Step by Step

September 4th, 2008

Before I delve further into the merits and demerits of RPC, SOAP, REST, and so on, I want to take a look back at the history of distributed computing. Not because I'm a connoisseur of history, but because that history has shaped much of the technology we see today and, without that history, today's distributed computing options would likely be different.

First Steps

Back in the eighties and early nineties, we saw the first few steps toward a kind of distributed computing that did not require developers to create half a networking stack on top of raw sockets. Sun ONC, Apollo NCS, and DCE marked these early steps. These technologies were a big improvement over sockets, and certainly made the job of writing a distributed application a lot easier. However, they also had their share of drawbacks, among them that they only provided a C API, and that much of the underlying network semantics were still exposed (such as the distinction between TCP/IP and UDP, or the need to explicitly deal with domain names and port numbers). In addition, these technologies were not widely available.

A Step Forward

During the nineties, the industry witnessed the rise of C++ (together with other object-oriented languages) and anointed OO programming the latest silver bullet. (It was impossible to open any trade magazine in those days without finding "OO" in the title of just about every article.) The interest in OO also brought about a shift in distributed computing. In particular, we saw increasing research in component models and, naturally, *distributed* component models. This led to a whole raft of new technologies, particularly on the Microsoft side, which created DDE, OLE, OLE Automation, COM, ActiveX, DCOM, COM+, and MTS. (Isn't it wonderful how developers were bestowed with successive new technologies? The list was later extended with .NET Remoting, which in turn was superseded by WCF.)

Meanwhile, the rest of the world wasn't asleep either… DEC morphed DCE into ObjectBroker, IBM created SOM and, later, DSOM, Trinity College created the beginnings of Orbix, ILog invented ILog Broker, Post Modern released Black Widow, HP developed ORB Plus, and Expersoft came up with Visibroker (among others).

So, there was plenty to choose from on both sides of the camp. While Microsoft continued to release a new spin on its distribution technology every few months, the other vendors closed ranks and joined the OMG, which published CORBA. During that time (the mid- to late-nineties), the industry witnessed a fierce battle between DCOM and CORBA—the Microsoft world mostly stuck with DCOM (although many CORBA implementations also ran on Windows), while the rest of the world went ahead and used CORBA.

A New Stomping Ground

By the late nineties, it had become clear that Microsoft were losing their battle with CORBA: CORBA could parade a large number of high-profile success stories, while there was never even a single large-scale or mission-critical DCOM application. (Even on Windows, CORBA had begun to displace DCOM, due to its platform-agnostic nature—by then, CORBA actually ran on more *Windows* versions than DCOM.)

Concurrently with these developments (beginning around 1995), a number of things happened that, together, reshaped the entire computing industry:

- The world-wide web began its meteoric rise.
- HTML created the world's first truly platform-independent user interface.
- Sun invented Java and created the world's first popular platform-independent programming language.
- The common household discovered the Internet.

Taken together, these factors meant that, suddenly, there were mega-dollars to be earned with distributed computing whereas, only a few years before, computer networking was strictly an activity for nerds. (As late as 1993, Bill Gates had said "The Internet? We are not interested in it.") All of a sudden, distributed computing had become the key to commercial success—there was no way that Microsoft was going to cede the distributed computing market to CORBA.

The explosion of the web and the ensuing high-tech bubble meant that anything based around web technology was in. (The OO silver bullet of the early nineties had given way to the XML silver bullet of the late nineties.) Rather than fight a battle they could not win, Microsoft created a completely new battlefield: the new paradigm was to be distributed computing based around the web and XML, and it begat SOAP in 1999, followed by Web Services shortly thereafter. Because the shine had worn off both DCOM and CORBA, many of the distributed computing vendors flocked to WS as the saviour of their ailing businesses. (Microsoft discontinued DCOM shortly thereafter, and CORBA had dwindled into insignificance by 2003.)

A Step Backwards

The shift to SOAP and WS was a severely retrograde step for a number of technical reasons:

- Many years of software engineering experience had demonstrated that encapsulation and separation of interface and implementation were key strategies in controlling cost and complexity. SOAP threw all this hard-earned wisdom out the window: there were no objects, there was no encapsulation, there were no standardized APIs and language mappings, and the data, once again, reigned supreme.
- SOAP and WS provided an excuse for hoards of developers with no prior distributed computing experience to reinvent the wheel (something that this industry loves to do more than anything else). Not only did they reinvent the wheel, but they reinvented it badly, repeating many of the mistakes of the

past, while adding a considerable share of new mistakes that no-one had ever thought of before.
- The bandwidth requirements of XML made it staggeringly wasteful; SOAP messages require twenty to more than one hundred times the bandwidth of the average binary protocol.
- XML is expensive to marshal and unmarshal; it requires hundreds of times the number of CPU cycles needed by a binary protocol.
- SOAP was so retarded and low-level that the marketing machinery had to be called into action. It duly bestowed us with an entire new service-oriented architecture ("SOA"), which was necessary to sell the botched technology to the industry.

In addition, proponents of SOAP made various claims that did not stand up to scrutiny, such as the importance of having a character-based protocol (as opposed to a binary one), the self-describing nature of XML, the inherent security gained by sending everything through port 80, and the loose coupling of service-oriented applications. (I will return to these topics in more detail in future posts.)

A Step Sideways

While the SOAP crowd was busy re-inventing a square RPC-wheel, another paradigm shift started to take hold: representational state transfer (REST). In contrast to RPC and SOAP/WS, REST is not a technology, but a set of architectural principles. In a nutshell, it argues that the success of the web is due to a number of constraints that guided its design. By making these constraints and designs explicit, we can arrive at an architectural style that benefits scalability, extensibility, and performance.

Like SOAP, REST was strongly inspired by the web, and its principles are closely linked to web technologies, such as HTTP, URIs, caching, and processing of documents by intermediaries. Like the web, REST focuses on "Internet-scale distributed hypermedia interaction", that is, it models distributed computing as an exchange of documents.

Where We Stand Today

Today, we can separate general-purpose distributed computing into three different camps.

- The RPC camp. This camp is populated by applications that use various forms of RPC, such as CORBA or Ice.
- The web camp. This camp is populated by applications that use HTTP as a substrate. It includes applications built around web browsers, HTML, XML, applets, servlets, SOAP, and web services.
- The REST camp. This camp is populated by applications that apply RESTful principles to a variety of technologies (even though a majority use the web for their implementation).

These three broad categories capture most of the action commercial and enterprise distributed computing. (Of course, there are many other technologies in use, such as SCADA, RTCP, or home-grown technologies; but these are not a part of general-purpose, B2B and e-commerce distributed computing.)

Now, regardless of the various merits of these camps, it is important to realize *why* they exist:

- Historically, the RPC camp came first. As developers applied the lessons of the past, RPC improved with successive iterations of the technology and managed to accumulate a number of considerable successes.
- The web and REST camps came second. The success of browsers, Java, XML, and the exploding e-commerce market inspired REST, and provided a convenient wave for SOAP, WS, and SOA to ride.

Another important point here is that the web camp exists not necessarily because of any inherent technological advantage over the RPC camp. Historically, *technology had little to do with the use of the web for distributed computing*. Instead, the web camp exists because of political manoeuvring, fighting for market dominance, opportunism, and the desire to leverage an industry trend. If there *is* a technological advantage for the web camp, that advantage is accidental because technological excellence could not have been further from the participants' minds at the time they created the schism.

Where We Are Going

Clearly, these distributed computing camps are here to stay for the foreseeable future. I simply cannot see a world where the majority of distributed computing would be built around RPC, or a world where most applications would be built around SOAP/WS, or a world where distributed applications would be exclusively RESTful.

To declare one camp "good" and another camp "bad" is naive and falls into the trap of searching for yet another silver bullet. As with all designs and technologies, there are trade-offs, and correctly matching these trade-offs to the requirements of a particular project is what separates "good" from "bad"—not a belief in the superiority of one camp over another.

With that in mind, we can delve deeper into these trade-offs, which I will do in future posts.

Cheers,

Michi.