Concise paper

A document-centric design pattern for Service Oriented Architectures

Mike Lopez **John Peppiatt**

Manukau Institute of Technology
Manukau, New Zealand
mike.lopez@manukau.ac.nz

ABSTRACT

A Service Oriented Architecture (SOA) is an architectural style that values coarse-grained loosely-coupled components that relate closely to real-world business functions.

This paper explores the issues involved in developing and deploying applications that use SOA. It summarises the best practice guidelines advocated by leading practitioners and evangelists and presents a document-centric design pattern that enables an application's logic to be decoupled from the underlying technologies chosen.

The design pattern is standards-based and technology-neutral. It allows a unified approach to be taken to connected and disconnected modes of operation in distributed systems.

The document metaphor provides a well-understood basis for communication between managers, technical specialists, business process analysts and users.

1. INTRODUCTION

A Service Oriented Architecture (SOA) is an architectural style that values coarse-grained loosely-coupled components that relate closely to real-world business functions.

This paper discusses SOA in the context of web services. These, however, are only one method of implementing a SOA; other options include email, TCP/IP and HTTP. For this paper, we define a web service as a network addressable end-point that has a number of methods with signatures defined in Web Service Definition Language (WSDL) as described by Christensen et al (2001).

The key goal of SOA is to enable agile systems; agility is defined as the ability for a system to cope with requirements that were not known when the system was developed. SOA achieves this by separating the definition of what a service does from its implementation. This separation leads to three roles:

- A provider offers an implementation of a defined service
- A consumer can use any implementation of a defined service.
- A broker can connect a consumer to a provider.

The de-coupling of a service definition from its implementation enables component-based development but, as with previous component technologies such as Microsoft's Component Object Model (COM) and the Common Object Request Broker Architecture (CORBA) as defined by the Object Management Group (OMG 2000), the major obstacle to the development of such re-usable components is the difficulty of making a business case to fund the necessary generalisation of functionality; typically, only package developers view solutions generically rather than focus on immediate implementation goals.

2. SUMMARY OF BEST PRACTICE

The first three items below stem directly from the SOA definition given above; the remaining items are a summary of best practice recommendations from Microsoft, Sun, IBM and Bea.

2.1 Loosely-coupled

The de-coupling of the specification of a service from its implementation allows a service to be replaced or re-engineered without affecting the consumers of the service. In the authors'

opinion, a broader concept of loose-coupling is useful. Consumers can be loosely coupled as to service and version so that they can work with a range of compatible services and versions. Providers could provide multiple versions and presentations of their functionality. Such loose-coupling is achieved by a generalisation of the abstraction modelled by a service.

2.2 Coarse-grained

It makes sense for messages sent between objects within a local address space to be finely grained. These messages are typically implemented as method calls and as such have little overhead. Messages sent over a network have substantial overhead and in particular high latency; there is a substantial performance benefit from sending a smaller number of larger messages.

Benefits of coarse granularity go beyond efficiency gains; larger messages allow context to be passed, enabling methods to be stateless. From a SOA perspective, all services should be stateless; state implies some knowledge of the inner workings of an implementation, breaking the loosely-coupled objective.

2.3 Business-related

Services should relate to real-world business functions. The key idea here is that at a high enough level of abstraction some concepts are invariant. Businesses can agree readily on what is meant by entities such as an employee, a customer, or an order, even though they might each hold different data for these or have different functionality. The more closely services express the functionality of invariant concepts, the more likely the lifetime of the service definition will transcend the lifetime of a particularly implementation. To achieve this, it is clear that such functionality must be expressed in a generic manner, rather than just exposing a specific implementation.

2.4 Location-transparent

This goal is stated by all the vendors and supported by standards such as UDDI (Oasis, 2004). Few implementers, however, have adopted this aspect. In the authors' view this is because the goal muddles two distinct requirements: the need to choose a service and the need to locate a chosen service. Choosing a service is itself a business process and should be modelled as such, rather than by a separate set of protocols. Locating a service is better expressed by the concept of location mobility. This is a superset of location transparency that also allows directory services to be relocated rather than hard coded.

2.5 Technology-agnostic

An implementation of a service should use the implementer's technology of choice. This freedom is enabled by the Ws-* standards and in particular the basic profile interoperability standard (WS-I, 2004).

2.6 Composable and Orchestratable

It should be possible to compose new services from a set of existing services and processes by orchestration or choreography of services. Arkin et al (2005) describe the Business Process Execution Language (BPEL) which is a proposed standard for the choreography of business processes.

Orchestrating stateless services into processes requires some means of state management. In general this requires both a decision on where to hold the process state, and a decision on how to hold it.

2.7 Secure

The ability to access the functionality of an organisation from the internet enables business to business automation and flexibility in working patterns, but also brings a significant security risk.

The combination of privacy measures, such as an SSL encrypted channel (Freier et al, 1996), together with authentication and authorisation mechanisms will provide an acceptable level of security for most businesses.

It is important, however, to be both vigilant and agile in this area; as vulnerabilities are detected, appropriate counter-measures need to be deployed rapidly. It is unlikely that the necessary agility will be achieved unless security is implemented as an application-neutral capability.

2.8 Manageable

There is an operational need to monitor the workload on servers and to re-deploy services to balance workload. New versions of services need to be deployed in a managed way without the need for concurrent updating of providers and consumers.

Arsanjani (2005) argues the case for a life-cycle view of service orientation and the need for Service Oriented Design and Analysis (SODA) tools.

3. PROPOSED DESIGN PATTERN

The proposed design pattern uses a metaphor of a document to represent all interactions between a service provider and a service consumer. It should be emphasised that this is a metaphor; although "real" documents could be retrieved or stored, in many cases "virtual" documents will be used and software will interface the virtual storage and retrieval to a database.

Retrieving state: All retrieval of information from a service is considered to be the retrieval of a document or set of documents. The consumer will send the service a document request and receive a document in response. The document request is itself a document that is populated with enough information to specify the content of the required response.

Updating state: Updating of system state is expressed by storing a document or set of documents that describe the changes required.

Both storage and retrieval operations also need information about the credentials, capabilities and preferences of the consumer. Given this, all services can be implemented with a single method that has two input parameters (an input document and a consumer description document) and returns a document.

The key benefit of using this design pattern is that by abstracting web services into a single generic method, a great deal of functionality can be implemented in an application-neutral manner. To achieve this, documents should be self-describing.

3.1 Security, Caching, Versioning and Instrumentation.

These can each be implemented in an application-neutral manner. A mature technology such as XSL Transformations (W3C, 1999b) can be used to translate between versions.

3.2 Location mobility

Redirection allows a realm to be redirected to another service location on a temporary or permanent basis. **Routing** allows the service to forward document storage and retrieval requests to another service location

3.3 Validation

The use of a single signature for all methods removes the capability of validating input parameters but it is simple to validate the documents containing the parameters by using XML Schema (Fallside & Walmsley, 2004)

3.4 Audit trail

Every system state change is represented by the storage of one or more documents; an audit trail can be derived from a filtered view of document storage.

3.5 Composition and processes

A document set may be composed from other documents (or document sets). This can be defined in an application-neutral manner. Process definitions such as BPEL may be stored as documents and the state of a process instance can be considered a document.

3.6 Transaction support

Transaction support may be added in an application-neutral, technology-neutral manner. A service can guarantee that a document set is accepted in its entirety, or not at all.

3.7 Technology neutrality

The document metaphor is readily implemented with technology-neutral mature standards such as XML (W3C, 2004).

3.8 Events

Every system state change is, by definition, represented by the storage of one or more documents. Standard events based on document

storage can therefore capture all possible system state changes. Custom events can be defined by applying filters to documents using mature technology such as XPath (W3C, 1999a). Event handlers can be associated with both standard and custom events. The separation of the definition of events from the specification of event handlers facilitates the decoupling of processes from services.

3.9 Connected and disconnected operation

When connected, an application will typically request a set of documents from a service that represents a view of state to be presented to a user. Changes to state will be represented as storage of one or more documents.

When disconnected, the application will typically retrieve its view of state from cached documents and store its updates in an "outbox". Synchronisation involves uploading the outbox contents to the service, and refreshing the cached view of state by retrieving up to date documents from the service.

Substantial support for local cache management and disconnected operation can be provided in an application-neutral manner.

4. CONCLUSIONS

The process of abstraction and generalisation is natural for software architects, but we have broadly failed to communicate the benefits of such an approach to management.

In order to make a sound business case for abstract architectures, we need a metaphor that is readily understood by management, users, business analysts and technical specialists.

This paper proposes the use of a document as that metaphor. It does not solve all the issues, but it does allow many of the issues involved in adopting SOA to be implemented in an application-neutral manner.

REFERENCES

Arkin et al, (2005) "Web Services Business Process Execution Language Version 2.0 (Draft)" downloaded from http://www.oasis-open.org/committees/download.php/11601/wsbpel-specification-draft-022705.htm , April 2005

Arsanjani, A., (2005), "How to identify, specify, and realize services for your SOA", downloaded from http://www.webservices.org/index.php/ws/content/view/full/55442, April 2005

Christensen, E. , Curbera, F. , Meredith, G. , Weerawarana, S. , (2001), "Web Services Description Language (WSDL) 1.1", downloaded from http://www.w3.org/TR/wsdl, April 2005.

Fallside, D., Walmsley, P., (2004)," XML Schema Part 0: Primer Second Edition", downloaded from http://www.w3.org/TR/2004/REC-xmlschema-0-20041028, April 2005

Freier, A., Karlton, P., Kocher, P, (1996),"The SSL Protocol Version 3.0", downloaded from http://wp.netscape.com/eng/ssl3/draft302.txt, April 2005.

OASIS, (2004), "UDDI Version 3.0.2", downloaded from http://uddi.org/pubs/uddi-v3.0.2-20041019.htm, April 2005

OMG, (2000), "Corba Overview", downloaded from http://www.omg.org/docs/formal/00-10-06.pdf, April 2005, see also http://www.omg.org/corba/

WS-I, (2004), "Basic Profile Version 1.1", downloaded from http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html, April 2005

W3C, (1999a), "XML Path Language (XPath) Version 1.0", downloaded from http://www.w3.org/TR/xpath, April 2005

W3C, (1999b), "XSL Transformations (XSLT) Version 1.0", downloaded from http://www.w3.org/TR/xslt, April 2005.

W3C, (2004), "Extensible Markup Language (XML) 1.0 (Third Edition)", downloaded from http://www.w3.org/TR/2004/REC-xml-20040204, April 2005