



OOPSLA
CONFERENCE

Building Service-Oriented Architectures (SOAs) with Web Services

OOPSLA 2008 – Tutorial 12

Olaf Zimmermann
IBM Research

Updates available from: <http://www.soadecisions.org>

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

1

readme.txt

Terms and Conditions of Reuse

- This is the handout material for OOPSLA 2008, © Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, IBM Corporation and Springer-Verlag 2003-2007 (except if indicated otherwise on individual slides).
- As the copyright is held by the author/owner(s) of the referenced material, any form of reuse requires explicit written permission from the authors/owner(s).

Request for Feedback

- We welcome feedback – good or bad – on the contents of the presentation. Drop us a note at info@perspectivesonwebservices.de

This presentation contains excerpts from the book "Perspectives on Web services" by Olaf Zimmermann, Mark Tomlinson, and Stefan Peuser, Springer-Verlag Berlin Heidelberg New York 2003, ISBN 3-540-00914-0.

This work is subject to copyright. © Springer Verlag Berlin Heidelberg 2003. All rights reserved. This material must not be published as a whole or in part without permission.

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

2

Building SOAs with Web Services

Tutorial Introduction

Abstract

- Service-Oriented Architecture (SOA) concepts and Web services technologies have matured into highly attractive architecture and implementation alternatives for building distributed systems. SOA concepts and Internet protocol-based implementation stacks are a powerful combination that is well-suited for crafting heterogeneous business-to-business and enterprise application integration solutions.
- In this tutorial, we introduce SOA as an architectural style defined by patterns such as Enterprise Service Bus (ESB) and service composition. We present two industry case studies that demonstrate where and how these patterns can be applied in practice. Next, we present selected elements of the Web services technology standards stack from an application programmer's perspective, for example SOAP and WSDL.
- In a third module, we design and develop a complete sample application applying the introduced concepts and technologies, making use of the Web services editors and code generators in the Eclipse Web Tools Project (WTP). We conclude with a discussion of the key architectural decisions on SOA and Web services development projects – for example REST vs. SOAP message exchange; service interface creation process and granularity concerns; security, reliability and other quality-of-service factors; server-side deployment and client-side invocation guidelines. This discussion centers around the lessons learned on large-scale industry case studies.

Tutorial positioning

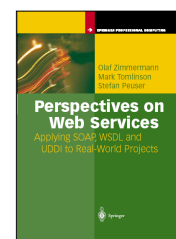
- Attendee background:
 - ▶ The tutorial is targeted at practitioners with general background in software engineering and object-oriented programming; familiarity with XML and Java is a plus.
- Tutorial objectives:
 - ▶ The tutorial is aimed at software architects wanting to design end-to-end SOAs and developers wanting to develop and deploy Web services requestor and provider applications. Attendees will develop an understanding of the principles and patterns defining SOA as an architectural style, and will learn how to leverage open source tools such as Apache Axis to create Web services descriptions, server-side stubs, and client-side invocation proxies for several languages.

Instructor(s)



- **Olaf Zimmermann** is an Open Group Master Certified and IBM Senior Certified IT Architect and Research Staff Member. His areas of expertise include distributed computing and Service-Oriented Architecture (SOA) in general and Web services in particular.

Over recent years, Olaf has conducted numerous SOA/Web services engagements, and educated practitioners around the world on this technology. He is an author of the Springer text book "Perspectives on Web Services", ISBN 3-540-00914-0. Olaf also contributed to several IBM ITSO Redbooks such as "Web Services Wizardry with WebSphere Studio Application Developer", SG24-6292-00. Olaf holds a graduate degree "Diplom-Informatiker" (awarded with distinction) in Computer Science from the Technical University in Braunschweig, Germany.



Agenda

- Module 1: Introduction to SOA
 - Defining principles
 - SOA patterns: enterprise service bus, service composition
- Module 2: Introduction to Web Services
 - SOAP and WSDL
 - JAX-RPC and other Java standards
- Module 3: Developing Web Service Provider and Requestor Applications
 - Java and PHP
- Module 4: Putting the 'A' back in SOA
 - Module 4a: SOA decision modeling concepts and tool support
 - Module 4b: SOAP, WSDL, and SOA best practices

Building Service-Oriented Architectures with Web Services

Module 1: Introduction to SOA

There have been other
distributed computing models,
but **this time it's serious.**

This is just another reinvention
of the wheel, **the most pointless
hype in years.**

Objectives and agenda for module 1

Learning Objectives:

- Understand the motivation and market forces behind SOA
- Learn which architectural principles define SOA as architectural style
- Get an overview of SOA patterns and how they are applied in the industry

Agenda:

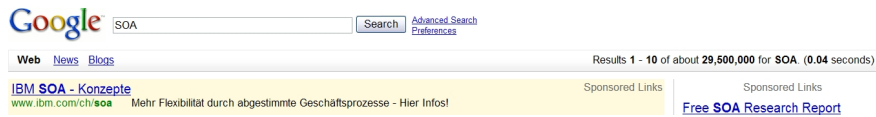
1. Motivation
2. Defining principles
3. SOA patterns and case studies
4. SOA and Web services market
5. Summary and discussion

Module 1: Agenda

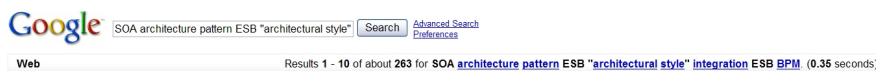
- **Motivation for SOA**
- Defining principles
- Key SOA patterns and project examples
- SOA and Web services market
- Summary and discussion

SOA – hype vs. reality

- Google search for SOA: 30 million hits; Amazon.com: 13,729 books (07/2007)



- Refined to technical essentials, as taught in this tutorial: 263 hits; 0 books



General challenges in enterprise application development

- **Many users and backend systems** interact with the system
 - Numerous functional and technical entry channels
 - Custom applications and software packages to be integrated
- Sophisticated **Quality of Service (QoS)** requirements and other **Non-Functional Requirements (NFRs)**, for example:
 - Response times to be guaranteed, even under heavy load
 - Transactional integrity in long-running workflow scenarios
- **Hardly any green field**
 - Multiple technology stacks, stovepipe architectures, interface spaghetti
 - Valuable data and business logic in systems that have been developed under tight budget and scheduling constraints
- **The only constant is change**
 - Requirements, technology, environment dynamics

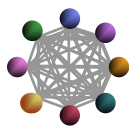
Typical status quo in many enterprise IT architectures

- Functional and technical **application monoliths** ubiquitous
 - Stovepipe architectures, application scope creep, redundant implementations, data management and many other agility issues
 - Architectural governance or guidance missing
- Development and integration projects **costly and long running**
 - Proprietary point-to-point connections, often developed from scratch
 - File transfer is a frequently used integration pattern with numerous architectural drawbacks
 - Roll-your-own philosophy works short term, but leads to maintenance headaches
- As a result, horizontal initiatives are **much harder to implement** than they have to be
 - Example: single customer relationship management solution on top of several line-of-business applications (packages and custom developed)

Solution: Evolve and integrate former monoliths into a Service-Oriented Architecture (SOA) ecosystem

Component-Based Development

Messaging Backbone



- Point-to-Point connection between applications
- Simple, basic connectivity

Enterprise Application Integration (EAI)



- EAI connects applications via a centralized hub
- Easier to manage larger number of connections

Service-Oriented Architecture

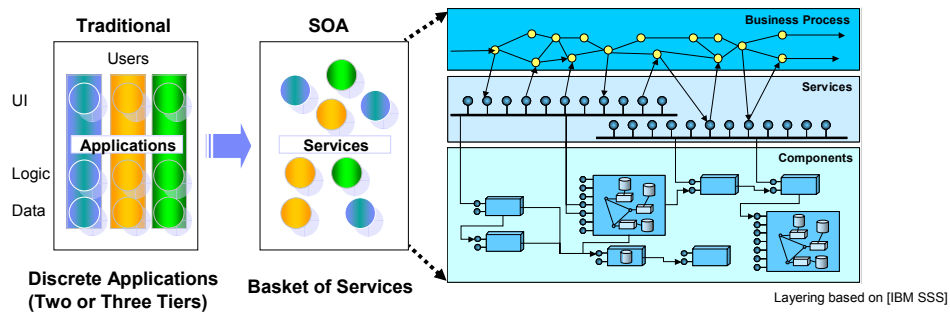


- Integration and choreography of services through an Enterprise Service Bus
- Flexible connections with well defined, standards-based interfaces

Flexibility

Source: [IBM SOA]

SOA fundamentals: Modularity, layering, and loose coupling



Example:

An insurance company uses three SAP R/3, MS Visual Basic, and COBOL applications to manage customer information, check for fraud, and calculate payments. The user interfaces (UIs) are the only access points.

A multi-step, multi-user business process for claim handling, executing in IBM WebSphere, is supposed to reuse the functions in the existing applications. How to integrate the new business process with the three legacy applications in a flexible, secure, and reliable way?

Module 1: Agenda

- Motivation for service-oriented computing
- **Defining principles**
- Key SOA patterns and project examples
- SOA and Web services market
- Summary and discussion

What is a Service-Oriented Architecture (SOA)?

No single definition – “SOA is different things to different people”

- ▶ A *set of services* that a business wants to expose to their customers and partners, or other portions of the organization.
- ▶ An architectural style which requires a *service provider*, a *service requestor* (consumer) and a *service contract* (a.k.a. client/server).
- ▶ A set of architectural patterns such as *enterprise service bus*, *service composition*, and *service registry*, promoting principles such as *modularity*, *layering*, and *loose coupling* to achieve design goals such as separation of concerns, reuse, and flexibility.
- ▶ A *programming and deployment model* realized by standards, tools and technologies such as Web services and Service Component Architecture (SCA).

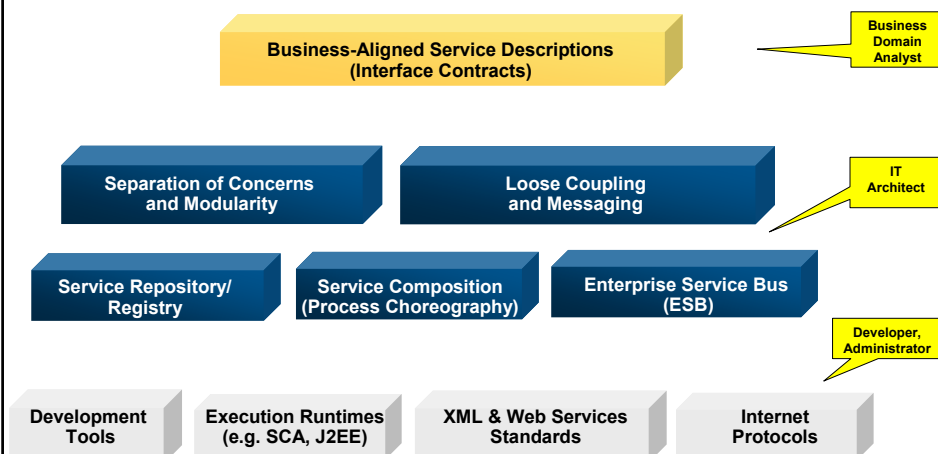
Business
Domain
Analyst

IT
Architect

Developer,
Administrator

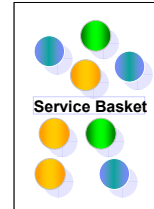
Adapted from: [IBM SSS]

SOA building blocks on the three levels of abstraction



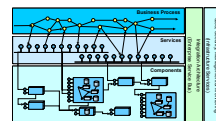
SOA principle 1: Modularity (a.k.a. separation of concerns)

- Motivation
 - Integrating monolithic applications ("stovepipes") is hard (e.g., traditional Enterprise Resource Planning packages)
- Solution
 - Refactor to services, expose service interface only, hide implementation details (a.k.a. encapsulation)
- Forces and consequences
 - Service contracts have to be defined and interpreted (by tools and/or runtimes)
 - Services have to be located and invoked in a coordinated manner
 - Service invocations have to be free of undesired side effects (data management?)
- Roots and known uses
 - [Parnas] and [Dijkstra] introduce modularization and separation of concerns
 - [Meyer] adds formal contracts with pre- and postconditions and invariants
 - Component models such as [CORBA], [J2EE], [Jini], and [OSGI] promote the concept



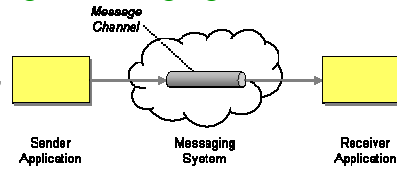
SOA principle 2: Layering (logical and/or physical)

- Motivation
 - Service characteristics such as interface granularity and lifecycle vary: e.g., technical logging service vs. claim checking business process
- Solution
 - Organize the SOA into 3++ architectural layers
- Forces and consequences
 - More indirections, requiring communications infrastructure
 - First law of distribution: "the best remote call is the one you don't make"
- Roots and known uses
 - Seven networking layers defined by [OSI]
 - Layers pattern originally described by Buschmann et al. in [POSA]
 - Patterns of Enterprise Application Architecture [Fowler]
 - e-business, on demand and WBI reference architectures from [IBM SOA]



SOA principle 3: Loose coupling through messaging

- Motivation
 - Once applications have been modularized, dependencies between services occur
- Solution
 - Couple services loosely (several dimensions)
 - E.g., messaging system decouples in time, location, and language dimensions
- Forces and consequences
 - Messaging means single implementation/endpoint by default (no remote objects)
 - Receiver is stateless per se, so conversational sessions require correlation logic
 - Asynchronous communication complicates systems management
- Roots and known uses
 - Enterprise application integration vendors have been promoting the concept for a long time; origin of term hard to track down [looselycoupled.com]
 - Hohpe and Woolf define a pattern language for message-based integration [Hohpe]

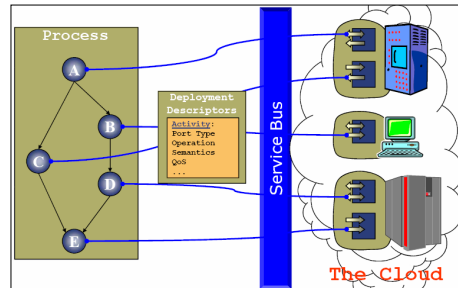


More SOA principles

- Business-IT alignment and SOA governance
- Model-driven, process-centric development methodology
 - Model-assemble-deploy-monitor development lifecycle, industry models
 - Formalization of process and service semantics, code generation
- Data- and meta data-centric SOA
 - Information as a service (master data management), semantic brokering
- Integrated service management
 - Contract-aware service monitoring
 - Feedback loop to business modeling
- Standardization and openness of service models and runtimes

SOA principle n: Service virtualization and flexible infrastructure

- Motivation
 - "I don't care about a particular provider, just chose the one that at this point in time is best for me"
- Solution
 - From WWW to service bus/cloud
 - Two-level programming
- Forces and consequences
 - Many open research issues
 - E.g., trust and privacy, precise semantics, QoS, multi tenancy, provisioning
- Roots and first isolated steps
 - Software as a service, e.g. Salesforce.com CRM and Amazon storage service
 - Dynamic matchmaking, grid and utility computing, on demand



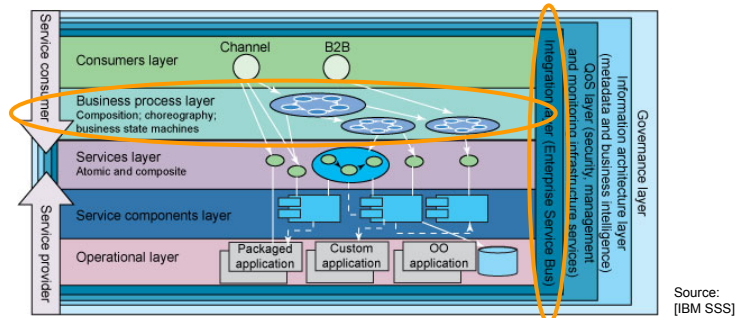
Source: [Leymann]

Module 1: Agenda

- Motivation for SOA
- Defining principles
- **Key SOA patterns and project examples**
- SOA and Web services market
- Summary

Layered SOA reference architecture revisited

Several strict and semi-strict logical layering schemes exist; defining service, integration and process layers is common. [IBM SOMA] adds a component and a QoS layer.



Source:
[IBM SSS]

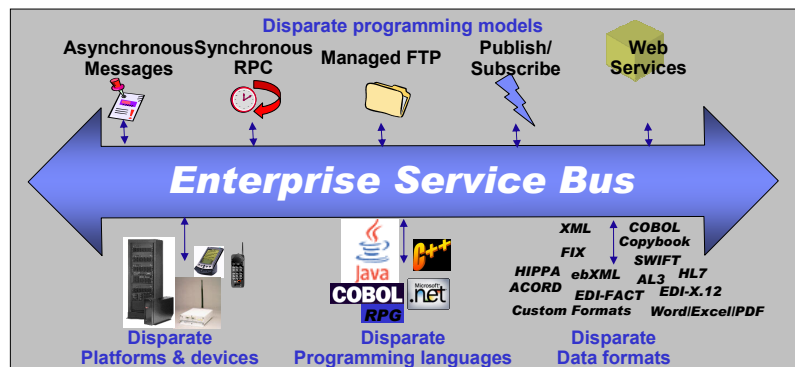
Some reference architectures define additional layers, for example the IBM SOA Solution Stack project defines 5+4 layers (semi-strict layering) [IBM SSS]. The service composition and the ESB patterns are two of these layers.

SOA pattern 1: Enterprise Service Bus (ESB)

A communications “architecture” that enables software applications that run

- on different platforms and devices
- written in different programming languages
- use different programming models
- require different data representations

to communicate with no disruption to existing applications or interfaces.



Source:
[IBM SOA]

SOA pattern 1: Enterprise Service Bus (ESB)



- Refinement of well-established **broker** pattern described in [POSA]
 - *Hub-and-spoke* architecture known from many Enterprise Application Integration (EAI) products, providing many-to-many connectivity between loosely coupled parties – the ‚B‘ in ESB
 - Plus explicit, *machine-readable service interface contracts* – the ‚S‘ in ESB
 - Plus *business alignment* and high-end *Quality of Service (QoS)* – the ‚E‘ in ESB
- Key **capabilities** defined in [Keen] and “Enterprise Integration Patterns” [Hohpe]:
 - Multiple *transport layers and message exchange patterns*:
 - Synchronous service invocations, e.g., via simple Web protocols (HTTP)
 - Asynchronous messaging (JMS, MQ) – key for loose coupling!
 - Mediations providing *content-based routing, message format conversions, instrumentation for QoS management* (logging, billing, security controls)
 - Declarative, *policy-based* configuration and management



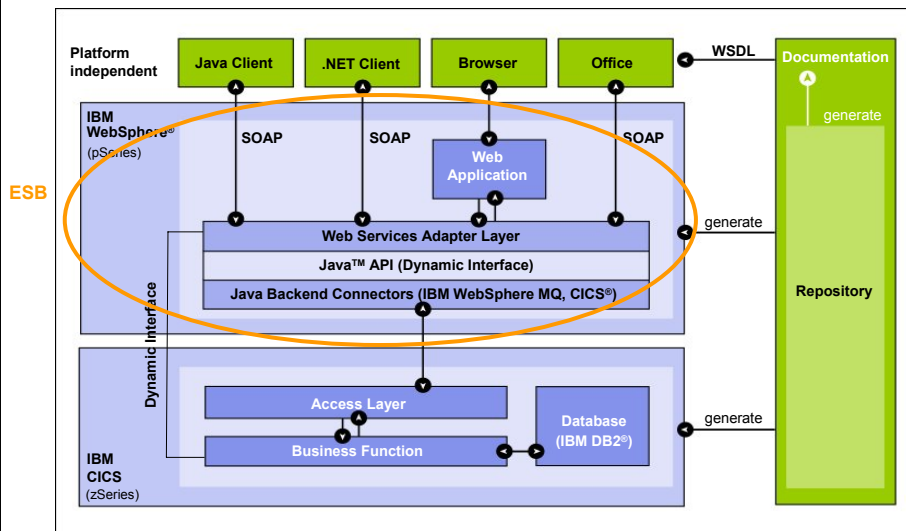
Source: [Hohpe]

27

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

ESB and Web services case study [OOPSLA 2004]

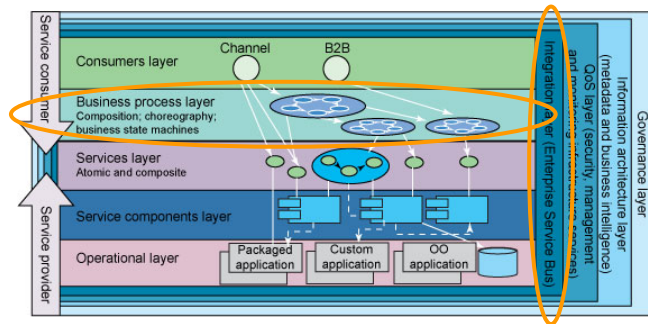


28

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Layered SOA reference architecture revisited



SOA pattern 2: Service composition

- The **business logic layer** of n-tiered enterprise applications can be divided into two sub-layers (architectural “refactoring”):
 - ▶ Processing steps assigned to roles placed in *process layer* (pattern a.k.a. workflow, business process choreography)
 - ▶ Basic computations, validation logic, manipulation of persistent business entities placed in atomic *services layer*
- **Foundations** for process layer execution semantics (workflows):
 - ▶ *Business Process Management (BPM)*, Petri nets, Pi-calculus, [Leymann]
 - ▶ One technology option is the *Web Services Business Process Execution Language* (WS-BPEL or BPEL4WS), standardized by [OASIS]
- **Key issues:**
 - ▶ Where to draw the line between the two sublayers?
 - ▶ How to interface with the presentation layer?
 - ▶ Integration of legacy workflows, e.g., those residing in software packages?



Source: [Hohpe]

Multi-channel order management supporting a wholesaler-retailer business model in the telecommunications industry [OOPSLA 2005]

Functional domain:

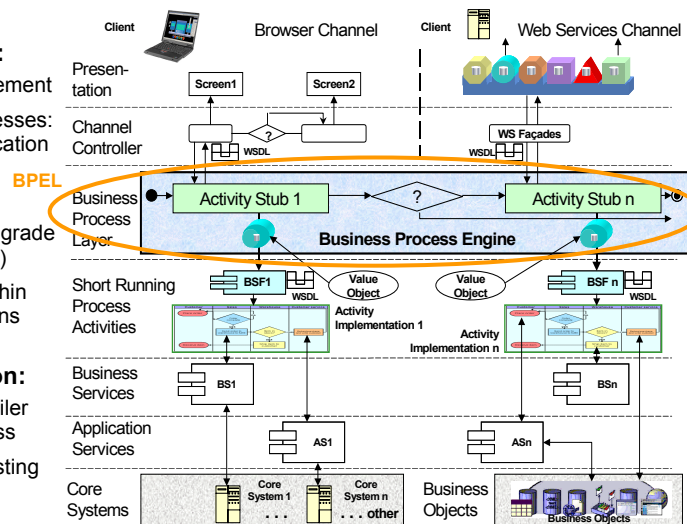
- Order entry management
- Two business processes: new customer, relocation

Main SOA drivers:

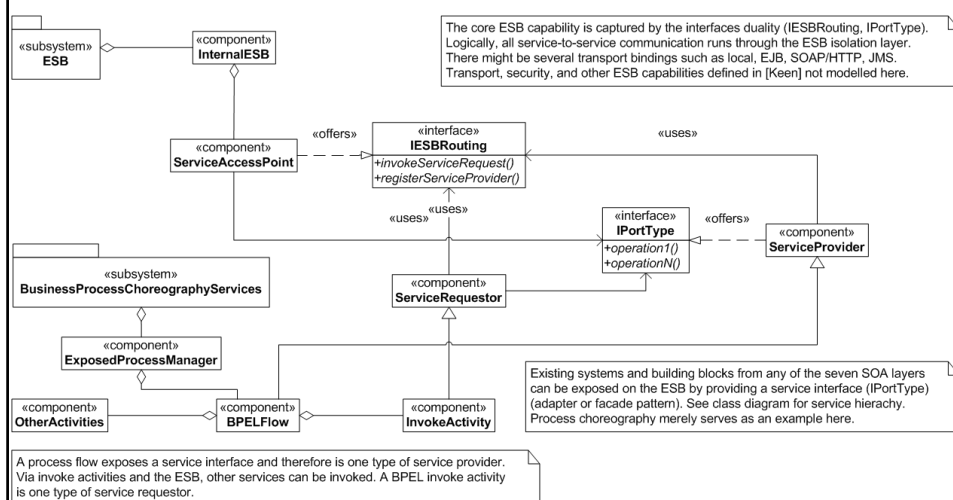
- Deeper automation grade (e.g., compensation)
- Services shared within and between domains

Service composition:

- Top-down from retailer interface and process
- Bottom-up from existing wholesaler systems



UML representation of ESB and BPEL service composition



SOA pattern 3: Service registry

- SOA incarnation of **naming and directory services** known from CORBA, J2EE, DCE, and other distributed computing technologies
- **Key capabilities:**
 - ▶ Build time service publishing and lookup
 - Human user (developer)
 - Tools
 - ▶ Runtime registration and lookup of service providers
 - ▶ Semantic annotations
 - ▶ Matchmaking
- **Known uses:**
 - ▶ Web services specification: UDDI
 - ▶ IBM WebSphere Service Repository and Registry (WSRR)
 - ▶ Most of today's SOA references use custom built repositories tightly integrated into inhouse development processes

General-purpose patterns still applicable in SOA

- All messaging patterns from Enterprise Integration Patterns website
 - ▶ Message queue, channel, broker, process manager [Hohpe]
- Domain model, service layer, universal language [Fowler], [Evans]
- Distributed computing and remoting patterns [POSA], [Zdun]
- Client-side remote proxy and server-side invocation stub [POSA]
- [GoF] patterns such as factory, singleton, observer
- Security patterns [Schumacher]
- Information integration patterns [Sauter]

Module 1: Agenda

- Motivation for SOA
- Defining principles
- Key SOA patterns and project examples
- **SOA and Web services market**
- Summary

Motivation for SOA and Web services initiatives

- **Business drivers**
 - ▶ Customer-focused initiatives involving multiple sales channels
 - ▶ Merger, acquisition, and divestiture readiness
 - ▶ Agility and flexibility issues
- **Technical reasons**
 - ▶ Web services seen as a simple and universal connectivity technology for heterogeneous worlds (B2B and EAI)
 - ▶ Replacement of proprietary with standards-based technology, replacement of custom middleware and tools with vendor components
 - ▶ Sharing of application building blocks across lines of business and projects (reuse)
 - ▶ Modernization of enterprise architecture due to maintenance, productivity, and cost issues

Widespread interest in SOA concepts and technologies

- Consumers of Information Technology (IT)
 - ▶ Lines of business in various industries: business-IT alignment, flexibility
 - ▶ Enterprise architecture groups, other corporate IT functions: governance
 - ▶ Solution development teams: modularity, layering, loose coupling, patterns
 - ▶ System administrators: declarative deployment, service management
- Suppliers of IT, e.g., application software and infrastructure vendors
 - ▶ Open existing applications for new usage scenarios (business model?)
 - ▶ Middleware, middleware, middleware
 - ▶ Strong need for tools
- Professional services (consulting) firms
 - ▶ Process and service modeling methods and practices
 - ▶ SOA strategy consulting, development, integration, and hosting services

Some industry incarnations of SOA (in alphabetical order)

Vendor	Selected SOA/WS Offering (s)
IBM	J2EE application server; SOA is an inherent part of strategy, products and services. The developer portal features an SOA and Web services zone: http://www.ibm.com/developerWorks/webservices
IONA	CORBA products and open source and commercial ESBs: Orbix, Artix, Celtix, see http://www.ionac.com/products
Microsoft	e.g., .NET SDK, Web Services Extensions (WSE), Information Bridge Framework (IBF)
Oracle, BEA	J2EE application server; SOA infrastructure management offering: http://dev2dev.bea.com/soa
SAP	Enterprise Service Architecture (ESA), NetWeaver: http://www.sdn.sap.com/irj/sdn/developerareas/esa
Various system integrators and consulting firms	SOA practices evolving from EAI/J2EE/BPM base, e.g., SOA white paper from ThoughtWorks [Hohpe], available from http://www.eaipatterns.com

A few of *many* Web services implementation assets

Language	Web Services Support
Java	Web services support is mandatory since J2EE 1.4 BEA WebLogic IBM WebSphere Other J2EE 1.4-compliant application servers Other commercial offerings Systinet WASP, registry (acquired by HP) Open source assets Apache Axis2, Axis 1.2/1.3, Codehaus Xfire
Microsoft languages (C#, VisualBasic)	Native support in .NET 1.1, Web Services Extensions (WSE), Information Bridge Framework (IBF) MS Office support via SOAP Toolkit (now deprecated)
Perl	SOAP::Lite and others
PHP	NuSOAP (SourceForge project) PHP 5 SOAP Extension (native C implementation)
Ruby	Basic support in Ruby on Rails via Action Web Service

Module 1: Agenda

- Motivation for SOA
- Defining principles
- Key SOA patterns and project examples
- SOA and Web services market
- **Summary and discussion**

How do Web services-based SOAs help addressing common enterprise application development and integration challenges?

Challenge	Response
Flexibility and agility	Logical layering Separation of concerns Matchmaking via bindings rather than name and type
Bridging business-IT gap	Business-aligned service models Business performance management and composition enabled through WSDL contracts and BPEL
Evolve proprietary monoliths into reusable components	Standardized interface contracts (WSDL, semantics, policy) Integration via ESB Split business logic layer into process and atomic service layer
Many legacy applications (no green field)	Document messaging rather than tightly coupled RPC Interoperable protocols such as SOAP Patterns such as adapter and facade
Time-to-market	Increase productivity through standardization and off-the-shelf tool support

Object-Orientation (OO) vs. Service-Oriented (SO)

- OO and SO share many characteristics, but differences exist as well
 - ▶ Encapsulation, information hiding through interfaces (OO and SO)
 - ▶ Remote objects and call stack (OO) vs. document-centric messaging (SO)
 - ▶ Name and type as linking element (OO) vs. bindings and contracts (SO)
- Rule of thumb: *develop object-oriented and integrate service-oriented* (SO on macro level vs. OO on micro level):
 - ▶ OO is a general-purpose programming paradigm
 - ▶ SOA is an architectural style for enterprise application integration
 - ▶ Web services consumers and providers (the enterprise systems to be integrated) can be OO applications or other
- *Service autonomy* should be strived for, see [Ferguson]

Module 1: Concluding thoughts

- SOA is a state-of-the-art architectural style for crafting and integrating enterprise applications of quality and longevity – considered to be a “paradigm shift” for the software industry
 - Benefits include agility, flexibility, reuse, productivity gains, openness
- On the other hand, SOA comprises many well-established software engineering principles and patterns (“same old architecture”)
 - Some of which are 30+ years old (nothing wrong with that!)
 - SOA adopts and combines them for one particular problem domain – enterprise application development and integration
 - Strong emphasis on modularity, layering, loose coupling via messaging
- ESB, process choreography, and service repository/registry are key SOA patterns; non-SOA patterns continue to be relevant
 - Refining general-purpose patterns such as broker, workflow, directory
 - Applicable across industries, both internally and externally

References (1/2)

- [CBDI] Sprott, D.: *On SOA Methodology*, Editorial March 2005 CBDI Journal, <http://www.cbdiforum.com/>
- [Dijkstra] Dijkstra, E. W., *A Discipline of Programming*, Prentice Hall, 1976
- [Evans] Evans E., *Domain-Driven Design*, Addison Wesley, 2003
- [Ferguson] Ferguson D., Storey T., Lovering B., Shewchuk J., *Secure, Reliable, Transacted Web Services*, <http://www.ibm.com/developerworks/webservices/library/ws-secutrans/index.html>
- [Fowler] Fowler M., *Patterns of Enterprise Application Architecture*, Addison Wesley, 2003
- [GoF] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995
- [Hohpe] Hohpe G., *Developing Software in A Service-Oriented World*, White Paper, January 2005, *Enterprise Integration Patterns* website, <http://www.eaipatterns.com>
- [IBM SOA] Service-Oriented Architecture from IBM – Success Stories, Products, Services <http://www.ibm.com/software/solutions/soa>
- [IBM SOMA] Arsanjani A., *Service-Oriented Modeling and Architecture*, <http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/>
- [IBM SSS] Ibrahim M., Long G., *Service-Oriented Architecture and Enterprise Architecture*, <http://www.ibm.com/developerworks/webservices/library/ws-soa-enterprise1/>
- [IBM ITSO] Wahli U., *Application Developer Version 6 Web Services*, IBM ITSO Workshop 2005, <http://www.redbooks.ibm.com>

References (2/2)

- [Keen] Keen M. et al, *Patterns: Implementing an SOA using an ESB*, IBM Redbook 2004
- [Meyer] Meyer B., *Object-Oriented Software Construction*, Second Edition, Prentice Hall 1997
- [OASIS] Web Services Business Process Execution Language Version 2.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
- [OSI] International Standardization Organisation, Open System Interconnection Basic Reference Model, http://en.wikipedia.org/wiki/OSI_model
- [POSA] Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., *Pattern-Oriented Software Architecture – a System of Patterns*. Wiley, 1996
- [PoWS] Zimmermann O., Tomlinson M., Peuser S., *Perspectives on Web Services – Applying SOAP, WSDL and UDDI to Real-World Projects*, Springer-Verlag, 2003
- [RAMP], Reliable, Asynchronous Messaging Profile 1.0, IBM, Ford Motor Company, DaimlerChrysler, <http://www.ibm.com/developerworks/webservices/library/specification/ws-ramp>
- [SAP] ESA zone of SAP Developer Network (SDN), via <http://www.sdn.sap.com/sdn/esa.sdn>
- [Sauter] Sauter G. et al, *Information Service Patterns, Part 1: Data Federation Pattern* <http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv1/>
- [Schumacher] Schumacher M., Fernandez E.B., Hybertson D., Buschmann F., and Sommerlad P., *Security Patterns: Integrating security and systems engineering*, Wiley 2006.
- [Zdun] Zdun U., Dustdar S., *Model-Driven and Pattern-Based Integration of Process-Driven SOA Models*, <http://drops.dagstuhl.de/opus/volltexte/2006/820>

Building Service-Oriented Architectures with Web Services

Module 2: Web Services Concepts

Web services reuse **well-established and proven** concepts.

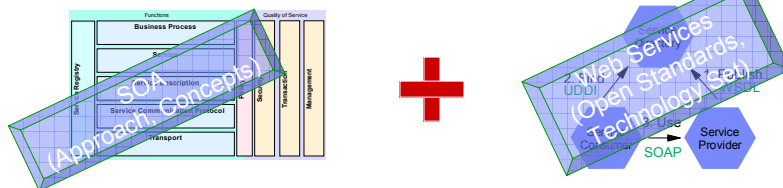
I've already skimmed through some WSDL, and **I didn't understand a single line.**

This module is excerpted from the book "Perspectives on Web services" by Olaf Zimmermann, Mark Tomlinson, and Stefan Peuser, Springer-Verlag Berlin Heidelberg New York 2003, ISBN 3-540-00914-0. This work is subject to copyright. © Springer Verlag 2003. All rights reserved.

Module 2: Agenda

- Building blocks for delivering SOA with Web services
 - XML
 - SOAP
 - WSDL
 - UDDI
 - Security
 - Interoperability
 - Java and J2EE

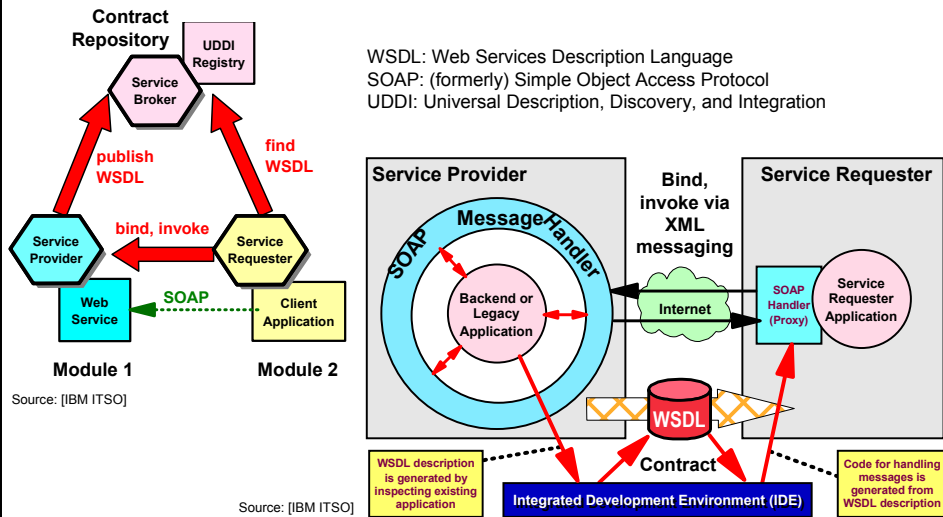
While SOA is a conceptual architectural style, Web services provide enabling technology standards for SOA.



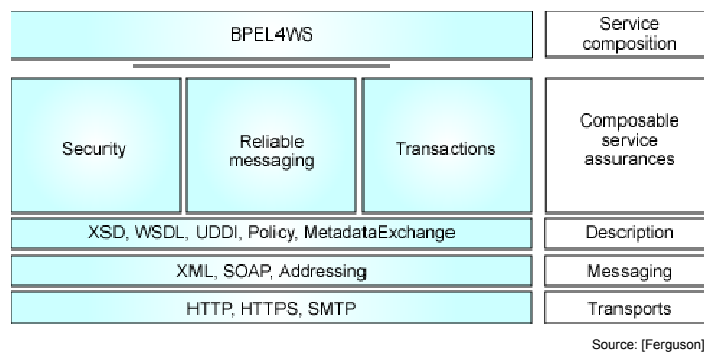
- SOA is an architectural style whose goal is to achieve *loose coupling* among interacting software agents *through modularization, layering, and messaging*
 - SOA enables advancement in the programming model, e.g. Service Component Architecture (SCA), an XML-based deployment model for service providers
- SOA and Web services are not identical (concepts vs. technology):
 - Many existing production SOAs do not primarily use Web services – they are built on Message-Oriented Middleware (MOM)
 - Not all deployed Web services-based systems necessarily embrace all the guiding principles of SOA (e.g., loose coupling through messaging)

Source: [IBM SOA]

Web services technologies use eXtensible Markup Language (XML) documents to describe, invoke, and publish services.



Completing the Web services layers (WS-* stack) :
Secure, reliable, transacted Web services

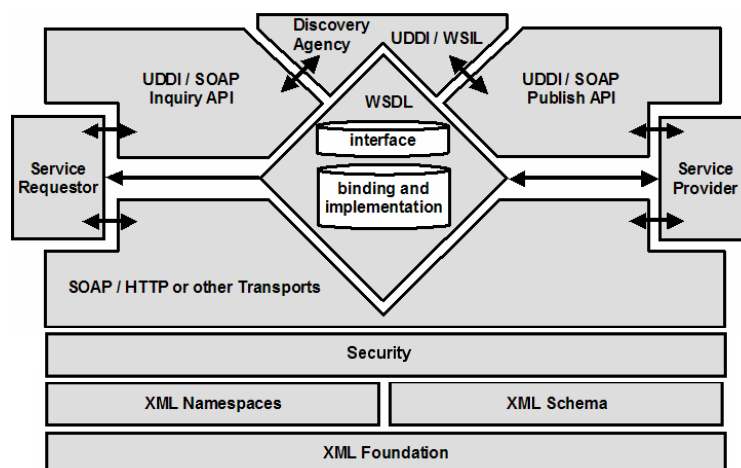


- WS-Security suite
- WS-ReliableMessaging
- WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity (under standardization at OASIS and WS-I)

SOAP/WSDL Web services (WS-* stack) vs. REST

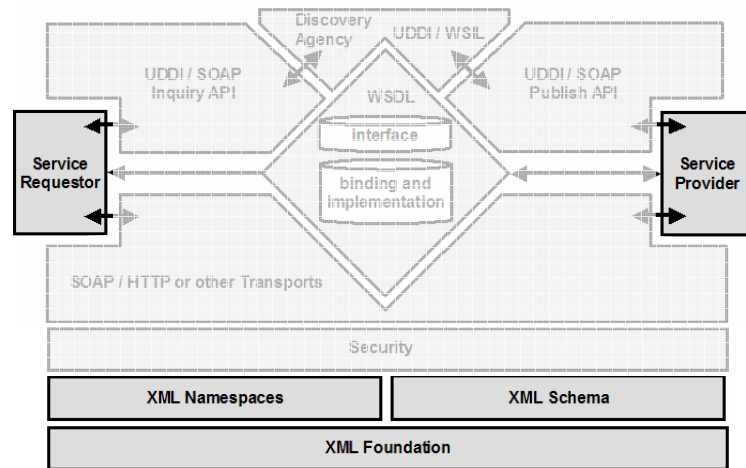
- **WS-*:** W3C, OASIS, WS-I standardization effort
 - SOAP, WSDL, UDDI ... as introduced in Module 2
 - Many additional specifications for all areas of concern in enterprise application development (principle of *composability*)
- **REpresentational State Transfer (REST)**
 - Lightweight XML over HTTP runtime based on protocol primitives like HTTP-GET [Fielding]
- Vendor support typically centers around WS-*; many public Web services are available via both channels, SOAP/WSDL and/or REST
 - See Amazon, eBay, Google, Yahoo! Web services for examples
- This module focuses on the core of the first of the two approaches, which is ready for production usage (XML, SOAP, WSDL)

Building blocks for delivering a service-oriented architecture implemented with Web services

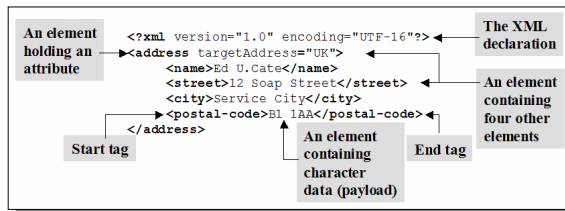


Interpretation of the core specifications and links through the WS-I profiles [WSI]

Building blocks: XML



XML, XML Namespaces and XML Schema introduction



XML instance document example

XML [XML]

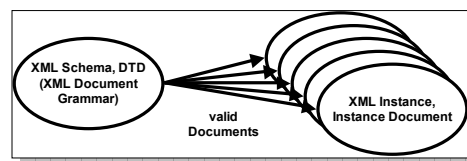
- ▶ Markup language composed of tags and data
- ▶ Elements and attributes
- ▶ Read by an XML processor
- ▶ Requires grammar definition
- ▶ Valid and well-formed

XML Namespaces [XMLNS]

- ▶ Global naming mechanism for XML
- ▶ Qualified names: prefix and local parts
- ▶ Multiple namespaces in same document

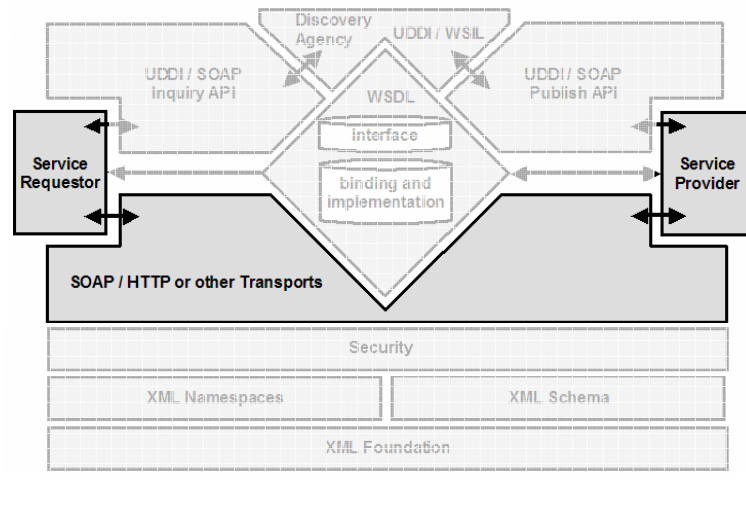
XML Schema [XMLSch]

- ▶ Provides grammar for XML instance docs
- ▶ Built-in types
- ▶ Simple and complex custom data types



XML document grammar and valid XML instances

Building blocks: SOAP



55

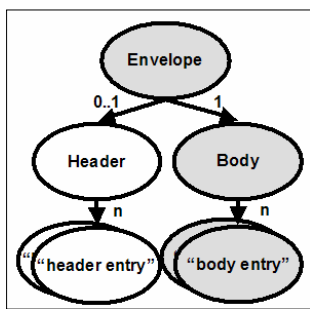
Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

SOAP basics: XML documents as service invocation messages

- SOAP message elements: Envelope, Headers, Message Body and Faults
- Two communication styles: Document style, RPC style
- Literal or SOAP encoding of message body plus attachments support

Reference: [SOAP]



SOAP message containment structure

```

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    <pQP:priority
      xmlns:pQP="http://premierquotes.com/ns/priority"
      SOAP-ENV:mustUnderstand="1">
      high
    </pQP:priority>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <pQP:getPhoneNumber
      xmlns:pQP="http://premierquotes.com/ns/employees"
      SOAP-ENV:encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/">
      <first-name xsi:type="xsd:string">Ed U.</first-name>
      <last-name xsi:type="xsd:string">Cate</last-name>
    </pQP:getPhoneNumber>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
  
```

Annotations in the example:

- Header entry**: Points to the `<pQP:priority>` element.
- Header must be processed**: Points to the `SOAP-ENV:mustUnderstand="1"` attribute.
- Method invocation**: Points to the `<pQP:getPhoneNumber>` element.
- Use SOAP encoding**: Points to the `SOAP-ENV:encodingStyle` attribute.
- Method parameters**: Points to the `<first-name>` and `<last-name>` elements.
- Method parameter types**: Points to the `xsi:type="xsd:string"` attributes.
- Method parameter values**: Points to the text content of the `<first-name>` and `<last-name>` elements.

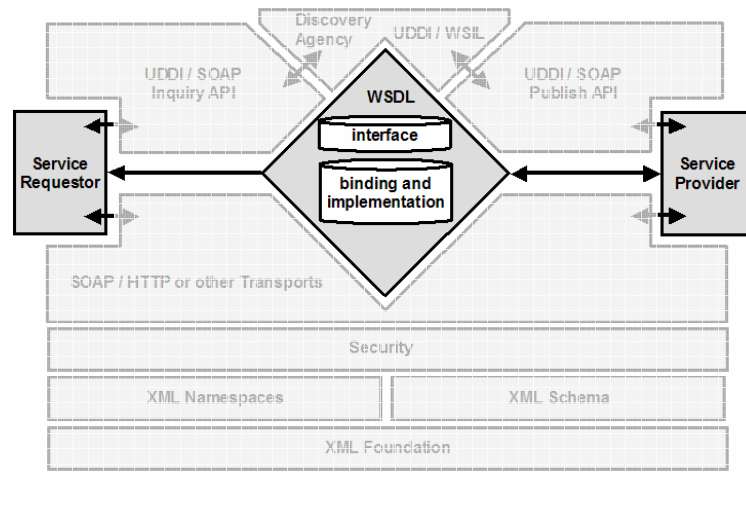
SOAP message example

56

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Building Blocks: Web Services Description Language (WSDL)

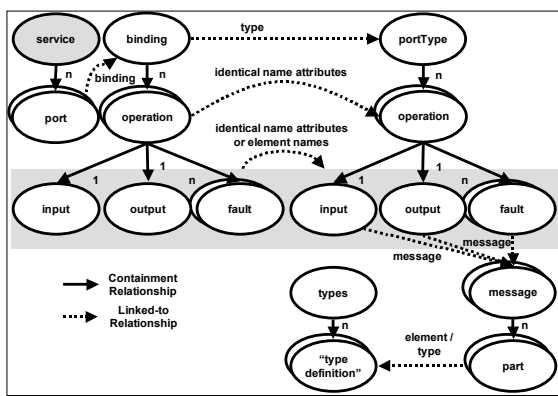


57

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

WSDL basics: XML language for service descriptions



Logical relationships between WSDL elements

- WSDL document elements
 - ▶ Type definitions and imports
 - ▶ Interface description (Port Type, Operations, Messages)
 - ▶ Extensible binding section
 - ▶ Implementation description (Ports)
- WSDL SOAP binding
 - ▶ Defines header and fault support
 - ▶ Extensibility element for addressing
- HTTP binding also defined

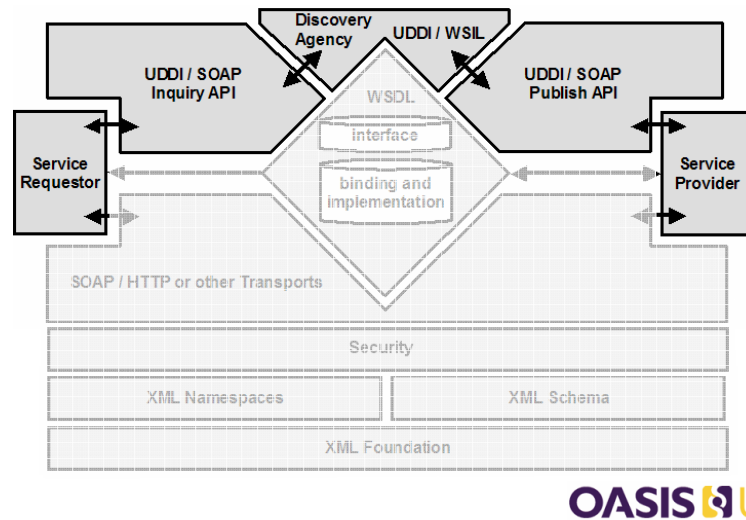
Reference: [WSDL]

58

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Building Blocks: UDDI



59

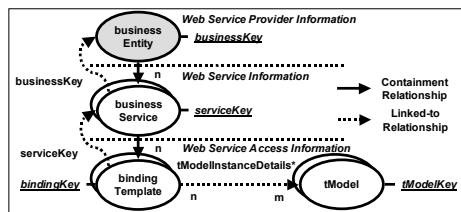
Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

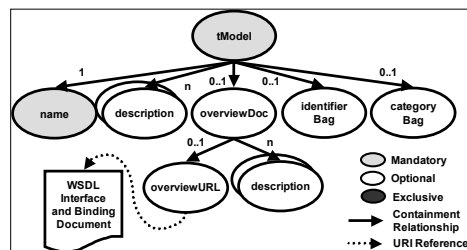
Universal Description, Discovery and Integration introduction

- Provides a Web services directory platform
 - Businesses and organisations providing Web services
 - Descriptions of the services provided
 - Information about technical interfaces
- Sophisticated taxonomy
 - Supports business identification systems (D-U-N-S, GLNs etc.)
 - Also supports business and product classification systems (UNSPSC, NAICS etc.)
- Contains references to WSDL interfaces
- Programmatic interface
 - Posting and requesting service information
- Global operator cloud
 - Test and Production registries
- Private internal registry implementations more frequently used

Reference: [UDDI]



Containment and reference relationship of data structures



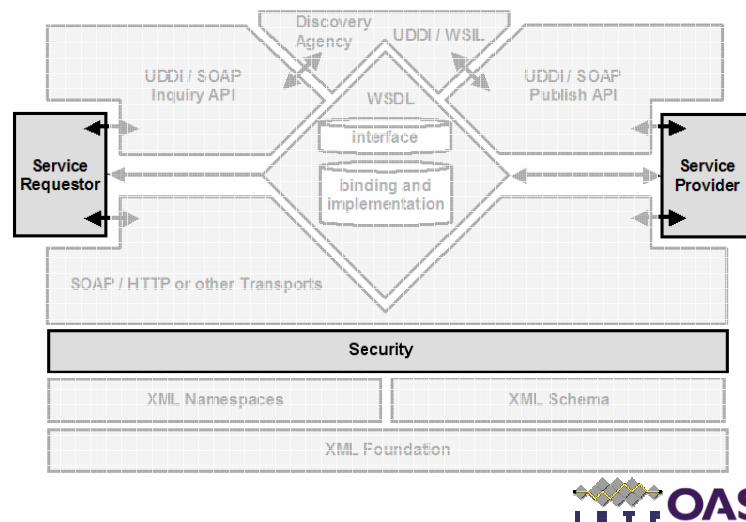
The tModel structure

60

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Building Blocks: Security



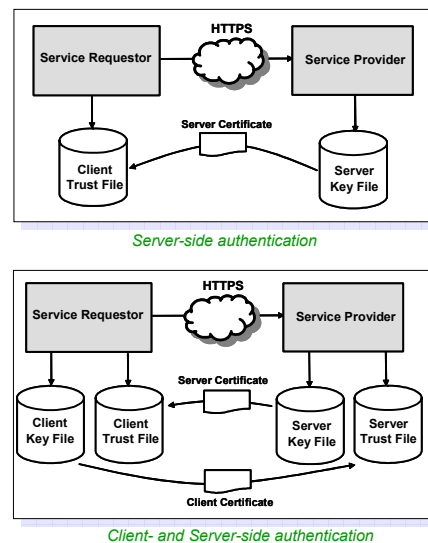
61

Building SOAs with Web Services
Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Securing Web services with HTTPS (HTTP over SSL or TLS)

- Most Web services providers and requestors are able to support SOAP over the HTTPS protocol
- This provides message-level integrity and confidentiality and also provides an authorisation model
- SSL certificates can be requested from a certificate authority or self-certified
 - Key file and trust file used
 - Password-protected certificates shared between parties
- Easy to implement, but has several limitations
 - Message cannot be processed by intermediaries

References: [SSL, TLS]

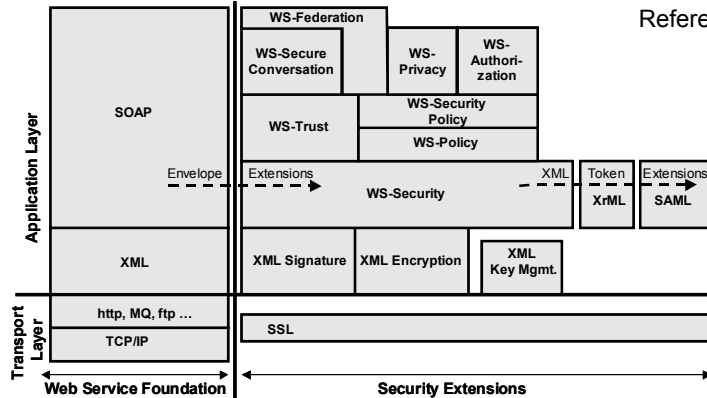


62

Building SOAs with Web Services
Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

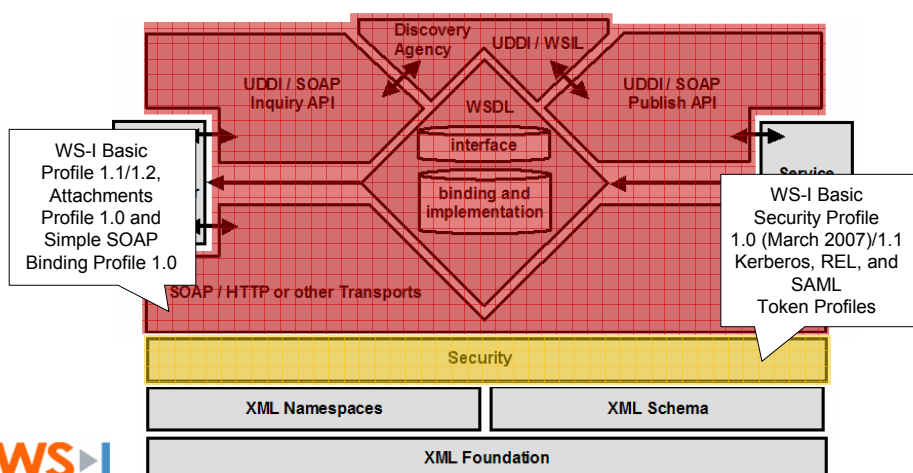
WS-Security specifications

Reference: [WSS]



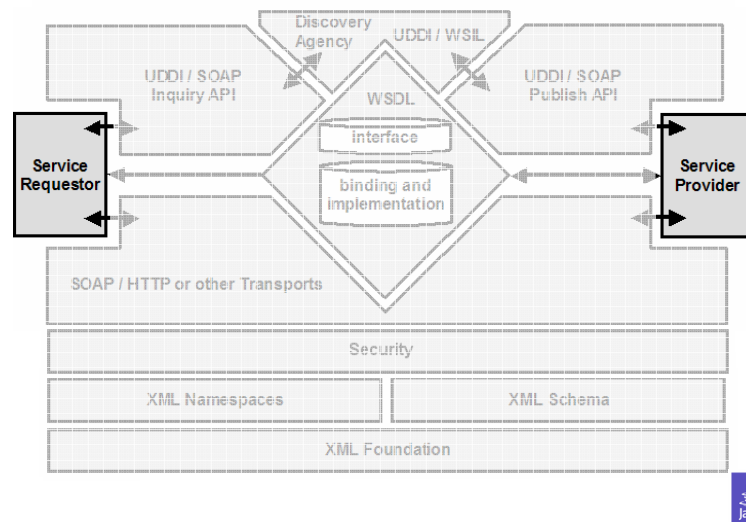
- WS-Security is a building block for security token propagation, message integrity and message confidentiality which can be combined with other Web services extensions
- Implementations available today, vendors and open source

Building Blocks: Interoperability and WS-I.org



WS-I deliverables: profiles, sample applications, testing tools [WSI]

Building Blocks: Java and Web services

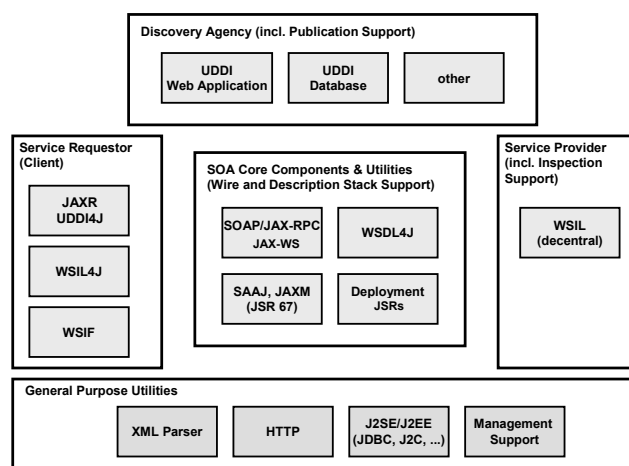


65

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Base architectural building blocks for a J2EE solution

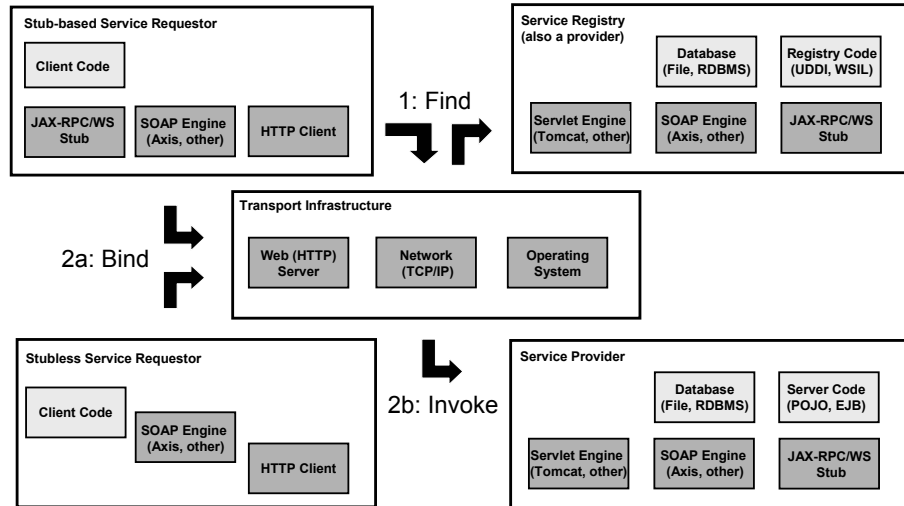


66

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Java Web services architecture – logical and process view

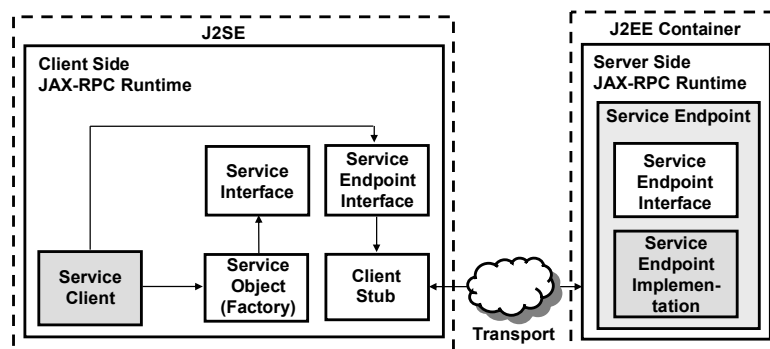


67

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

JAX-RPC 1.1 API (Java 1.4 and earlier)



- Java XML API for Remote Procedure Calls (JAX-RPC) uses design patterns such as proxy and factory to provide consumer (client) and provider (server) side access to SOAP messaging
- Defines WSDL and XML Schema to Java mapping

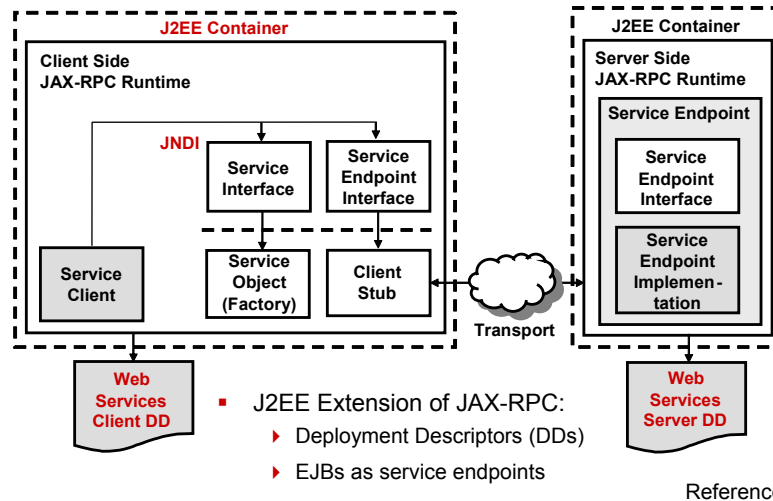
Reference: [JAXRPC]

68

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

JAX-RPC 1.1 and Enterprise Web Services 1.1 (Java 1.4)



69

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

Java XML API for Web Services (JAX-WS) 2.0 (Java 5)

- Follow-up to JAX-RPC 1.1:
 - Support for latest SOAP, WSDL, and WS-I specifications
 - SOAP 1.2, WSDL 2.0 (continued support for 1.1 versions)
 - WS-I Basic Profile 1.1
 - Data binding related tasks now delegated to JAXB 2.0
- Annotations and Web services metadata – JSR 175, JSR 181
 - JAX-WS 2.1 defines the use of Java annotations (JSR 175) to simplify the most common development scenarios for both clients and servers, and aligns with and complements the annotations defined by JSR 181
 - Class level: `@WebService()`, method level: `@WebMethod()`
- Extension for Implementing Enterprise Web Services (JSR 109)
 - The `jaxrpc-mapping-info` deployment descriptor (JSR 109) provides deployment time Java-to-WSDL mapping functionality. In conjunction with JSR 181, JAX-WS 2.1 complements this mapping feature with development time Java annotations controlling the mapping

Reference: [JAXWS]

70

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

ESB and service composition patterns in Java

- Implementing the Enterprise Service Bus (ESB) pattern is straightforward if SOAP is used as messaging format:
 - Java standards for Web services support provide basic ESB support (loose coupling, location transparency, message routing, etc.)
 - Support for advanced ESB features such as mediation in numerous commercial products and open source assets (via SOAP headers)
- Several alternatives for service composition:
 - Write your own composition code
 - Using JAX-RPC or JAX-WS for service invocation
 - Business Process Execution Language (BPEL) support in Java application/integration servers, both open source and commercial
 - Use BPEL for programming-in-the-large and Java for programming in-the-small
 - Use jOpera, a composition framework developed by ETH Zürich (not based on BPEL, defining its own composition language)
 - <http://www.jopera.org>



Module 2: Summary

Web services reuse well-established and proven concepts.

I've already skimmed through some WSDL, and I didn't understand a single line.

- Having a solid XML background is halfway towards understanding Web services
- The base Web services stack is now solid and well established
- The WS-I profiles are an important milestone on the way towards real interoperability between implementations
- Basic security has been robust for several years, but more sophisticated security standards are still emerging

References

- [XML] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, 6 October 2000, <http://www.w3.org/TR/2004/REC-xml-20040204/>
- [XMLNS] Namespaces in XML, W3C, 14 January 1999, <http://www.w3.org/TR/REC-xml-names/>
- [XMLSch] XML Schema W3C Recommendation Parts 0-2, 2 May 2001, <http://www.w3.org/XML/Schema>
- [SOAP] SOAP Version 1.1 W3C Recommendation, (1.2 Rec. since 2007), <http://www.w3.org/2000/xp/Group/>
- [WSDL] WSDL Version 1.1 W3C Note, March 2001 (2.0 Rec. since 2007), <http://www.w3.org/2002/ws/desc/>
- [UDDI] UDDI Version 3.0.2 OASIS Draft, October 2004, http://uddi.org/pubs/uddi_v3.htm
- [SSL] SSL Protocol Version 3.0, Netscape Communications, 1996, <http://wp.netscape.com/eng/ssl3/>
- [TLS] Transport Layer Security 1.0, Internet Engineering Task Force, January 1999, <http://www.ietf.org/html.charters/tls-charter.html>
- [WSS] Web Services Security: SOAP Message Security 1.0 Specification, OASIS, March 2004, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- [WSI] Web Services Interoperability Profiles, WS-I.org, 2004-2008, <http://www.ws-i.org/deliverables/matrix.aspx>
- [JAXRPC] Java API for XML Remote Procedure Calls 1.1 Specification, Java Community Process, October 2003, <http://www.jcp.org/en/jsr/detail?id=101>
- [EWS] Enterprise Web Services 1.1, Java Community Process, November 2003, <http://www.jcp.org/en/jsr/detail?id=921>
- [JAXWS] The Java API for XML Web Services, (JAX-WS) 2.1, May 2007

Building Service-Oriented Architectures with Web Services

Module 3: Web Services Construction

Web services programming **isn't fundamentally different** from what I've been doing with J2EE and XML.

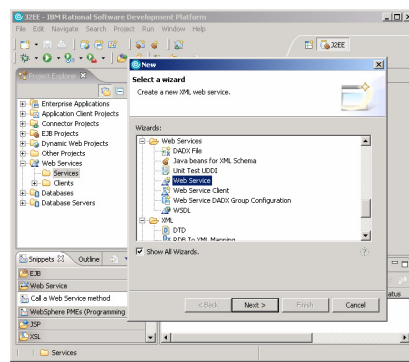
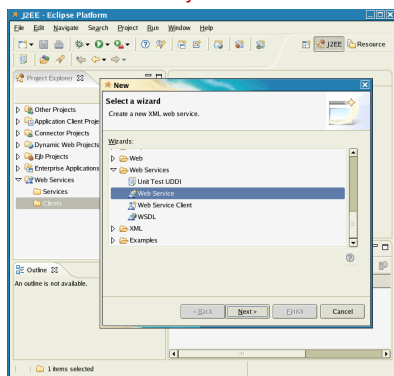
I bet I can get some of these **new wizards and tools** to do most of the hard work.

Module 3: Agenda

- Technology implementations
 - Web services tools
 - J2EE Web services implementations
 - Other programming languages
- End-to-end example using open source technologies
 - Introduction and getting started
 - WSDL definition
 - Service provider creation from WSDL
 - Test service implementation
 - Web service publishing and discovery
 - Create a Web service client from WSDL
 - Secure communication with SOAP/HTTPS

Eclipse-based Web services tools

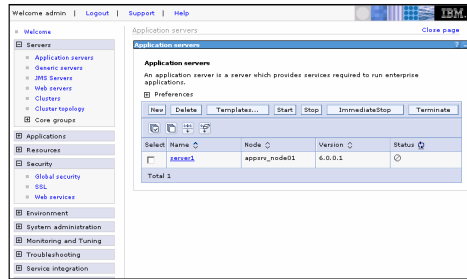
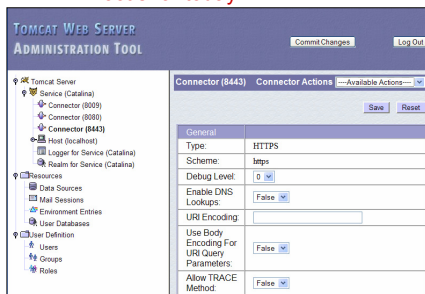
- Eclipse 3.3/3.4 Web Tools Project (WTP)
 - Open source downloads from eclipse.org
 - Web services wizards, WSDL editor, WS-I validator
 - Focus for today



- IBM Rational Application Developer (RAD) 6.x and 7
 - Based on Eclipse and WTP
 - Plus many more Web services tools

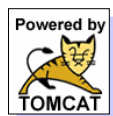
J2EE Web services implementations

- Apache Tomcat + Axis
 - ▶ Open source download
 - ▶ J(2)EE compliant Web container plus SOAP engine with JAX-^{*} support
 - ▶ Basic administration tools
 - ▶ Integration with Eclipse + WTP
 - ▶ **Focus for today**



- IBM WebSphere Application Server
 - ▶ J(2)EE compliant
 - ▶ First class administration and clustering support
 - ▶ Integration with RAD
 - ▶ WS-Security, WS-AtomicTransaction, Service Gateway, SDO support + others

Axis / Tomcat / J2SE compatibility matrix



Axis, Axis2	Axis 1.2	Axis 1.3	Axis2 1.2-1.4
J2SE	1.4.x	5.0 (1.5)	Java 5
Tomcat	5.0.28	5.5.12	5.0 and higher

- Tomcat 5.5.x requires J2SE 5.0, Axis 1.2 J2SE 1.4.x
- Eclipse WTP 1.0 supports Axis 1.2, WTP 1.5 Axis 1.3; WTP 2.0 added support for Axis2 (new Axis code base!); latest version is WTP 3.0
- The combination we use is **Eclipse WTP 2.0 RC2, Axis2 1.2, Tomcat 5.0.28/5.5.26, and Java 5**

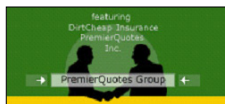
PHP introduction



- PHP: Hypertext Processor
- Portable scripting language especially suited to Web development
- Open source implementation
- Embedded inline with HTML, Syntax similar to C / Java / Perl
- For simplicity, we have used the XAMPP implementation from [apachefriends.org](http://www.apachefriends.org)
 - Single distribution containing Apache, MySQL, PHP and Perl
 - <http://www.apachefriends.org/en/xampp.html>
- SOAP support now native in PHP 5
 - Service consumer programming easier than service provider
 - Implementation in C provides good performance
 - Also NuSOAP 0.7 from sourceforge.net for PHP4 clients



Example introduction



Archie Tekt



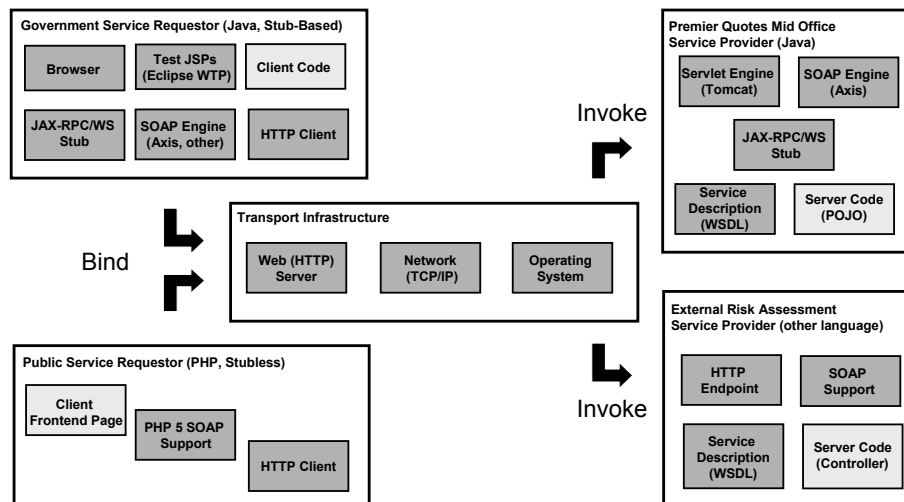
Zippy Coder



Ed U. Cate

- Taken from Perspectives on Web Services Chapter 4.11
 - “Creating a document/literal Service from WSDL”
- Insurance scenario with a fictitious Insurer called PremierQuotes
- Example shows how PremierQuotes generates a report on the total risk under management by the company for a government agency
 - Broken down by year
- XML schemas are defined to represent documents received and returned from the service (standards body)
- WSDL created which references the schemas
- Java service implementation created from WSDL (top-down) by PremierQuotes
- Java service requestor created from WSDL by government agency
- PHP service requestor and Ruby service provider created to show interoperability

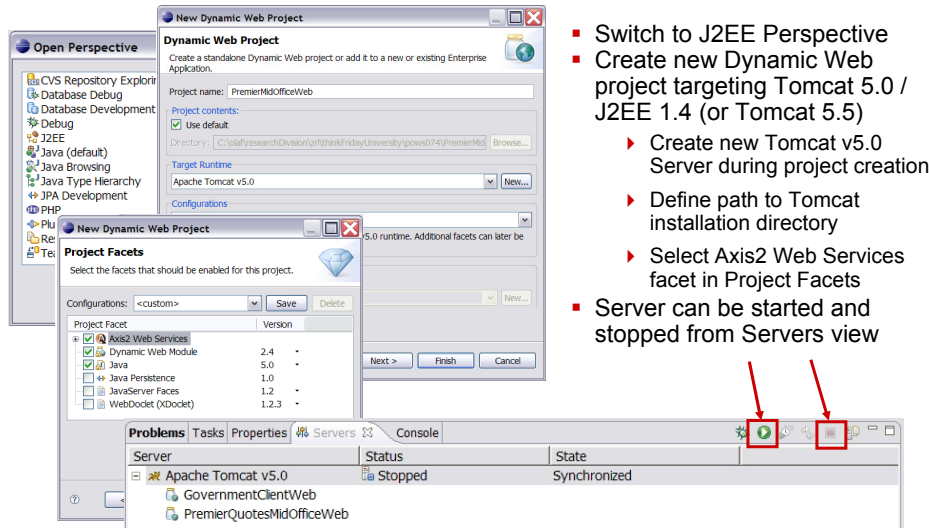
Web services architecture for insurance scenario



Green thread through the development steps

- Step 1: Create WSDL service definition
 - From scratch (top-down) or from existing asset (bottom-up)
- Step 2: Generate server-side implementation stub from WSDL
 - WSDL port type mapped to Java interface and classes
 - WSDL operations mapped to Java methods
 - XML Schema (XSD) types mapped to Java value objects
- Step 3: Complete server-side implementation and test
- Step 4 (optional): register service into UDDI or other registry (publish)
- Step 5: Generate client-side invocation proxy from WSDL, write client
 - Same WSDL to Java mapping steps as on server side
- Step 6 (optional): secure service on network, transport, message layer

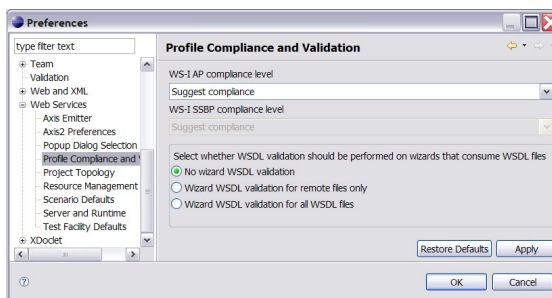
Getting started with Eclipse Web Tools and Tomcat



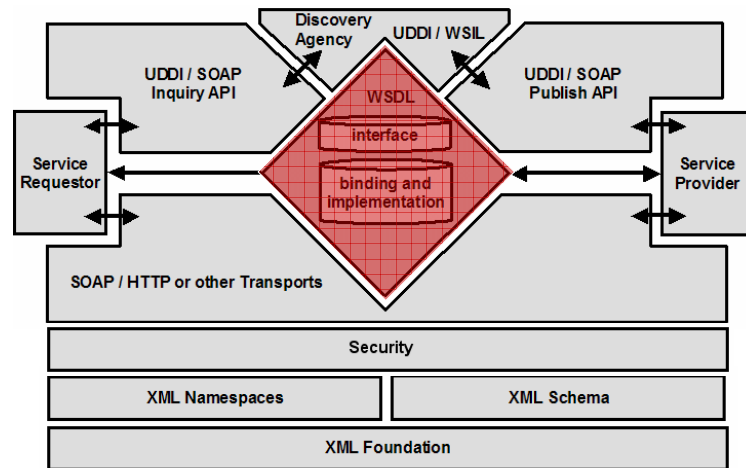
- Switch to J2EE Perspective
- Create new Dynamic Web project targeting Tomcat 5.0 / J2EE 1.4 (or Tomcat 5.5)
 - Create new Tomcat v5.0 Server during project creation
 - Define path to Tomcat installation directory
 - Select Axis2 Web Services facet in Project Facets
- Server can be started and stopped from Servers view

Configuring the Eclipse Web services tools

- Selection of Web Services Preferences available for customisation
- Window -> Preferences Menu
 - Find Web Services entry
- Mandatory step: *define location of Axis2 runtime*
- Optional step: modify setting for the WS-I compliance, default is "Suggest Compliance"
 - WS-I Attachments Profile (WS-I AP)
 - WS-I Simple SOAP Binding Profile (WS-I SSBP)

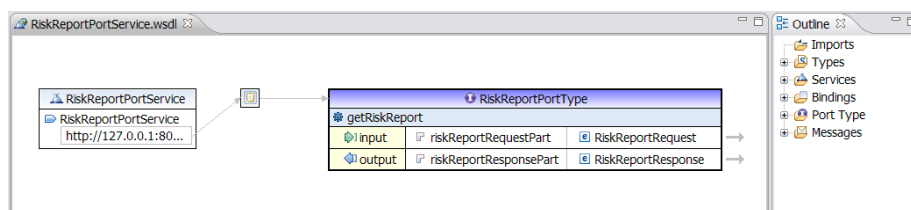


Step 1: WSDL definition



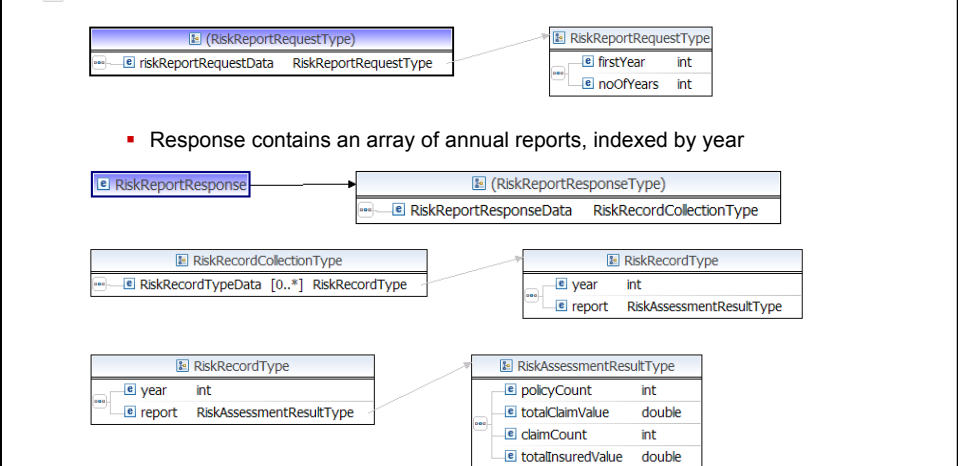
Graphical WSDL Editor

- Create a new WSDL file RiskReport.wsdl using WSDL Editor
 - Alternatively, you can import an existing file

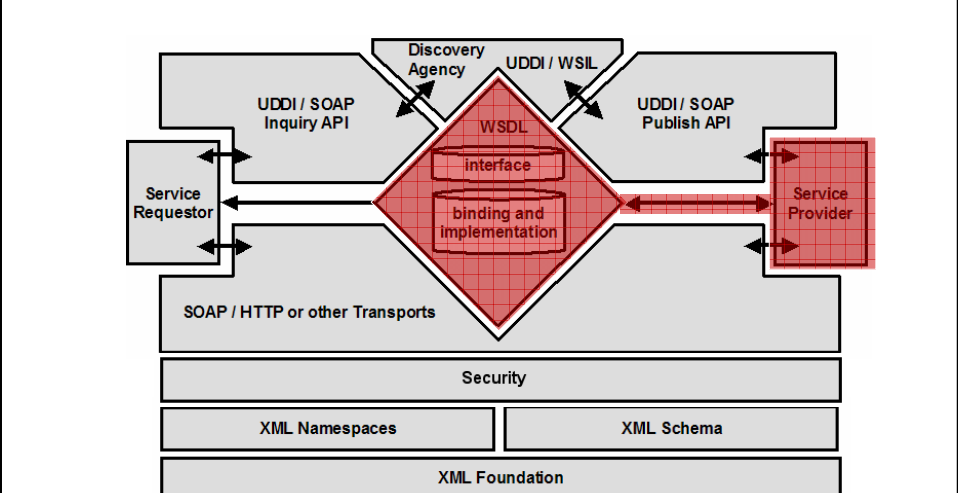


WSDL input and output message contents

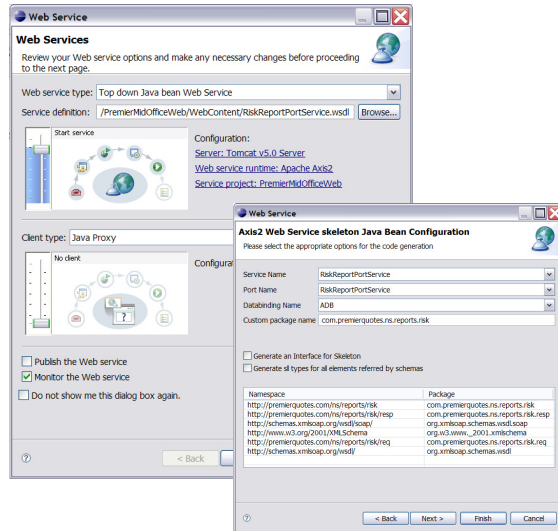
- Request contains two integer values to create report



Step 2: Service provider creation from WSDL



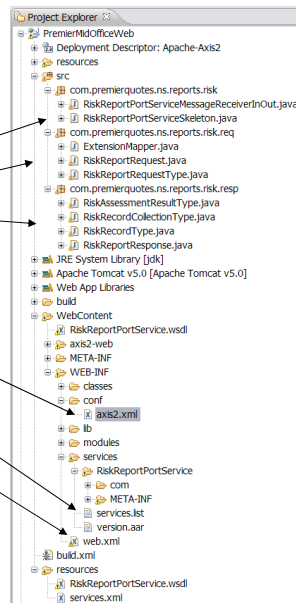
Creating a WS-I compliant service in Java from WSDL



- Select WSDL
- Select Web Services -> Generate Java bean skeleton context menu
- Wizard based on Axis2 WSDL to Java command line tool
 - ▶ Select Axis2 as Web service runtime
 - ▶ Optionally, define custom mappings

Generated code

- JAX-RPC/JAX-WS Service Endpoint Interface (optional) and Service Implementation Skeleton
- Value objects for message parameters
- Axis2 configuration
- Service deployment
- Updated web.xml
- Classpath updates include Axis2 libraries
- Project deployed to server
- Server started



Complete service implementation

- To simplify this example we have stubbed out the service implementation with a random number generator
- Use generated data types and package names from Axis2 WSDL to Java
- Book example based on Cloudscape / Derby DB
- **Note:** Axis emitter interface different in IBM WAS/RAD
- **Note:** Axis and Axis2 use different interfaces

```
package com.premierquotes.ns.reports.risk;
/**
 * RiskReportPortServiceSkeleton java skeleton for the axisService
 */
import com.premierquotes.ns.reports.risk.resp.*;

public class RiskReportPortServiceSkeleton {
    public com.premierquotes.ns.reports.risk.resp.RiskReportResponse getRiskReport(
        com.premierquotes.ns.reports.risk.req.RiskReportRequest riskReportRequest0 ) {

        System.out.println("## Started MidOffice riskReportRequest");
        int noOfYears = riskReportRequest0.getRiskReportRequestData().getNoOfYears();
        int firstYear = riskReportRequest0.getRiskReportRequestData().getFirstYear();
        RiskRecordCollectionType resultArray = new RiskRecordCollectionType();

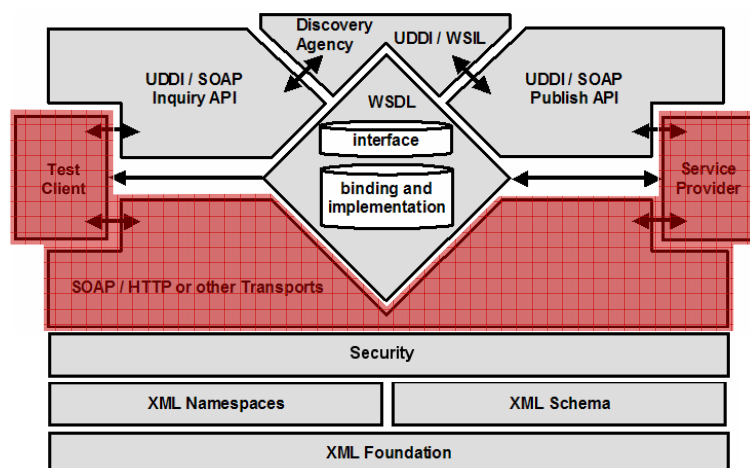
        // loop for each year requested
        for (int i=0; i < noOfYears; i++) {
            RiskRecordType rec = new RiskRecordType();
            rec.setYear(firstYear+i);
            RiskAssessmentResultType report = new RiskAssessmentResultType();
            report.setClaimCount((int) Math.round(Math.random()*100.0));
            report.setPolicyCount((int) Math.round(Math.random()*10000.0));

            report.setTotalClaimValue(Math rint((Math.random()*1000000.0)*
            report.getClaimCount()) / 100.0);
            report.setTotalInsuredValue(Math rint((Math.random()*10000000.0)*
            report.getPolicyCount()) / 100.0);

            rec.setReport(report);
            resultArray.addRiskRecordTypeData(rec);
        }

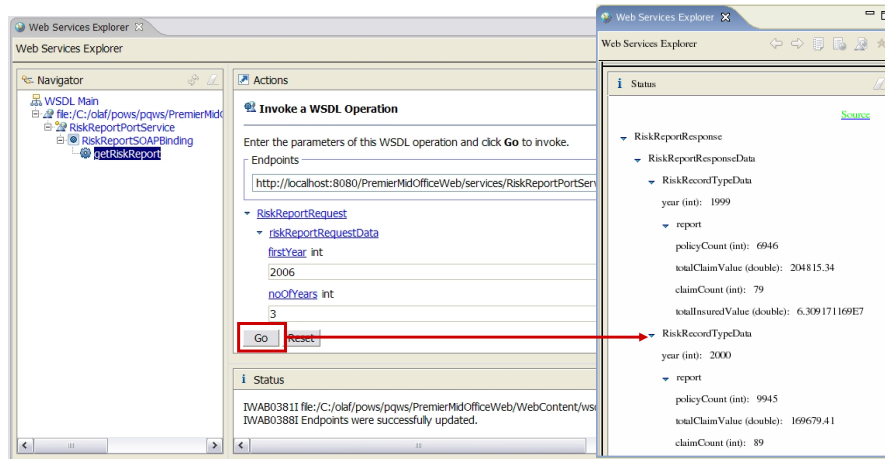
        RiskReportResponse resp = new RiskReportResponse();
        resp.setRiskReportResponseData(resultArray);
        System.out.println("## Completed MidOffice riskReportRequest");
        return resp;
    }
}
```

Step 3: Test service implementation

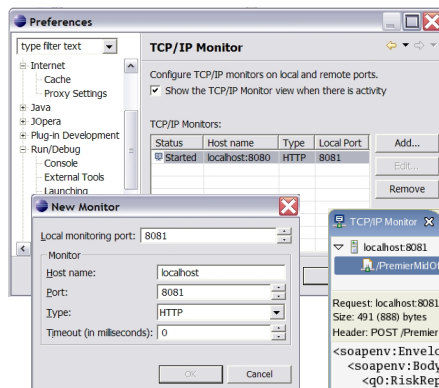


Web Services Explorer

- Test service implementation from WSDL definition – no coding
- View Form or Source of results

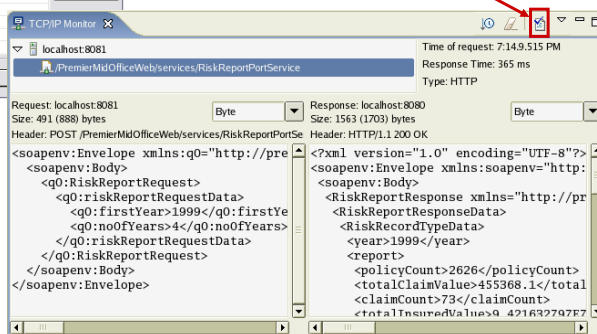


TCP/IP Monitor

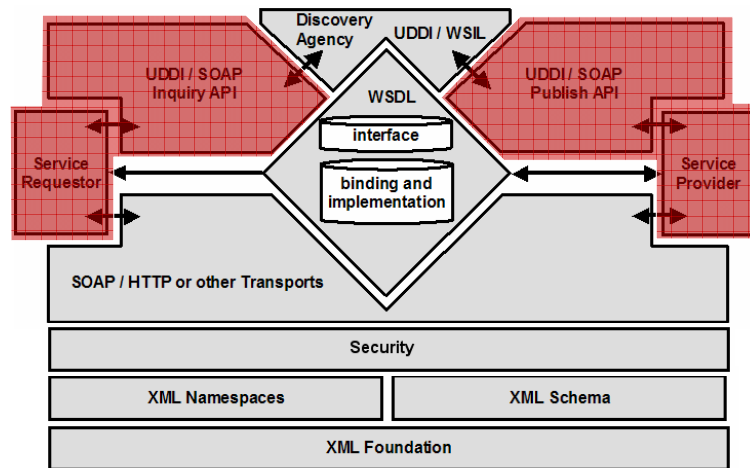


- Creates a new TCP/IP port which listens on requests and responses and forwards to another address
 - ▶ Local or remote
- Integrated WS-I Compliance checking for SOAP messages

- Enable through Window -> Show View -> Other menu
- Under Debug category



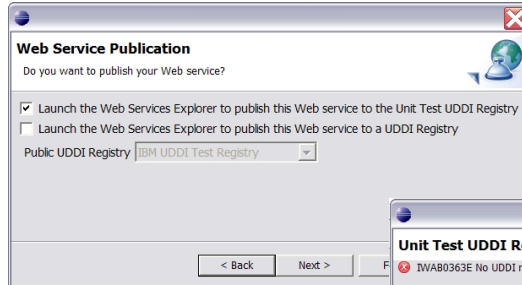
Step 4: Web service publishing and discovery



Publishing with WS-Inspection

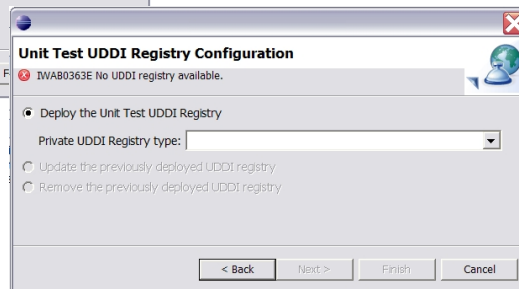
- Generate and update WS-Inspection Language (WSIL) documents
- Browse and import services

Publishing with UDDI

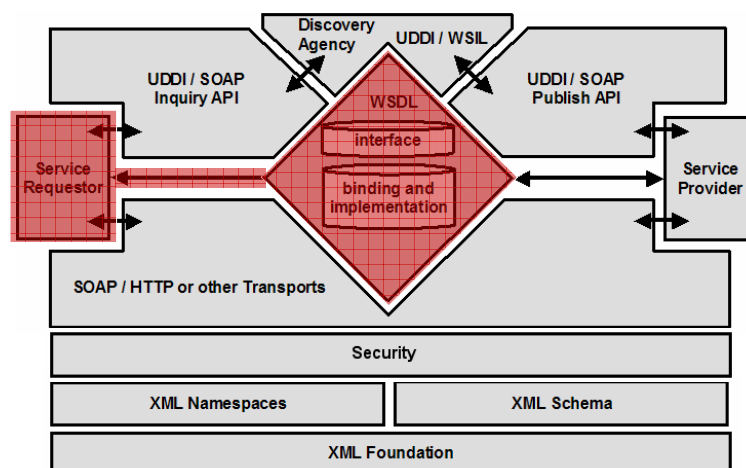


- UDDI Publish feature in Eclipse Web Tools includes references to "Unit Test" UDDI Registry

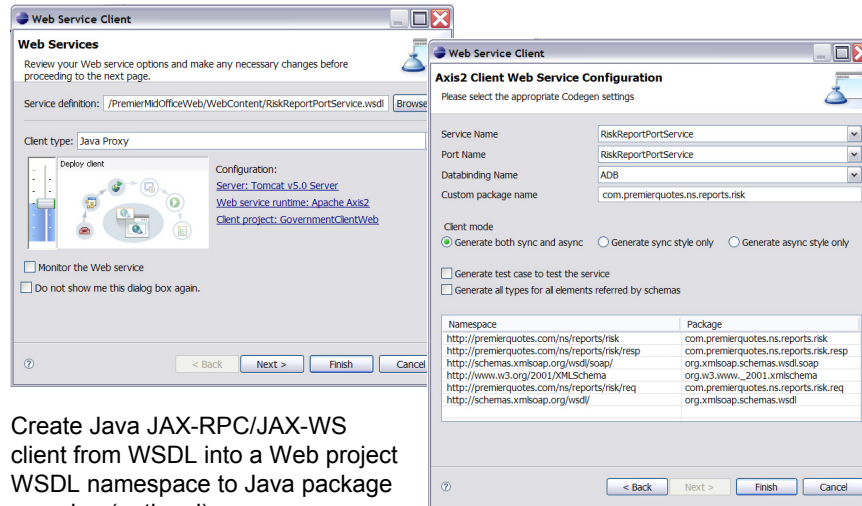
- ▶ Local J2EE UDDI Implementation
- ▶ Not shipping with WTP, but packaged with WAS/RAD



Step 5: Creating a Web service client from WSDL



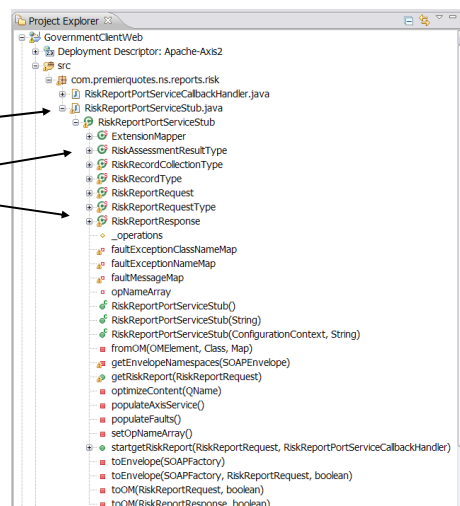
Creating a Web service client in Java



- Create Java JAX-RPC/JAX-WS client from WSDL into a Web project
- WSDL namespace to Java package mapping (optional)

Generated client-side artefacts

- JAX-RPC/JAX-WS Service Endpoint Interface (optional) and service proxy/stub
- Value objects for message parameters
- Classpath updates include Axis2 libraries
- Project deployed to server
- Server started



Creating a client

- Create a new Java client with a **main()** method called **RiskReportClient**
- Implement using the generated Service Endpoint Interface and Service Locator (Axis) and Service Stub (Axis2)
- Once complete, use Run -> Run ... menu
 - Add program arguments **<first year> <no years>**
 - e.g. **1999 5**
- **Note:** Axis emitter interface different in IBM WAS/RAD
- **Note:** Axis and Axis2 use different interfaces

```
import com.premierquotes.ns.reports.risk.RiskReportPortServiceStub;

public class RiskReportClient {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: RiskReportClient <first year> <no of years>");
            return;
        }
        try {
            RiskReportPortServiceStub sei = new RiskReportPortServiceStub(
                ("http://127.0.0.1:8080/PremierMidOfficeWeb/services/RiskReportPortService"));

            // populate request with passed values
            com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskReportRequestType data =
            new com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskReportRequestType();
            data.setFirstYear(Integer.parseInt(args[0]));
            data.setNoOfYears(Integer.parseInt(args[1]));
            com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskReportRequest req = new
            com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskReportRequest();
            req.setRiskReportRequestData(data);

            // invoke the service
            com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskReportResponse resp =
            sei.getRiskReport(req);
            com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskRecordCollectionType
            recordscoll = resp.getRiskReportResponseData();
            com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskRecordType[] records =
            recordscoll.getRiskRecordTypeData();

            // loop through each record printing results to console
            for (int i=0; i < records.length; i++) {
                System.out.println("REPORT FOR " + records[i].getYear() + " :");
                System.out.println("-----");
                com.premierquotes.ns.reports.risk.RiskReportPortServiceStub.RiskAssessmentResultType
                result = records[i].getResult();
                System.out.println("Total # policies      =" + result.getPolicyCount());
                System.out.println("Total # claims      =" + result.getClaimCount());
                System.out.println("Total value of policies =" + result.getTotalInsuredValue());
                System.out.println("Total value of claims =" + result.getTotalClaimValue());
                System.out.println("-----");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

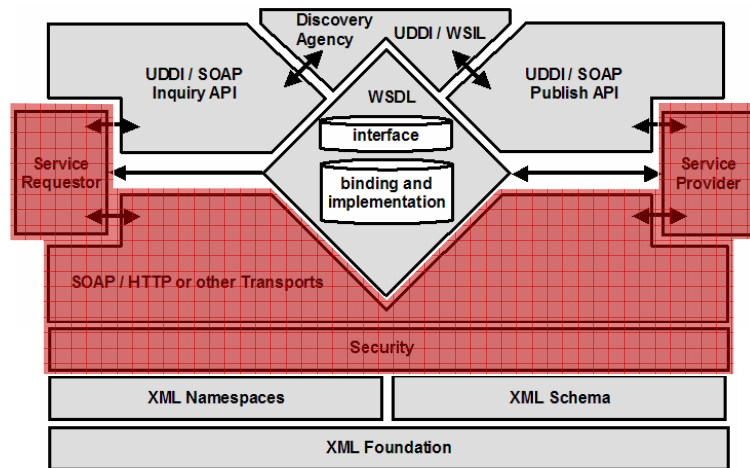
Creating a Web service client in PHP 5

- PHP SOAP extension provides a simple to use class called SoapClient
 - Create a new instance, passing WSDL URL as a parameter
 - Call the WSDL operation directly, passing in an array of parameters
 - Extract data from returned structure and populate table
- Copy PHP file into XAMPP **/htdocs** directory – invoke from browser
 - Check file permissions

```
<?php
/* Display Risk Report */
function displayRiskReport ($firstYear, $years) {
    try {
        $soapClient = new SoapClient(
            "http://localhost:8080/PremierMidOfficeWeb/services/RiskReportPortService?wsdl");

        /* Retrieve the report */
        $rrPart->firstYear = $firstYear;
        $rrPart->noOfYears = $years;
        $rr->riskReportRequestData = $rrPart;
        $rrResponse = $soapClient->getRiskReport($rr);
        $rrReturn = $rrResponse->RiskReportResponseData;
    } catch (SoapFault $soapFault) {
        echo $soapFault, "<br/>";
    }
}
?>
```

Step 6: Secure communication with SOAP/HTTPS



Securing a Web services implementation on Tomcat with SSL

- Use the J2SE keytool command to create a certificate keystore with a self-signed certificate using the RSA algorithm
 - The **keystore** contains certificates used by the server
 - The **truststore** contains certificates trusted by the server
 - `<JAVA_HOME>/bin/keytool -genkey -alias tomcat -keyalg RSA -keystore <TOMCAT_HOME>/conf/.keystore`
 - Complete questions when prompted
- Edit Tomcat configuration file in local workspace (not Tomcat install dir)
 - Java Perspective: Servers -> Tomcat @ localhost-config -> server.xml
 - Remove comments and specify as follows

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector className="org.apache.coyote.tomcat5.CoyoteConnector"
  port="8443" minProcessors="5" maxProcessors="75"
  enableLookups="true" disableUploadTimeout="true" acceptCount="100"
  debug="0" scheme="https" secure="true"; clientAuth="false"
  sslProtocol="TLS" keystoreFile="/opt/tomcat-5.0.28/conf/.keystore"
  keystorePass="your_password"/>
```

- Restart and open the Tomcat home page using <https://localhost:8443>
- Update client code to reference keystore/truststore and HTTPS URL

Module 3: Summary

Web services programming isn't fundamentally different from what I've been doing with J2EE and XML.

I bet I can get some of these new wizards and tools to do most of the hard work.

- For both simple and sophisticated Web service development, you can rely on tools support plus some basic Java programming skills
- The Eclipse Web Tools project provides all of the features you need when using the basic Web services building blocks (SOAP, WSDL, and UDDI)
- Both Axis2 and commercial tools and runtimes support specifications like WS-Security, WS-BPEL, or WS-ReliableMessaging

Module 4a: Putting the A back into SOA – Architectural Decision Modeling Wiki

How can SOA decisions be identified, made, and enforced?

All these options to implement SOA... **has anybody done it before?**
If so, how?

Decisions and outcome are obvious...
just put the right people on the project,
they know better anyway.

Grady Booch in his SW Handbook blog (October 2006)

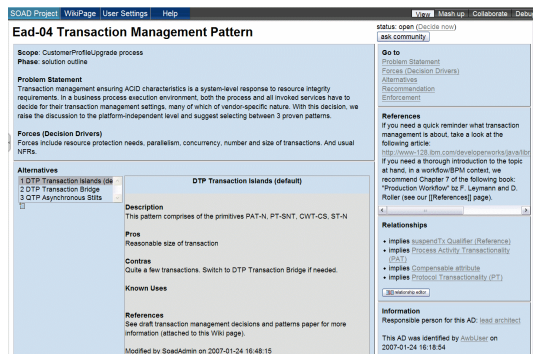
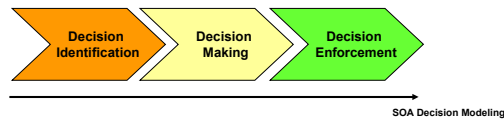
- “IMHO, SOA's value proposition begins with the A in its acronym: architecture”
 - ▶ “There are many things we already know about what constitutes a good architecture and what does not. *Stripped away of all the hype, a Service-Oriented Architecture is essentially a variant of well-proven message-passing architectural patterns.* The variance comes in the form that services are cleverly designed to take advantage of the Web-centric infrastructure that pervades many organizations: services allow you to send and receive semantically rich messages through firewalls.”
 - ▶ “What distinguishes a good service from a bad one? what should the granularity of a service be? when should I offer up a stateless service versus a stateful one? as for the stateful ones, how to I express their semantics, and how do I ensure their misuse doesn't corrupt my system? how do I express the semantics of a society of services (only the most trivial services work in isolation)? how do I decide upon the semantics of the information transmitted by these services so that locally they are efficient and useful but that also globally they are consistent? how do I expose some services to some clients and hide them from others? how do I offer up variants on a service, so that different clients see a different face to that service? how do I ensure the security of critical services, such that I am confident I'm not opening up holes in my enterprise that will let the bad guys in? what services should I expose to the world, and what services should I keep hidden? where are services appropriate, and where are they not? how do I best expose services in a legacy system? who should own/maintain these services? are there alternative architectural patterns I should employ instead of services, and where, and why?”

How to structure and share answers to these questions?

- Patterns...
- Methodologies...
- Reference architectures...
- ... do not provide sufficient answers to these and other *architectural decisions*:
 - ▶ Presentation layer
 - Rich vs. thin client, Web 2.0 vs. plain HTML, etc.
 - ▶ Business modeling interface and process (service composition) layer
 - Process modeling and execution language, control flow, data flow
 - ▶ Integration layer
 - ESB technology and product selection, configuration

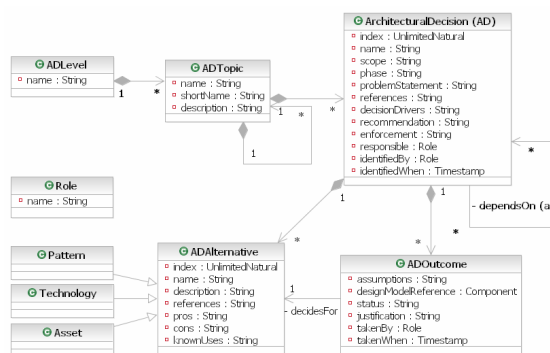
SOA Decision Modeling (SOAD) in a nutshell

- Technical decision *identification*, *making*, and *enforcement* often disconnected, leading to cost, quality, and risk mitigation issues
 - One-of-a-kind transaction, security, and reliability design (no reuse)
 - Gaps between logical and physical models, LOBs, presales and post sales
- SOAD provides innovative method, content, and tools for *end-to-end* decision maker collaboration and best practices exchange:
 - SOA decisions, patterns, and policies *continuum*
 - Reusable, machine-readable decision *models*
 - Web 2.0* tool support

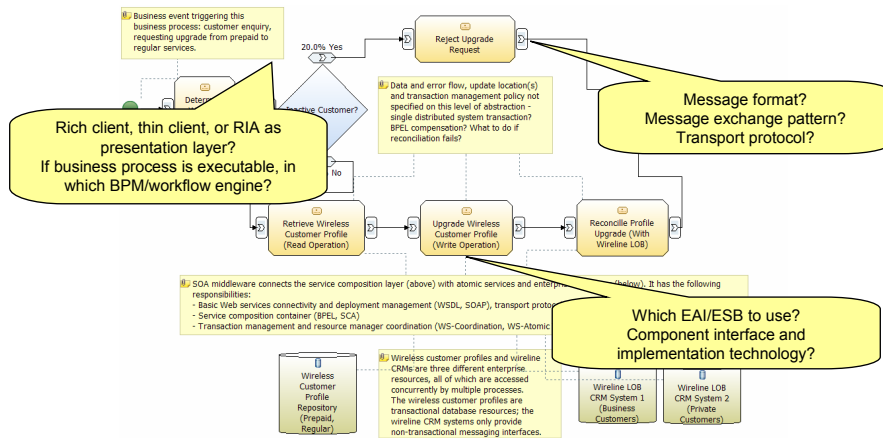


Architectural decision modeling for reuse [QoSA 2007]

- Architectural decisions capture key design issues along with the identified alternatives and their tradeoffs, as well as the rationale behind a design:
 - Conscious design decisions concerning a software system as a whole, or one or more of its core component
 - Determining the non-functional characteristics and quality factors of the system
- Documenting architectural decisions *should* be state of the practice
- Using pre-populated AD *models* in an active, guiding role is a novel approach
 - Our work is based on existing work and ongoing research e.g. by Philippe Kruchten



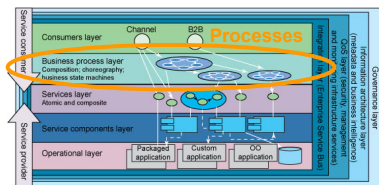
Examples for SOA design issues: Process realization



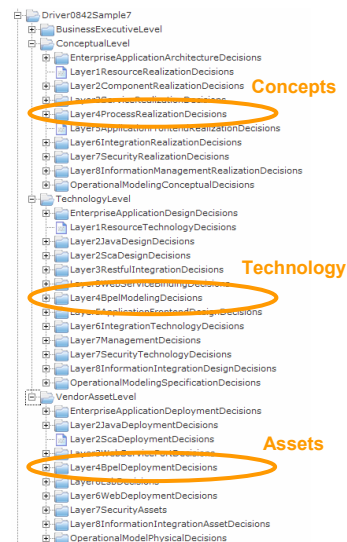
- Solution architects concerned with many questions regarding process transactionality, protocols, product selection/configuration, logical and operational modeling, etc.

Layered SOA reference architecture and SOA decisions

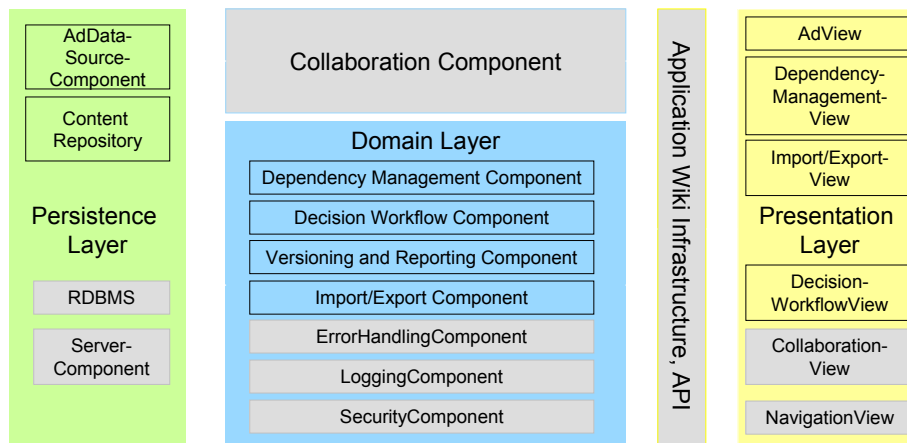
- Decision space organized by layers of a SOA reference architecture
 - IBM SOA Solution Stack by default [IBM SSS]
 - Other ones possible



- Refinement dependencies from conceptual to technology to vendor asset level
 - Process layer: workflow *concepts*, BPEL *technology*, IBM Process Server product *asset*
 - Same for integration layer: messaging, ESBs



Collaboration tool support: Layered application Wiki



Rationale for using an application Wiki as SOAD Front end

- Decision drivers:
 - ▶ User experience, small footprint, support for collaboration use cases, API
- Architecture alternatives and their pros and cons:
 - ▶ Rich user experience, e.g., via Eclipse plugin
 - Only way to share information is via version control system or import/export
 - ▶ Plain HTML
 - Small footprint, but content has to be managed centrally
 - ▶ Plain wiki
 - Self-governing content management, but content lost in presentation layer
 - ▶ Application wiki
 - Benefits of Wiki plus server-side APIs for automated processing

Benefits of model-driven decision capturing

- Overcome shortcomings existing knowledge management approaches
 - Existing practices (structured text templates) lack precision and do not scale
 - Existing decision guides focus on single sub domain/scenario and jump from concepts straight to assets (or start at asset level)
- In line with stepwise refinement concept in Rational Unified Process and IBM Global Services Method
 - Three levels of refinement for component and operational modeling work products
 - Decision models bridge the gap between these concerns
- Structured way of stating best practices
 - Bound to architectural context: decision drivers, design model
 - Big picture preserved via decision relationships

Summary of benefits and novel contributions of SOAD

- Time savings
- Risk mitigation
- Quality improvements
- Active, guiding micro-methodology
 - Anticipating required decisions, giving concrete advice
 - Complementing Rational Unified Process (RUP), Service Modeling and Architecture (SOMA), and IGS Method
- Model-driven approach to AD capturing
 - Reusable ADMs replacing text tables
- Best practices bound to architectural context

Module 4b: SOA and Web Services Best Practices

Which are the pitfalls to be avoided during service design?

We have to **manage expectations** so that we can be sure to deliver in time and on budget.

We **don't want to repeat all the mistakes** made by the very early adopters of this technology.

This module is excerpted from the book "Perspectives on Web services" by Olaf Zimmermann, Mark Tomlinson, and Stefan Peuser, Springer-Verlag Berlin Heidelberg New York 2003, ISBN 3-540-00914-0. This work is subject to copyright. © Springer Verlag 2003. All rights reserved.

Building SOAs with Web Services

Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

117

Module 4b: Agenda

- Usage scenarios for Web services
 - Business litmus test
 - Technical litmus test
- Architectural decisions
 - Addressing non-functional requirements
 - Gaps and countermeasures
- Best practices
 - SOAP, WSDL, UDDI
 - SOA in general
- Questions and answers session

Building SOAs with Web Services

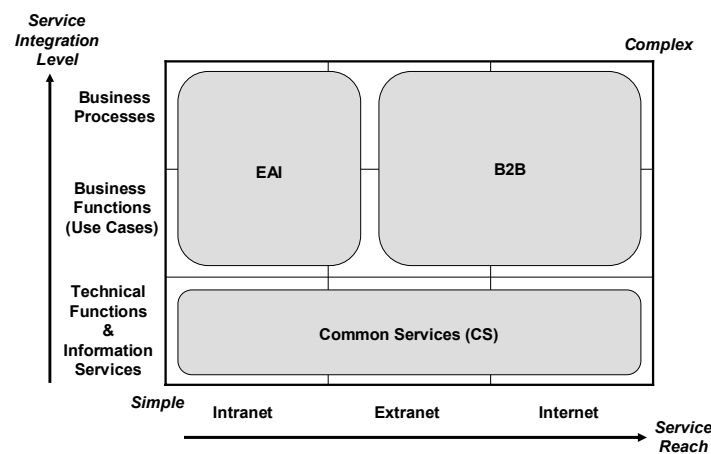
Copyright is held by the author/owner(s). OOPSLA 2008, October 19–23, 2008, Nashville, Tennessee, USA. ACM 978-1-60558-220-7/08/10.

118

Web services – holy grail or déjà vu?

- SOA concepts and Web services technologies have been used successfully to address the following requirements:
 - ▶ Automation through application clients
 - ▶ Connectivity for heterogeneous worlds
 - ▶ Information and process sharing
 - ▶ Reuse and flexibility
 - ▶ Dynamic discovery of service interfaces and implementations
 - ▶ Business process orchestration without programming
- Additional advantages include:
 - ▶ Non-invasiveness of the technology
 - ▶ Productivity boost and industry momentum
 - ▶ Standardisation and openness
 - ▶ Low project risk

Usage scenarios



- Plus: EDI replacement, portal adaptors, competency-focussed organisations, mobile device communication, RMI/IIOP substitute, file transfer, grid computing ...

Take the business litmus test – are Web services for you?

- If you answer “Yes” to any of the following business questions, consider using Web services:
 - ▶ Do you want to interact with your business partners in a more automated fashion?
 - ▶ Is there a requirement to connect stovepipe applications/packages?
 - ▶ Do you want to make legacy assets available for reuse?
 - ▶ Looking for a more flexible IT architecture that can easily adapt to change? (*agility / competitiveness / responsiveness*)
 - ▶ Is your system environment heterogeneous?
- Note that there is a place for both Web services and more “traditional” EAI approaches. They also complement J2EE.

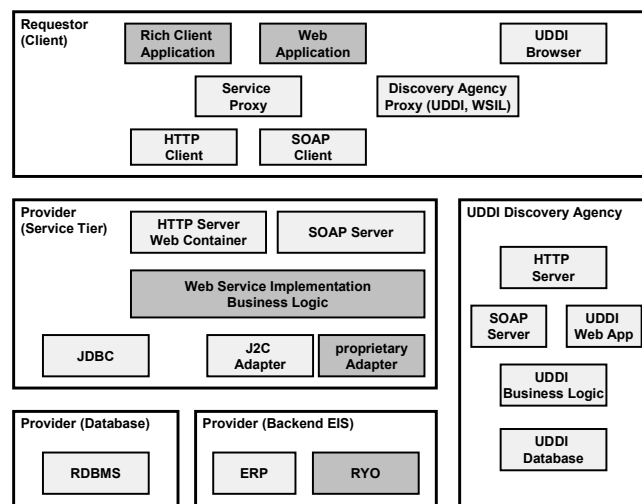
The technical litmus test

- If you answer “Yes” to any of the following technical questions, consider using Web services:
 - ▶ In your use case model, are other systems the primary actors in your system?
 - ▶ Do you have to support a heterogeneous or unknown client environment?
 - ▶ Do you plan to extend the reach of J2EE applications to application clients?
 - ▶ Do you already transfer XML documents via HTTP-GET or -POST?
 - ▶ Do your rich application clients use proprietary communication channels and are your firewall administrators unhappy about this?
 - ▶ Does the number of service providers in your environment vary?
 - ▶ Is your existing infrastructure capable of handling a rather verbose text-based, self-describing message exchange format?

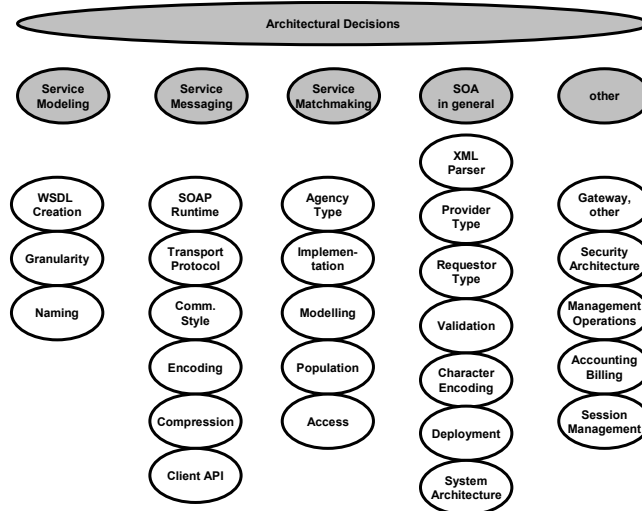
Addressing potential inhibitors

- The following are the most typical inhibitors to adoption. Most can be overcome:
 - Over-enthusiastic expectations
 - Goal conflicts
 - General scepticism regarding maturity of new technology
 - Security and performance concerns
 - Logistical and organisational issues
 - Skill deficiencies
 - Roll-your-own temptation
- The exact ROI and TCO is often difficult to determine up-front

W3C WSA solution architecture



Key architectural decisions which must be made



Handling non-functional requirements (NFRs)

- **Performance**
 - Ensure that requirements are realistic
 - Build a small prototype at start of project to check if criteria can be met
- **Scalability**
 - Design your services to be as stateless as possible
 - Normal J2EE scaling strategies can be applied
- **Availability**
 - Normal J2EE availability strategies can be applied
- **Robustness**
 - Create an effective error handling mechanism with SOAP fault handling
 - The product building-blocks are now mature enough for prime-time
- **Portability**
 - Stick to agreed industry standards/specifications such as JAX-RPC, where they exist, and be prepared for changes (JAX-WS)

Gaps and countermeasures

- The **XML language binding and encoding maze**
 - ▶ WSDL and SOAP do not define any language bindings
 - ▶ Adhere to the WS-I Basic Profile 1.1, use (wrapped) document/literal
- **Security solutions**
 - ▶ *Network Layer* security (IPSec, VPNs)
 - ▶ *Transport Layer* and *Application Server* security (Basic vs. Keys)
 - ▶ *XML-based* security (XML-Signature, XML-Encryption, SAML)
 - ▶ *WS-Security* and its additional specifications (WS-Policy, WS-Trust etc.)
 - ▶ Or *Application Layer* security if all else fails
- **Web service management approaches**
 - ▶ Look for SOAP runtimes which have JMX instrumentation
 - ▶ OASIS Web Services Distributed Management (WSDM)
- **Transactional and context semantics plus orchestration**
 - ▶ Still emerging: WS-Coordination, transaction support, WS-BPEL

Best practice highlights I: WSDL and modeling

- Follow the *design-by-contract* principle
- Separate concerns and isolate interfaces
- Provide interoperable versions of interfaces
- Follow the *bottom-up* design approach
- Expose coarse-grained interfaces
- Avoid complex operation signatures
- Stick to standard XML schema constructs

Follow the **design-by-contract** principle

Always describe your services using WSDL and XML schema. Add comments for human consumption, and put the documents on a Web server. Consider developing your own WSDL generator if many similar processes need to be supported.

Follow a **meet-in-the-middle WSDL design approach**

Generating WSDL from server side Java can be a good idea, if you make sure no programming language specifics make it into your interface. This provides a jump-start for beginners.

and type names small and simple schema best practices

Best practice highlights II: SOAP and messaging

- Use HTTP as the default transport
- Carefully observe the messaging overhead
- Be aware of the trade-off between performance and reliability
- Design your Web services to be stateless
- Avoid custom mappings, write standard mappings
- Include, but do not rely on the HTTP action header
- Be careful with message handling
- Try to leverage existing transport mechanisms

Include, but do not rely on, the HTTP SOAP action header

This should not be used for routing purposes – this should be based on the namespace attribute of the body element. The feature is now deprecated, but certain SOAP engines might still expect it to be present.

Carefully observe the messaging overhead

Also known as SOAP verbosity. The overhead can be three to 20 times, depending on the naming conventions and the nesting levels of the document. Use TCP monitors and try different runtime parsers/engines.

Best practice highlights III: UDDI and matchmaking

- Carefully evaluate which type of UDDI registry (private vs. public) is suited for your scenario
- Consider lightweight alternatives
- Obey the best practices already in place

Carefully evaluate which type of UDDI registry is suited for your scenario

Using UDDI on the Web is problematic for organisational reasons. UDDI can be useful in intranet and extranet scenarios where the user groups are well known. However, the API is rather complex to use. Defining specific tModels and UUIDs may relieve some of the data consistency and trust issues.

Best practice highlights IV: SOA and project approach in general

- Clearly identify business needs
- Decide carefully whether Web services are the right solution for your problem at hand
- Apply standards pragmatically:
- Use stateless session EJBs and RESTful architecture
- Do not over-architect, do not over-engineer

Resist the temptation to be over-creative

Do not implement your own SOAP layer; any Roll-Your-Own (RYO) approach compromises the Web services value proposition. Let the vendor labs and open source community worry about the runtime, otherwise RYO is likely to become *rewrite your own* (every time).

Apply standards pragmatically: follow the 80-20 rule

Do not always use all of the features in each and every specification. Upgrade to high specification levels only if there is a concrete need, not for its own sake (e.g. SOAP 1.1 vs. 1.2). The 80-20 or “keep it simple” rule helps with interoperability.

Follow best practices

Interoperability

Any questions?



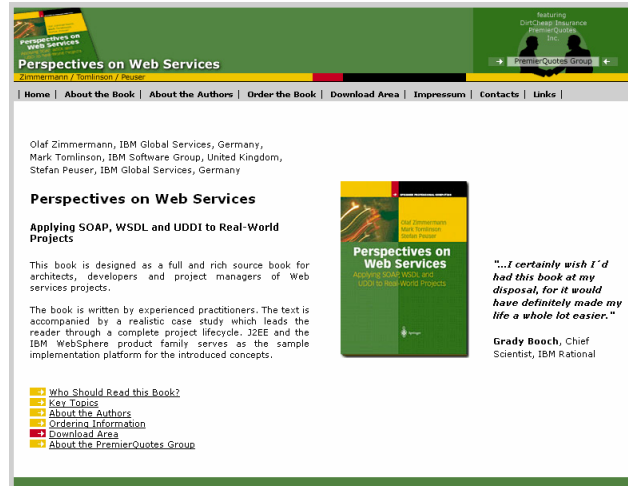
If you don't have questions for us... we have some for you

- Which business challenges does the demonstrated sample SOA/Web services solution address?
- Which core Web services specifications were involved? And where?
- Where in the practical demonstration was the ESB?
- Same for service composition?
- Could you identify any familiar design patterns in the generated JAX-RPC artifacts?
- Apart from function and parameter names, what does the interface contract agreed between GovernmentOffice and PremierQuotes have to define?
- What information is required to get the Web services enablement project started?

Optional exercise: Customer Information (CI) Service

- Install Eclipse Web Tools Project
- Start from a Java class providing a customer information method:
 - ▶ In parameter: name
 - ▶ Out parameters: email address, areas of interest
- Apply Java-to-WSDL wizards to create Web service provider
- Apply WSDL-to-Java wizard to create invocation stub for test client
- Test the CI Web service with Web Services Explorer
- Compose risk report and customer information service into one service
 - ▶ Custom code or jOpera

Check out our Web site – www.soadections.org



Additional Materials for Handouts

Tomcat & Axis install

- Tomcat 5.0/5.5
 - ▶ Extract Tomcat
 - ▶ Add admin/manager user to XML configuration
 - ▶ Tomcat homepage is <http://localhost:8080>
- Java Activation Framework (JAF) 1.0.2 and Java mail library
 - ▶ Download JAF from Sun site might be required (Axis prerequisite)
 - ▶ Copy [mail.jar](#) and [activation.jar](#) to [tomcat/common/lib](#) directory, and add these libraries to client project build path
- Axis 1.2/1.3 (earlier versions of this tutorial)
 - ▶ Ships with Eclipse WTP 1.0, so no need for additional installation steps
 - ▶ Optional: run Axis validation to check if any dependencies are missing
- Axis2 1.2 (2007)
 - ▶ Binary image has to be loaded into Eclipse via Window -> Preferences... (under Web Services and then Axis2 Preferences)

Eclipse Web Tools install

- Install Eclipse 3.3/3.4
- Launch [./eclipse](#)
- Install Web Tools Project (WTP) via Eclipse update manager (or manually):
 - ▶ Select Help, then Software Updates, then Find And Install...
 - ▶ Click on Search for new features to install, click on Next
 - ▶ Create a New Remote Site... for WTP, pointing to <http://download.eclipse.org/webtools/updates>
 - ▶ Select WTP 2.0 and all required prerequisites
 - ▶ Perform the installation
- Create workspace
- Create Tomcat 5.0/5.5 server configuration
- Start and stop server to check configuration

RiskReport.wsdl – WSDL used in example (1/4)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="RiskReport"
  xmlns:impl="http://premierquotes.com/ns/reports/risk"
  xmlns:req="http://premierquotes.com/ns/reports/risk/req"
  xmlns:resp="http://premierquotes.com/ns/reports/risk/resp"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://premierquotes.com/ns/reports/risk">
  <wsdl:types>
    <!-- Schema definition for request document -->
    <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
      targetNamespace="http://premierquotes.com/ns/reports/risk/req"
      xmlns:req="http://premierquotes.com/ns/reports/risk/req"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <complexType name="RiskReportRequestType">
        <sequence>
          <element name="firstYear" type="xsd:int"/>
          <element name="noOfYears" type="xsd:int"/>
        </sequence>
      </complexType>
      <element name="RiskReportRequest">
        <complexType>
          <sequence>
            <element name="riskReportRequestData" nillable="true" type="req:RiskReportRequestType"/>
          </sequence>
        </complexType>
      </element>
    </schema>
```

RiskReport.wsdl – WSDL used in example (2/4)

```
<!-- Schema definition for response document containing year and data -->
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://premierquotes.com/ns/reports/risk/resp"
  xmlns:resp="http://premierquotes.com/ns/reports/risk/resp"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <complexType name="RiskAssessmentResultType">
    <sequence>
      <element name="policyCount" type="xsd:int"/>
      <element name="totalClaimValue" type="xsd:double"/>
      <element name="claimCount" type="xsd:int"/>
      <element name="totalInsuredValue" type="xsd:double"/>
    </sequence>
  </complexType>
  <complexType name="RiskRecordType">
    <sequence>
      <element name="year" type="xsd:int"/>
      <element name="report" nillable="true" type="resp:RiskAssessmentResultType"/>
    </sequence>
  </complexType>
  <complexType name="RiskRecordCollectionType">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="RiskRecordTypeData" nillable="true"
        type="resp:RiskRecordType"/>
    </sequence>
  </complexType>
```

RiskReport.wsdl – WSDL used in example (3/4)

```
<element name="RiskReportResponse">
  <complexType>
    <sequence>
      <element name="RiskReportResponseData" nillable="true"
        type="resp:RiskRecordCollectionType"/>
    </sequence>
  </complexType>
</element>
</schema>
</wsdl:types>

<!-- Input and output messages -->
<wsdl:message name="riskReportRequestMsg">
  <wsdl:part element="req:RiskReportRequest" name="riskReportRequestPart"/>
</wsdl:message>
<wsdl:message name="riskReportResponseMsg">
  <wsdl:part element="resp:RiskReportResponse" name="riskReportResponsePart"/>
</wsdl:message>

<!-- Port type - links messages and operation -->
<wsdl:portType name="RiskReportPortType">
  <wsdl:operation name="getRiskReport">
    <wsdl:documentation>Returns a collection of risk reports for the period specified
    </wsdl:documentation>
    <wsdl:input message="impl:riskReportRequestMsg" name="riskReportRequestMsg"/>
    <wsdl:output message="impl:riskReportResponseMsg" name="riskReportResponseMsg"/>
  </wsdl:operation>
</wsdl:portType>
```

RiskReport.wsdl – WSDL used in example (4/4)

```
<!-- Binding - defines invocation style and encoding for port type -->
<wsdl:binding name="RiskReportSOAPBinding" type="impl:RiskReportPortType">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getRiskReport">
    <wsdlsoap:operation
      soapAction="http://premierquotes.com/ns/reports/risk/getRiskReport"/>
    <wsdl:input name="riskReportRequestMsg">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="riskReportResponseMsg">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<!-- Port - defines endpoint for a specified binding -->
<wsdl:service name="RiskReportPortService">
  <wsdl:documentation>Example Risk Reporting Service from Perspectives on Web Services
  </wsdl:documentation>
  <wsdl:port binding="impl:RiskReportSOAPBinding" name="RiskReportPortService">
    <wsdlsoap:address
      location="http://localhost:8080/PremierMidOfficeWeb/services/RiskReportPortService"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

RiskReportPortService.wsil – WS-Inspection file

```
<?xml version="1.0" encoding="UTF-8"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
  xmlns:wsilwsdl="http://schemas.xmlsoap.org/ws/2001/10/inspection/wsdl/"
  xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/"
  xmlns:uddi="urn:uddi-org:api">
  <service>
    <abstract xml:lang="en-US">
    </abstract>
    <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
      location="RiskReportPortService.wsdl">
    <wsilwsdl:reference endpointPresent="true">
      <wsilwsdl:referencedService xmlns:impl="http://premierquotes.com/ns/reports/risk">
        impl:RiskReportPortService
      </wsilwsdl:referencedService>
    </wsilwsdl:reference>
    </description>
  </service>
</inspection>
```

Trademarks

AmberPoint is a trademark of AmberPoint, Inc.
 Amazon is a registered trademark of Amazon.com, Inc. and its affiliates
 Apache Tomcat, Axis and Xerces are trademarks of The Apache Software Foundation
 BEA is a registered trademark of BEA Systems, Inc.
 CDDI is a trademark of CDDI Forum Limited
 CORBA is a registered trademark of the Object Management Group, Inc.
 DCE is a trademark of The Open Group
 D-U-N-S® is a registered trademark of Dun & Bradstreet Corporation
 eBay is a registered trademark of eBay Inc.
 Eclipse is a registered trademark of the Eclipse Foundation
 GLN is a trademark of the Uniform Code Council Inc.
 Google™ is a trademark of Google Technology Inc.
 HP is a registered trademark of Hewlett-Packard Development Company, L.P.
 IBM, AlX, alphaWorks, CICS®, Cloudscape, Crossworlds®, DB2®, developerWorks, Domino, e-business on demand, eServer, HACMP/6000, IMS, MQSeries®, Tivoli®, pSeries, Rational®, Rational Unified Process®, RUP®, Rational ClearCase®, Rational XDE, Redbooks, SOMA, VisualAge, WebSphere®, all WebSphere based trademarks, zSeries and z/OS are trademarks of International Business Machines Corporation in the United States, other countries, or both.
 IETF is a trademark of The Internet Society
 Iona is a registered trademark of IONA Technologies
 Linux is a registered trademark of Linus Torvalds.
 Microsoft, .NET, Microsoft Office, Visual Basic, Win32, Windows®, Windows® NT, and Windows® 2000 are either registered trademarks or trademarks of Microsoft Corporation in the United States, other countries, or both.
 NAICS was developed in cooperation with the US Economic Classification Policy Committee, Statistics Canada, and Mexico's Instituto Nacional de Estadística, Geográfica e Informática

OASIS is a registered trademark of OASIS Open
 OMG is a registered trademark of the Object Management Group, Inc.
 Perl is a registered trademark of the Perl Foundation
 PHP is a registered trademark of the PHP Group
 Red Hat® is a registered trademark of Red Hat, Inc.
 SAP and R/3 are trademarks or registered trademarks of SAP Aktiengesellschaft.
 SourceForge is a trademark of the Open Source Technology Group
 Sun, Sun Open Net Environment, Sun ONE, Solaris, Java and all Java-based trademarks (e.g. EJB, J2EE, J2SE, JavaBeans, Java Connector, JCA, JDK, JVM) are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
 Thoughtworks is a registered trademark of ThoughtWorks, Inc.
 UDDI is a trademark of OASIS Open
 UNDP is a trademark of United Nations Development Programme (UNDP) and Dun & Bradstreet Corporation (D & B)
 UML is a registered trademark of the Object Management Group, Inc.
 UNIX® is a registered trademark of The Open Group in the United States and other countries.
 W3C® is a registered trademark of the Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, or Keio University on behalf of the World Wide Web Consortium.
 WS-I is a trademark of The Web Services-Interoperability Organization
 XAMPP is a registered trademark of Apache Friends
 Yahoo! is a registered trademark of Yahoo! Inc.

Other company, product or service names may be trademarks or service marks of others