Analyzing Service Contract with Model Checking

Joseph C. Okika, Anders P. Ravn

Department of Computer Science
Aalborg University, Denmark

FLACOS 2009 - Toledo, Spain, September 24-25, 2009
SOA - better Service still a major concern

Sales Processing Services

Ordering Services

Bank Services

Compatible

Conformance Interoperability Consistency
BPEL – Business Process Execution Language

- Emerging Web Service standard
- Specifies behavioral aspects of a service
- Partnerlinks and activities to model the service interaction

```xml
<?xml version="1.0" encoding="utf-8" ?>
<process name="Travel"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/"
business-process/>

...  
<partnerLinks>
  <partnerLink name="client"  
    partnerLinkType="trv:travelLT"  
    myRole="travelService"/>
  ...
  <partnerLink name="employeeTravelStatus"  
    partnerLinkType="emp:employeeLT"  
    partnerRole="employeeTravelStatusService"/>

...  
</partnerLinks>
```

```xml
...  
<partnerLink name="AmericanAirlines"  
  partnerLinkType="aln:flightLT"  
  myRole="airlineCustomer"  
  partnerRole="airlineService"/>
<partnerLink name="DeltaAirlines"  
  partnerLinkType="aln:flightLT"  
  myRole="airlineCustomer"  
  partnerRole="airlineService"/>

</partnerLinks>

<!-- Variables are declared here-->
<sequence>
  <receive partnerLink="client"  
    portType="trv:TravelApprovalPT"  
    operation="TravelApproval"  
    variable="TravelRequest"  
    createInstance="yes" />
```
Why BPEL?

- Real world examples.
- European Council for Nuclear Research
- CJIB under Dutch Ministry of Justice
Real world examples...
- European Commission
- EMCS

Vivansa
Simply because you need results.

BPEL has some unique features

Business Process Management Suite (BPMS) for creating sophisticated business processes (including long running, asynchronous processes). The modelling is achieved using BPMN as the standard graphical notation, and KPLC as the XML-based process definition language or BPEL as the serialized XML programming language for the specification of executable business processes (applied primarily to the orchestration of Web services). These languages improve communication and portability of process models. They facilitate designing, defining, implementing, and deploying composite applications and services from a number of distributed and autonomous software components, offering a flexible way to achieve the required business collaborations.

BPEL has some unique features
Event Handler

Two Constructs

1. OnEvent – message event
2. OnAlarm – alarm event
Fault Handler

- Reverse work – Undo a partial/unsuccessful work of a scope that a fault has occurred in.
- Not same as compensation handler.
- Eg. Internal process error, platform specific fault, a web service operation cannot complete successfully, a throw activity
Compensation Handler

- Execution serves to reverse some previously executed application logic
  - i.e. When the scope is completed
- No automatic restoration of data during compensation
- Up to the application to define its own compensation behavior
- Eg. Cancelling reservation, putting an order on hold, removing a charge on a credit card.
Putting them together

1) Event handler receives a cancellation message and throws a fault to the Fault Handler

2) Fault Handler executes a compensate for the previously completed (linked) scope

3) Compensation Handler for the completed scope rolls back the work of InvokeOrderInventory
Does the Service Work?

Does it really compensate the charge on a credit card?
We can do … Software Testing

Black box, White box testing
Unit testing
Incremental Integration testing
Integration testing
Functional testing
System testing
End-to-end testing
Sanity testing
Regression testing,
Acceptance testing
Load testing
Stress testing
Performance testing
Usability testing
Security testing
Compatibility testing, etc.

http://www.xbosoft.com/images/index/index800_07.jpg
We can do … Analysis

- based on:
  - Automata
  - Petri nets
  - Abstract State Machines
  - Process Algebra
  - etc.

✓ Automata/Model Checking

Model Checking has made some progress
Model Checking tools are reaching maturity
Tools: SPIN, NuSMV, UppAal, CWB
Related Work I

- Intermediate Representation: automata with XPath guards (called GFSA) as an intermediate representation for web services.
- A translator from BPEL4WS to GFSA is developed,
- Model checker SPIN used as the back-end of WSAT to check LTL properties.

**WSAT: A Tool for Formal Analysis of Web Services:** Xiang Fu, Tevfik Bultan, and Jianwen Su
Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004
Related Work III

Using RT-UML for modelling web services
María-Emilia Cambronero, Juan José Pardo, Gregorio Díaz, Valentin Valero, SAC 2007:643-648
Related Work IV

- Exhaustive simulation based on a formalisation of BPEL semantics using the Algebra of Timed Processes (ATP).
- Models analysed by model checking value-based temporal logic properties using the CADP toolbox.

Formal Modeling and Discrete-Time Analysis of BPEL Web Services.
Radu Mateescu and Sylvain Rampacek.
Approach

Many theoretical results, tools proposals
- semantic definition/mapping to target language/ applicability.
- Mostly fragments of BPEL without the intricate features

Our approach: - make a formal model of a service
  - WS-BPEL
  - Analysis of behaviour
    - Orchestration
    - Choreography
  - UppAal
Previous Efforts

- Derive semantic models in the form of (timed) automata
- Several specifications – different aspects.
- Simulation relation

**Consistency Checking of Web Service Contracts**

*International Journal On Advances in Systems and Measurements,*

issn 1942-261x vol. 1, no. 1, year 2008,

http://www.iiariajournals.org/systems_and_measurements/"

Previous Efforts ...

- Specification of the CoCoME case study
  - temporal logics, operational, deontic specification
- Comparison between contract specifications
  - card pay, express mode, sales process
- Discussion on how easy it is to analyze the specifications

Current effort

- Full behavior of BPEL
- Timed Automata for the model with rendering to UppAal
- Semantics based on UppAal
  - Semantic preserving extraction/translation

- Semantics based on Rewriting Logic
  - Executable operational semantics
### SOS for full BPEL (I)

- **Service Interaction**

#### Activity | Semantic Rules
--- | ---
receive | \[
\begin{align*}
(receive \ p, \ s, (\rho, \sigma)) \xrightarrow{\gamma_p} (\epsilon, s, (\rho, \sigma')) \\
(receive \ p, \ s, (\rho, \sigma)) \xrightarrow{\chi} (receive \ p, \ s, (\rho, \sigma))
\end{align*}
\]

reply | \[
\begin{align*}
(reply \ p, \ s, (\rho, \sigma)) \xrightarrow{\lambda_p} (\epsilon, s, (\rho, \sigma)) \\
(reply \ p, \ s, (\rho, \sigma)) \xrightarrow{\chi} (reply \ p, \ s, (\rho, \sigma))
\end{align*}
\]

invoke | \[
\begin{align*}
(invoke \ p, \ s, (\rho, \sigma)) \xrightarrow{\lambda_p} (\epsilon, s, (\rho, \sigma)) \\
(invoke \ p, \ s, (\rho, \sigma)) \xrightarrow{\chi} (invoke \ p, \ s, (\rho, \sigma))
\end{align*}
\]

\[
\begin{align*}
(invoke \ p_1, \ s, (\rho, \sigma)) \xrightarrow{\lambda_{p_1}} (\epsilon, s, (\rho, \sigma')) \\
(invoke \ p_1, \ s, (\rho, \sigma)) \xrightarrow{\gamma_{p_2}} (\epsilon, s, (\rho, \sigma)) \\
(invoke \ p_1 \ p_2, \ s, (\rho, \sigma)) \xrightarrow{\lambda_{p_1}, \gamma_{p_2}} (\epsilon, s, (\rho, \sigma'))
\end{align*}
\]
SOS for full BPEL (II)

- Scope + handlers

\[
\begin{align*}
\text{scope 1:} & \quad \left( \text{scope } s_0 \ A \ F \ E \ C \ T, s, (\rho, \sigma) \right) \xrightarrow{\tau} \left( \text{sequence } A \ \text{endscope}, s_0, (\rho', \sigma') \right) \\
\text{where} & \quad \rho' = \rho + [s_0 \mapsto (s, \rho, t_{\text{new}})] + [f \mapsto A](f, A) \in F] + [e \mapsto A](e, A) \in E] \\
& \quad \sigma' = \sigma + [t_{\text{new}} \mapsto (C, T, \rho')]
\end{align*}
\]

\[
\begin{align*}
\text{scope 2:} & \quad \left( \text{endscope } s, (\rho, \sigma) \right) \xrightarrow{\tau} (\epsilon, s', (\rho', \sigma)) \text{ where } s' = \rho[s]_1, \quad \rho' = \rho[s]_2 + [s \mapsto \rho[s]_3] \\
\text{scope 3:} & \quad \left( \text{exit } s, (\rho, \sigma) \right) \xrightarrow{\tau} (\epsilon, s', (\rho', \sigma)) \text{ where } s' = \rho[s]_1, \quad \rho' = \rho[s]_2
\end{align*}
\]

\[
\begin{align*}
\text{scope 4a:} & \quad \rho[f] = A \\
& \quad \left( \text{throw } f, s, (\rho, \sigma) \right) \xrightarrow{\tau} (A', s, (\rho, \sigma)) \text{ where } A' = \text{sequence } \rho[f] \ \text{endscope} \\
\text{scope 4b:} & \quad \rho[f] = A \\
& \quad \left( \text{throw } f, s, (\rho, \sigma) \right) \xrightarrow{\tau} (A', s, (\rho, \sigma)) \text{ where } A' = \text{sequence } \rho[f] \ \text{endscope} \\
\text{scope 4c:} & \quad \rho[f] = A \\
& \quad \left( \text{throw } f, s, (\rho, \sigma) \right) \xrightarrow{\tau} (A', s, (\rho, \sigma)) \text{ where } A' = \text{sequence } \rho[f] \ \text{endscope} \\
\text{scope 5:} & \quad \left( \text{rethrow } f, s, (\rho, \sigma) \right) \xrightarrow{\tau} (\text{sequence exit throw } f, s, (\rho', \sigma))
\end{align*}
\]
Lessons learned so far …

- Better Service is still a major concern
- Analyzing service orchestration: important for SOA and Cloud Computing
- Few semantics for orchestration analysis
- Intricate features difficult to formalize
- Difficult to create an UppAal model directly from the standard
- But automated analysis will
  - improve quality
  - can reduce cost; example testing cost
Some Issues

Suppose two concurrent isolated scopes, S1 and S2, access a common set of variables and partner links (external to them) for read or write operations. The semantics of isolated scopes ensure that the results would be no different if all conflicting activities (read/write and write/write activities) on all shared variables and partner links were conceptually reordered so that either all such activities within S1 are completed before any in S2 or vice versa.

- Concurrent scope with compensation
- Non-deterministic behaviour due to cascading compensation
- Non-termination
- Data (infiniteness) handling
- Properties (common to every service)
- …