TECHNICAL SPECIFICATION

SOA Blueprints Concepts

Draft v0.5 (For Public Review)

A move to drive industry standardization of SOA concepts and terminology

http://www.MiddlewareRESEARCH.com

The Middleware Company
Research Team

Steve Wilkes & John Harby
June 2004

research@middleware-company.com

Expert Review Group Members:

Will Pugh (BEA), Paul Patrick (BEA), Steve Jones (Cap Gemini),
Hugh Grant (Credit Suisse First Boston), Chris Peltz (Hewlett-Packard),
Scott Metzger (Truelink), Bola Rotibi (Ovum), Annrai O'Toole (CapeClear),
Frank Cohen (PushToTest), Alan Zenreich (Ness Technologies),
Michele Leroux Bustamante (IDesign), Vince Salvato (E2E Consulting), David Murrell (Xede),
Rob Castaneda (CustomWare), Doron Sherman (Individual), Chris Judson (E2E Consulting),
John Cheesman (7irene), Rainer Ammon (FH Wels), Mike Sawicki (Compuware),
Shigeru Urushibara (UL Systems), Akira Hirasawa (UL Systems Inc), Koichi Hayashi (UL Systems)

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION TO SERVICE ORIENTED ARCHITECTURE

A service-oriented architecture can be defined as a way of designing and implementing enterprise applications that deals with the intercommunication of loosely coupled, coarse grained (business level), reusable artifacts (services). Determining how to invoke these services should be through a platform independent service interface.

While Web services and SOA are usually thought to be synonymous, they are not. It should be made clear that Web services are an important tool and one implementation method for SOA, but there are other patterns that may be more appropriate for any given use-case.

In general, SOA can be thought to consist of service providers and service consumers. The providers define what the service looks like and how to invoke it through an implementation independent service interface. The consumers use this interface to construct the necessary data and invoke the service.

An optional construct is the introduction of a discovery mechanism that acts as an intermediary to which providers publish the service interface and from which consumers discover it. This is useful for enterprises with many services, but is not covered in this specification.

One of the keys to SOA is defining the correct level of granularity. This is a fairly subjective thing, but generally speaking services exposed to other systems should provide operations that correspond to business functions. This does not mean that all services are coarse grained. Finely grained component services may be used by business services, but would not be exposed to other systems.

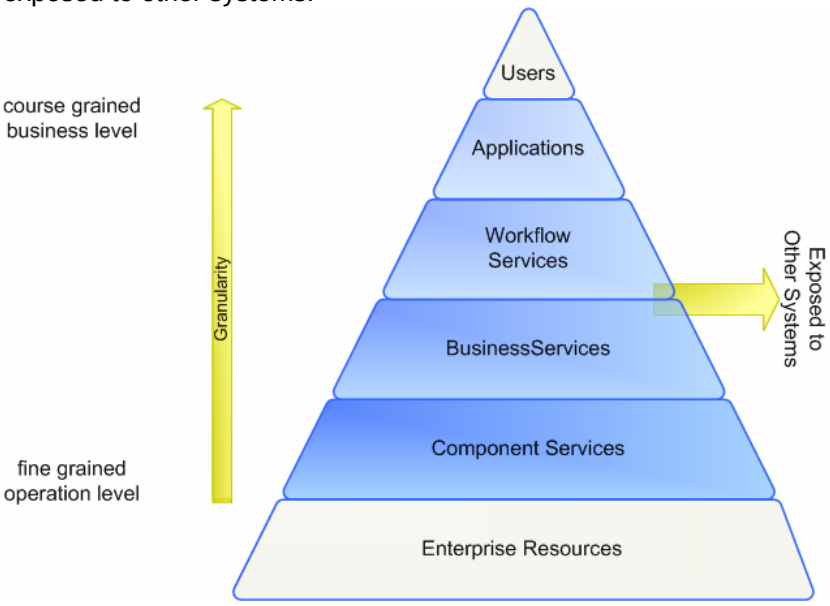

Figure 1 - Service Granularity

For example, in the case of an Expense Reporting Application, Expense Processing may be a workflow service triggered by submitting an Expense Report document. This may invoke business services to update the report, or cut a check, at various places in the workflow. The Expense Processing, report update and cut check services are all exposed (with sufficient security) so they can be used by other applications. Component services to add a line item or get the employees name are used internally, but not exposed.

2 COMMON SOA TERMS

2.1 Synchronicity

Services can be invoked in one of two modes, synchronously or asynchronously. The mode chosen for a particular service depends upon its potential usage, how long the service takes to run and how reliable the service invocation needs to be.

2.1.1 Synchronous Services

Synchronous services return a response to the invoker of the service after the service has completed processing. Some usage patterns (such as a Web application retrieving data via a service) require this kind of interaction. Such services cannot take more than a couple of seconds to execute and are not inherently reliable. The service invocation is not guaranteed and may terminate due to transport issues. The transport for such services is usually local invocation or remote invocation utilizing HTTP.

2.1.2 Asynchronous Services

Asynchronous services do not return any response to the invoker, although they may return an acknowledgement of receipt. The communication of the status of processing or the return of any requested information is usually handled by sending a return asynchronous message through a callback or other mechanism. These messages have to be correlated in order that they can be matched with the original request (standards such as WS-Addressing and BPEL4WS aim to help define the correlation mechanism). The transport for such messages can be over http, but is often through a message broker or other asynchronous transport such as SMTP (email).

2.2 Component Services

A component service is a simple atomic action on a simple entity that does not depend on another service to function. For example, database access to a single table can be thought of as a component service. Operations such as get, add, update or delete would invoke equivalent SQL statements against the database. There are typically no internal rules or invocations of other services through a component service.

2.3 Composite (Business) Services

A composite service, termed a business service in this specification, is also atomic in nature, but orchestrates the invocation of component services into a business level process.
For example, submitting an expense report may invoke component services to add an entry to the ExpenseReport table, add multiple entries to the ExpenseReportItem table, send an email to an employee, create a task and place this in the employee's manager's task list. A composite service is stateless as viewed by the consumer, however, and does not manage a long lived transaction, as opposed to a workflow service. Composite services may be invoked synchronously, or asynchronously and may hold internal state while invoking other services.

2.3.1 Serial Service Orchestration

Within a composite (or workflow) service component services (or other composite services) can be invoked in order. Serial orchestration is the process by which these services are invoked in order, waiting for the completion of one before the next is executed.

2.3.2 Parallel Service Orchestration

Parallel orchestration involves the execution of services concurrently. That is many services may be invoked at one time. Certain sections of a composite service may be concurrent, with a join point at which all services must be complete before moving on.

2.4 Conversational (Workflow) Services

A conversational service typically has state attached to it and looks like a classical finite state machine. A certain operation on that service will start the conversation and set some item into a specific state. Subsequent operations may continue the conversation and change the state of the item. The conversation is ended with an operation that sets the item to a final state. Operations within the service may be invoked synchronously or asynchronously and the service needs to have a mechanism for correlating individual operations. The industry is working on specifications, such as BPEL4WS that aim to standardize the definition of the conversations and correlation.

2.5 Data Services

A data service provides a mechanism for querying a datasource or multiple datasources through a message based request response mechanism. The user of the data service is not aware of the actual physical source of the data, nor its storage format. Data services can be combined together to provide a single response containing data from multiple services. The data service is responsible for routing query parameters to the correct source and combining resultant data in the correct response message format.

2.6 Publish-Subscribe Services

Publish-subscribe services are ones in which interested parties may request notification of certain events. Some entity manages a list of subscribed parties and publishes notification in the form of a message when the event takes place. Services of this type are not bound to any form of transport and the services invoked upon publishing can be of any type. Subscription and subsequent publishing of messages could be through a Message Broker, or managed explicitly by a set of subscribe/unsubscribed messages sent to a subscription manager.

2.7 Service Brokers

This specification defines a service broker as an intermediary service that manages the invocation of a set of registered services based on a set of rules. This incorporates routing of the messages and possibly data transformation between the incoming message and the requirements of the brokered service. A broker may itself be configured to be invoked synchronously or asynchronously.

The routing to registered services can be message based (only the actual message name matters) or content based (some part of the data contained in the message is used in the routing rules).

The broker may invoke one or many services concurrently depending on how it is configured. If many services are invoked it may wait for all to complete or just one to complete before notifying the client, if running synchronously.

2.8 Exception Handling and Compensating Services

When a service invocation fails with an exception, there usually needs to be some way of handling this failure. Timeout or watchdog mechanisms can also raise exceptions in the cases where a certain time period has expired, or a value has crossed some threshold. A simple mechanism of handling such exceptions is to log or report them via some notification to the invoker of the service. In a complex business transaction, however, some action may need to be taken if a service that was expected to succeed as part of the transaction, fails. A compensating transaction is a mechanism for undoing some actions that were already completed that are now inconsistent because the service failed. This can be extended for workflow services, where some transaction may have to compensate for actions committed previously in the workflow.

2.9 Interception and Extensibility

Interception is a mechanism for inserting additional functionality into a system without modifying or affecting existing components. Functionality that cuts across many aspects of a system can be inserted via interceptors, extending the capabilities of the system. For example, a logging or auditing service could be added to a security service by adding a logging interceptor to the access points of all security services.

2.10 Interoperability

Interoperability is a requirement of any service that may be accessed from multiple platforms. It essentially means that the invocation mechanism, message format, data format and security requirements of a service can be interacted with successfully by any SOA implementation. In the context of this specification this means that any service consumer should be able to utilize any provided service without modification.

2.11 Service Security

To ensure protection of confidential resources a variety of techniques are available within SOA to ensure security. This can be applied at various levels within the transport stack.

2.11.1 Http Authentication

Basic http authentication will secure web services on the http transport layer. This requires that a client to that service passes some credential to gain access to the service.

2.11.2 Https Encryption

In addition to authentication, more security can be applied by using https with the Secure Sockets Layer (SSL) protocol. This utilizes a public / private key algorithm to exchange a (symmetric) key used to encrypt and decrypt data. Such security reduces the chances of a third party intercepting and understanding information passed to and from a service.

2.11.3 XML Signing

Authentication validates the identity of a party and encryption helps to make the information contained in a service invocation secure. XML Signing adds to this security by providing a mechanism to ensure validate the identity of the sender and to check that the actual data transmitted has not been tampered with between the consumer and provider of a service. This is achieved by applying an algorithm that provides a signature corresponding to sender and the contents of a message. If the data is modified the key will no longer match the contents and such situations can be caught immediately.

2.11.4 XML Encryption

At a higher level in the message stack than Https encryption, XML encryption provides requirements for a XML syntax and processing for encrypting digital content, including portions of XML documents and protocol messages. It describes how to use XML to represent a digitally encrypted Web resource (including XML itself) in which the XML representation of the encrypted resource must be a first class object (i.e., referenceable and consequently describable, signable, etc.) and represented by a distinct element type.

3 REQUIREMENTS FOR IMPLEMENTING SOA

A service-oriented architecture provides the implementation patterns required to construct applications from loosely coupled services. In order to build such applications, an implementation environment should provide the following capabilities:

- Definition of services independent of their implementation, location or use
- Implementation and hosting of services as a provider
- Location and usage of services as a consumer
- Assembly of services from other services and business rules
- Support for synchronous, asynchronous and conversational services
- Orchestration of application presentation built on services and rules
- Support for multiple forms of human interaction (such as portal, email, wireless, etc.)
- Automated data transformation between disparate data structures
- Provisioning of local and remote services
- Support for simulating, testing and debugging of services

The reference example requirements specification is designed to test the abilities of an implementation environment for each of these requirements.

4 GLOSSARY

To be completed

application

asynchronous

implementation

interface

security

service

synchronous