

Middleware and Web Services

Lecture 6: Service Concepts

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • <http://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <http://vitvar.com/courses/mdw>



Modified: Mon Oct 27 2014, 22:03:01
Humla v0.3

Service Oriented Architecture



- SOA supports two core business strategies
 - Growing top-line revenue
 - Enterprise reacts quickly to requirements from the market
 - Business processes can be reconfigured rather than reimplemented
 - Improving bottom-line profit
 - Saving development costs by reusing existing services
- Pre-integrated solutions
 - Out-of-the-box applications and integration solutions among them

Overview

- Integrating Applications
- Web Service Architecture

Integration and Interoperability

- Integration
 - *A process of connecting applications so that they can exchange and share capabilities, that is — information and functionalities.*
 - *Includes methodological approaches as well as technologies*
- Interoperability
 - *Ability of two or more applications to understand each other*
 - *Interoperability levels*
 - *Data – syntax/structure and semantics*
 - *Functions/Processes – syntax and semantics*
 - *Technical aspects – protocols, network addresses, etc.*

Integration Approaches Overview



Data-oriented Integration



- Third-party database access
 - Application D accesses a database of application B directly by using SQL and a knowledge of database B structure and constraints
 - In the past: monolithic and two-tier client/server architectures
 - Today: ETL (Extract, Transform, Load) technologies
- Problems
 - App D must understand complex structures and constraints
 - Data – very complex, includes structure and integrity constraints
 - Functions/processes – hidden in integrity constraints
 - Technical – access mechanisms can vary

Service-oriented Integration



- Integration at the application layer
 - Application exposes services that other applications consume
 - Services hide implementation details but only define interfaces for integration
- Problems
 - Can become unmanageable if not properly designed
 - Interoperability
 - Data – limited to input and output messages only
 - Functions/processes – limited to semantics of services
 - Technical – access mechanisms can vary

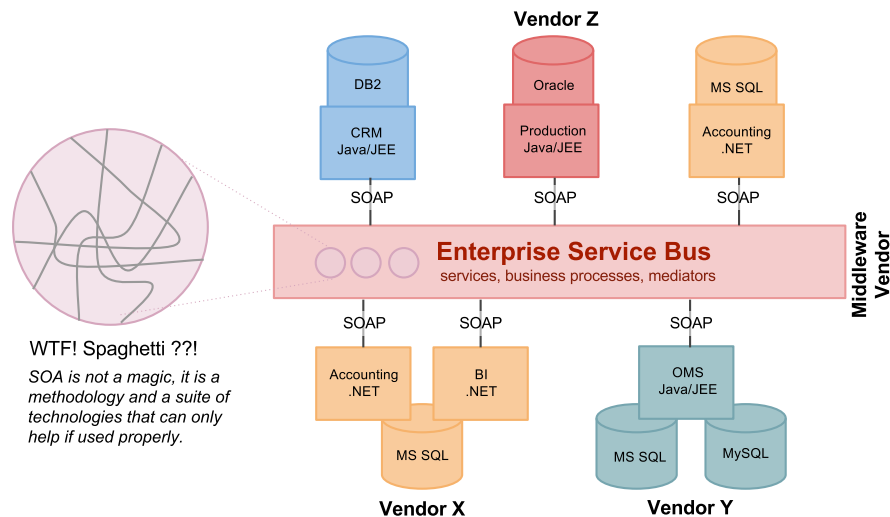
One-to-One Service Integration

- Direct integration of applications
 - Multiple protocols problem, multiple vendor problem
 - Replication of integration functionalities such as interoperability solutions



Many-to-Many Service Integration

- Enterprise Service Bus – central integration technology
 - Realizes so called *Service Oriented Architecture (SOA)*
 - Contains various integration components such as process server, mediators, messaging middleware, identity management, etc.



Integration and Types of Data

- Transactional data – Web services
 - *Service-oriented integration*
 - *online, realtime communication between a client and a service*
 - *Usually small amount of data and small amount of service invocation in a process*
- Bulk data – ETL
 - *Data-oriented integration*
 - *processing of large amount of data in batches*
- **ESB provides both Web service and ETL capabilities**

Overview

- Integrating Applications
- **Web Service Architecture**
 - *Definition of a Service*
 - *Service Interface Components*

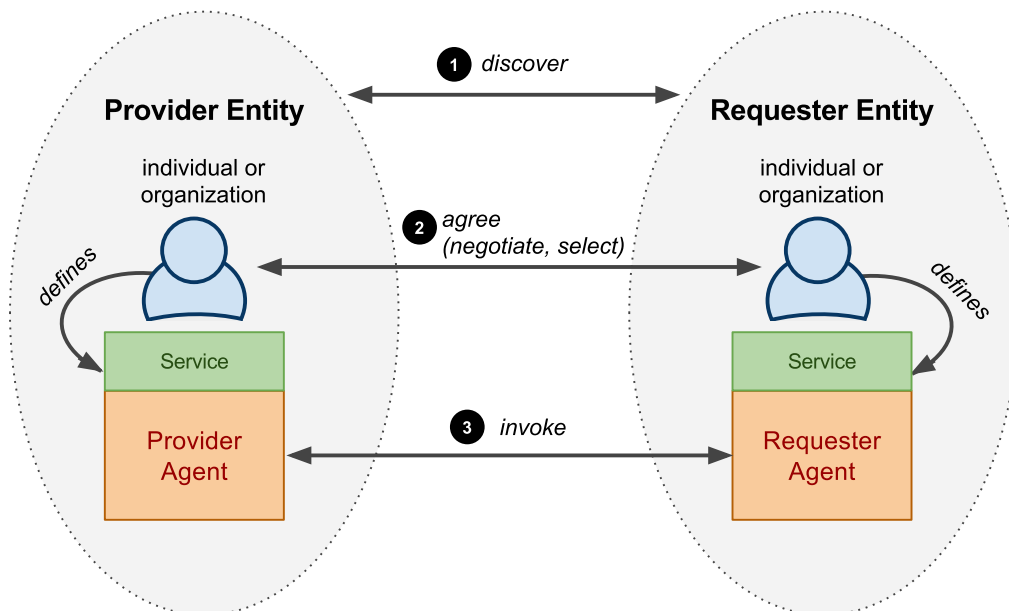
Web Service Architecture

- Web Service Architecture
 - *Defined by W3C in Web Service Architecture Working Group Note* [🔗](#)
 - *Defines **views***
 - *message-oriented view (WSDL and SOAP)*
 - *resource-oriented view (REST and HTTP)*
 - *Defines **architecture entities** and their **interactions***
 - *Abstraction over underlying technology*
 - *Basis for service usage processes and description languages*
- Service Oriented Architecture
 - *Collection of tools, methods and technologies*
 - *There is some implicit understanding of SOA in the community such as*
 - *SOA is a solution for Enterprise Application Integration*
 - *SOA is realized by using SOAP, WSDL, (and UDDI) technologies*
 - *SOA utilizes Enterprise Service Bus (ESB)*
 - ⇒ *~ a realization of Web Service Architecture message-oriented view*

Basic Entities

- **Agent**
 - *software or hardware that sends/receives messages*
 - *concrete implementation of a service*
- **Service**
 - *abstract set of functionality and behavior*
 - *two different agents may realize the same service*
- **Provider**
 - *owner (person or organization) that provides an agent realizing a service*
 - *also called a service provider*
- **Requester**
 - *a person or organization that wishes to make use of a provider's service*
 - *uses a requester's agent to exchange messages with provider's agent*

Interaction of Entities



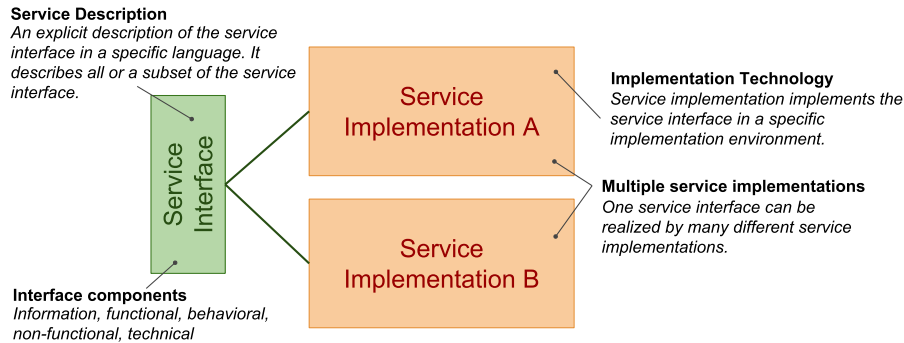
Overview

- Integrating Applications
- Web Service Architecture
 - *Definition of a Service*
 - *Service Interface Components*

Service

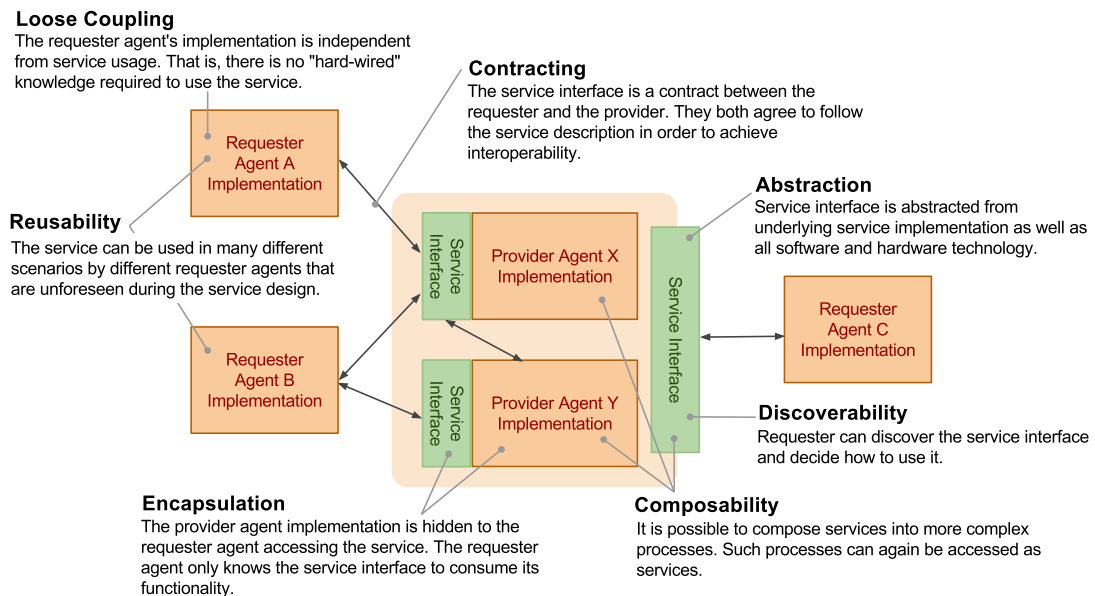
- Difficult to agree on one definition
- Business definition
 - *A service realizes an effect that brings a business value to a service consumer*
 - *for example, to pay for and deliver a book*
- Conceptual definition
 - *service characteristics*
 - *encapsulation, reusability, loose coupling, contracting, abstraction, discoverability, composability*
- Logical definition
 - *service interface, description and implementation*
 - *service usage process*
 - *service use tasks, service types*

Interface, Description and Implementation



- Terminology clarification
 - *service ~ service interface + service implementation*
 - *WSDL service ~ service description in WSDL language*
 - *SOAP service ~ a service interface is possible to access through SOAP protocol; there is a WSDL description usually available too.*
 - *REST/RESTful service ~ service interface that conforms to REST architectural style and HTTP protocol*

Service Characteristics



Service Description

- Standards-driven
 - *Standards that define service description*
 - *They give a space for variability*
 - *too much – big flexibility but increases complexity (~WSDL)*
 - *limited – enforce agreement and interoperability (~REST)*
(as long as parties correctly implement the standard)
- Languages to describe service interfaces
 - *formal – machine processable*
 - *textual – natural text description*
- Comparison of WSDL and REST models for service interfaces

Model	Standards-driven	Languages
WSDL	XML-based WSDL, XML Schema for input/output/fault messages; big space for variations (operations, exchange patterns, protocols)	WSDL+XML, textual description for rules of public processes
REST	Web Architecture, HTTP, XML Schema, JSON; little space for variations (uniform interface, statelessness, etc.)	HTML – mostly textual description, AtomPub, WADL

Overview

- Integrating Applications
- Web Service Architecture
 - *Definition of a Service*
 - *Service Interface Components*

Service Interface

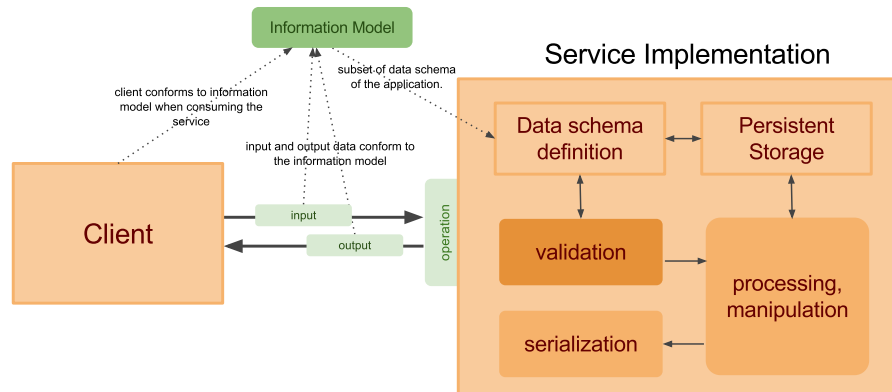
- They correspond to interface definition
- Service interface components
 - *Information*
 - data used by the service
 - for example, input and output messages, resource representations
 - *Functional*
 - capability: operations, preconditions, effects
 - pointer to a classification hierarchy
 - *Behavioral*
 - public process: how to consume the service's functionality
 - orchestration: realization of service's functionality
 - *Non-Functional*
 - security, financial, descriptive info (author, date)
 - *Technical*
 - technical details such as IP addresses, ports, protocols, etc.

Running Example

- Textual service description
 - *Service name: Order Book Service*

```
1  * the service provides three operations: 'open', 'add', 'close'
2  * operation 'open' opens the order
3    - input: none
4    - output: text informing that the order was opened
5
6  * operation 'add' adds an item to the order
7    - input: an item name, the syntax is [0-9A-Za-z\-\-]+
8    - output: text informing that the item was added to the order
9
10 * operation 'close' closes the order and returns all items in the order
11   - input: none
12   - output: list of all items previously added to the order
13
14 * the public process is: S0--open--S1, S1--add--S1, S1--close--S0, where
15   S0, S1 are states such that S0 = order is closed, and S1 = order is opene
16
17 * protocol is HTTP, method POST for all operations,
18   running locally, tcp/8080, stateful server
```
- Service implementation
 - Will go through the code in Java
 - Will use the *session object* (see Lecture 3) for the state management

Information Component



- Information Component
 - Defines models for all data used by the service as input/output messages, states
 - Data in formats mostly XML and JSON or plain text (our example)
 - Languages: XML Schema, or other—regular grammars or plain text.
- Tasks
 - Validation – check the syntax and validates the data against rules
 - Processing and manipulation – process and manipulates the data
 - Serialization – transforms the data to transportation formats (XML, JSON, text)

Example

- Description

```
1 | ...
2 | * operation 'add' adds an item to the order
3 |   - input: an item name, the syntax is [0-9A-Za-z\-\-]+
4 |   - output: text informing that the item was added to the order
5 | ...
```

- Service implementation

- Validation – syntax checking

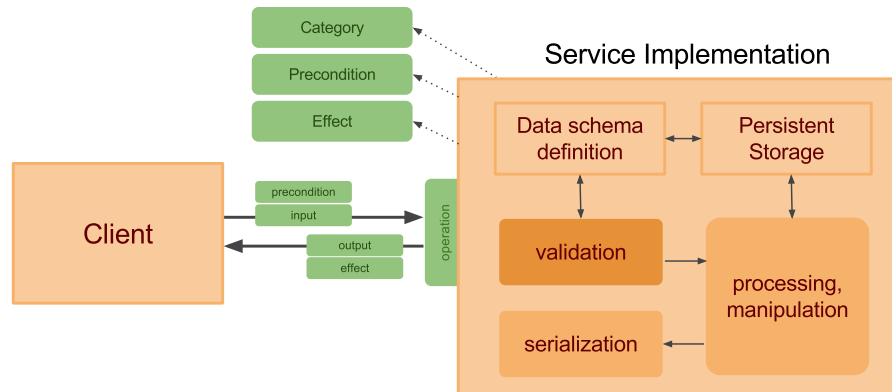
```
1 | // check the syntax of the item name
2 | if (item.matches("[a-zA-Z0-9\\-]+"))
3 |     // ... process operation
4 |     return "Item added.";
5 | else
6 |     throw new Exception("Invalid item name.");
```



Tasks

- Describe a complex item using XML Schema and learn how to validate it in Java.

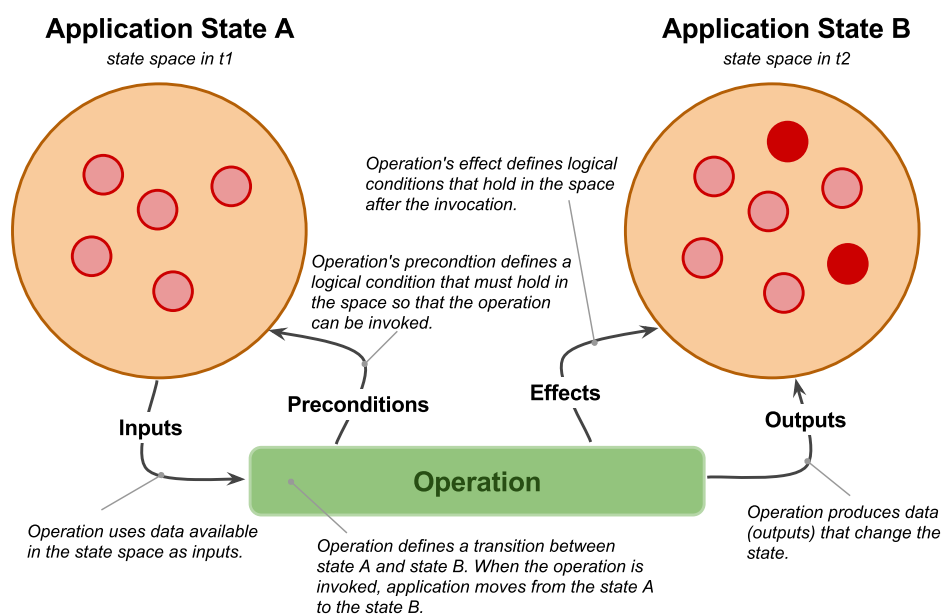
Functional Component



- **Functional component**
 - Service has a set of operations
 - each operation has input and output data from the information model
 - each operation has a capability (a precondition and an effect)
 - Service has a functional category – pointer to a classification hierarchy
 - Service has a capability (a precondition and an effect)
- **Tasks**
 - Validation – checks a precondition holds in a state before processing

Preconditions and Effects

- **Preconditions and effects on an operation**
(Note that preconditions and effects on a service are analogical)



Example

- Description

```
1 * the service provides three operations: 'open', 'add', 'close'
2 ...
3 * the public process is: S0--open--S1, S1--add--S1, S1--close--S0, where
4   S0, S1 are states such that S0 = order is closed, and S1 = order is opened.
5   ...
```

- ⇒ *There is an order of operations such that*
 - before invoking **add**, the client must invoke **open**
- ⇒ *operation **add** has*
 - precondition **order.isOpen()==true**
 - effect **item** in **order.items**

```
1 if (order.isOpen()) {
2     // ...
3     order.getItems().add(item);
4     return "Item added.";
5 } else
6     throw new Exception("An order must be opened before adding the item!");
```

Functional Category Example

- Classification schema
 - Describes taxonomy of services (outside of service interface)
 - Functional category points to a term in the taxonomy

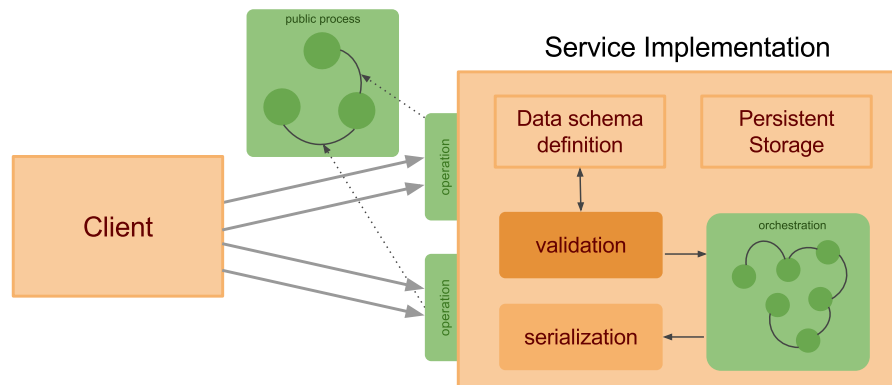
- Example

- Classification schema in XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <root xmlns="http://example.org/service-classification-schema">
3     <order>
4         <book>
5             <adventure/>
6             <travel/>
7         </book>
8         <electronics>
9             <TV/>
10            <computer/>
11        </electronics>
12    </order>
13    <shipment><!-- shipment services -->
14 </root>
```

- functional category as XPath expression: **/root/order/book**
- implicit assumption: XML hierarchy is a **sub-class-of** hierarchy

Behavioral Component



- Behavioral component
 - *public process*
 - order of operations for the correct functionality consumption
 - can be derived from preconditions of service operations
 - Note that not all operations must participate in the public process
 - A service can have more than one public processes
 - *orchestration*
 - How service's functionality is composed out of other services

Example

- **processOrder** method implements a public process

```
1 public String processOrder(String op, String item, SessionData sessionData) throws Except
2     if (op.equals("open")) {
3         if (sessionData.isOpen())
4             throw new Exception("Order was already open");
5         else {
6             sessionData.open();
7             return "The new order has been opened";
8         }
9     }
10    if (op.equals("add")) {
11        if (sessionData.isOpen()) {
12            if (item.matches("[a-zA-Z0-9\\-]+"))
13                sessionData.getItems().add(item);
14            else throw new Exception("Invalid item name.");
15            return "Item added.";
16        } else
17            throw new Exception("An order must be opened before adding the item!");
18    }
19    if (op.equals("close")) {
20        if (sessionData.isOpen()) {
21            String response = "The order has been closed, the ordered items are:\n";
22            for (String i : sessionData.getItems())
23                response += " " + i + "\n";
24            return response;
25        } else
26            throw new Exception("Cannot close an order that has not been opened!");
27    }
28    throw new Exception("Invalid operation: " + op);
29 }
```

Example (Cont.)

- RequestHandler implementation

```
1 public void handleRequest(HttpServletRequest request,
2     HttpServletResponse response) throws IOException, ServletException {
3     // get the session id and create the new session data if none exist
4     String sid = sessions.getSessionID(request);
5     SessionData sessionData = sessions.getData(sid);
6     if (sessionData == null) {
7         sessionData = new SessionData();
8         sessions.setData(sid, sessionData);
9     }
10
11     try {
12         if (request.getMethod().equals("POST")) {
13             String responseText = processOrder(request.getParameter("op"),
14                 request.getParameter("item"), sessionData);
15             response.setStatus(200);
16             response.setHeader("cookie", "session-id="+sid);
17             response.setContentType("text/plain");
18             response.getWriter().write(responseText);
19         } else {
20             response.setStatus(405); // Method not allowed
21             response.setHeader("Allow", "POST");
22         }
23     } catch (Exception e) {
24         response.setStatus(400); // client-side error
25         response.setContentType("text/plain");
26         response.getWriter().write(e.getMessage());
27     }
28     response.flushBuffer();
29 }
```

Evaluation

- How "good" is our Order service?
 - *Analysis of the service by service characteristics.*

Principle	+/-	Comment
Loose Coupling	+	Uses standard response codes.
	–	Unforeseen clients will have to know the service's public process to work with it.
	–	Uses operation names that clients must understand.
Reusability	–	Can be reused but is subject to loose coupling issues.
Contracting and Discoverability	–	Textual description is informal, it is hard to agree on the service interface.
Composability		N/A
Abstraction	+	Service description can be implemented by various implementation technologies.
Encapsulation	+	Distinguishes interface from implementation, processing logic is not exposed to clients through the interface.