

Analysing Web Service Composition with PEPA

Jane Hillston
LFCS, Edinburgh.
Joint work with Bryce Mitchell

PASTA'04, 11th June 2004

Overview

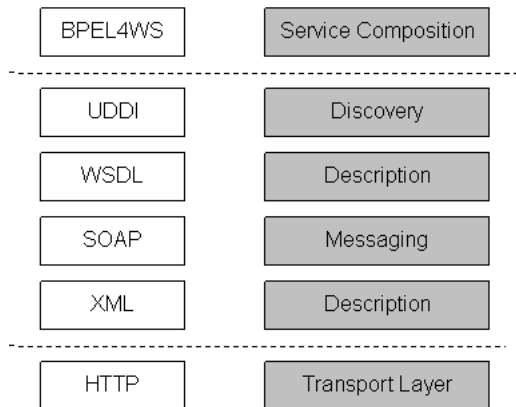
- ▶ Web Services, BPEL4WS and WSDL
- ▶ Mapping to PEPA
- ▶ Example
- ▶ Conclusions

Web Services

- ▶ An emerging paradigm for developing distributed applications.
- ▶ The **interoperability** associated with web applications combined with **rich client interaction** associated with distributed object systems such as CORBA.
- ▶ Web service **composition** or **choreography** constructs a distributed application from a number of previously published web services.

The Web Service Stack

Specifications



BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS or simply BPEL) is an XML-based language intended to specify an application as the composition of a number of web services.

It captures the different organisations or **partners** involved in the process; **ordering constraints** between operations provided by different web services and how **data** is shared between web services.

May be regarded as an **abstract** or an **execution** process.

BPEL

There are four key sections in a BPEL service definition:

Process: root element of the document providing the process name and other information.

Partner Links: record the information about interactions between partners.

Variables: used to store intermediate values that relate to the state or history of the process.

Fault Handlers: contingency activities in the event of a failure.

We currently essentially ignore variables and fault handlers.

BPEL (2)

A number of activities are provided in the language. Some of these, such as `<assign>` and `<receive>` are **atomic**.

There are also **structural** activities: namely, `<sequence>` `<flow>` and `<pick>` These activities record the constraints between operations within a business process.

Mapping BPEL to PEPA

- ▶ Each BPEL atomic activity is mapped to a PEPA activity of the same name (operations) or a derived name (internal activities).
- ▶ Each partner is mapped to a PEPA component.
- ▶ An additional PEPA component is used to represent the BPEL process itself. Data and state handling activities are attributed to this component
- ▶ The interactions within the process are captured in a system equation.

Mapping structural activities

In summary,

`<sequence>` maps to a sequence of prefix operations;

`<pick>` maps to choice; and

`<flow>` within a partner maps to a sequence of prefix operations.

This is based on an assumption that each partner runs on a single-threaded processor. A flow indicates that there is no causal ordering between the activities involved. Thus an arbitrary order is chosen for the activities present and they are mapped to a sequence of prefix operations accordingly.

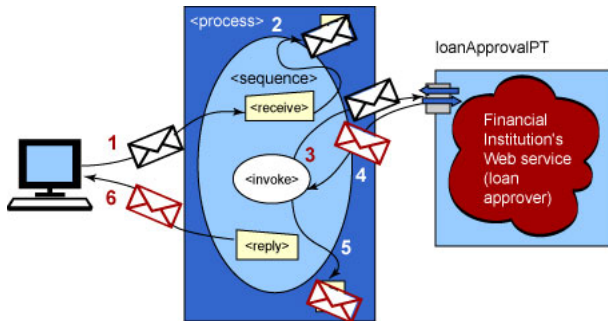
Parameterisation via WSDL

- ▶ A PEPA model combines timing information (in the form of rates) with functional information.
- ▶ The BPEL specification gives us the structure or software architecture of the system (functional information) but nothing about the timing information.
- ▶ The most logical place for such information seems to be in the WSDL specification of each operation.
- ▶ Thus the schema was enriched with an optional **latency** for each offered service.
- ▶ The rate is then simply calculated as $1/\textit{latency}$.

Implementation

- ▶ The mapping is implemented in a Java tool, which has, embedded within it, a copy of the PEPA workbench.
- ▶ The JAXB framework is used to access the data held within the BPEL document via a set of Java classes.
- ▶ An internal representation of the PEPA model is then built.
- ▶ This is then used to write the PEPA syntax to a file which can be loaded into the PEPA workbench.
- ▶ By default all rates are set to the value 1.0 but these may be overridden when a corresponding latency is defined in the WSDL specification of an operation.

Example: The Loan Approval Service



In this process we have the customer and the two web services provided by the financial institution: *loan assessor* and *loan approver*.

The Loan Approval Service (2)

In BPEL each of these are represented as a *partner*

- ▶ The process waits to receive a request.
- ▶ The process invokes a web service to perform a risk assessment based on the application details
- ▶ The result of the assessment is assigned to an internal variable within the process
- ▶ A further web service is invoked to decide if the application should be approved
- ▶ The result is sent back to the customer

Partners in the Loan Approval Process

```
<partnerLinks>
  <partnerLink name="customer"
    partnerLinkType="lns:loanApprovalLinkType"
    myRole="approver"/>
  <partnerLink name="approver"
    partnerLinkType="lns:loanApprovalLinkType"
    partnerRole="approver"/>
  <partnerLink name="assessor"
    partnerLinkType="lns:riskAssessmentLinkType"
    partnerRole="assessor" />
</partnerLinks>
```

Activities in the Loan Approval Process

```
<sequence>
  <receive name="receive1" partnerLink="customer" portType="apns:loanApprovalPT"
    operation="approve" variable="request" createInstance="yes" />
  <invoke name="invokeAssessor" partnerLink="assessor" portType="asns:riskAssessmentPT"
    operation="check" inputVariable="request" outputVariable="riskAssessment" />
  <assign name="assign">
    <copy>
      <from expression="'yes'" />
      <to variable="approvalInfo" part="accept" />
    </copy>
  </assign>
  <invoke name="invokeapprover" partnerLink="approver" portType="apns:loanApprovalPT"
    operation="approve" inputVariable="request" outputVariable="approvalInfo" />
  <reply name="reply" partnerLink="customer" portType="apns:loanApprovalPT"
    operation="approve" variable="approvalInfo" />
</sequence>
```

Annotated WSDL for Risk Assessment Web Service

```
<portType name="riskAssessmentPT">
  <operation name="check" latency="4">
    <input message="loandef:creditInformationMessage"/>
    <output message="tns:riskAssessmentMessage"/>
    <fault name="loanProcessFault"
      message="loandef:loanRequestErrorMessage"/>
  </operation>
</portType>
```

The generated model

```
/*
 * This file is autogenerated from a BPEL4WS document
 * by s0094815 using bpel2pepa
 * on Mon May 24 22:24:05 BST 2004
 */
BPEL = (assign,1.0).BPEL;
CUSTOMER = (receive1,1.0).CUSTOMER+(reply,1.0).CUSTOMER;
APPROVER = (invokeapprover,1.0).APPROVER;
ASSESSOR = (invokeAssessor,0.25).ASSESSOR;

P = (receive1,1.0).(invokeAssessor,0.25).(assign,1.0)
  .(invokeapprover,1.0).(reply,1.0).P;

P<receive1,invokeAssessor,assign,invokeapprover,reply>
  (BPEL||CUSTOMER||APPROVER||ASSESSOR)
```

Steady state solution for Loan Approval Process

1	0.125000000000000022
2	0.500000000000000007
3	0.124999999999999997
4	0.124999999999999993
5	0.124999999999999997

From this we could obviously easily derive measures such as the throughput of the various activities in the BPEL model.

Conclusions

- ▶ A proof-of-concept implementation has been completed, using both BPEL and (annotated) WSDL.
- ▶ Parameterising the model has several outstanding problems:
 - ▶ Communication delays are currently ignored;
 - ▶ The WSDL annotations give a static/"effort" view of each operation — instead need to take into account current capability of service.
 - ▶ The default setting of local operations needs to be more realistic.
- ▶ A reflector is needed to provide the results of the analysis in a form which is comprehensible to a BPEL modeller.