From the Editor-in-Chief . . .

WRITE ASYNCHRONOUS, RUN SYNCHRONOUS

Munindar P. Singh • *North Carolina State University* • *singh@ncsu.edu*

Last issue, I wrote about interaction via "live," or dynamic, documents as analogs of communication. This time I will probe the concept of communication a little further.

Why should an Internet specialist care about communication? Because, fundamentally, communication is all you ever do with the Internet. And communication is more than just bit transport. Communication happens not only when you talk to someone, but also when you work together or carry out a trade. This isn't simply because information is passed around in these activities, but because the activities are based upon the parties interacting in certain ways. Likewise, in the realm of computing, we must think of communication as more than just bit transport, and consider explicitly the interactions among the computations.

Briefly, my claim is that the forms of communication we see in current programming models are limited and, consequently, so are the applications we can build over the Internet. If we are to engineer the next generation of applications—from negotiation-based electronic commerce to tools for people to come together in online communities—we must come up with higher level abstractions than, say, push and pull. These abstractions will prove as important as having interpretive programming environments, such as Java, that support the "write once, run anywhere" paradigm.

If we are to engineer the next generation of applications, we must come up with higher level abstractions than push and pull.

Synchrony vs. Asynchrony

There are two main kinds of communication: synchronous and asynchronous. Sometimes we distinguish synchronous communication by calling it "real time," but in my view, synchrony is not about timing guarantees but about the communicating parties being simultaneously engaged in communication.

Over the Internet, asynchrony is the default and synchrony is something you have to work for. However, synchrony is simpler both conceptually and computationally. Under synchronous communication, the parties involved share more of their context and can thus make stronger assumptions about each other. By contrast, under asynchronous communication, none of the participants knows for sure when the other participants will be engaged, and thus the communications must have a meaning that to some extent is independent of the context in which they were created. That's why programming under the assumptions of synchrony is easier and that's why many of the earlier programming models assumed synchrony.

Expressiveness vs. Reusability

Independence from context is essential to reuse, whereas sensitivity to context is essential for capturing the nuances of dynamic situations. For example, a traditional electronic commerce system might provide forms for a user to fill out or it might notify a user when a product becomes available. The system's behaviors are reusable, but not especially sensitive to what the user needs at any given time. In business-to-business settings, can one party change its mind or interrupt the other? Do the parties modify their behavior based on the situations of the other parties? In real life, they can, but as computations in our programming models, they usually cannot. A shopper, who in a traditional store could make arbitrary requests and express a variety of preferences, would be locked into whatever the form allows.

Human language allows us to say things out of context. We can write prose that will be read in contexts different from that in which it was produced. However, fixed descriptions can become cumbersome if they try to cover several possible

contexts of use: This is why legal documents are often so long-winded.

As Internet professionals, our interest of course is not in writing stories or laws, but in designing systems whose components interact with each other as well as with users. These interactions must be reusable, but they must also be sensitive to the contexts in which all parties operate. If we neglect the importance of context, we will end up producing unwieldy specifications that essentially attempt to enumerate all possibilities or fail unexpectedly, because they neglected some crucial situation. What are the abstractions with which we can "write once, run repeatedly" in different contexts?

Ideally, interactions as *conducted* should have the expressiveness of synchrony, while interactions as *designed* should have the reusability of asynchrony. Before the Internet, we had to choose between structure (for example, rigid forms) and flexibility (such as unrestricted conversation). But now, by distributing intelligence and decision making across the entire system, we can hope to capture structure without sacrificing flexibility. Doing so properly is what I call the challenge of writing asynchronously and running synchronously. To address this challenge will require developing high-level abstractions of the sort I alluded to above.

Constructive Communication

Deconstructionism in simple terms (and, I confess, I don't understand it in any but these simple terms) opposes any objective, context-independent meaning of communications. Further, the implied context of the sender is suspected while the context of the receiver is emphasized. I agree with this concept in that it maintains that meaning at some level is contextual, but disagree with the suggestion that independence from context is undesirable. What I propose here is the systematic reference to context in any and all interactions. For want of a better term, I call this *constructive communication*.

An important part of being interactive is being collaborative, so as to communicate constructively. New technologies for collaboration are the theme of this issue. I believe such technologies provide many of the ingredients we will use in times ahead. ■

IEEE INTERNET COMPUTING

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