many subscribers, while big opportunities exist for profiting from usage volume. The volume of API calls Google and Facebook report exceeds 5 billion daily. Twitter, meanwhile, reports more than 13 billion API calls per day, with 75 percent of all Twitter traffic coming from third-party applications via the Twitter API (www.slideshare.net/raffikrikorian/twitter-by-the-numbers).

Challenges and Opportunities

The increasing popularity of programmatic Web interfaces and the growth of third-party applications that use them raise questions about how developers should design services and maintain those services' levels of performance and scalability. Of the services in the ProgrammableWeb service directory, 75 percent claim to use REST and 25 percent use SOAP, XML-RPC, and other technologies. REST APIs, or RESTful APIs implemented with HTTP, inherently adopt Web architecture principles

1089-7801/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

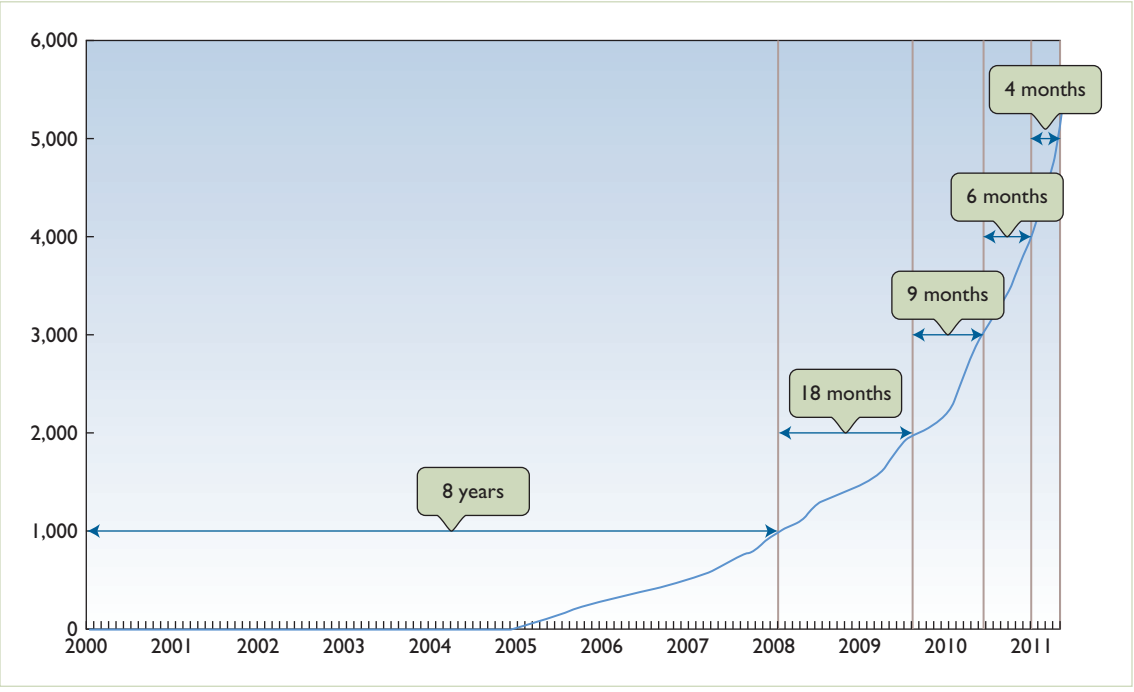

Figure 1. Total APIs over time. The number has steadily increased since 2008. (Figure courtesy John Musser, from http://vitvar.com/events/aaai-ss12/slides/jmusser-keynote.pdf; reprinted with permission.)

and can exploit already existing Web technology. It isn't unusual, however, for APIs claiming to be RESTful to actually fail to fulfill such claims. They overload the meaning of HTTP methods, for example, or they lack hypermedia support for representing relationships among application states. Most programmatic interfaces also don't fully take advantage of dynamic negotiation among various Web data formats, caching metadata, or design principles for ensuring loose coupling, scalability, extensibility, and versioning of services. A growing number of Ajax applications that use Web APIs from JavaScript in a browser also require them in order to support new protocols to better manage cross-origin communications. For example, a Web API that implements the Cross-Origin Resource Sharing protocol (CORS; www.w3.org/TR/cors/) can better control clients that come from domains other than the Web API's domain.

Developing a programmatic Web interface also requires tight integration with already existing back-end applications and infrastructures, and sometimes needs a new, highly dependable back-end technology. For example, to deliver the real-time message traffic at peaks of more than 12,000 tweets per second to a social graph of more than 140 million users (http://mashable.com/2012/03/21/history-of-twitter-timeline/), Twitter has developed its own distributed database called FlockDB that supports a high rate of database operations (http://engineering.twitter.com/2010/05/introducing-flockdb.html).

Public and Enterprise Services

Any Web application vendor's ultimate goal is to enable the visibility of its Web APIs and increase their reuse, eventually leading to profits. ProgrammableWeb is the largest repository of Web APIs that adopts the characteristics of a social-Web platform. Web application vendors can use it to publish, share, and discuss, thereby increasing the visibility of their APIs. Enterprises, on the other hand, build service-oriented architectures (SOAs) with additional goals. Not only do they want to maximize service reuse, they also have requirements to shorten the time to deliver new services, control and track services' reuse, ensure compliance with IT policies, or measure dependencies and the impact of change. SOA governance and its core enterprise repository technology define a methodological and technological framework for building the enterprise SOA that supports such requirements. Enterprises can use SOA governance for centralized asset management (such as management of services, applications, runtime environments, design patterns, projects, and users)

and to build the service environment through a well-defined service life cycle.

Web technologies' evolution has enabled RESTful Web APIs to grow while the Web has brought many improvements that, if properly used, can ease Web application development and integration. On the other hand, many services in enterprise environments still use traditional technologies based around SOAP and the Web Services Description Language (WSDL). Today's stack of SOA technologies provides a comprehensive set of tooling that, in contrast, eases the development and integration of enterprise applications. For example, making a service available in an enterprise service bus is only a matter of importing a WSDL document. It's also possible to generate code for a Web service client from a WSDL document, gather various service endpoint metrics from a running SOA, and manage service security, scalability, and performance.

Note, however, that organizations today are looking at how they can use RESTful services to integrate applications exposed on the Web in conjunction with their enterprise SOA technology. For instance, within the public administration sector, running a very complex enterprise SOA means performing systems integration spanning several domains, such as transport, police, revenue, and citizen registries. A robust, industrial SOA solution is required to effectively manage this integration. However, for opening up the data and exposing e-government services on the Web to enable third-party applications to reuse them, Web API technology is better suited to take advantage of all available dissemination channels the Web offers. Integration and better cooperation among SOA technologies, SOA governance, public Web API directories, and social Web platforms is thus another step toward connecting the enterprise service and Web API worlds.

In This Issue

Given the increasing popularity of programmatic interfaces for Web applications, we invited researchers and practitioners to submit articles to this special issue that describe topics related to emerging technologies and best development practices that underpin any modern programmatic Web interface.

In "Toward an Open Cloud Standard," Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson present a case for an open standards-based approach to APIs for programmatic access and control of cloud computing infrastructures. The authors argue that a standards-based approach can reduce the risk of vendor lock-in and increase the potential reach of cloud computing technology. Technical work for reaching an agreement on an actual standard is still under way, but the proposed open cloud API the article describes will be based on HTTP.

Web services are delivered and supported via software framework stacks, with each stack designed with a specific service model in mind. For RESTful HTTP services, many competing frameworks unfortunately interpret REST's constraints in different ways. In "ArRESTed Development: Guidelines for Designing REST Frameworks," Ivan Zuzak and Silvia Schreier distill several valuable guidelines for designing frameworks that encourage developers to make full and correct use of HTTP. The article also provides a comparison on how well four existing server- and client-side frameworks fit with the guidelines.

In "Welcome to the Real World: A Notation for Modeling REST Services," Olga Liskin, Leif Singer, and Kurt Schneider propose using a simplified version of UML state charts as a notation to support the design process and enable the documentation of REST services. Their intent is to create a notation that promotes the design of services that conform to the constraints of the REST architectural style. The authors have applied the notation to describe a set of randomly selected public services and bring new evidence that not all services claiming to be RESTful really are. The proposed notation thus also helps to detect whether services explicitly or implicitly use resources, and whether resources are preferable to methods. It also identifies isolated resources that aren't reachable by navigating through a service's hypermedia graph.

As more and more Web services provide access to streaming data sources, selecting the appropriate interaction protocol and patterns becomes important so that the resulting application architecture can satisfy real-time requirements. In "Communicating and Displaying Real-Time Data with WebSocket," Victoria Pimentel and Bradford G. Nickerson compare the impact of stateless, poll-based interactions,

supported by standard HTTP, with stateful, push-based protocols, such as WebSocket, that have recently appeared. The result is that WebSocket outperforms techniques such as HTTP polling and long polling, especially if the latency between the Web service and its client exceeds the interval between successive stream elements.

In "SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing," Fatna Belqasmi, Jagdeep Singh, Suhib Bani Melhem, and Roch H. Glitho compare two kinds of Web services technology in the context of a real-world case study. Both SOAP and REST are expressive enough to support the design and construction of an API for a conferencing gateway. Still, when it comes to evaluating the performance of the alternative approaches, measurements indicate that avoiding SOAP will significantly reduce delay.

Even with the recent explosive growth of programmatic interfaces across industry websites — and the advances and achievements in programmatic Web interface research and development these articles describe — much investigation and implementation work remains. Significant attention and effort is still needed to address numerous problems and questions remaining in areas such as protocols, media formats, agent-server communication patterns, caching, scalability, operations, versioning, and development tools and languages.

Tomas Vitvar is a senior technical architect at Oracle and an associate professor in the Faculty of Information Technology at the Czech Technical University in Prague. His research interests are in distributed systems and applications, including service discovery, REST architectures, Semantic Web, and SOA governance. Vitvar has a PhD in computer science from the Czech Technical University in Prague. He's codeveloped architectures and technologies for Web services and middleware in numerous international projects and contributed to standardization at the W3C and OASIS. Contact him at tomas@vitvar.com or on Twitter at @tomasvitvar.

Steve Vinoski is an architect at Basho Technologies in Cambridge, Massachusetts. He writes the "Functional Web" column for *IEEE Internet Computing*. Vinoski is a senior member of IEEE and a member of ACM. You can read his blog at http://steve.vinoski.net/blog, and contact him at vinoski@ieee.org or on Twitter at @stevevinoski.

Cesare Pautasso is an assistant professor in the Faculty of Informatics at the University of Lugano, Switzerland. His research group focuses on the design of experimental, self-adaptive systems to explore the intersection of service- and resource-oriented architectures. Pautasso has a PhD in computer science from ETH Zurich, Switzerland. He was the general chair of the 9th IEEE European Conference on Web Services (ECOWS), coedited a book on REST, *From Research to Practice* (Springer, 2011), and is a senior member of IEEE. Contact him at c.pautasso@ieee.org and on Twitter at @pautasso.