Content

- SOA
- Web Services
- Dataflow/Workflow
  - BEPL
- Services & Components
  - SCA
  - STCM
SOA – Service Orientated Architecture

“This term is increasingly used to refer to an architectural style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services.”

OGSA Glossary
Publish, Find and Bind Triangle

Service Oriented Architecture

Characteristics of SOA – WS-Architecture

- **Logical view**: The service is an abstracted, *logical* view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.
- **Message orientation**: The service is formally defined in terms of the messages exchanged between provider and requester agents, and not the properties of the agents themselves.
- **Description orientation**: A service is described by machine-processable meta data.
Characteristics of SOA – WS-Architecture

- **Granularity**: Services tend to use a small number of operations with relatively large and complex messages.

- **Network orientation**: Services tend to be orientated toward use over a network.

- **Platform neutral**: Messages are sent in a platform-neutral, standardized format delivered through the interfaces, XML is the most obvious format that meets this constraint.

SOA References


- **OGSA Glossary** - [https://forge.gridforum.org/projects/ogsa-wg](https://forge.gridforum.org/projects/ogsa-wg)

- For discussions on SOA see:
  - [http://savas.parastatidis.name/](http://savas.parastatidis.name/)
  - [http://jim.webber.name/](http://jim.webber.name/)
  - WS-GAF mailing list
Why Web Services

- Execute everywhere
  - Multi-platform
  - Multi-languages
- From everywhere
- Through everything
  - In particular, through firewalls
### What is needed?

- **To execute on a platform**
  - Description language
    - Interfaces
    - Data types
  - Mappings to/from programming languages
- **To execute remotely**
  - A service of naming or discovery
  - A communication protocol

### Web Services: Elements

- **WSDL**
  - Web Services Description Language
- **SOAP**
  - Simple Object Access Protocol
- **XML**
  - eXtended Markup Language
- **UDDI**
  - Universal Description, Discovery, and Integration
Web Services: Architecture

- **UDDI** (Universal Description, Discovery and Integration)
- **WSDL** (Web Services Description Language)
- **Library**
  - Java - SOAP for Apache
  - Java - GLUE
  - Perl - SOAP::Lite
  - C/C++ - gSOAP
  - Python - ZSI
  - Microsoft SOAP (part of .NET)
- **XML Document to SOAP Specifications**
- **Server**
  - Executes Request and Generates Response
  - HTTP
- **Client**
  - Calls
  - Receives Responses
  - HTTP

SOAP [Simple Object Access Protocol]

- Provides
  - RPC
  - User Defined Data Types
  - Localization (English, Chinese, etc.)
- Uses widely adopts standards
  - HTTP
  - XML
- Multi-platform (contrary to DCOM)
- Multi-language (contrary to Java RMI)
- Independent of the protocol (~contrary to CORBA)
SOAP

Each message part is of some data type. Use XML predefined types, e.g., xsd:int or define your own.

The input and output messages form an operation. A collection of these operations forms a portType.

A binding specifies how operations are accessed using a particular protocol, e.g., SOAP or HTTP GET.
WSDL

- Specifications (09/2000)
  - Ariba, IBM, Microsoft
  - TR W3C v1.1 (25/03/2001)
- Goals
  - WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.
- XML grammar (schema XML)
  - Modular (import of other documents WSDL and XSD)
- Non targeting human beings
  - There are generator of WSDL from programming languages

Elements of a WSDL definition

- `<types>`
  - Embeds the type definition using a type system (such as XSD).
- `<message>`
  - Describes the names and the types of the set of fields to transmit
    - Parameters of an invocation, answer, ...
- `<porttype>`
  - Describes a set of operations. Each operation has zero or one message as input, zero or several message(s) as outputs, or faults
- `<binding>`
  - Specifies a link between a `<porttype>` and a concrete protocol (SOAP1.1, HTTP1.1, MIME, &). A `<porttype>` can have several bindings!
- `<port>`
  - Specifies an endpoint as the combination of a `<binding>` and a network address.
- `<service>`
  - A collection of endpoints.
An example of WSDL

```xml
<?xml version="1.0"?>
<wsdl:definitions name="EndorsementSearch" targetNamespace="http://namespaces.snowboard-info.com"
xmlns:es="http://www.snowboard-info.com/EndorsementSearch.wsdl"
xmlns:esxsd="http://schemas.snowboard-info.com/EndorsementSearch.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
>
<wsdl:types>
<xsd:schema targetNamespace="http://namespaces.snowboard-info.com"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<xsd:element name="GetEndorsingBoarder">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="manufacturer" type="string"/>
<xsd:element name="model" type="string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GetEndorsingBoarderResponse"/>
</xsd:schema>
</wsdl:types>
<wsdl:message name="GetEndorsingBoarderRequest">
<wsdl:part name="body" element="esxsd:GetEndorsingBoarderRequest"/>
</wsdl:message>
<wsdl:message name="GetEndorsingBoarderResponse">
<wsdl:part name="body" element="esxsd:GetEndorsingBoarderResponse"/>
</wsdl:message>
<wsdl:portType name="GetEndorsingBoarderPortType">
<wsdl:operation name="GetEndorsingBoarder">
<wsdl:input message="es:GetEndorsingBoarderRequest"/>
<wsdl:output message="es:GetEndorsingBoarderResponse"/>
<wsdl:fault message="es:GetEndorsingBoarderFault"/>
</wsdl:operation>
</wsdl:portType>
...</wsdl:definitions>
```

UDDI: Registry of Web Services

- Specification (09/2000)
  - Ariba, IBM, Microsoft +260 other companies
  - OASIS standard (04/2003)
- Goals
  - Worldwide registry of companies/services.
  - Several indexed entries: name, company ID, description of product, of services, remotely accessible software services (endpoints)
  - Indexation of proprietary catalogues (ebXML, RosettaNet, Ariba, Commerce One, etc.)
- XML grammar (schema XML)
  - Submission/Request based on SOAP et WSDL
**UDDI: What’s inside?**

1. SW companies, standards bodies, and programmers populate the registry with descriptions of different types of services.

2. Businesses populate the registry with descriptions of the services they support.

3. UBR assigns a programmatically unique identifier to each service and business registration.

4. Marketplaces, search engines, and business apps query the registry to discover services at other companies.

5. Business uses this data to facilitate easier integration with each other over the Web.

---

**UDDI: Information Organization**

- Businesses register public information about themselves
  - White Pages
  - Yellow Pages
  - Green Pages

- Standards bodies, Programmers, Businesses register information about their Service Types
  - Service Type Registrations
UDDI: White pages

- Business Name
- Text Description
  - list of multi-language text strings
- Contact info
  - names, phone numbers, fax numbers, web sites...
- Known Identifiers
  - list of identifiers that a business may be known by - DUNS, Thomas, other

UDDI: Yellow Pages

- Business categories
  - 3 standard taxonomies in V1
    - Industry: NAICS (Industry codes - US Govt.)
    - Product/Services: UN/SPSC (ECMA)
    - Location: Geographical taxonomy
      - Implemented as name-value pairs to allow any valid taxonomy identifier to be attached to the business white page
UDDI: Green Pages

- New set of information businesses use to describe how to do e-commerce with them
  - Nested model
    - Business processes
    - Service descriptions
    - Binding information
  - Programming/platform/implementation agnostic
  - Services can also be categorized

UDDI

- XML document
- Created by end-user company (or on their behalf)
- Can have multiple service listings
- Can have multiple taxonomy listings
Bibliography

- Web Services
  - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, Ethan Cerami, February 2002 O'Reilly

- WSDL
  - W3C specification
  - http://www.w3.org/TR/wsdl

- UDDI
  - http://www.uddi.org
  - Java and SOAP, Robert Englander, May 2002 O'Reilly

Workflows
Service Composition

- **Composite Service**
  - a service implemented by combining other web services

- **Service composition**
  - the process of developing a composite web service

- **Composition as a way to master Complexity**

Orchestration models

- **Goals**
  - Specifying the order of service invocations depending on conditions
  - Need for abstraction models and languages
    - activity diagrams
    - statecharts
    - petri-nets
    - pi-calculus
    - activity hierarchies
    - rule-based orchestration approaches
    - gamma-calculus
## Service Selection

- During execution, a composition engine has to target messages to specific services, which are defined in the composition schema (typically in the form of a port type).

- The question is how to select and bind the services:
  - Static binding
  - Dynamic binding by reference
  - Dynamic binding by lookup
  - Dynamic operation selection

## Dependencies between Coordination and Composition

- **Composition protocols**
  - private documents that define the internal implementation of a Web service.

- **Coordination protocols**
  - public documents focusing on external interactions of a Web service.

- Coordination Protocols and Composition Schemas
- Conversation Controllers and Composition Engines
Workflow based models

- Bibliography
  - http://www.gridworkflow.org/snips/gridworkflow/
  - Askalon-AGWL (Abstract Grid Workflow Language) - Innsbruck, Austria
  - Triana - Cardiff, UK
  - GriCoL (Grid Concurrent Language) - HLRS Stuttgart, Germany
  - Kepler - SEEK, SDM, GEON, …

- Generic models
  - Task
  - Ports
    - Input/output data
  - Assembly
    - Control/data flows

---

BPEL
BPEL for Web Services
Combining Atomic Web Services into a Composite Web Service

BPEL is a second generation language, building on prior work by Microsoft, IBM. Standardisation now managed by Oasis.

BPEL: Construct Overview

- Process declaration
  <process>
    <partners/>
    <receive/>
    <invoke/>
    <invoke/>
    <invoke/>
    <reply/>
  </process>

- Business logic
  - <sequence/>
  - <flow/>
  - <link/>
  - <switch/>
  - <throw/>
  - <scope/>
  - <while/>
  - <pick/>
  - <copy/>
  - <assign/>
Loan Processing Orchestration

1. ReceiveCustomerRequestForLoan
2. If loan < $10,000, go to risk = low
3. If loan > $10,000, go to risk = high
4. If risk = low, go to assignYesNoToAccept
5. If risk = high, go to invokeLoanApprover
6. If assignYesNoToAccept == yes, go to invokeLoanApprover
7. Reply: AcceptMessageToCustomer
Service Component Architecture

- A vendor-, technology-, language-neutral model for the creation of business systems using SOA by the composition and deployment of new and existing service components

Business Drivers

- Flexible businesses require flexible IT
  - Globalization demands greater flexibility
  - Global supply chain integration
  - Business processes
    - Daily changes vs. yearly changes
  - Growth through flexibility is at the top of the CEO agenda
  - Reusable assets can cut costs by up to 20%
  - Crucial for flexibility and becoming an On Demand Business
What We have Today

- Complexity
- Rigid, brittle architectures
- Inability to evolve

What we want to get to

- Well-defined interfaces with business-level semantics
- Standardized communication protocols
- Flexible recombination of services to enhance software flexibility

Service-Oriented Architecture is one of the key technologies to enable flexibility and reduce complexity
Service-oriented Architecture

- SOA derives its technical strategy and vision from the basic concept of a *service*:
  - "A service is an abstraction that encapsulates a software function."
  - "Developers build services, use services and develop solutions that aggregate services."
  - "Composition of services into integrated *solutions* is a key activity"

SOA Core Elements

- **Service Assembly**
  - technology- and language-independent representation of the composition of services into business solutions

- **Service Component**
  - technology- and language-independent representation of a service which can be composed with other services
SCA: Simplified Programming Model for SOA

What is SCA:
- Model for assembly of service components into business solutions
- Simplified component programming model for implementation of services:
  - Business services implemented in any of a variety of technologies
    - e.g. EJBs, Java POJOs, BPEL process, COBOL, C++, PHP ...

Key Benefits of SCA:
- **Loose Coupling:** Components integrate with other components without needing to know how other components are implemented
  - Loose coupling - KEY requirement for SOA
- **Flexibility:** Components can easily be replaced by other components
  - Flexibility - KEY requirement for SOA
- **Services** can be easily invoked either synchronously or asynchronously
- **Composition** of solutions: clearly described
  - Composition of services - KEY requirement for SOA
- **Productivity:** Easier to integrate components to form composite application

SCA simplifies development experience for all developers, integrators and application deployers

SCA: What is it NOT

- Does not model individual workflows
  - use BPEL or other workflow languages

- Is not Web services
  - SCA can use / may use Web services, but can also build solutions with no Web services content

- Is not tied to a specific runtime environment
  - distributed, heterogeneous, large, small

- Does not force use of specific programming languages and technologies
  - aims to encompass many languages, technologies
Example SCA assembly

Assembly Model Concepts

- Component
- Implementation
- Composite
- Service
- Reference
- Wire
- ComponentType
- ConstrainingType
- Domain
- Contribution
SCA Composite Component

Service:
- Java Interface
- WSDL PortType

Reference:
- Java Interface
- WSDL PortType

Composite A

Component A

Component B

Properties

Service

Reference

Wire

Wire

Wire

Binding
- Web Service
- SCA
- JCA
- JMS
- ...

Properties

Example

bigbank.accountcomposite

AccountService Component

Reference

AccountData Service Component

Service

AccountService
SCA Interaction Model

- **Synchronous** & **Asynchronous** service relationships
- **Conversational** services
  - stateful service interactions

Asynchronous support
- "non-blocking" invocation
- asynchronous client to synchronous service
- callbacks

Policies Framework and Infrastructure Capabilities

- **Infrastructure** has many configurable capabilities
  - Security: Authentication and Authorization
  - Security: Privacy, Encryption, Non-Repudiation
  - Transactions, Reliable messaging, etc.
  - Complex sets of configurations across multiple domains of concern

- SCA abstracts out complexity with a **declarative model**
  - no implementation code impact
  - simplify usage via declarative policy intents
  - simple to apply, modify
  - complex details held in PolicySets
Java Common Annotations

- Java Annotations for generating corresponding componentType
- Common across all Java-related specifications
- Implementation annotations
  - @Service
  - @Reference
  - @Property
  - @Scope @Init @Destroy @EagerInit
  - @ConversationID @ConversationAttributes
  - @ComponentName
  - @Constructor
- Interface annotations
  - @AllowsPassByReference
  - @Callback
  - @Remotable
  - @Conversational
  - @Oneway

Java Annotation Example

```java
package services.account;
...
public class AccountServiceImpl implements AccountService {
  @Property
  private String currency = "USD";
  @Reference
  private AccountDataService accountDataService;
  @Reference
  private StockQuoteService stockQuoteService;
  ...
  public AccountReport getAccountReport(String customerID) {
    ...
  }
  ...
```
Java Common APIs

- Common across all Java-related specifications
- APIs for:
  - Component context
  - Request context
  - Callable reference
  - Service reference
  - Conversation
  - Exceptions

BPEL Component Implementation

- SCA and BPEL are complementary
  - BPEL provides business orchestration view of the component
  - SCA provides a compositional view of interconnection among service components
- Supports WS-BPEL 1.1 and 2.0
- Requires WSDL interfaces
- SCA service = partnerLink with a single role belonging to the BPEL process
- SCA reference = partnerLink with a single role belonging to a partner
- When partnerLink defines two role, directionality defines whether it is a service or a reference
- SCA extensions
  - Attribute “sca:property” on a variable declaration defines a property
  - Element “sca:multiReference” on a variable declaration defines a multivalued reference
STCM
Spatio-Temporal Component Model

Workflow models vs. Component models

<table>
<thead>
<tr>
<th></th>
<th>assembly</th>
<th>simplicity</th>
<th>Code coupling</th>
<th>Resources usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow models</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Component models</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>
Limitation of existing approaches

- **Software component models**
  - Adding meta-data about component’s behavior (exp: ICENI)
  - Objective: compute an optimal placement of components
  - Require code knowledge
  - Complicate application design

- **Workflow models**
  - Encapsulate spatial composition within tasks implementations
  - Objective: offer a level of composition for coupled codes
  - Limits the hierarchy to two levels
  - Limits re-usability

- **Limitations because of**
  - Spatial and temporal compositions are not at the same level

Principle of STCM

- **Combination of component and workflow models**
  - Spatial and temporal dimensions at the same level of assemblies

- **Component-task**
  - Input and output ports (temporal)
  - Spatial ports
  - Task

- **Assembly model**
  - Adaptation of a workflow language
Temporal ports & task

```c
void setIn_inA(...) { d_inA = ..; }
void task() {
    d_outA = pUses.mult (d_inA, 50);
}
double getOut_outA() { return d_outA; }
```

input double inA

```
output double outA
```

Life cycle

- States of a component instance at execution

- non existent
- inactive
- active
- running
Temporal ADL: Primitive Component à la Fractal

```xml
<component name="name" (extends="parentType")>
  <clientPort name="..." type="itfName" (set="...") ?*/
  <serverPort name="..." type="itfName" (set="...") ?*/
  <attribute name="name" type="attributeType"/>
  <dataIn name="..." type="dataType" (set="...") ?*/
  <dataOut name="..." type="dataType" (set="...") ?*/
  <impl type="exe|dll|.." signature="sign" />
  <controllerDesc desc="desc"/>
</component>
```

Temporal ADL of Composite based on AGWL (1/2)

```xml
<component name="name" (extends="parentType")>
  <dataIn name="..." type="dataType" (set="...") ?*/
  <dataOut name="..." type="dataType" (set="...") ?*/
  <body>
    <component>
      <instance name="i1" compRef="C1" />
      <instance name="i2" compRef="C2" />
      <setPort client="i2.p2" server="i1.p1" />
      <setPort in="i2.d2" out="i1.d1" />
    </component>
  </body>
  <controllerDesc desc="desc"/>
</component>
```
Temporal ADL of Composite based on AGWL (2/2)

Sequence
<sequence name="name">
  <dataIn name="name" type="..." (set=..)?/>*
  <dataOut name="name" type="..."/>*
  <clientPort name="name" type="..." (set=..)?/>*
  <serverPort name="name" type="..."/>*
  <!-- other spatial ports -->
  <instruction1>...
  <instructionN>
</sequence>

Condition
<if name="name">
  <!-- like in sequence-->
  <condition>
    boolean expression
  </condition>
  <then>
    <instruction1>
  </then>
  <else>
    <instruction2>
  </else>
</if>

Example
<component name ="example">
  ...
  <parallel name="ParallelCtrl">
    <section>
      <component name="B">
        <dataIn name="inB" ... set="init.out1"/>
        <serverPort name="pB" type="Foo"/>
      </component>
    </section>
    <section>
      <component name="A">
        <dataIn name="inA" ... set="init.out2"/>
        <clientPort name="pA" type="Foo" set="B.pB"/>
      </component>
    </section>
  </parallel>
  ...
</component>
Summary

- Combination of component and workflow models
  - Component-task
  - Temporal ports
  - Assembly model “à la workflow”

- STCM
  - Extension of GCM (CoreGrid)
    - Temporal ports and task
  - Adaptation of AGWL (Abstract Grid Workflow Language)
    - Component and spatial composition