SOA-World

ANALYST BRIEFING

Loek Bakker, IT strategy consultant, Gartner

A Consultant's View

Proprietary SOA

[March 7, 2006]

One of the most often overlooked things about service-oriented architecture (SOA) is that it is not about services. No matter how much the name may imply that we are talking about an architecture that is oriented towards services, the most important part of the SOA picture are not services. From what we have seen in practice, we can say that the key of SOA is about sharing services.

In our research we have identified three fundamental benefits of service-oriented architecture (SOA): architectural partitioning, incremental deployment, and maintenance and reuse of services. All these three elements can be reduced to the “simple” concept of loose coupling. A couple of months ago I had a discussion with one of our senior analysts on SOA, and he stated that SOA “was just a long-winded way of stating the goal of loose coupling”. He was right, I guess.

One description of the concept loose coupling I really like is: “a state in which the impact of change is minimized across dependencies”. Loose coupling is important at both the business and the technical level. From a business perspective, it is about loosely coupling a business process to the technology implementation of that process. At a technical level, loose coupling is about separating code and data, and separating interface and technical implementation. At the business level decisions on service granularity are important, at the technical level we often talk about things such as “Contract first!” development. It all sounds so easy and straightforward.

In practice, it appears that loose coupling in general, and how to incorporate this concept into your architecture in particular, is simple, but not so simple. And it takes time and creativity to achieve simplicity. In the last couple of months, I have been at different clients to assess and review their architectures. One of the questions that kept coming back was the level in which their architectures were ‘service-oriented’. Which should not come as a surprise, as ‘service-oriented’ and ‘future-proof’ are used interchangeably in the architecture context, as these are what CxO level management wants. CxO’s and other decision makers cannot be bothered what the orientation of your architecture is, objects, services, components, events: they don’t care. What they are interested in, is reuse, flexibility and scalability. This was no different for the sponsors of the architectures I reviewed at our clients. Both the goals of reuse and flexibility are realized by designing around the aforementioned paradigm of loose coupling.

The three architectures I reviewed in the last months had in common that they were based on a modern platform (Java or .NET) and that the architects had really put some effort in defining and identifying reusable services. In fact, they even spoke about things such as service modeling, service granularity and contract first development.

Sounds like their architectures are service-oriented and thus future-proof, right? Unfortunately it is not that simple. The point (maybe I could say problem) with these architectures was that they relied heavily on either the Java Virtual Machine (JVM) or the .NET runtime, both for the service and the service consumers. The services in the architecture were exposed through respectively RMI (Remote Method Invocation, a proprietary Java communication mechanism) and .NET remoting (proprietary .NET communication mechanism). Clients and service consumers that want to use the services therefore also need to communicate through the proprietary mechanism. It required not only the JVM (or .NET runtime) on the platform running the service, but each and every service consumer also needs the JVM. In other words: we were dealing with proprietary service-oriented architectures, as much as this may sound like a contradictio in terminis to some people.

Most vendors do not emphasize this, but one of the potential benefits of an SOA is that it allows enterprises to avoid vendor lock-in, as SOA is often associated with the use of open standards, i.e. non-proprietary standards that are not controlled by some vendor, and whose specifications are publicly available. SOAP, WSDL and UDDI, the famous threesome for web services, are excellent examples of (mature) open standards.

If for your business needs it is sufficient to have only the ability to plug in other JEE (or J2EE as Sun used to call it) or .NET services, it is okay to have an SOA based on proprietary standards. But this is no different from method invocation of subroutine calling. Arguments to go for a proprietary, pure Java/.NET solution are often that it provides superior performance, stability or robustness compared to web services based on open standards. However the value of being able to grow the network effectively, and being able to plug in any service that complies to this open and universal interface, often provides far more business value than any (temporary) technology advantage. The services itself can be implemented with JEE or .NET, but the interface should be based on an open standard.

All this does not mean that we must expose everything as a service based on open standards. There are numerous situations that I can think of where it is either better not to distribute at all (hence not to define a service for a specific function and expose it), or where it is better to create a pure Java or .NET service. However, since we were dealing with architectures that were branded “service-oriented”, we pointed out that these were proprietary SOAs, and that the expected benefits from generic reuse were limited to the level of which the services are proprietary.

Yes: you can have a fully Java (or .NET)-based service-oriented architecture… in theory. However in practice it lacks so much of the unique selling point of SOA, namely true loose coupling, that it will be very hard to create a solid business case for switching to an

SOA. And be prepared that you will hear a lot of people in your company talk about ‘old wine in new bags’.

I started by stating that an SOA is not about services, but about sharing services. And the more you can share services, the more potential benefits an SOA can provide.