Web Services Choreography and Process Algebra
29th April 2004

Steve Ross-Talbot
Chief Scientist, Enigmatec Corporation Ltd
Chair W3C Web Services Activity
Co-chair W3C Web Services Choreography
Agenda

• Orchestration vs Choreography
• WS-BPEL
• WS-CDL
• Underpinnings
• Status
• Q&A
Orchestration vs Choreography

• Consider a dance with more than one dancer.
• Each dancer has a set of steps that they will perform. They orchestrate their own steps because they are in complete control of their domain (their body).
• A choreographer ensures that the steps all of the dancers make is according to some overall scheme. We call this a choreography.
• The dancers have a single viewpoint of the dance.
• The choreography has a multi-party or global viewpoint of the dance.
Orchestration vs Choreography

• Orchestration is about describing and executing a single view point model.

• Choreography is about describing and guiding a global model.

• You can derive the single view point model from the global model by projecting based on participant.
WS-BPEL and WS-CDL

• WS-BPEL
  – Orchestration implies a centralized control mechanism.

• WS-CDL
  – Choreography has no centralized control. Instead control is shared between domains.
Orchestration of Web Services

• **The Oasis WS-BPEL TC**
• Summary: Orchestration of web services and recursive composition thereof.
• Style: Scoped programming language (BPEL) with behavioural interfaces (Abstract BPEL).
• Uses: Orchestration of Web Services in a single domain of control (i.e. order flow within institution).
• Status: Currently X issues to resolve and based on WSDL1.1 and some proprietary specs. Due to deliver Q4.
• Issues: Licensing. Based on some proprietary specifications
WS-BPEL

• Is a Web Service
  – Runtime semantics
  – Centralised orchestration

• Abstract
  – Defines end-point protocols

• Executable
  – Executes the necessary WSDL calls effecting message exchange between services

• Benefits
  – Higher reuse of WSDL collateral
WS-BPEL

- Sequence,
- Fork,
- Join,
- Parallel threads,
- Computation (Turing Complete)
WS-BPEL - Problems

• Centralised execution
• Lack of formal semantics
• Non-scalable (requires the concept of dual connectivity)
• Non-collaborative
Choreographing Web Services

- **W3C Web Services Choreography Working Group**
- **Summary:** Describing peer to peer interaction in a global model by means of a CDL
- **Style:** Formalized description of external observable behavior across domains
- **Use for:** Modeling cross domain protocols, protocol enforcement, skeletal code generation (i.e. for FIX)
- **Status:** Requirements document formally published, Model Overview document published to mailing list. Due to deliver end 2004.
What is a Choreography

- WS-Choreography concerns the collaboration protocols of cooperating Web Service participants
  - WS act as peers
  - WS interact in long-lived, stateful & coordinated fashion
- A WS-Choreography description is a multi-participant contract that describes, from a Global Viewpoint, the common observable behavior of the collaborating WS participants
- WS-CDL is a language in which such a contract is specified
  - Standardization underway in the W3C Choreography WG
Using a WS-CDL

- promote a common understanding between WS participants;
- automatically guarantee conformance;
- ensure interoperability;
- increase robustness;
- generate code skeletons.
Benefits of a WS-CDL

• more robust Web Services to be constructed;
• enable more effective interoperability of Web Services through behavioral multi-party contracts, which are choreography descriptions;
• reduce the cost of implementing Web Services by ensuring conformance to expected behaviour;
• increase the utility of Web Services as they will be able to be shown to meet contractual behavior.
Overview of WS-CDL

• Interactions
• Channels
• Participants
• Roles
• State
WS-CDL Approach

• Simple contract-like mechanisms are exhibited in the literature for capturing
  – Deadlock-freedom (Kobayashi, 99, 00)
  – Liveness (Kobayashi, 01; Yoshida, et al, 02)
  – Security (Abadi et al; Cardelli and Gordon; Berger, Honda, Yoshida)
  – Resource management (Tofte; Kobayashi; Gordon and Dal Zillio; Yoshida, et al)

• A contract language that guaranteed even basic versions of these properties (at the compatibility level) then that would be a significant advance over the state of the art.
WS-CDL Approach

This work needs to be carried out using formal basis. To the extent possible, *technical* design deliberations can and should be a matter of calculation.

Mobile process calculi provide a natural candidate.

<table>
<thead>
<tr>
<th>Web service Implementation</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does roughly what client wants it to do</td>
<td>Bisimulation ‘approximation’</td>
</tr>
<tr>
<td>Contract</td>
<td>Behavioral type</td>
</tr>
</tbody>
</table>
Why process calculi?

<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>$\pi$</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
## Global Models

### State Transition Table for a Garage Door Opener

<table>
<thead>
<tr>
<th>Current State</th>
<th>INPUT</th>
<th>Effect</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Door closed / Motor off</td>
<td>Button pressed</td>
<td>Start Motor</td>
<td>Motor Running Up</td>
</tr>
<tr>
<td>Motor Running Up</td>
<td>Door Open detected</td>
<td>Stop Motor</td>
<td>Door Open / Motor Off</td>
</tr>
<tr>
<td>Motor Running Up</td>
<td>Button pressed</td>
<td>Stop Motor</td>
<td>Door Partially Open / Motor Off</td>
</tr>
<tr>
<td>Door Partially Open / Motor Off</td>
<td>Button pressed</td>
<td>Start Motor</td>
<td>Motor Running Down</td>
</tr>
<tr>
<td>Door Open / Motor Off</td>
<td>Button pressed</td>
<td>Stop Motor</td>
<td>Door Partially Open / Motor Off</td>
</tr>
<tr>
<td>Motor Running Down</td>
<td>Door Closed detected</td>
<td>Stop Motor</td>
<td>Door Closed / Motor off</td>
</tr>
<tr>
<td>Door Partially Open / Motor Off</td>
<td>Button pressed</td>
<td>Start Motor</td>
<td>Motor Running Up</td>
</tr>
<tr>
<td>Door Partially Closed / Motor Off</td>
<td>Button pressed</td>
<td>Start Motor</td>
<td>Motor Running Up</td>
</tr>
</tbody>
</table>

### Example State Chart

- **On**
  - Turn on
  - Off
- **Off**
  - Turn off
  - Turn on
Collaboration Diagram for two States

- **toggle()**
  - <<Singleton>>
  - **S1**: On
  - **setState(S2)**
  - **S2**: Off

- **toggle()**
  - <<Singleton>>
  - **S2**: Off
  - **setState(S1)**
  - **S1**: On
Global Models

Example flow for Pre-Trade Allocation (using Allocation Instruction message)

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Respondent</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Order-Single (OrderQty=35000)</td>
<td></td>
</tr>
<tr>
<td>Execution Report (ExecType = “0” [New]</td>
<td></td>
</tr>
<tr>
<td>Allocation Instruction (AllocType=&quot;P&quot; Preliminary provided without MiscFees or NetMoney)</td>
<td></td>
</tr>
<tr>
<td>Allocation Instruction Ack (AllocStatus=R Processed)</td>
<td></td>
</tr>
<tr>
<td>Allocation Instruction Ack (AllocStatus=Accept)</td>
<td></td>
</tr>
<tr>
<td>Execution Report (ExecType = “F”) [Trade] (optional Execution Report (ExecType = “3”))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 1 – entire part-filled quantity is warehoused</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bayside</strong></td>
</tr>
<tr>
<td>New order single</td>
</tr>
<tr>
<td>G7Bootstrad = 1</td>
</tr>
<tr>
<td>Execution report (done for day)</td>
</tr>
<tr>
<td>Allocation report (AllocReportType = 5)</td>
</tr>
</tbody>
</table>

1. Bayside sends new GT order with instruction to warehouse any part-filled quantity until the order fills or expires (i.e. G7Bootstrad is 1).
2. SellsSide accepts the order. They sends 1 or more partial fill execution reports.
3. SellsSide sends a “done-for-the-day” (DFT) execution report when execution completes for the day.
4. SellsSide sends a warehouse recap allocation report.

Note: A ‘warehouse instruction’ allocation instruction message from the bayside is not required at this point due to the use of G7Bootstrad when placing the order.

<table>
<thead>
<tr>
<th>Day 2 – further executions; entire part-filled quantity is again warehoused</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bayside</strong></td>
</tr>
<tr>
<td>Execution report (more partial fills)</td>
</tr>
<tr>
<td>Execution report (done for day)</td>
</tr>
<tr>
<td>Allocation report (AllocReportType = 5)</td>
</tr>
</tbody>
</table>

2. SellsSide sends 1 or more partial fill execution reports.
3. SellsSide sends a “done-for-the-day” (DFT) execution report when execution completes for the day.
4. SellsSide sends a warehouse recap allocation report.

Note: A ‘warehouse instruction’ allocation instruction message from the bayside is not required at this point due to the use of G7Bootstrad when placing the order.
WS-CDL Global Models

- A sequential process
  \[
  \text{Client(open, close, request, reply)} = \text{open}.\text{request}_1.\text{reply}_1.\text{request}_2.\text{reply}_2.\text{close}.0
  \]
WS-CDL Global Models

• A repetitive process

Client(open, close, request, reply) =

open.request_1.reply_1.request_2.reply_2.close.Client(open, close, request, reply)
WS-CDL Global Models

- A process with choices to make

\[
\text{IdleServer}(o, \text{req, rep, c}) = \\
o. \text{BusyServer}(o, \text{req, rep, close})
\]

\[
\text{BusyServer}(o, \text{req, rep, c}) = \\
\text{req.rep.BusyServer}(o, \text{req, rep, c}) + \\
c. \text{IdleServer}(o, \text{req, rep, c})
\]
WS-CDL Global Model

• Communication, Concurrency and Replication

SYSTEM = (!Client | IdleServer)  

Client\textsubscript{i} | IdleServer
Client\textsubscript{i} | BusyServer
Client\textsubscript{j} | IdleServer
Client\textsubscript{j} | BusyServer
......

When Client\textsubscript{i} has started an exchange with IdleServer
No other Client can then communicate with the server
Until Client\textsubscript{i} has finished and the server is once again Idle
## WS-CDL and the pi-calculus

<table>
<thead>
<tr>
<th>Operation</th>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prefix</strong></td>
<td>$\pi.P$</td>
<td>Sequence</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>$a(y)$, $\overline{a(y)}$</td>
<td>Communication</td>
</tr>
<tr>
<td><strong>Summation</strong></td>
<td>$a(y).P + b(x).Q$</td>
<td>Choice</td>
</tr>
<tr>
<td></td>
<td>$\sum \pi_i P_i$</td>
<td></td>
</tr>
<tr>
<td><strong>Recursion</strong></td>
<td>$P={}.....{}.P$</td>
<td>Repetition</td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>$!P$</td>
<td>Repetition</td>
</tr>
<tr>
<td><strong>Composition</strong></td>
<td>$P</td>
<td>Q$</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>$(\nu x)P$</td>
<td>Encapsulation</td>
</tr>
</tbody>
</table>

Collapse send and receive into an interact on channels
WS-CDL and the pi-calculus

• Static checking for livelock, deadlock and leaks
  – Session types and causality

• Robust behavioral type system
  – Session types
WS-CDL - Status

• Where are we today?
• Working Draft V2
• Looking for comments
• Lots of work with vertical standards
• Looking to last call end Q404
WS-CDL Summary

• Global model
  – Ensured conformance

• Description language
  – Not executable

• Tools
  – Generators for end points
  – Advanced typing

• Status
  – Moving for last call end of Q404
References

- WS-CDL Working Draft
- WS-CDL Overview
- BPEL4WS 1.1
- Enigmatec