Editor: Charles Petrie • petrie@stanford.edu

It's the Programming, Stupid

Charles Petrie • *Stanford University*

We often, if not always, discuss the virtues of the Semantic Web (and services) in terms of e-commerce — virtual enterprises and all that jazz. But as it turns out, that's not what's so important about enterprise semantics.

The Semantics of Semantics

If you weren't sufficiently put off by the title and actually read my inaugural column as editor of the Peer to Peer department ("Pragmatic Semantic Unification," Sept./Oct. 2005, p. 96), I wouldn't have to repeat myself now, but you probably were and, of course, that's my fault. Briefly, I put forth the thesis that the semantics of semantics is operational for both computers and humans. We might talk to people and think we know what they're saying, but we don't really know until we try to use the language to actually do something together, such as hunt or build a software system. Until then, it's all just talk.

The same is true with computers. If you say your ontology unifies with mine, all I really know is that you've made some mapping between them, which might or might not make sense to me. So what? Where are the mapping's semantics? They exist only if you have an application that uses your ontology and I have one that uses mine, and the two applications interoperate to our mutual satisfaction using the mapping. That's my story, and I'm sticking to it.

The SWS Challenge

I'm sticking with the story so much that since that column's publication, I've devoted a lot of time — along with Michal Zaremba, Holger Lausen, and others — to developing the Semantic Web Services Challenge Workshop (www.sws-challenge.org). The 2006 workshop's first phase met in March to mutually define and refine a set of problems that semantics-based technologies should solve. The idea isn't to show computational speed but rather to operationally explore the semantic technologies space.

Based on the challenge of processing and fulfilling a RosettaNet (www.rosettanet.org) purchase order (PO), we've collectively defined a set of problem levels as well as processes for adding problems and for evaluating how well the technologies solve them. A RosettaNet PO is a complex XML specification as well as a simple protocol for exchanging messages about its processing. The workshop's level-0 problem is to match this RosettaNet specification to a legacy system that doesn't conform to it. In level 0, we simply test that the participant system actually exchanges the right messages under different circumstances, including errors and incomplete orders.

At the workshop's second phase this June in Montenegro, the invited participants will face problems identified as levels 1a, b, c, and 2a. (We've identified several kinds of problems, represented by sublevels within each.) This operational evaluation is designed to define the technologies' semantics and let us collectively evaluate what success level the solution reaches. If you can say that your semantics-based technology at least solves the basic services-discovery problem of 2a to success level 0, then we have a common understanding of your technology based on the workshop tests. And how do we evaluate your solution? By the amount of human effort it takes to go from one problem (sub)level to the next. The idea is that a semantics-based solution should semiautomate reactions to a change in an e-commerce partner's system or the need to find a new partner. (If you're reading this, I'll assume you're interested in semantics and have seen the standard e-commerce examples, so I won't repeat them here.)

That's a lot of background, even though I've omitted the specifics, which you can read about at the SWS Challenge site. Yet, the interesting part is in the technologies we've received as entries so far. You might be as surprised by them as we were.

Scaling Enterprise Programming

If you thought as we did, you'd expect most of the entries to be AI-like — relying on knowledge representation through declarative statements that could be easily modified without modifying the code that compiled or interpreted them. Instead, we received a lot of software engineering technologies, most showing that it was very easy to modify the code via some nifty GUI.

Well, in retrospect, duh! The arguments for declarative semantics all boil down to ease of programming. If the definition of "department manager" changes, for example, how hard is it to change the code in different applications that depended on the old definition? If you find a new supplier for a

continued on p. 95

 Published by the IEEE Computer Society 1089-7801/06/$20.00 © 2006 IEEE

continued from p. 96

part with defined specs, how much do you need to change in your legacy system to interoperate? If the specs change, what code needs to be changed? How hard is it?

Christoph Bussler, of Cisco, illustrates the idea with the following example: How will your accounting system react to a supplier that adds, at no cost, a power cord to a product you ordered? Standard accounting systems reasonably reject invoices that contain unordered parts. How many programmers in a developing country does it take to produce an accounting system that understands "power cords" as a human accountant would?

The main challenge facing modern enterprises is how to scale programming costs in the face of rapid change and the increasing requirement for interoperation with other enterprises. Some researchers (including me) have posed the problem of the Semantic Web, and services, as an interoperability issue. Thinking about it carefully, we see that the issue is one of employing the least possible effort to reconcile systems programmed by different people.

Given a system with some sort of really terrific semantics magic, this task would take no effort at all: at least one system would automatically adapt when another system changed some interaction. So it boils down to programming effort. If semantic technology has a future — and I'm sure that it does — it's in software engineering. It remains to be seen whether a real semantics-based approach can do better than other software engineering techniques. Our challenge is designed to help provide the answer.

You Can Help

In phase II of the 2006 SWS Challenge Workshop, this June, participants will report their success in solving not only the base-level problems but also a set of problem changes that we'll define in detail roughly one month prior to the workshop. As a basic measure for each level, the SWS Challenge Web services must be invoked with correct messages and in the correct sequence, but we'll also evaluate participants on how much they had to alter their systems to respond to our released changes. The first changes to which the participants will have to react will be sublevels in which the legacy system changes and the RosettaNet specification changes. In level 2a, the participants will be challenged to discover a shipper in order to fulfill the order.

If semantic technology has a future — and I'm sure that it does — it's in software engineering.

We'll continue to build out the challenge problems and corresponding system as a community effort. Holger Lausen will serve as the "Linus Torvalds" of this open-source initiative. Our intent is to establish a living online system that can be used to evaluate different approaches. Future face-to-face workshops will look not only at technologies but also at how the community initiative is developing. In the meantime, there's an issue that you, *IC* readers, can help us with.

One of our naive thoughts was that we'd determine whether any code or data had to be changed in going from one problem level to the next. One approach would be better than another if only data (preferably semantic declarations) had to be changed, and even better if the developers had to only add new data and not delete any.

Then someone asked if Extensible Stylesheet Language Transformations (XSLT) constituted code or data, and I thought about Lisp, in which the functions and declarations are defined in the same language. And then there's Java. Oops! So, we decided to let challenge participants self-decide what's code and what's data. They'll declare what they've changed and how, but those declarations will need to be peer reviewed.

Phase II workshop participants will peer review each other's work, but as we extend the problems online, we'll need to extend the community of reviewers. Yes, we know you're busy. But if you want your technology evaluated, you need to volunteer to review someone else's. And if you're not participating this year, we hope that you'll follow along on the wiki and think about how your favorite technology might be able to solve some of the challenge problems (www.sws-challenge.org/wiki/index.php?title=Special:Userlogin).

Finally, whether you're participating this year, or in subsequent workshops, we encourage you to cheat. All technologies in this workshop will be publicly accessible. We encourage you to "steal" them, build on them, improve them, combine them, and do something better. Come to one of our workshops and show us how to really do it.

We hope to hear from you on the participant's mailing list (http://lists.deri.org/mailman/listinfo/sws-challenge-participants) and, when you get your password by joining the effort, on the wiki.

Charles Petrie is a senior research scientist and consulting associate professor at Stanford University. His research interests include concurrent engineering, virtual enterprise management, and collective work. Petrie has a PhD in computer science from the University of Texas at Austin. He is EIC emeritus and a member of *IC*'s editorial board. Contact him at petrie@stanford.edu.