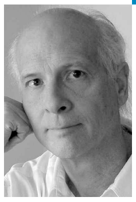
RIK FARROW

promises, promises: an interview with Mark Burgess



When Rik Farrow is not acting as editor of *;login:*, he is working to improve computer security as a consultant and researcher.

rik@usenix.org

Mark Burgess is professor of network and system administration at Oslo University College, Norway. He is the author of Cfengine and many books and research papers on system administration.

Mark.Burgess@iu.hio.no

RIK: SO LET'S BEGIN AT THE BEGINning, shall we? Assume your gentle reader is clueless in these discussions. What led you to promise theory?

Mark: Well [laughs], as always, ideas come by a circuitous route. Promise theory sort of grew in my mind over a few years from thinking about everything I was doing: network graph theory, game theory, host autonomy (as in cfengine), policy-based management, anomaly detection, fault analysis, etc., etc. I think I was getting interested in others' modal logic approaches to policy and was at the same time getting depressed that they seemed to be a complete waste of time. Then, I kind of have this thing where I challenge myself to break with convention and say: The hell with what everyone else is doing; I've got to think again from the beginning! How would *I* do it?

At the time, I was doing some work on network ranking in search algorithms (like Google's PageRank) with two colleagues at Telenor and I was wondering whether there was a way of using those ideas for pinpointing problems in computer networks. It occurred to me that the reason networks succeed or fail to make a working system is all about what the individual nodes do for each other. So that might lead you to thinking about a kind of Service Oriented Architecture, but I'm always trying to look beyond the obvious answer. It occurred to me (from what I know about events and anomalies) that what we *do* or what actually *happens* is far too ephemeral to be interesting. It is not the issue. Rather, it's the *average* behavior of systems that tells you what's of lasting value about it. You know, if we design systems by thinking about mosquito bites we'll wallow in details that will mostly average out to nothing. But what if we could simply describe how individual (i.e., autonomous) components *behave* toward each other on average, and see how that ends up leading to a working system? In other words, stop thinking about the networks as communication, and start thinking about them as *interaction*, or what behaviors the components exhibit toward one another. That was sort of what got me going.

Rik: And you described the nature of these interactions between components as "promises." Why "promises"?

Mark: Yeah, that's important. We're trying to get to a simple summary of how parts in a system will behave when we put them together, and that includes both machines and people—human-computer systems, as I like to say. I think we have to realize that systems don't always do what we want them to, and that we sometimes take it for granted that they should. A promise (if made) is a good way of summarizing how a component will try to make its best effort to contribute to a system voluntarily. Voluntary cooperation, in turn, is an important viewpoint (though you might at first think it's a bit odd) because it says that whatever autonomous decisions have been made by the component, you know either behind the scenes by its owner or programmed within it, these decisions are going to support this behavior that has been promised. So a promise captures the essence of what behavior is advertised and planned for. Now let's say a system does not behave the way we want. This could mean either that it has not promised to behave in that way (i.e., choice was involved) or that it has indeed made a promise but was not able to comply for reasons beyond its control (i.e., a fault).

Without a promise we can't tell the difference between not being willing to cooperate and not being able to cooperate, that is, design error or a fault. So we would miss a vital part of the specification, something like half a contract. If we have all necessary promises to guarantee success, the only reason for failure in a system is an unforeseen fault. So we distinguish between what is promised or "expected" (with inevitable uncertainty) and what is simply "unknown." I think this helps us make an important conceptual step away from believing that we can magically force components to do their jobs.

Rik: I find myself wondering about systems that make promises that they cannot fulfill. I don't see anything in promises that would prevent a system from "lying" about its capabilities and thus tricking other systems into choosing to rely on a system that will fail.

Mark: That's true. It's the way the world really is. Just as there is nothing to prevent any service provider or component in a system from lying about its service delivery. A good example of this is a power supply we bought recently that is rated with a certain current delivery that was simply false. I should be clear: We talk about voluntary cooperation not because it is necessarily desirable but because it is the only realistic viewpoint. Actually, you can't squeeze blood from a stone, or force someone to deliver on something without their voluntary cooperation. At best you might be able to refuse them something in return, if there is a trade of some kind.

Take a peering agreement: If you promise to carry my traffic and then later refuse, I can withdraw my promise to carry yours. In the case of a power supply, there is no trade. The power supply is a "slave" component and if we anthropomorphize, we have no threat of reprisal if it lies about its capabilities—we are simply screwed (pardon my French). Of course, in reality the manufacturer of the power supply lied about its capabilities, so we could either withdraw our promise of money to them or make a threat of litigation (which is a promise to do something that would have a negative value to the other party). The promise paradigm still fits. Promises only help us to manage this uncertainty by indicating an intention to behave in a certain way.

Rik: I can see how promises are a realistic way of representing the relationship between components. How do promises fit into configuration management?

Mark: Configuration management is a service that essentially makes a number of promises about how a system will be configured.

Rik: You and Alva Couch had a paper at LISA '06 that ties promises in with two other concepts: closures and aspects [1]. Can you explain briefly how these three concepts fit together?

Mark: Yes, Alva and I don't get to talk half often enough, but when we do we click on things quickly. We were actually walking around Vancouver at NOMS2006, lapping up some sunshine after I had been really sick in my hotel room for a couple of days. My eye for interesting graffiti was about to take us into "Death Alley" when a guy climbed out of a pile of garbage and tried to sell us drugs (carefully pointing out that we would probably be murdered if we walked down said alley). Anyway, this gentleman then showed us an alternative route through the neighborhood and gave up only when he finally believed that all I needed was aspirin. The promise made by "Death Alley" could be thought of as one of any number of aspects of Vancouver—crime, life, death, violence, drugs, etc. An aspect is a very high-level idea, much higher-level than a promise, but it is nonetheless a thing we use all the time to organize our thinking. Paul Anderson points this out in his SAGE booklet. Think of a Web page, if you like. Its specification makes certain promises about text, palette, images, etc. An aspect of these promises could be "color contrast." If we want to increase or decrease the contrast, we might have to reevaluate promises made about the text, the images, and the colors in a coordinated way even though they are closed-off and separate categories that we deal with in quite different ways. Aspects often cut through several specific issues.

If we think of aspects of configuration management such as backup, host naming, and service delivery, we can express them (compile them down, if you like) into low-level promises between specific components. The promises are much closer to showing you how to implement these concerns. In fact, cfengine statements are essentially promises (to fix a system in a particular way if anything should go bad). Now, closures are a software idea that represents somehow the autonomous nature of agents in promise theory.

Alva was thinking about closures even before I was thinking of promises. It turns out that these are all very complementary concepts. The agents interact only through the promises they give and receive. Otherwise, they are independent. A closure is essentially an agent that interacts only through its computing interface, that is, in accordance with the promises it makes to others. Like an agent, it does what is inside it; it cannot be forced by an outside influence (for instance, there can be no global variables or shared memory leaking control information in or out). Closures cooperate essentially voluntarily, by virtue of their internal specifications. They can't be obliged to change their behaviors. An agent follows a certain internal discipline, and a promise is a piece of glue that binds closures into cooperative patterns of behavior. These then result in certain "aspects" of configuration being promised.

Rik: Are closures about voluntary cooperation?

Mark: Again, promises go beyond mere change management. They say: Forget about obligations, deontic logic, and all of the barely plausible security models policy people talk about, and think realistically about what happens when you put closed components into a system. You buy a resistor or a capacitor that makes a certain promise, usually printed on its side. You can't force a resistor to be a transistor, so why even try to talk about obligations? The component does what it can, or what it "wants," *voluntarily*. There are no genies in the bottles. When we ask ourselves what promises are required to build a radio, we go beyond one instance or one application to what could be. I am a resistor and I promise to resist by one hundred ohms plus or minus five percent—not by the number you first thought of! That is an obvious but crucial philosophical aspect of management thinking that we seem to have forgotten in computing.

REFERENCE

[1] "Modeling Next Generation Configuration Management Tools": http://www.usenix.org/events/lisa06/tech/burgess.html.

[2] "Introduction to Promise Theory": http://research.iu.hio.no/promises.php.