What Would Smart Services Look Like*

And How Can We Build Them on Dumb Infrastructure?

Keith Duddy

Queensland University of Technology / Smart Services CRC
126 Margaret St, Brisbane, QLD 4000, Australia

Abstract. The research for the next generation of service oriented systems development is well under way, and the shape of future robust and agile service delivery and service aggregation technologies is taking focus. However, the distributed computing infrastructure on which these systems must be built is suffering from years of "worse is better" thinking, and an almost invisible vendor fragmentation of the Web Services and Business Process Modelling spaces. The balkanisation of the basic technologies of service deployment and business process implementation threatens to undermine efforts to build the next generation of Smart Services. This paper is a summary of the keynote talk at WESOA 2008, which accounts for its informal style. It paints a picture of services futures, reveals the problems of the present tangle of technologies, and points to some practical initiatives to find the way out of the mess.

1 The Speaker, and His Journey "Up the Stack"

Keith Duddy is a graduate of the University of Queensland, and went from Honours in Computer Science to industry in 1990 where he worked on extending PC-Unix systems to use plug-in hardware which supported up to 32 serial ports for multiple terminal, printer and modem connections. His work included a mixture of hacking C code and supporting customer applications in 12 European countries. In 1993 he returned to the University of Queensland as a Research Assistant, and apart from a two year sojourn in the Health sector working for a government institute, he has been a professional researcher with Cooperative Research Centres – Australia's flagship program for industry, government and university collaboration.

Keith's research focus moved "up the stack" from serial communications protocols to working with CORBA Middleware and its supporting infrastructure services, including contributions to the CORBA Trader and CORBA Notifications standards. In 1996 Keith and a colleague organised for his employer, the Distributed Systems Technology Centre (DSTC), to be contracted by the Object Management Group (OMG) to develop and host *CORBAnet*, the CORBA

* The work reported in this paper has been funded in part by the Smart Services Co-operative Research Centre through the Australian Federal Government's CRC Programme (Department of Innovation, Industry, Science and Research).

G. Feuerlicht and W. Lamersdorf (Eds.): ICSOC 2008, LNCS 5472, pp. 5–14, 2009.

2.0 Interoperability showcase. This became an ongoing web-accessible demonstration of 12 separate CORBA implementations making calls on one another's distributed objects.

In the late nineties Keith became leader of the Pegamento project at the DSTC's successor, the Enterprise DSTC, investigating the confluence of middleware and workflow technologies, and began making extensive use of the DSTC's implementation of the MOF meta-modelling standard of OMG, which his DSTC colleagues had played a major role in specifying. Keith was the first author of a standard UML profile – for the representation of CORBA interfaces. As an OMG Architecture Board member, he was a co-author of the first technical white paper on the Model Driven Architecture, which was a concept already in use in EDSTC at the time.

His research focus continued to raise the level of abstraction in the specification and development of distributed systems – but with a focus on the flip side of that coin, reification of abstract specifications to real implementations by capturing parametric transformations from specification to implementations on multiple platforms. After being part of the large team that standardised the EDOC Metamodel and UML Profile in OMG, which resulted in some influential ideas being incorporated into UML 2.0, Keith was one of the authors and sponsors of the OMG's RFP for Model Transformation, to which his team contributed their in-house technology for MOF model transformations.

After EDSTC closed in 2004, Keith went to the Australian National e-Health Transition Authority (NEHTA) to apply his architectural and modelling experience in the health sector. This organisation has been applying Service Oriented Architecture principles, along with standards-based interoperability in an environment where documents and services need to be shared between a large number of public and private sector organisations, across three tiers of government, with diverse funding arrangements, and involving a large number of software vendors and developers. The experience of attempting to use COTS Web Services middleware which *claims* conformance to a variety of W3C and OASIS standards was eye-opening to say the least.

The latest phase of Keith's career is back in the CRC research framework – at the Smart Services CRC, where he is involved in two projects called *Service Aggregation* and *Service Delivery Framework*. These newly formed projects aim to bring the best of breed research in Service Oriented Computing at four Australian universities, together with applied research at SAP, Infosys and some government departments to offer a framework for practical SOA deployment.

2 What Is a Smart Service?

The short answer is that it is a marketing term to bring together various meanings of the term Service (economic, technical, political, business- and end-user-oriented) with an adjective to make it sound clever. The Smart Services CRC has an impressive portfolio of 11 projects with focus on finance, media and government which cover the spectrum of these meanings.

What we mean by *Smart Service* in the context of the Service Aggregation and Service Delivery Framework projects is easier to narrow down. We use Service Oriented Architectures as a basis for the meaning of *Service* – a function of an enterprise that is exposed through various technology-supported channels, and is amenable to re-use and composition into larger services which add value. Therefore a *Smart* Service is one that is better enabled for this purpose by approaches and technologies that are being developed in the CRC.

Smart Services, or the infrastructure in which they are deployed, will have some or all of the following properties.

2.1 Metadata

In order for a service client to be able to select the most appropriate service in a given environment for its needs and budget, the services available must be able to expose both their functional interface, as well as a set of non-functional properties as metadata. This includes concepts such as Quality of Service (QoS), which usually include things characterised as *-ilities*, like reliability, availability and execution time; as well as other properties like trust and reputation, operating hours, and other aspects of service provision usually found in Service Level Agreements (SLAs). In current service provision models, these are usually agreed on paper in a static framework of guarantees and compensations, and not exposed in a computer-readable form. Section 7 provides a suggested path forward for exposing service metadata.

2.2 Recursive Process Orientation

The emerging literature at ICSOC and other conferences and journals in the SOA space makes it clear that the name of the services game is to expose encapsulated business processes on one hand, and to create additional value by reusing services in the context of new business processes. The array of technologies used to choreograph and/or orchestrate service invocations is as wide as the Business Process Management field itself. The term *recursive* in this section header is important, however, as it requires the generalisation of the concepts of using services to enact business processes, as well as the creation of new services by exposing a wrapper interface to a business process itself. A process uses a service, which encapsulates a process, which invokes services, which encapsulate other processes, and so on.

The degree to which the process nature of services is exposed will vary, depending on factors such as traditional information hiding (which dictates opacity) and automated monitoring and use of formal methods for validation (which require increased transparency). To a large extent this will be dictated by organisational boundaries, with process exposed inside an enterprise, and hidden outside its borders to protect competitive advantage. A middle path being investigated by some researchers is one in which an abstract process definition is exposed, reflecting the logical execution of a service, but the complexity of the actual implementation is hidden.

Smart infrastructure to support service aggregation based on process definitions will be able to navigate the recursive process enacapsulations as deep as they are exposed within its context.

2.3 Service Selection and Substitution

When metadata is available to technical service execution frameworks, such as the increasingly dominant BPM paradigm, a service can be compared to others implementing the same logical function, using the same interface, and the one most fit for use can be selected. This is very simple when comparing only a single QoS parameter, however, various techniques exist to allow the optimisation of a whole process execution, involving multiple services, and taking into account multiple non-functional properties. These include Integer Programming, and Generic Algorithms, among many others. Some approaches apply local optimisation of the properties of each task in a process, and others consider the execution of a whole process and perform global optimisations.

However, the case where a large number of equivalent services with the same interface, varying only by QoS, are available is rare, and usually happens only inside a large "hub" or marketplace where the host defines the interfaces and makes them standards for provision of services into the market. Finding two Web Services "in the wild" with the same interface is exceedingly rare. This implies that some combination of the following approaches needs to be used:

1. Industry sectors and other groups using services will need to produce standardized "commodity" interfaces that will be implemented the same by all participants in that sector or grouping. (The political solution.)
2. Techniques need to be used which analyse services for compatibility, and transform similar interfaces to make them suitable for substitution for one another. (The technical solution.)

Although the precedent for industries in standardising data transfer formats, and sometimes interface definitions for their exchange, is there in bodies such as ISO, IEEE, ITU and even OMG, there is little evidence of this work being applied to things like standard WSDL definitions for domains. And for leading edge service integration activities in the current world, the need for Semantic-Web and AI-based techniques to discover service equivalences is obvious. Some of these technologies are well advanced in a research context, but mainstream efforts are still bound to manual wrappers around interfaces to bridge their syntactic differences, based on a human analysis of their semantic similarities. Once again, whether using ontologies and other semantic approaches, or AI techniques, the candidate services require good metadata and documentation above and beyond their interface definitions.

2.4 Constraint Enforcement

When using service aggregation techniques to bind multiple services into roles in a larger context, the ability for a high-level specification to indicate appropriate resource and other constraints is necessary, as the implementations of the

services are probably hidden from their BPM invokers. The classical example of such a constraint is the encapsulation of a legislative requirement for different entities to play the roles of *bank* and *auditor* in financial processes. Other example constraints include the selection of a service provider only from a list of nominated business partners, or the same storage provider to be used for all storage-related activities in a process for financial reasons. Smart infrastructure for service usage will extend the kinds of resource management functions of current Workflow environments to include the use of the metadata available about services (and their providers) to include constraint enforcement.

2.5 Metering and Billing

Ever wondered why there are only trivial examples of Web Services available on the open web? E.g. check the ISBN of a book, convert degrees Fahrenheit to degrees Celsius. It's probably because, without a standard per-use or per-volume metering and billing infrastructure available right alongside the WSDL endpoint information, no-one wants to make anything of value available when there's no person at the end of the interaction to view an advertisement – which is the standard entry-level charging model for the provision of high-value services to people through web browser interfaces.

To provide the incentive for service providers to offer their non-trivial services to a set of clients, the infrastructure needs to support a metering and billing apparatus. The current web economy is supported by consumers providing credit card details (or PayPal credentials) when they pay for services (usually on a subscription basis), or by advertising revenue when those services are presented through the web browser. The credit card/PayPal model typically applies when purchasing goods or services ordered, but not provided, via the web. There is no widespread model for charging for services used by aggregators to provide value-added services to users.

Truly smart services will have the ability to charge their users using multiple charging models, and unless a financial services company steps into this niche, other brokers will need to assume this role, and ultimately use the banking system to transfer funds at billing time. However, this is a space where a standard or *de facto* standard (analogous to a PayPal) for metering, billing and payment is needed.

3 Traders, Directories and Brokers

The idea of an *Open Service Marketplace* has been around since the early 1990s, and by 1995 ISO and the OMG had begun jointly standardising a "Trader" specification, which was one of the roles identified in the then recently published *ISO Reference Model for Open Distributed Processing*. RM-ODP (as it known for short) is a landmark publication which gave distributed systems researchers and practitioners the same kind of language for discussing distributed systems that the OSI specification had given network protocol designers a decade earlier. Both models continue to be relevant as reference sources today, even though networks still do not have seven layers, and the Trader role in distributed systems has failed to manifest.

Trader was a repository that contained *Service Offers* which had a *Service Type* describing both the interface and non-functional properties of a Service[1]. It was meant to allow service clients to specify queries across the properties which identified valid service offers for the needs of the client. Implementations of the OMG Trader, however, failed to find an application outside a few niche financial and telecoms management scenarios.

The next major hype about repositories of services came about with the release of the UDDI specification. The story that seemed to be believed by major vendors, including HP, IBM, Microsoft and SAP was that a global repository of Web Services would be available which would be populated by millions of endpoints, which could be navigated using their type and other metadata. The dream was backed up by a global scale implementation sponsored by IBM, Microsoft and SAP, and which lasted from 2001 until 2005, when the lights were turned off, due to lack of interest by Web Services developers.

So what's different about the concept of a Service Broker – a component that will store endpoints to multiple services and act as the place to discover the services needed in a particular marketplace? Sounds like déjà vu. Well, firstly, a broker will have a trusted relationship with both client and service provider (although it does not require that these parties are initially aware of one another), and a contract that permits it to charge for the services it delivers on behalf of the service providers, as well as taking a cut to sustain its own role in the marketplace.

Secondly, it will not only store the service endpoints for discovery by clients – it will be a logical intermediary in every interaction between clients and services. This mediation is mostly for the purposes of metering and billing. The broker knows which services are used, by whom, and for how long (or with what payload, or any other metering criteria that are useful), and has a contract with the clients which allows them to be billed for what they use.

4 How Does All This Relate to Web Services Standards?

The unstated assumption, since UDDI, is that when we speak of a "Web Service", that we are referring to a network-accessible endpoint which behaves in a predictable way (as specified in an interface description) when sent a message – much like its middleware predecessors, CORBA, DCE and Sun RPC. On the face of it, this appears be true, with the standards SOAP and WSDL being what makes this possible. But when one looks closer, it is clear that these standards offer so many variations in style (RPC, Document, Literal, Wrapped) that no vendor of Web Services toolkits offers all of the variations... and what's more, the intersection of the subset of the standards that they support is empty (or very small, depending on which vendor's products are under consideration).

[1] Interestingly, the Type Management function of RM-ODP which was intended to model these properties morphed into the OMG's Meta Object Facility (MOF), which emerged at the same time as the UML, and has since become intervolved with that standard.

A whole industry consortium was formed on the basis that the standards from W3C and OASIS offer no profiles for their usage which can foster interoperability: Web Services Interoperability (WSI). This body attempted to overcome the ambiguities and range of implementation choices offered by the standards, but the evidence seems to be that they have failed to do so, with even the WSI-Basic profile for the use of Web Services offering three variations, and some of the major toolkit vendors (like Microsoft) opting out of supporting even one of these. Web pages like [7] and [8] give an indication of proliferation of N by N interoperability problems that programmers face when using more than one Web Services toolkit. An example of an attempt to specify a common profile that makes no specific product references can be seen in [1], specified by NEHTA. The problem with this sort of profile is that its implementation in a particular toolkit often requires hand modification of SOAP messages after they have been created in a non-conformant manner by the generated code, and no documentation exists in the toolkit to assist.

The current facts of the situation are that out-of-the box interoperability, without hacking the XML in the SOAP messages for a service by hand, does not exist between the two most popular WS frameworks: JAXWS and .Net. And this is just using the most common WS-* standards: WSDL, WS-Addressing and WS-Security. There are more than 50 current standardisation efforts in W3C and OASIS that attempt to augment these basic components to do every other middleware function imaginable, from Transaction and Policy, to Reliable Messaging, to BPM. The dependency matrix between the standards is highly complex, and often simply contradictory, with many of the standards gratuitously overlapping in functionality. The principle of separating interface from implementation is paid lip service, but in reality the only way to get interoperability is to implement all services and their clients using the same vendor's platform.

The current .Net toolkit does not even allow the use of a WSDL with arbitrary XML payloads – even when using the right combination of literal and wrapped conventions and the right choice of security signing before encrypting, rather than the other way around. The Microsoft response to support calls to assist in delivering HL7 (a heath domain standard) XML payloads using Web Services is a recommendation to design everything from scratch in C# using Visual Studio, and then generate the WSDL, as "WSDL-first development is not supported".

5 What about the Web Services Success Stories?

A number of web-based platforms and Software-as-a-Service offerings are usually held up by Web Services spruikers as *WS Success Stories*. These include Rearden Commerce, which is the platform used in American Express's Axiom corporate travel spend-control site; RightNow.com and SalesForce.com, which are CRM systems hosted on behalf of clients in larger and smaller companies respectively; and last, but not least, Amazon.com, which integrates thousands of retailers' product offerings through its portal using WS-* specifications. So when these successful businesses base their multi-billion dollar strategies on WS, how can I claim that it's a big non-interoperable mess?

What is really happening is that WS-* is hidden from the vast majority of users of these systems - who mainly access their services using POWP (plain old web pages). The WS part of these companies' solutions is very carefully managed between the hub, and their strategic partners – a so-called Walled Garden. The hub controls all of the WSDL defined, and they use only the simplest WSDL definitions. These interfaces convey only very simple XML payloads (in the Amazon.com case, everything in the WSDL is a string element). Virtually none of the other WS-* standards outside of WSDL and SOAP are used (the exception being some use of WS-Addressing). WS is used to connect to a proprietary set of functionality to support user profiles, events and notifications, and payment and charging infrastructure.

Another interesting thing to note is that Google has decided that WS-* is not ready for prime time, and instead they support a large set of language library APIs that do proprietary communications back to the Googleplex.

6 What Other Challenges Do We Face?

6.1 Business Process Management

One of the key emerging technologies for aggregating services into useful business functions is Business Process Management (BPM), which is being used increasingly to choreograph services as well as the activities of people. Unlike the service invocation space, in which WSDL has emerged as the *lingua franca* of interface definitions (despite the support for the standard being partial and often non-overlapping in the various implementations), BPM is still in a state where a number of competing languages are being used to define processes.

BPMN is favoured by analysts, and has increasing tool support, although the standard does not have a well defined execution semantics. UML Activity Graphs have a petri-net inspired token-passing semantics (with "semantic variations"), and is supported as a graphical language by most UML tools, although most of these do not perform the necessary well-formedness checks, and thus leave most Activity Graphs that I have ever seen with errors which make them essentially semantic-free. BPEL is based on Pi-calculus, and has no formal mappings to the graphical languages, and no graphical syntax of its own. YAWL is formally defined based on modified petri-nets, but has only a single open-source implementation.

There are myriad other languages for BPM, all of which are fundamentally incompatible, and therefore there does not exist any general way of translating between them, as can be done with RPC interface descriptions of many kinds. This is a field in which it is also unlikely that the industry will settle on a single formalism, as there are constant innovations (of the reinventing the wheel sort), which throw in new concepts and notations (as well as a host of partially defined, or informal, description techniques) to muddy the waters.

6.2 Quality of Service

The term QoS has been refined and formalised in the context of networking and telecoms management over the last decade or so, and companies like Cisco have

made a fine art of managing basic networking services. However when applying the basic idea – of dynamically available meta-data about the performance of the non-functional aspects of a service – to general computing services, a lot of confusion exists in the literature, and some basic questions are unanswered:

- Is *execution time* end-to-end, or just at the server. Is it average, or maximum, and who measures it?
- Is *price* the access price (e.g. searching the catalogue) or the price of the eventual commodity (e.g. buying an MP3). Is price even a QoS?
- Who rates *reputation and trust*? Who stores it? Is the mechanism revealed (number of reviews, etc)?
- Who stores any QoS property? And in what format?

Most academics working with optimising service aggregations make assumptions that QoS is available and accurate, that it is variable and differentiated between otherwise identical services. Whereas the large deployments of WS that we know of use mechanisms like Service Level Agreements to ensure that alternative providers of services all comply to the same level of quality, and none of them store and/or calculate, or query, QoS properties for individual services at runtime.

7 Some Practical Initiatives to Overcome Challenges

7.1 Service Description Meta-models

The PhD work of Justin O'Sullivan [4] is fully documented at http://service-description.com/. It resolves many of the questions asked about QoS, and has been used in European Union projects, and is included in new product development underway at SAP.

7.2 Domain Specific Languages

DSLs raise the level of abstraction when specifying the operation of software (and hardware) in a particular domain setting. Many industries have successfully standardised documents and processes that apply across a sector, using MOF, XML, UML and other abstractions. Much of the promise of OMG's Model Driven Architecture is embodied in the use of DSLs to abstract away from much of the technology and implementation detail when specifying a system. Even when no automatic code generation is possible, the effort of extracting irrelevant detail from a problem specification makes the design more understandable, more portable, and more amenable to automated code generation or bespoke platform construction techniques (such as software factories).

7.3 KISS Initiative

The KISS (Knowledge Industry Survival Strategy) Initiative puts forward a Modeling Tool Interoperability Manifesto and Projects. In addition, the initiative in running a Workshop Series at major conferences throughout 2009/2010. See http://www.industrialized-software.org/kiss-initiative.

8 Conclusion

Standards development in the Web Services space has hit an all-time quality low, with conformance being left unconsidered, or at best patched in after the fact by other standards attempts. Architectural oversight is almost non-existent across the fifty plus ongoing WS-* standards processes in W3C and OASIS, and many potential standards overlap in functionality and intent, with support from vendors very fragmented.

Thankfully in another decade it is very likely that we will be throwing this all away and starting over for a third generation of middleware standards, according to the perceived needs and fashions of the time.

Smart Services will be the combination of a range of techniques to describe, coordinate, allow invocations of, and allow billing of individual Web Services. These services are likely to be kept to a small common subset of interface types and styles, and use simple XML types to convey payloads.

The discovery, substitution and construction of variations of services will also be facilitated by Smart Services infrastructure. This will contain the "duct-tape and ticky-tacky" needed to glue together the incompatible parts of the WS world.

The aggregation of simple (and wrapped complex) services will increasingly use BPM techniques, but this field will necessarily be fragmented according to the kind of BPM language used, as there is no logical path to translating BPM models between the different formalisms.

References

1. NEHTA: Web Services Profile, Version 3.0 (December 1, 2008)
2. Hamadi, R., Benetallah, B.: A Petri Net-based Model for Web Service Composition. In: Proceedings of Fourth Australasian Database Conference (2003)
3. Zeng, L., Benetallah, B., Ngu, A.H.H., Dumas, M., Kalaggnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30(5) (May 2004)
4. O'Sullivan, J., Edmond, D., ter Hofstede, A.: What's in a Service? Distributed and Parallel Databases 12(2-3), 117–133 (2002)
5. Recker, J., Mendling, J.: On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages, `http://citeseer.ist.psu.edu/recker06translation.html`
6. Cohen, F.: Understanding Web service interoperability: Issues in integrating multiple vendor Web services implementations (Februray 2002), `http://www.ibm.com/developerworks/webservices/library/ws-inter.html`
7. Simon Guest: Top Ten Tips for Web Services Interoperability (August 2004), `http://blogs.msdn.com/smguest/archive/2004/08/12/213659.aspx`
8. Ye, W.: Web Services Interoperability between J2EE and .NET (April 2005), `http://www.developerfusion.com/article/4694/web-services-interoperability-between-j2ee-and-net-part-1/`