From the Editor in Chief ...

Services and Situations

Munindar P. Singh • *singh@ncsu.edu*

Forget objects. The killer buzzword of our era is now services. And who isn't for services? Services are the "motherhood and apple pie" of modern computing. But unlike motherhood — or maybe like motherhood, for all I know — services can mean different things to different people.

Litmus Test

Ask five people what a service is and you will get five answers. Here's a sampling of the answers from different communities, reflecting their backgrounds and prejudices:

Composability is a powerful property for engineering software.

- Networking: a service is characterized by bandwidth and such properties.
- Telecommunications: it's a telephony feature such as caller ID and call forwarding, and basic connection services like narrowband versus broadband (itself of a few varieties).
- Systems: services are components for billing and storage and other key systems functions.
- Web applications: services correspond to Web pages, especially those with forms or a programmatic interface thereto.
- Wireless: services are wireless versions of the Web, but also things like the exceedingly popular short message service (SMS).

If there is agreement here, it is just at a dictionary level — that a service is some capability that is provided and exploited. I will use this as a working definition. The different answers offer a litmus test, however, for judging the roles played in constructing distributed systems.

Using Services

Services provide a programming metaphor that supports the right kinds of programming models for open distributed systems. Service architectures are modular, because each service inherently offers a certain provider-subscriber interface. This interface enables much flexibility, for instance, by allowing proxy agents to transparently provide new services based on old services and to compose services as appropriate. Composability is a powerful property for engineering software and more than sufficient justification for all the interest in services.

Although services must be invoked, their invocations will often be implicit. For example, many networking and telecom services are not invoked as such; they are merely variants of some basic services that are invoked. That is, you might invoke a packet-delivery service to send a series of packets. With the same programming interface and depending on what underlying service is provisioned, you might have different guarantees as to the packet delivery in terms of, say, jitter. In telecom, whether is of regulatory (and hence economic) interest. For example, looking up a phone number is a stand-alone service, whereas call forwarding is a feature of telephony.

To get a feel for these distinctions, see the Federal Communications Commission's ruling on reverse lookup (http://www.fcc.gov/Bureaus/Common_Carrier/Orders/1996/da961069.txt).

Architectural Dilemma

If we view services as invocations, the building blocks of a service are the classical pull and push approaches. Both pull and push typically involve filters or constraints that are associated with invocations. How precise or elaborate the constraints are is often a matter of performance. If the constraint in the invocation is weaker — that is, more liberal — then the subscriber (perhaps a computation) must apply additional constraints to filter out undesirable results. But the only effect of where you do the filtering is on the performance.

 1089-7801/01/$10.00 ©2001 IEEE

The simple service architecture plays host to an interesting dilemma. On one side, doing the filtering as part of the subscriber can affect the performance of the composed system noticeably over some infrastructures. For example, if a service pushes notifications with minimal constraints, it might overwhelm a wireless device with the data. Of course, a good application would shield its user from masses of data by applying additional constraints.

On the other side of this dilemma, the best constraints are situational, that is, dependent on the user's situation as inferred by the subscriber.

Current provider-centric approaches to services ignore the user's situation altogether. But how could they *not* ignore it? Situational information is usually not available to the service and there are good reasons, such as preserving privacy, why it should not be made available. You wouldn't want your location disclosed to all services just because they might send you an alert relevant to your location.

Standardizing Situations

Services can take some profiles, usually rigid, to accommodate the interests of users but not their changing situations. The usual service architecture makes the subscriber responsible for managing the situation. This is reasonable, because keeping an interface generic broadens the applicability of a given service. Today, making a service open means enabling access through a well-known protocol, such as HTTP, and maybe a published syntax, such as an XML DTD. But true openness requires publishing the semantics as well.

Even a generic interface can be rich. It should enable a wider variety of constraints. In particular, services should have declarative, inspectable representations of knowledge, so that a discerning subscriber can determine how the capability provided by a service relates to its situational constraints. The subscriber should be able to probe the service in various ways to obtain the best utility from the service, but without compromising his, her, or its privacy.

For example, in the context of a notifications service, this would mean telling the subscriber what kinds of attributes the available notifications have and how they might be accessed, but not asking the details of their situation. To do so generically presupposes the development of standards for the kinds of situational variants appropriate in different domains.

IEEE Internet Computing

Editorial: *IEEE Internet Computing* targets the technical and scientific Internet user communities as well as designers and developers of Internet-based applications and enabling technologies. Instructions to authors are at http://computer.org/internet/edguide.htm. Articles are peer reviewed for technical merit and copy edited for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

Circulation: *IEEE Internet Computing* (ISSN 1089-7801) is published bimonthly by the IEEE Computer Society. IEEE headquarters: 3 Park Avenue, 17th Floor, New York, NY 10016-5997. IEEE Computer Society headquarters: 1730 Massachusetts Ave., Washington, DC 20036-1903. IEEE Computer Society Publications Office: 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720; (714) 821-8380; fax (714) 821-4010. Subscription rates: IEEE Computer Society members get the lowest rates and choice of media option — US$36/29/47 for print/electronic/combination. For information on other prices or to order, go to http://computer.org/subscribe. Back issues: $10 for members, $20 for nonmembers. Also available on microfiche.

Postmaster: Send undelivered copies and address changes to *IEEE Internet Computing*, IEEE Service Center, 445 Hoes Ln., Piscataway, NJ 08855-1331. Periodicals postage paid at New York, N.Y., and at additional mailing offices. Canadian GST #125634188. Canada Post International Publications Mail Product (Canadian Distribution) Sales Agreement #1008870. Printed in USA.

New Editorial Board Members

The IEEE Computer Society Publications Board accepted two new members of the editorial board for *IEEE Internet Computing*.

Jean Bacon is a Reader in distributed systems at the University of Cambridge Computer Laboratory and director of studies in computer science at Jesus and New Hall Colleges. She is editor in chief of *IEEE Distributed Systems Online* (http://computer.org/dsonline/) and former EIC of *IEEE Concurrency* magazine. Bacon received her PhD in computer science from the Council for National Academic Awards (CNAA). Her research interests include distributed systems, operating systems, concurrent systems, systems software, and computer architecture. She is author of *Concurrent Systems* (Addison Wesley, 2nd edition 1998).

Chris Metz is a Technical Leader in the service provider engineering group at Cisco Systems. Metz has been a regular contributor to *IC*'s "On the Wire" tutorials since 1998. His research interests include current and emerging IP network technologies including IP/multiprotocol label switching (MPLS) signaling and routing, IP multicast, quality of service, and IP-optical integration. Prior to joining Cisco in 1998, Metz spent 14 years at IBM. He is author of *IP Switching: Protocol and Architectures* (McGraw-Hill, 1999) and coauthor of *ATM and Multiprotocol Networking* (McGraw-Hill, 1997).

How to Reach IC

Articles

We welcome submissions about Internet application technologies. For detailed instructions and information on peer review, *IEEE Internet Computing*'s author guidelines are available online at http://computer.org/internet/author.htm.

Letters to the Editor

Please send letters, including reference to articles in question, via e-mail to internet-computing@computer.org.

Reuse Permission

For permission to reprint an article published in *IC*, contact William J. Hagen, IEEE Copyrights and Trademarks Manager, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855-1331; w.hagen@ieee.org. Complete information is available online at http://computer.org/permission.htm. To purchase reprints, see http://computer.org/author/reprint.htm.