JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 6, July-August 2007

SOA and the Clash of Technocultures *Classes versus Infosets versus Business Process*

By **Dave Thomas**

THE GOOD, THE BAD AND THE VICTIMS

SOA is a vendor and business driven train that is coming down the tracks at all large enterprises and many SME developers. It is leading to a clash of technocultures as the XML and BPM troops join the development battlefield. Recently, many have asked about SOA, the associated technologies, and their impact on development. SOA is a mix of good ideas, a huge amount of infrastructure and tool marketing, and a mixture of vendor, pseudo and real standards.

SOA SERVICES MEANS INTERFACES ARE IMPORTANT THEREFORE SERVICES ARE GOOD

One of the major disappointments of OO in practice is that even with explicit interface support in Java and C#, thus far interfaces are seldom or poorly used. Only recently has API [1] design[1] been recognized as a challenging and important task that is critical to the minimal yet functional access to the components of a system. Although it has been years since the dependency inversion principle was advocated, far too many class libraries and frameworks force developers to reach into the framework and/or import all sorts of irrelevant dependents, etc. The lack of separation of concerns and the associated compounding tangle of dependencies makes many OO programs bigger legacy liabilities than their older predecessors. Further, it makes the deployment of code across multiple tiers or threads high-risk work. SOA mandates the definition of interface separate from implementation, thereby providing architectural enforcement of this critical principle of architecture and design. The KISS principle should, of course, apply to services as well as

[1] It is important to note that API design is largely a component/library focused activity to support development while service design is largely a system or application architecture activity to support the assembly of services into a larger service or business process.

it does to APIs. The tension between simplicity and functionality is clearly visible in Amazon S3, Google and Yahoo Web Services where the challenge is to provide just what is really needed, allowing more complex things to be built on top rather than built in to a large monolith API.

INVESTMENT PROTECTION - SOA IS TECHNOLOGY NEUTRAL, WHICH IS GOOD

One of the unfortunate problems of diverse technology and platforms is that organizations are encouraged or even forced to abandon software assets unnecessarily to obtain the benefits of new platforms. In a world where 75% or more of the effort is expended in software evolution (enhancements, defects), it makes little sense to try to keep rewriting large portions of a working product in a new language to obtain a new UI, etc. SOA enables the large grained reuse of current legacy assets by reprovisioning them or portions of their service interfaces. It provides a standards-based approach for integrating existing assets with new assets and capabilities. Software as a Service (SAAS) extends this externalizing service for use or delivery across the Internet. It removes the ridiculous technical bias that just because S is written in language L or only runs on platform P, we can’t use it!

SOA IS MARKETECTURE: PROCEED WITH CAUTION

The mainstream vendors, while touting the many benefits, see SOA as a vehicle to entrench their platforms deeper into the business. They embrace SOA, BPM and XML standards as a means to reposition their offerings as being about the business rather than about technology platforms, which is really what they are offering. They have embraced the reengineering and BPM modeling community in the hopes that they can lift UML from a rather complex and cumbersome modeling tool to something much more. SOA has also allowed them to use web services and enterprise BUS to attack the EAI vendors who were taking too much of their business. Why would you want to use a perfectly good EAI product when you could buy a marketecture for your favorite platform or enterprise application? This is reminiscent of the days when 4GLs were replaced by much inferior, poor performing relational technology backed by these same vendors. In the end, the platform game is about winner takes all. Now they have seized upon XML, which is of course not a language but a meta-language. They don’t want a new language, new languages are problems for platform vendors; they can’t allow a new language to threaten the productivity of their current complex stack! So instead they will encrypt programs in XML. This promising marketecture is propped up by emerging standards with unknown implementation dates. Even basic interoperability with WSDL/SOAP is problematic. The scary thing is that the usually enlightened open source community, rather than innovating, seems to be playing the “me too” game following the same marketecture.

It is interesting to point out that those successful companies such as Google, Yahoo and Amazon have in general opted for much simpler service definitions often using RESTful style protocols.

XML AND ATTRIBUTE PROGRAMMING CONSIDERED HARMFUL

Just to be clear, I am a big fan of XML as a documented language of exchange between machines where XML plays a critical and important role as a vehicle for carrying well defined data and meta-data between systems. It would be nice if there were a binary format to reduce the overhead but perhaps this is being left as an opportunity for hardware vendors. However, XML definitely doesn't meet the test with respect to readable or writeable syntax, or comprehensive semantic account for use as programming notation! Given the number of XML files out there one would think that there would at least be a well specified form of publication language such as Algol 60 with reasonable typography so that humans could at least read it.

Use of XML as a programming language should be constrained. It allows too many inadequate languages to be defined too quickly under the guise of "XML standards are good". XQuery made the effort to recommend the support of alternative syntactic forms. At least in Lisp/Scheme, the oldest and best-known symbolic language, there is a well-defined read eval print loop. Further macros and pretty printers support modularity and readability. It is time to stop penalizing people with explicit XML and provide at least reasonable surface languages that allow programs to be read and refined as literate programs.

To placate those who must do everything in a single OO language, the attribute mechanism has been abused to allow the meaning of a Java or C# program to be redefined through attribute programming. This leads to a world of <here is my language> or @ here is mine @. It takes simple programs and overcomplicates them with annotations, platform APIs and XML parsing. Perhaps vendors like this gratuitous complexity because it provides more demand for their expensive tools.

THE FICTION OF BUSINESS PROCESS = BPEL PROCESS

If you can't pronounce BPEL4WS, don't feel too bad, you won't be able to read it either.

Like XLST, BPEL4WS, it is yet another programming language that can't be printed or read after being written. In the rush to merge the disparate world of IT and BPM, the BPM Process companies are racing to embrace SOA and UML in order to survive the expanding platforms of the major vendors. There are many new process engines in the market place both from established BPM/workflow vendors that want to execute at the BPMN level and from platform vendors that offer their own XML BPEL execution engines. Each of these has their own idiosyncratic behavior, coupling to OO applications etc.

In the dream of the modelers (if their models ran we would call them developers), executives will be able to take their MIT Sloan, Harvard McKinsey, AT Kearney or Boston Consulting business process PowerPoint and magically produce BPMN notation models that will then magically be materialized in BPEL4WS and WSDL. What an idea: push button process reengineering. Sorry, there is a lot missing and a lot of assembly required. The major complexity of workflow is exception handling, which humans do amazingly well yet programmers in general do relatively poorly. The easy part of implementing a business process is describing the happy path. It is much more challenging to deal with all of the exceptions.

SERVICE COMPONENTS?

One of the more promising standard-seeking activities is the SCA/SDO [2] initiatives led by the Anything But Microsoft (ABM) lobby. SCA defines a technology independent component model that allows the definition of service components which are implemented in different languages such as PHP, Java and BPEL4WS. SCA also provides simple composition semantics for composing new services from existing ones. For those who don’t like XML programming one can choose the other messy alternative of using attributes (in languages which support them). SCA provides a separate policy configuration mechanism to cope with the challenges of different classes of service however it isn’t clear how well some of these ideas adopted from telecommunications systems will apply to business services.

A related standard, SDO provides a simple, storage API independent means for object developers to talk to a data service. SDO materializes an object graph which is navigated by the OO programmers. Proponents claim that this eliminates the need for a plethora of specific services. It is interesting that some vendors, such as SAP, are already supporting these emerging standards.

THE JAVA/C# VERSUS BPEL VERSUS XML WARS

In the rush to create a middleware platform dependency larger than your current legacy platform dependency, vendors and their well-paid industry analysts push a plethora of complex technologies on organizations that just want to run their businesses. Some of the most talented technical experts I know find the complexity overwhelming when there are at least 5 different ways to do the same thing. Non-productive skirmishes, such as JDO versus SDO, BPEL versus Ajax, XML to parse[2] versus Object to execute, seem to flair everywhere. More dangerous still is the war between Process Server and OO Server camps. Sadly neither group typically understands the arguments of the other. Despite

[2] It is amazing how much complexity creeps into every application when instead of a simple set of parameters or a serialized object each programmer is expected to parse and generate XML files. This requires that every programmer has to learn one or more sets of XML parsing APIs.

their so-called expert opinions neither likely understands what is really happening in an enterprise class Java or Process server infrastructure. Unfortunately, system management tools for modern VM and process runtimes are not able to see into the fine-grained process structure. Too often important issues such as versioning and version migration are completely ignored in the heat of battle. What a wonderful opportunity for teams of consultants!

USE LOTS OF SOAP OR JUST MAKE IT RESTFUL?

One of the most common misperceptions things about SOA is that SOA mandates WDSL and SOAP. It doesn't, indeed there are many thought leaders who prefer simple service state machines and REST versus the complexity WSDL and SOAP. Microsoft Indigo is a nice example of an architecture that really doesn't care which wire format is used to implement a service. It is also important to note that SOAP and WS standards inter-operability isn't pretty so make sure that whatever SOAP kits you are using are compatible. It is interesting that where platform vendors have opted for complexity, Yahoo, Google, Amazon and Salesforce have opted for simplicity.

SUMMARY

There is just too much stuff in the SOA for most developers to absorb. It is difficult to understand what is real and what ishype. Beyond that, it is difficult to understand the performance and inter-operability of different implementations approaches. Developers need guidelines for when and where to use XML, Objects and BPEL and for how to make them play together. They need a sense of the relative performances of the technologies or, better yet, a small set of benchmarks that they can run to characterize their own environments. It is important to make sure developers are aware of the classic distributed design patterns. We recently talked to a company with a major web service performance problem, only to find out that their developers were using a naïve web service design to implement fine-grained computations between two processes on different machines.

Many business processes, especially those which cross organisationational boundaries achieve transaction semantics through asynchronous services. This imposes a new currency paradigm on developers used to working in an RPC world. Expect to see a lot more UML interaction diagrams and a lot more tools for dealing with service deadlock. Services designers should have a strong background in protocol state machines and requires a new role of Service Architect in many organizations. Finally developers need to know how to design long running business processes and how to upgrade them. One major omission in many service definitions is versioning, which means that it is difficult to support legacy services and introduce new ones. At present I don't believe there is any accepted way for dealing with versions of a service.

The promises of SOA are great but the latent complexity and the disruption created by pitting different technology proponents indicates one should tread carefully. Focus on service definitions of high value services well before adopting a major platform or toolset. Leverage best practices in interface design, and distributed system design. Make sure that service designers are properly educated in the use of protocol state machines.

REFERENCES

1. Joshua Bloch: "How to Design a Good API and Why it Matters", http://lcsd05.cs.tamu.edu/slides/keynote.pdf

2. SCA/SDO http://www.osoa.org/display/Main/Home

About the author

Dave Thomas is cofounder/chairman of Bedarra Research Labs (www.bedarra.com), www.Online-Learning.com and the Open Augment Consortium (www.openaugment.org) and a founding director of the Agile Alliance (www.agilealliance.com). He is an adjunct research professor at Carleton University, Canada and the University of Queensland, Australia. Dave is the founder and past CEO of Object Technology International (www.oti.com) creator of the Eclipse IDE Platform, IBM VisualAge for Smalltalk, for Java, and MicroEdition for embedded systems. Contact him at dave@bedarra.com or www.davethomas.net.