Middleware and Web Services

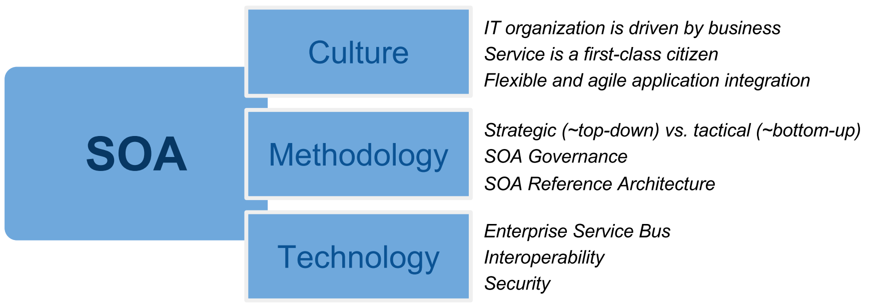
Lecture 6: Service Concepts

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: Tue Oct 28 2014, 00:03:01
Humla v0.3

Service Oriented Architecture


SOA
Culture
IT organization is driven by business
Service is a first-class citizen
Flexible and agile application integration
Methodology
Strategic (~top-down) vs. tactical (~bottom-up)
SOA Governance
SOA Reference Architecture
Technology
Enterprise Service Bus
Interoperability
Security

- 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 resuing existing services*
- Pre-integrated solutions
  – *Out-of-the-box applications and integration solutions among them*

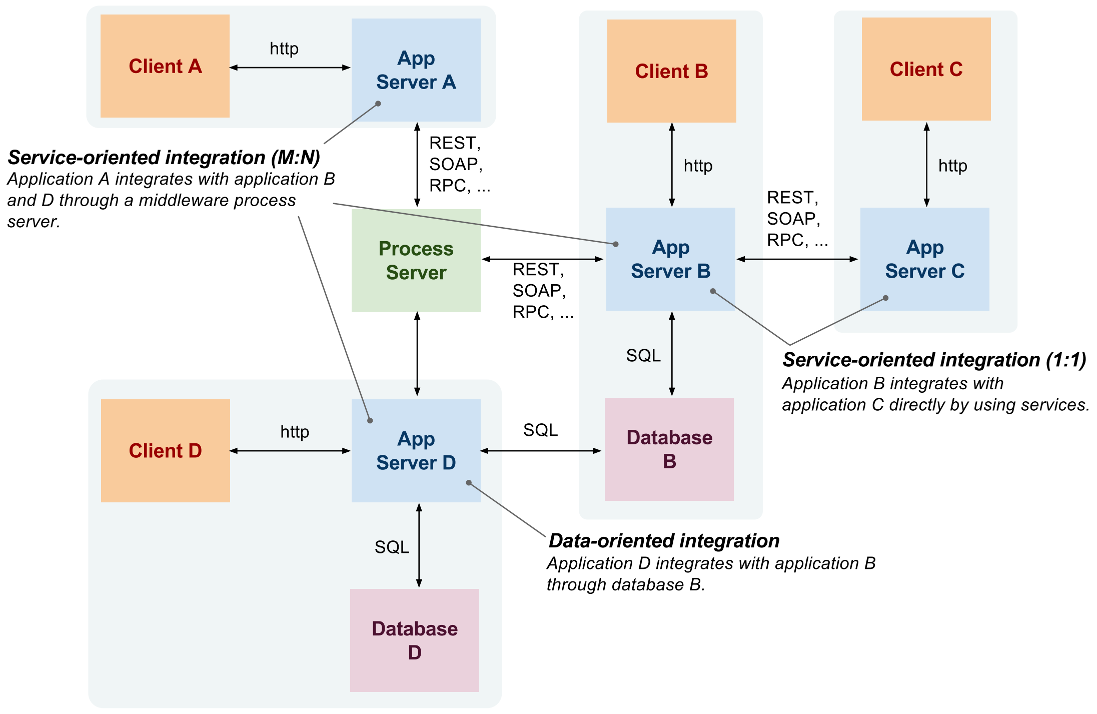
Overview

- Integrating Applications
- Web Service Architecture

Integration and Interoperability

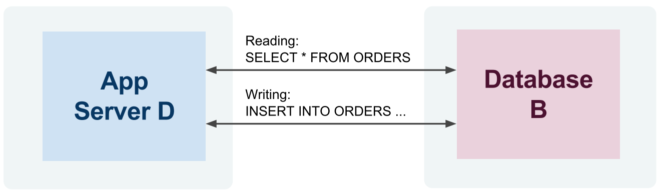
- Integration
  – *A process of connecting applications so that they can exchange and share capabilities, that is — information and functionalities.*
  – *Includes methodological approaches as well as technologies*
- Interoperability
  – *Ability of two or more applications to understand each other*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

Integration Approaches Overview



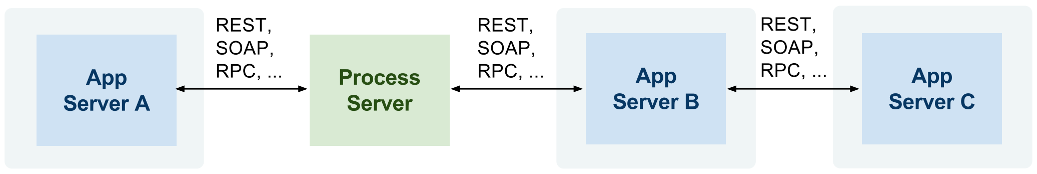
Client A
http
App Server A
REST, SOAP, RPC, ...
Service-oriented integration (M:N)
Application A integrates with application B and D through a middleware process server.
Client B
http
Client C
http
REST, SOAP, RPC, ...
Process Server
REST, SOAP, RPC, ...
App Server B
App Server C
SQL
Service-oriented integration (1:1)
Application B integrates with application C directly by using services.
Client D
http
App Server D
SQL
Database B
SQL
Data-oriented integration
Application D integrates with application B through database B.
Database D

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*

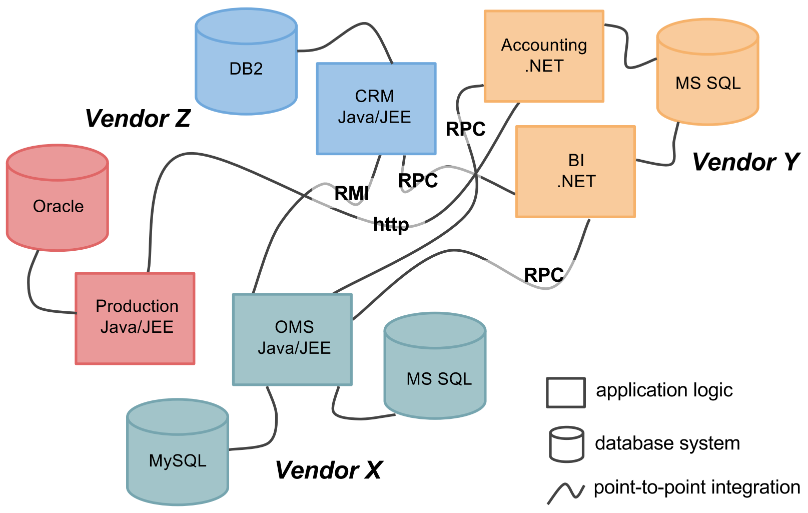
Service-oriented Integration



- 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*

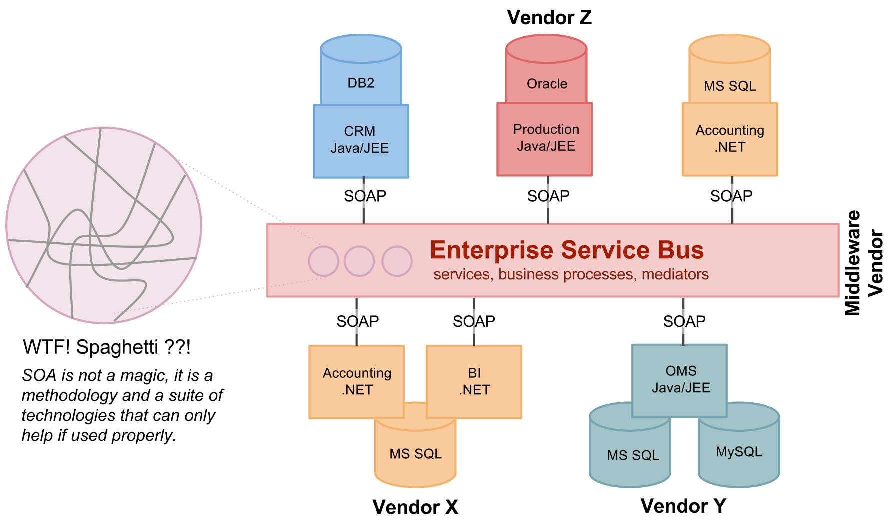
One-to-One Service Integration

- 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*

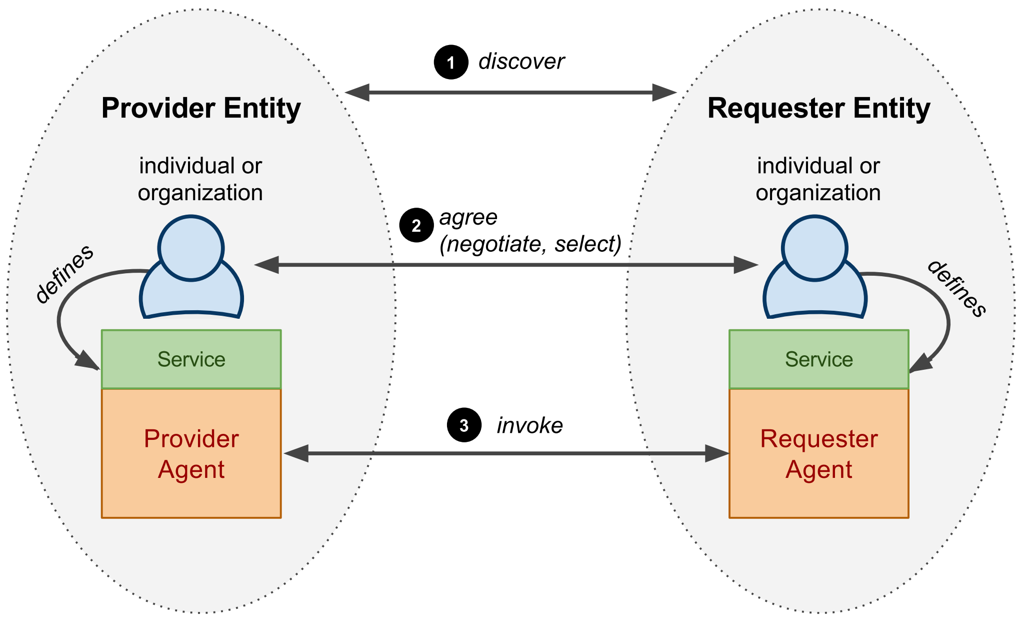
Web Service Architecture

- 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

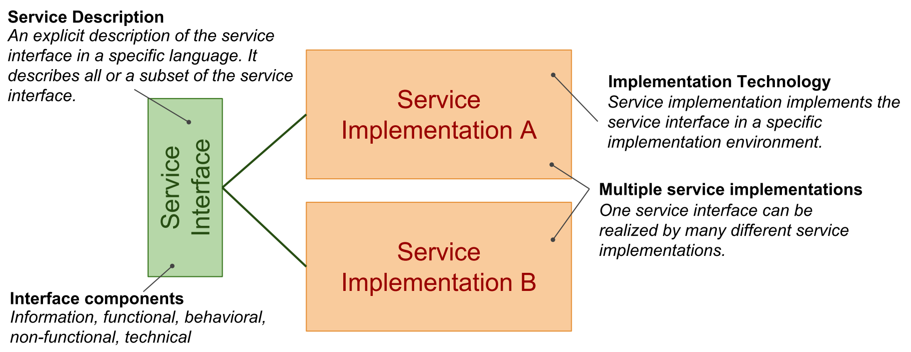
- Difficult to agree on one definition
- Business definition
  - *A service realizes an effect that brings a business value to a service consumer*

    → *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*

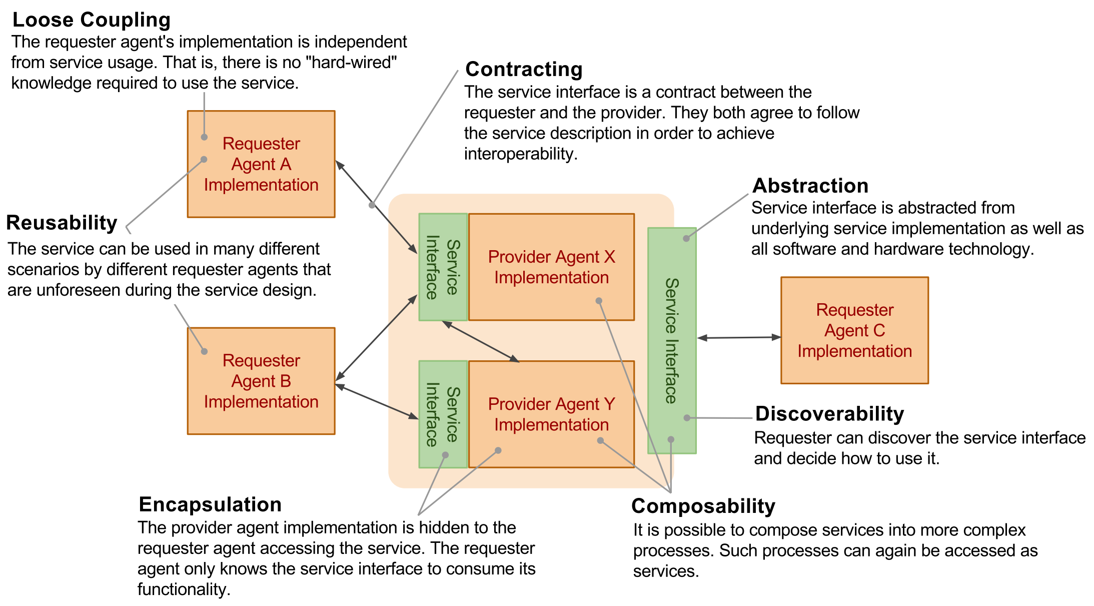
Interface, Description and Implementation



Service Description
An explicit description of the service interface in a specific language. It describes all or a subset of the service interface.
Service Interface
Service Implementation A
Service Implementation B
Implementation Technology
Service implementation implements the service interface in a specific implementation environment.
Multiple service implementations
One service interface can be realized by many different service implementations.
Interface components
Information, functional, behavioral, non-functional, technical

- 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



Loose Coupling
The requester agent's implementation is independent from service usage. That is, there is no "hard-wired" knowledge required to use the service.
Contracting
The service interface is a contract between the requester and the provider. They both agree to follow the service description in order to achieve interoperability.
Requester Agent A Implementation
Reusability
The service can be used in many different scenarios by different requester agents that are unforeseen during the service design.
Abstraction
Service interface is abstracted from underlying service implementation as well as all software and hardware technology.
Service Interface
Provider Agent X Implementation
Service Interface
Requester Agent C Implementation
Requester Agent B Implementation
Service Interface
Provider Agent Y Implementation
Discoverability
Requester can discover the service interface and decide how to use it.
Encapsulation
The provider agent implementation is hidden to the requester agent accessing the service. The requester agent only knows the service interface to consume its functionality.
Composability
It is possible to compose services into more complex processes. Such processes can again be accessed as services.

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.*

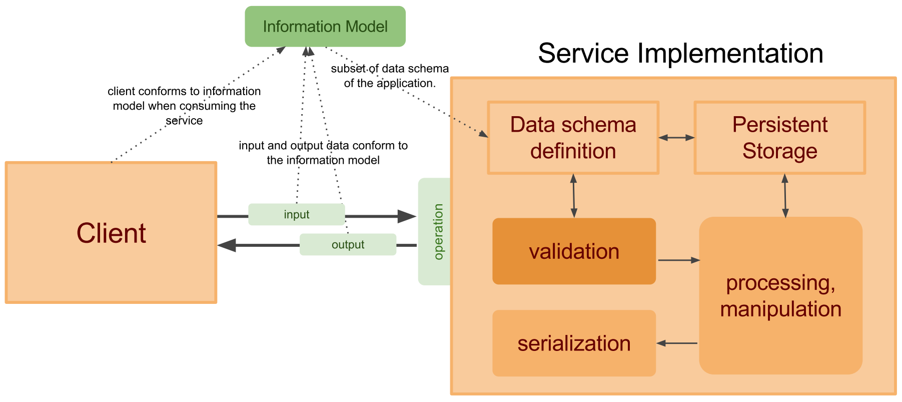
Running Example

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

```
* the service provides three operations: 'open', 'add', 'close'
* operation 'open' opens the order
    - input: none
    - output: text informing that the order was opened

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

* operation 'close' closes the order and returns all items in the order
    - input: none
    - output: list of all items previously added to the order

* the public process is: S0--open--S1, S1--add--S1, S1--close--S0, where
  S0, S1 are states such that S0 = order is closed, and S1 = order is opene

* protocol is HTTP, method POST for all operations,
  running localy, 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 Model
Service Implementation
subset of data schema of the application.
client conforms to information model when consuming the service
input and output data conform to the information model
Client
input
output
operation
Data schema definition
Persistent Storage
validation
processing, manipulation
serialization

- 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

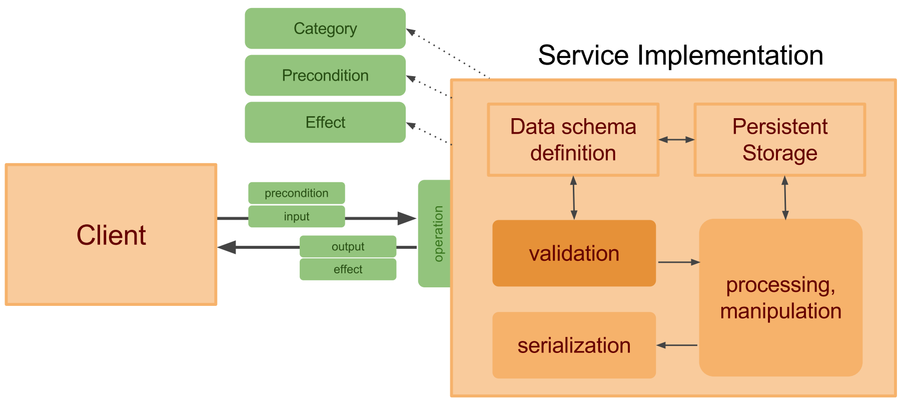
```
...
* operation 'add' adds an item to the order
    - input: an item name, the syntax is [0-9A-Za-z\-]+
    - output: text informing that the item was added to the order
...
```

- Service implementation
  - *Validation – syntax checking*

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

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

Functional Component

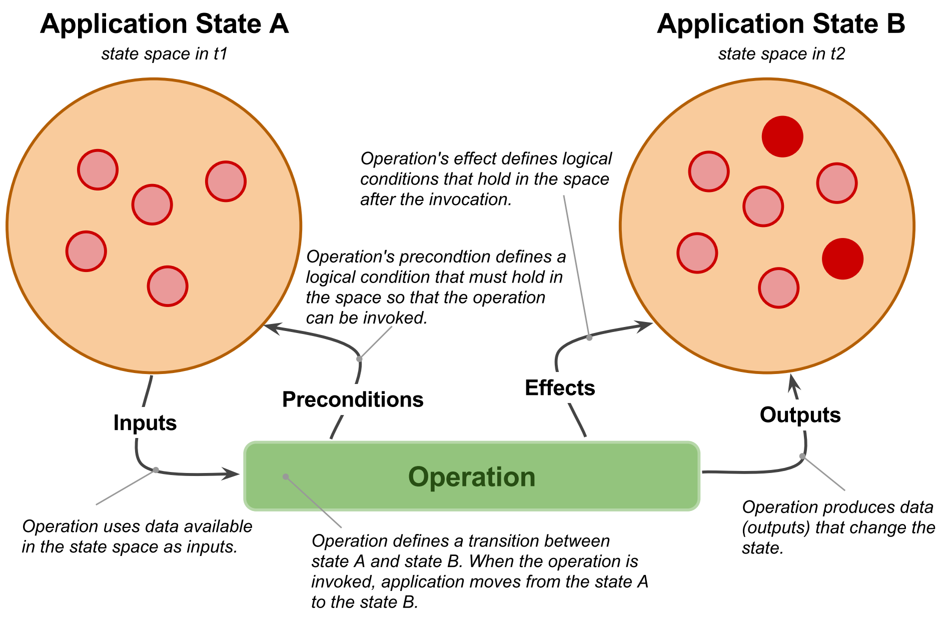


- Functional component
  - *Service has a set of operations*
    - *→ each operation has input and output data from the information model*
    - *→ each operation has a capability (a precondition and an effect)*
  - *Service has a functional category – pointer to a classification hierarchy*
  - *Service 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)


Application State A
state space in t1
Application State B
state space in t2
Operation's effect defines logical conditions that hold in the space after the invocation.
Operation's precondtion defines a logical condition that must hold in the space so that the operation can be invoked.
Preconditions
Effects
Inputs
Outputs
Operation
Operation uses data available in the state space as inputs.
Operation defines a transition between state A and state B. When the operation is invoked, application moves from the state A to the state B.
Operation produces data (outputs) that change the state.

Example

- Description

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

⇒ *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`

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

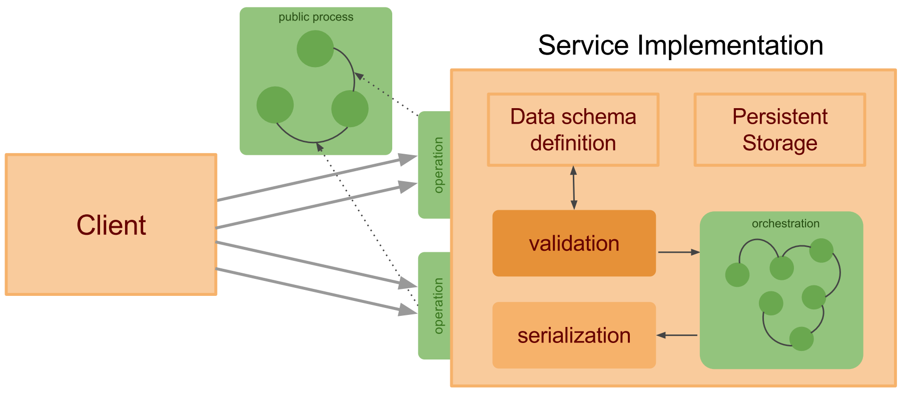
Functional Category Example

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="http://example.org/service-classification-schema">
    <order>
        <book>
            <adventure/>
            <travel/>
        </book>
        <electronics>
            <TV/>
            <computer/>
        </electronics>
    </order>
    <shipment><!-- shipment services -->
</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

```java
public String processOrder(String op, String item, SessionData sessionData) throws Excep
    if (op.equals("open")) {
        if (sessionData.isOpen())
            throw new Exception("Order was already open");
        else {
            sessionData.open();
            return "The new order has been opened";
        }
    }
    if (op.equals("add")) {
        if (sessionData.isOpen()) {
            if (item.matches("[a-zA-Z0-9\\-]+"))
                sessionData.getItems().add(item);
            else throw new Exception("Invalid item name.");
            return "Item added.";
        } else
            throw new Exception("An order must be openned before adding the item!");
    }
    if (op.equals("close")) {
        if (sessionData.isOpen()) {
            String response = "The order has been closed, the ordered items are:\n";
            for (String i : sessionData.getItems())
                response += "  " + i + "\n";
            return response;
        } else
            throw new Exception("Cannot close an order thas has not been openned!");
    }
    throw new Exception("Invalid operation: " + op);
}
```

Example (Cont.)

- `RequestHandler` implementation

```java
public void handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
    // get the session id and create the new session data if none exist
    String sid = sessions.getSessionID(request);
    SessionData sessionData = sessions.getData(sid);
    if (sessionData == null) {
        sessionData = new SessionData();
        sessions.setData(sid, sessionData);
    }

    try {
        if (request.getMethod().equals("POST")) {
            String responseText = processOrder(request.getParameter("op"),
                    request.getParameter("item"), sessionData);
            response.setStatus(200);
            response.setHeader("cookie", "session-id="+sid);
            response.setContentType("text/plain");
            response.getWriter().write(responseText);
        } else {
            response.setStatus(405); // Method not allowed
            response.setHeader("Allow", "POST");
        }
    } catch (Exception e) {
        response.setStatus(400); // client-side error
        response.setContentType("text/plain");
        response.getWriter().write(e.getMessage());
    }
    response.flushBuffer();
}
```

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 throught the interface.