Orchestration and Choreography: Standards, Tools and Technologies for Distributed Workflows

5-7 October, 2005, Second University of Naples, Naples, Italy

Steve Ross-Talbot
CEO Pi4 Technologies
Chair W3C Web Services Activity
Co-chair W3C Web Services Choreography
Agenda

- What are Workflow, Orchestration and Choreography?
- What is SOA?
- How do all of these fit together?
- WS-CDL and bioinformatics
- WS-CDL under the hood
- Formalisms and what they mean
- WS-CDL in practice (lets look at tools)
- What can I do with it
- Summary
What is a workflow?

The automation of business processes, in whole or in part, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules

WFMC
What is orchestration?

Orchestration [of web service] is a technique to recursively compose and orchestrate web services to provide a new composite web service

WS-BPEL
What is choreography?

A choreography is a global behavioral contract that describes (and therefore can be used to constrain) the valid ordering of messages between services that make up some flow that meets some [business] objective

WS-CDL
What is SOA?

- Service Oriented Architecture is a way of building distributed systems.
  - A service is a computational process with a WSDL interface.
  - A service may interact with other services through any communication mechanism (HTTP is but one).
  - Services are discovered at runtime.
  - SOA encourages document centric rather/messaging passing than function-centric programming (see WSDL2.0)
How does it fit together?

• An SOA is just a grid of services. Something has to give the grid and the services meaning.

• Meaning can be thought of as some notion of the common behavior of the services in achieving some goal. We might call this a workflow description.

• Distributed (peer to peer) workflows may be described using WS-CDL and executed at the peers using WS-BPEL. Thus orchestration is an execution idiom and choreography a description one.
How does it fit together?

WS-CDL (Choreography)

Other Languages (i.e. Java, C#, C++, etc)

WS-BPEL (Orchestration)
WS-Addressing

WSDL

SOAP

HTTP

TCP/IP

Legacy
Available
Nascent
Missing
Relevance to Bioinformatics

- **Formalism**

<table>
<thead>
<tr>
<th>Agents</th>
<th>Communication Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small molecules</td>
<td>Electron Sharing</td>
</tr>
<tr>
<td>Proteins</td>
<td>Protein-protein interaction, binding, phosphorylation, methylation, etc</td>
</tr>
<tr>
<td>Cells</td>
<td>Material consumption, environmental sensing, etc</td>
</tr>
</tbody>
</table>

**Common formalisms**

<table>
<thead>
<tr>
<th>Agents</th>
<th>Communication Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network service</td>
<td>TCP/IP read/write</td>
</tr>
<tr>
<td>Email client/server</td>
<td>SMTP read/write</td>
</tr>
<tr>
<td>Browser server</td>
<td>HTTP read/write</td>
</tr>
<tr>
<td>WS-CDL service</td>
<td>send/receive</td>
</tr>
</tbody>
</table>
Relevance to Bioinformatics

- **SOA**
  - Emergence of WSDL described services (KEGG, BLAST, FASTA, Swiss-Prot, … etc)
  - Componentization for reuse and greater availability
  - Decrease in costs and complexity of data integration
  
  - WSDL, SOAP, Java, .NET, …. and the rest of the stack presented in the context of a service grid.
Relevance to Bioinformatics

- Process volatility
  - In-silico experimentation as workflows across a peer to peer SOA
  - Contractual descriptions for clinical trial protocols (Vioxx) that can be proven to have been followed
  - Outsourcing and yet remaining in control

- WS-BPEL and WS-CDL
WS-CDL under the hood

- Participants, Roles, Relationships, Information, Channels
- Choreography, Interaction
- WorkUnits, Structured composition
- Non Observable Conditionals
- Observable Conditionals
- No State Mgmt
- State Mgmt
- Exception, Finalizers
WS-CDL under the hood
WS-CDL under the hood
package name="BuyerSellerCDL"
author="Steve Ross-Talbot"
version="1.0"
targetNamespace="www.pi4tech.com/cdl/BuyerSellerExample-1"

<description type="description">
This is the basic BuyerSeller Choreography Description
</description>

<sequence>
<interaction name="Buyer requests a Quote - this is the initiator" operation="requestForQuote" channelVariable="Buyer2SellerC" initiate="true">
<description type="description">
Request for Quote
</description>
<participate relationshipType="BuyerSeller" fromRole="BuyerRoleType" toRole="SellerRoleType" />
</interaction>
<exchange name="request" informationType="RequestForQuoteType" action="request">
<description type="description">
Requesting Quote
</description>
</exchange>
<exchange name="response" informationType="QuoteType" action="respond">
<description type="description">
Quote returned
</description>
</exchange>
</sequence>
</choreography>
</package>
WS-CDL under the hood

<workunit name="">
  <choice>
    <silentAction roleType="">
      <description type="">
        silentAction
      </description>
    </silentAction>
    <sequence>
      <interaction name="">
        <description type="">
          "operation","channelVariable"
        </description>
        <participation relationshipType="">
          <fromRole="" toRole=""/>
          <exchange name="">
            "informationType"
            "action"
          </exchange>
        </participation>
      </interaction>
      <interaction name="">
        <description type="">
          "operation","channelVariable"
        </description>
        <participation relationshipType="">
          <fromRole="" toRole=""/>
          <exchange name="" channelType="">
            "action"
          </exchange>
        </participation>
      </interaction>
      <assign roleType="">
        <copy name="">
          <source expression=""/>
          <target variable=""/>
        </copy>
      </assign>
    </sequence>
  </choice>
</workunit>
WS-CDL under the hood

<parallel>
  <workunit name="">
    <sequence>
      <workunit name="">
        <sequence>
          <workunit name="">
            <sequence>
              <workunit name="">
                <sequence>
                  <workunit name="">
                    <sequence>
                      <workunit name="">
                        <sequence>
                          <workunit name="">
                            <sequence>
                              <workunit name="">
                                <sequence>
                                  <workunit name="">
                                    <sequence>
                                      <workunit name="">
                                        <sequence>
                                          <workunit name="">
                                            <sequence>
                                              <workunit name="">
                                                <sequence>
                                                  <workunit name="">
                                                    <sequence>
                                                      <workunit name="">
                                                        <sequence>
                                                          <workunit name="">
                                                            <sequence>
                                                              <workunit name="">
                                                                <sequence>
                                                                  <workunit name="">
                                                                    <sequence>
                                                                      <workunit name="">
                                                                        <sequence>
                                                                          <workunit name="">
                                                                            <sequence>
                                                                              <workunit name="">
                                                                                <sequence>
                                                                                  <workunit name="">
                                                                                    <sequence>
                                                                                      <workunit name="">
                                                                                      
                                                                                      </sequence>
                        </sequence>
                    </sequence>
                  </sequence>
                </sequence>
              </sequence>
            </sequence>
          </sequence>
        </sequence>
      </sequence>
    </sequence>
  </sequence>
</parallel>
WS-CDL under the hood

A typical order of evaluation is as follows:

\[(G) \text{ Body} (R G) \text{ Body} (R G) \text{ Body}\]

With respect to a G then the G is only evaluated when the variables are available and evaluate to True and otherwise we wait at the guard condition. Thus the Body after the first G only gets executed when G is True. Or put another way Body is primed ready for action and then is executed when G evaluates to True.

IF G is unavailable or evaluates to False THEN it equates to:

\[\text{when (G) \{ Body \} until (!R)}\]

IF G is always True THEN it equates to:

\[\text{repeat \{ Body \} until (!R)}\]

IF R is always False THEN it equates to:

\[\text{when (G) \{ Body \}}\]
A typical order of evaluation is as follows:

\((G) \text{ Body } (R \ G) \text{ Body } (R \ G) \text{ Body}\)

Which equates to (in pseudo code):

\[\text{while (G) } \{ \text{Body} \} \text{ until (!R)}\]

IF \(G\) is always True THEN it equates to:

\[\text{repeat } \{ \text{Body} \} \text{ until (!R)}\]

IF \(R\) is always False THEN it equates to:

\[\text{if (G) } \{ \text{Body} \}\]
Formalisms

• Pi-calculus
  - Algebraic encoding of behavior
  - Reduction rules to show progress

• Static checking for liveness (livelock and deadlock)

• Composition

• Channel/port passing (dynamic topologies)

• Higher order calculi
  - Includes stochastic pi-calculus used for in-silico experiments to simulate interaction between molecules
## Formalisms

<table>
<thead>
<tr>
<th>Model</th>
<th>Completeness</th>
<th>Compositionality</th>
<th>Parallelism</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing Machines</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Lambda</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Petri Nets</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CCS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>π</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# Formalisms

<table>
<thead>
<tr>
<th>Operation</th>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>$^1.P$</td>
<td>Sequence</td>
</tr>
<tr>
<td>Action</td>
<td>$\Delta(y), x(y)$</td>
<td>Communication</td>
</tr>
<tr>
<td>Summation</td>
<td>$a.P + b.Q \cdot \sum_{i=1..n} P_i$</td>
<td>Distributed Choice</td>
</tr>
<tr>
<td>Recursion</td>
<td>$P={\varepsilon}P$</td>
<td>Repetition</td>
</tr>
<tr>
<td>Replication</td>
<td>$!P$</td>
<td>Repetition</td>
</tr>
<tr>
<td>Composition</td>
<td>$P</td>
<td>Q$</td>
</tr>
<tr>
<td>Restriction</td>
<td>$(v_x)P$</td>
<td>Encapsulation</td>
</tr>
<tr>
<td>Nothing</td>
<td>$0$</td>
<td>Do nothing</td>
</tr>
</tbody>
</table>

System = (!Client | !IdleServer)

$Client(o,c,req,rsp) = o.req_1.rsp_1 .req_2 .rsp_2 .c.Client(o,c,res,rsp)$

$IdleServer(o,req,rsp,c) = o.BusyServer(o,req,rsp,close)$

$BusyServer(o,req,rsp,c) = req.rep.BusyServer(o,req,rsp,c) + c.IdleServer(o,req,rsp,c)$
The channels | a pair of "ports" in the $\pi$-calculus
---|---
An interaction | message exchange that occurs between paired ports
message | a polyadic message
type | sorts
Guarded workunit | Process pattern
WS-CDL in practice

Figure 5. A Petri Net model of the invasion process, corresponding to the Workflow model shown in Figure 2. Places are shown as circles, and transitions as rectangles, and are labeled as $t_1, t_4$. The first and last places in the Petri Net are also labeled (as $P_1$ and $P_6$). Implicit XOR split and joins are marked as “XOR split” and “XOR join”, respectively. AND split and joins are also marked.
WS-CDL in practice
WS-CDL in practice
WS-CDL in practice

+ <variableDefinitions>
  - <sequence>
    + <description type="documentation">MainSequence</description>
    + <interaction name="start" operation="start" channelVariable="p1-t1Channel"/>
    - <choice>
      + <description type="documentation">XORSplitAndJoin</description>
      - <sequence>
        + <description type="documentation">Glycophorin B: Alt Pathway</description>
        + <interaction name="i1checkGlycophorinB" operation="checkGlycophorinB" channelVariable="t1-t2Channel"/>
        + <interaction name="i4checkGlycophorinB" operation="checkGlycophorinB" channelVariable="t2-t5Channel"/>
      </sequence>
      - <sequence>
        + <description type="documentation">Glocophorin A: Init attach</description>
        + <interaction name="i2checkGlycophorinA" operation="checkGlycophorinA" channelVariable="t1-t3Channel"/>
        + <interaction name="i5checkGlycophorinA" operation="checkGlycophorinA" channelVariable="t3-t5Channel"/>
      </sequence>
      - <sequence>
        + <description type="documentation">No sialic acid: Alt pathway</description>
        + <interaction name="i3checkNoSialicAcid" operation="checkNoSialicAcid" channelVariable="t1-t4Channel"/>
        + <interaction name="i6checkNoSialicAcid" operation="checkNoSialicAcid" channelVariable="t4-t5Channel"/>
      </sequence>
    </choice>
  - <parallel>
    + <description type="documentation">ANDSplitAndJoin</description>
    - <sequence>
      + <description type="documentation">Formation of tight junction</description>
      + <interaction name="i7checkForTightJunction" operation="checkForTightJunction" channelVariable="t5-t6Channel"/>
      + <interaction name="i9checkForTightJunction" operation="checkForTightJunction" channelVariable="t6-t8Channel"/>
    </sequence>
    - <sequence>
      + <description type="documentation">Processing of AMA-1</description>
      + <interaction name="i8ProcessingAMA-1" operation="processAMA-1" channelVariable="t5-t7Channel"/>
      + <interaction name="i10ProcessingAMA-1" operation="processingAMA-1" channelVariable="t7-t8Channel"/>
    </sequence>
  </parallel>
+ <interaction name="i11AttachmentDone" operation="start" channelVariable="p8-p1Channel"/>
</sequence>
</choreography>
</package>
What can I do with it?

• Simulation:
  - I can test it (simulate the message exchanges)

• Generation:
  - I can generate Java or BPEL code, deploy and execute or run it as a set of peer services (no single point of control)

• Documentation:
  - I can produce documentation (html) to describe it

• Execution:
  - I can monitor the threads of execution through it
What can I do with it?

p1-t1Channel

The variable's type is p1-t1.

p8-p1Channel

The variable's type is p8-p1.

Activities

- Sequence: MainSequence
  - start
  - Choice: XORSplitAndJoin
    - Sequence: Glycophorin B: Alt Pathway
      - i1checkGlycophorinB: Check for Glycophorin B
      - i4checkGlycophorinB: Response for checking Glycophorin B
    - Sequence: Glycophorin A: Init attach
      - i2checkGlycophorinA: Check for Glycophorin A
      - i5checkGlycophorinA: Response to checking for Glycophorin A
    - Sequence: No sialic acid: Alt pathway
      - i3checkNoSialicAcid: Check for no sialic acid
      - i6checkNoSialicAcid: Response after checking for no sialic Acid
  - Parallel: ANDSplitAndJoin
    - Sequence: Formation of tight junction
      - i7checkForTightJunction: Check for tight junction formation
    - Parallel: i9checkForTightJunction
  - Sequence: Processing of AMA-1
    - i8ProcessingAMA-1
    - i10ProcessingAMA-1
- i11AttachmentDone
Summary

- WS-CDL is the Web Services Choreography Description Language (CDL for short)
- Common formalism (the pi-calculus) between SOA/WS-CDL and bioinformatics
- WS-CDL to describe workflow, Orchestration to execute workflow
- WS-CDL description for compliance
Acknowledgments

- Greg Meredith of Djinnisys
- Maria Mirto (for some example and early access to papers)
- Dr Gary Brown (pi4tech)
Grazie
Thank You

Q & A