Towards Software Development Methodology for Web Services

George Feuerlicht and Sooksathit Meesathit
Department of Software Engineering, Faculty of Information Technology,
University of Technology, Sydney
jiri@it.uts.edu.au
Presentation Outline

- Introduction and Motivation
- Service-Oriented Computing
- Design Issues
- Research Approaches
- Design Framework for Web Services
- Conclusions
Motivation

“What is missing is a sound design methodology and engineering principles for developing web service based applications....”

[Mike Papazoglou, 2002]
The Shifting Technology Platform

Corba

J2EE

Web Services
Version 1 Syndrome

• Most applications developed early in the technology evolution cycle suffer from poor design and maintainability
• This translates into poor reliability and high maintenance costs
• Effective system development methodology is essential
Service-Oriented Computing

- Service is the right level of abstraction for Internet-scale distributed systems - good correspondence to the real-world
- Each partner (service provider) has disparate business semantics, development cycles, administrative boundary, ...
- No control over service provider’s system, i.e. service availability, QoS, ..
- Concept of domain-specific services, e.g. travel domain (airlines, travel agents, hotels, etc)
### Web Services Platform

<table>
<thead>
<tr>
<th>e-Business Application</th>
<th>Security Transactions Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPCs/Documents/Notifications</td>
<td>WS-Security</td>
</tr>
<tr>
<td>Web Services – XML/SOAP/HTTP/WSDL</td>
<td>WS-Transaction</td>
</tr>
<tr>
<td>Partner A (J2EE)</td>
<td>WS-Coordination</td>
</tr>
<tr>
<td>Partner B (.Net)</td>
<td></td>
</tr>
</tbody>
</table>
Main Benefit of Web Services

- Standard for programmatic access over the Internet/Intranet
- Overcomes technical level heterogeneity
  - Documents can be transmitted between diverse application platforms, .Net, J2EE
  - Remote procedures (RPCs) can be executed across diverse application platforms
- Does not address information level heterogeneity (i.e. structural and semantic heterogeneity) – some level of standardization across domain of interest is needed
- Currently, Web Services are mainly used to ship documents between applications
Web Services Application Examples

- Google (www.google.com)
- Amazon (www.amazon.com)
- eBay (www.ebay.com)
- Yahoo (www.yahoo.com)
- Galileo (www.galileo.com)
- Sabre (www.sabre.com)
Research Focus

- Intra-Enterprise applications
  - Wrapping existing (legacy) applications
  - Developing new Web Services applications
- Inter-Enterprise applications (e-business)
  - Industry-wide solutions, e.g. travel, healthcare, education, etc.
- Focus on industry-wide solutions
- Focus on design, (i.e. not on analysis)
Design Issues

• Service-oriented computing just another style of distributed computing
• Design (competing) concerns
  – Performance – response time, reliability, scalability, etc.
  – Software engineering – reuse, maintainability, extensibility (i.e. evolution)
• Most current systems are optimized for performance and scalability
Document-Centric Design

- Minimize number of messages
- Avoid the need to maintain state – i.e. message is *self-contained* – contains all information to execute a business transaction, e.g. flight booking
- Improves performance and reliability, BUT:
  - Complex message structure – implements a complex business process, embedded business rules (e.g. max 4 segments)
  - Duplication of data structures, optional elements, control elements (e.g. Direct Flights Only)
  - *Poor reusability, maintainability, extensibility*
OTA Example

- Open Travel Alliance ([http://www.opentravel.org](http://www.opentravel.org)) is a consortium of airlines, travel agents, car rental companies, hotel chains, and IT companies
- Air, Car, and Hotel Working Groups that focus on different aspects of the standardization process
- Define message payloads using XML Schema
- Standard implemented by Sabre, Galileo, etc. – shipping OTA messages using Web Services
OTA Air Availability Request (AirAvailRQ)
OTA Air Availability Response (AirAvailRS)
Document-Centric Model

- Document (message) as the basic unit of business communications
- Emulates paper/fax-based business communications
- User-centric design rather than data centric, i.e. based on paper documents
- No direct correspondence to business processes (i.e. poor cohesion)
- No comprehensive design methods for document (message) structures
Document-Centric Model ...  

- Exposes complex data structures and leads high levels of coupling (i.e. interdependencies) – *stamp coupling*
- Poor stability of the interface – redundant data
- Each partner has to map documents into local transactions and resolve data semantic issues
- *Data integration problem*
Web Services Design Research

- Object-Oriented and Component-based Approaches
  - Aggregation of components [e.g. Ambler, Levi and Arsanjani, etc.]
- Transformation of Industry Domain Specifications Approaches
  - RosettaNet specifications [e.g. Masud, etc.]
- Business Process Transformation Approaches
  - Business process analysis [Papazoglou and Yang, Radeka, etc.]
Components or Messages?

- Inter-enterprise applications use Internet as the underlying transport mechanism, and are subject to a different set of design considerations than intra-enterprise applications.
- Mapping existing components directly to Web Services leads to suboptimal design and results in poor performance and scalability.
- Most practitioners recommend the use of coarse-grained, message-oriented Web Services that minimize the number of messages and avoid the need to maintain state information between invocations.
- REST (REpresentational State Transfer) vs MEST (MESsage Transfer) debate.
Where is the Interface?
(Messages vs Methods)

FlightBooking{ DocIn: OTA_Air_AvailRQ, DocOut: OTA_Air_AvailRS }

- Contract formed by In/Out (XML) documents – *Stamp Coupling*

FlightEnquiry{ In: OriginLocation, DestinationLocation, DepartureDate, Out: FlightNumber }

- Contract formed by In/Out parameters – *Data Coupling*
Interface Design

- Transforming the problem of message (document) design into service interface design allows the application of well-established design principles
- The key issue is how design the interface
- What can we learn from Object-Oriented design?
- What can we learn from API design?
- Avoid re-inventing the wheel
Design Principles

- Key requirements
  - Reuse (eliminate code and data redundancy)
  - Maintainability (support for change)
  - Extensibility (support for evolution)
  - Consistency (across domain of interest)

- Key principles
  - Orthogonality
  - Maximize cohesion (maximize modularity)
  - Minimize coupling (minimize interdependencies)
  - Standardize interfaces (avoid run-time resolution)
Cohesion and Coupling

- Cohesion as a degree of interaction within programming modules – *should be maximized*
- Coupling as the degree of interaction between programming modules – *should be minimized*
- Functional cohesion - module performs a single function
- Data coupling - individual data elements are used as parameters
- Stamp coupling - modules use data structures as parameters (not as good as data coupling)

[Myers, 1978 – Structured Design]
Software Development Framework

- Analysis
  - define scope
  - identify/develop use cases
  - identify/develop business process models
  - identify/develop data models

- Initial Design
  - identify elementary business functions
  - map elementary business functions to service operations
Software Development Framework ...

- Detailed Design
  - identify parameters for each operation
  - normalize service interfaces
  - fine-tune granularity of operations

- Implementation design
  - map interfaces to WSDL
  - specify type of operation (i.e. RPC, document style)
Software Development Framework...

- Starting point – industry specification of standard messages and business functions (e.g. OTA specification)
- Identify elementary business functions (maximize cohesion)
  - Use Case Diagrams
  - Sequence Diagrams
  - Business Function Decomposition
- Use data engineering principles to refine interface design
- *Interface design as a data engineering problem; i.e. how to allocate data parameters to specific interfaces*
## Airline Booking Service Operations

<table>
<thead>
<tr>
<th>Service Operations</th>
<th>Input Parameters</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlightEnquiry</td>
<td>OriginLocation</td>
<td>FlightNumber</td>
</tr>
<tr>
<td></td>
<td>DestinationLocation</td>
<td>DepartureAirport</td>
</tr>
<tr>
<td></td>
<td>DepartureDate</td>
<td>DepartureTime</td>
</tr>
<tr>
<td>SeatEnquiry</td>
<td>FlightNumber</td>
<td>ArrivalAirport</td>
</tr>
<tr>
<td></td>
<td>DepartureAirport</td>
<td>ArrivalDate</td>
</tr>
<tr>
<td></td>
<td>ArrivalAirport</td>
<td>ArrivalTime</td>
</tr>
<tr>
<td></td>
<td>DepartureDate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CabinType</td>
<td></td>
</tr>
<tr>
<td>PriceEnquiry</td>
<td>FlightNumber</td>
<td>FareBasisCode</td>
</tr>
<tr>
<td></td>
<td>DepartureAirport</td>
<td>BaseFare</td>
</tr>
<tr>
<td></td>
<td>ArrivalAirport</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DepartureDate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CabinType</td>
<td></td>
</tr>
</tbody>
</table>
### Airline Booking Service Operations...

<table>
<thead>
<tr>
<th>Service Operations</th>
<th>Input Parameters</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BookFlight</td>
<td>FlightNumber, DepartureAirport, DepartureDate, TravelerName, CabinType</td>
<td>BookingReferenceID</td>
</tr>
<tr>
<td>SeatingRequest</td>
<td>BookingReferenceID, SeatPreference</td>
<td>SeatNumber</td>
</tr>
<tr>
<td>MealRequest</td>
<td>BookingReferenceID, MealPreference</td>
<td>MealType</td>
</tr>
</tbody>
</table>
Interface Design Guidelines

- Maximize cohesion (Functional Cohesion)
  - all input parameters must be used by the method as data, i.e. not to control the execution of the method
  - input parameters must generate output parameters (parameter dependency)

- Minimize coupling (Data Coupling via parameters)
  - data elements used as parameters
  - input and output parameter sets must be minimal
  - matching output parameters of one method to input parameters of another
Interface Normalization

- Rule 1: input parameters must be mutually independent (i.e. should form a minimal set)
- Rule 2: output parameters must be mutually independent (i.e. should form a minimal set)
- Rule 3: output parameters must be fully functionally dependent on input parameters
  [Rule 3 applies to query methods only]
Data Engineering Interfaces

- Dependencies are used to eliminate redundant parameters, e.g.:

  FD1: FlightNumber => DepartureAirport, DepartureTime, ArrivalAirport, ArrivalTime

Dependency FD1 can be used to eliminate:
DepartureAirport and ArrivalAirport from SeatEnquiry and PriceEnquiry
Functional Dependencies

FD1: OriginLocation, DestinationLocation, DepartureDate
    => FlightNumber

FD2: FlightNumber
    => DepartureAirport, DepartureTime, ArrivalAirport, ArrivalTime

FD3: FlightNumber, DepartureDate
    => ArrivalDate

FD4: FlightNumber, DepartureDate, CabinType
    => Quantity

FD5: FlightNumber, DepartureDate, CabinType
    => BasicFareCode, BasicFare
## Normalized Interfaces

<table>
<thead>
<tr>
<th>Service Operations</th>
<th>Input Parameters</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FlightEnquiry</strong></td>
<td>OriginLocation, DestinationLocation, DepartureDate</td>
<td>FlightNumber</td>
</tr>
<tr>
<td><strong>ScheduleEnquiry</strong></td>
<td>FlightNumber</td>
<td>DepartureAirport, DepartureTime, ArrivalAirport, ArrivalTime</td>
</tr>
<tr>
<td><strong>ArrivalEnquiry</strong></td>
<td>FlightNumber, DepartureDate</td>
<td>ArrivalDate</td>
</tr>
<tr>
<td><strong>SeatEnquiry</strong></td>
<td>FlightNumber, DepartureDate, CabinType</td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>PriceEnquiry</strong></td>
<td>FlightNumber, DepartureDate, CabinType</td>
<td>FareBasisCode, BaseFare</td>
</tr>
</tbody>
</table>
### Normalized Interfaces ...

<table>
<thead>
<tr>
<th>Service Operations</th>
<th>Input Parameters</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BookFlight</td>
<td>FlightNumber, DepartureDate, TravelerName, CabinType</td>
<td>BookingReferenceID</td>
</tr>
<tr>
<td>SeatingRequest</td>
<td>BookingReferenceID, SeatPreference</td>
<td>SeatNumber</td>
</tr>
<tr>
<td>MealRequest</td>
<td>BookingReferenceID, MealPreference</td>
<td>SpecialMealType</td>
</tr>
</tbody>
</table>

**New operations:** ScheduleEnquiry, ArrivalEnquiry added to maintain completeness
Adjusting Granularity of Operations

- Increasing number of service operations increases network traffic and interaction complexity
- Combining operations based on input parameters:

<table>
<thead>
<tr>
<th>Service Operations</th>
<th>Input Parameters</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeatPriceEnquiry</td>
<td>FlightNumber, DepartureDate, CabinType</td>
<td>Quantity, FareBasisCode, BaseFare</td>
</tr>
</tbody>
</table>
### Adjusting Granularity of Operations ..

<table>
<thead>
<tr>
<th>Service Operations</th>
<th>Input Parameters</th>
<th>Output Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeatingMealRequest</td>
<td>BookingReferenceID</td>
<td>SeatNumber</td>
</tr>
<tr>
<td></td>
<td>SeatPreference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MealPreference</td>
<td>SpecialMealType</td>
</tr>
<tr>
<td>Implementation: RPC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Combining operations reduces cohesion and results in non-homogeneity, i.e. some parameters do not take values, e.g. requesting a special meal, but not specifying seating preference.
WSDL Specification

- Map services to WSDL service definitions
- Map operations to WSDL operations
- Map input and output parameters to WSDL input and output messages
- Define binding style: document or RPC
Conclusions

- Document-centric approach provides only limited interoperability and scalability
- Service-Oriented approach based on Web Services technologies provides an opportunity to overcome limitations of earlier e-business approaches (e.g. EDI, ebXML, etc.)
- *Standardized, stable, and well-designed Web Services interfaces are needed to obtain the full benefit of the Service-Oriented computing*
Industry-Wide APIs

- e-business Application
- Industry-wide API
- RPCs/Documents/Notifications
- Core Web services - SOAP/HTTP/WSDL
- Partner A (J2EE)
- Partner B (.Net)
Travel Application APIs

- Hertz Car Hire
  - CarAvail
  - CarPrice
  - CarBook
  - CarCancel
- Sheraton Hotel
  - HotelAvail
  - HotelPrice
  - HotelBook
  - HotelCancel
- Travel.com.au
- KLM
  - AirAvail
  - AirPrice
  - AirBook
  - AirlItinerary
  - AirCancel
- Qantas
  - AirAvail
  - AirPrice
  - AirBook
  - AirlItinerary
  - AirCancel

Copyright © GF2005  SoMet 2005 – Tokyo Sept 2005
First International Workshop on Design of Service-Oriented Applications  
(WDSOA'05: http://elab.njit.edu/wdsoa05)  
3rd International Conference on Service Oriented Computing (ICSOC 2005: www.icsoc.org)  
December 12, 2005, Amsterdam, The Netherlands

• Design methods and guidelines for developing service-oriented applications  
• Design of industry-domain Web Services  
• Design of Web Services for enterprise applications  
• Methods for transforming component-based applications to Web Services  
• Design approaches for Web Services composition  
• Web Services design and interoperability case studies  
• Analysis and modeling approaches for service-oriented applications  
• Challenges and future directions for design of service-oriented applications