From the Editor in Chief

Physics of Service Composition

Munindar P. Singh • singh@ncsu.edu

All science, as Ernest Rutherford famously asserted, is either physics or stamp collecting. By physics he meant clean, succinct principles that apply to diverse phenomena, and by stamp collecting he meant the cataloging and organization of large sets and varieties of observations.

To me, the essence of the Web is its support for interaction with users and among users. This is interaction not at the level of bits but at the level of meanings. Let's consider the specific challenges of an increasingly important application of interaction, namely, *service composition* — that is, using existing Web services and building new customized services out of them. Composing Web services is the next logical step in the progression of the Web and is being promoted by Microsoft's .NET (http://www.microsoft.com/net/) and Sun's ONE (http://www.sun.com/software/sunone/) initiatives. In general, engineering is part physics and part stamp collecting, but here I will emphasize the physics of service composition, because the stamp collecting seems to get all the attention.

Building Blocks on the Web

The idea behind service composition is simple. If we think of Web sites as offering not only content, but also services, then we can say, for example, that Yahoo provides a news service and Amazon provides a book-selection service. We typically invoke these services by hand through a Web browser, but a program could invoke them directly. Thus, you might compose a service by finding the latest news headlines and then searching for books that match those headlines. Or you might take the news from one service, filter it through a service that selects news based on a given user's interests, and pass the selected news items through a transcoding service to create a personalized Web page that could be reviewed through a handheld device. Or, more conventionally, you could create a travel service that invokes hotel, airline, and car rental services.

The above *procedural* form of composition is exactly what you would see in a closed environment. The main compositional operation here is invoking the methods corresponding to the constituent services. Procedural composition yields a graph whose vertices are existing services or filters and whose edges represent data flows. The graph would typically be executed top-down with each vertex invoking its children; it could also be executed bottom-up when children push up results to their parents. Both Microsoft's .NET and Sun's ONE emphasize the procedural view of service composition.

Open Versus Closed Environments

When the execution model is trivialized, any complexity that remains lies primarily in how the data connectors are built. The Web simplifies the data connectors by standardizing the protocol as HTTP and the data representation as XML (or HTML). Aligning the semantics of the information exchange is still a challenge, of course, but once you choose a uniform description framework — say, RDF — the main problem is to agree about the terms. Agreement is typically reached through domain-specific standardization efforts. This is a challenge on the order of stamp collecting: solutions are engineered by organizing knowledge into acceptable ontologies.

More importantly, these procedural approaches are fundamentally insensitive to the challenges of an open environment. Their physics, as it were, is no different from the physics of a closed environment. Applying procedural approaches to open environments is like applying traditional mechanics to quantum mechanical problems.

Abstractions for Openness

So how would our abstractions for service composition differ to accommodate open environments? We would first have to recognize certain special features of open environments and then add capabilities to exploit those features.

- Because services are autonomous, we would not require them to be subservient to other services. Instead, we would enable them to be proactive and to interact flexibly on their own terms. We would capture their autonomy by expressing contractual guarantees between them regarding the quality of service they offer.
- Because services are heterogeneous, we would develop expressive, standardizable representations for them. We do this now for data, but not for processes and policies. For example, to be able to compare across travel services, it would help if the processing detail that hotel bookings are refundable and cheap airline fares are nonrefundable could be standardized.
- Because services can live long, evolve, and operate in environments that produce exceptions, our representations would have to handle the resulting dynamics. For example, how would flight cancellations be handled by our travel service? Would the airline service suggest alternatives?
- Because services can be cooperative, our abstractions would represent how they behave in awareness of the behaviors of other services. For example, if the airline service has to reschedule a flight, would the hotel service accommodate the change?

These are not easy challenges. In the end, they will require a lot of organizing, but the right abstractions will go a long way in streamlining the task of organizing. The recent W3C activity on the Web Services Description Language (http://www.w3.org/TR/wsdl) is interesting, but it too fails to address the challenging problems: it defines a service merely as a set of "ports" where operations can be invoked.

True, engineering often requires a lot of stamp collecting, but it is always the physics that scales.

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/edguide.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 at http://computer.org/permission.htm. To purchase reprints, see http://computer.org/author/reprint.htm.

IEEE INTERNET COMPUTING

IEEE Computer Society Publications Office
10662 Los Vaqueros Circle, PO Box 3014
Los Alamitos, CA 90720-1314