WS-BPEL Process Compiler for Resource-Constrained Embedded Systems

Hendrik Bohn, Andreas Bobek, Frank Golatowski
University of Rostock
Institute of Applied Microelectronics and Computer Engineering

3rd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments
SOCNE 2008

In conjunction with
IEEE 22nd International Conference on Advanced Information Networking and Applications (AINA-08)

March 25-28, 2008
Ginowan, Okinawa, Japan
0. Overview

• Motivation for process management on embedded systems
• State of the art
  • Criteria for the evaluation
  • Evaluation results
• WS-BPEL Process Compiler
  • Intermediate XML Process Description
  • Process logic in Java
  • Generation of skeleton and stubs
• Prototyping
• Conclusions and future work
1. Motivation – The innovation lies in Integration!

- A lot of heterogeneous devices from different manufacturers need to be connected and interact with each other.

- **Connection**: SOA approach forms a solid foundation but

- **Interaction**: Composition and automation of services/components will reveal the full potential.
1. Motivation – Problem!

• Connectivity between heterogeneous devices is provided by several SOA approaches

• Composition standards **only** specified for Web services being provided by software components (e.g. WS-BPEL)

**Conclusion:**
WS-BPEL for the Devices Profile for Web Services
→ Extending WS-BPEL
→ Implementing a compliant WS-BPEL engine
1. Motivation – Web Services Business Process Execution Language

- WS-BPEL allows the structured programming of processes (condition statements, loops, parallelism)
- Process is an executable Web service with its own WSDL being able to invoke other Web services, callback functionality
- Event, fault and compensation handling as well as data manipulation
- Data manipulation, callback functionality
2. SotA – Criteria for evaluation

- WS-BPEL enhancements and extension might result in changes to the Web services stack resp. WS-BPEL engine and vice versa – therefore,

- Effort should be minimized by using existing solutions/technologies and changing fewest layers,

- Legacy devices and environments must be supported, as well as

- Resource-constrained and embedded devices,

- Must be easily extensible,

- Process should be provided as Web services or as a stand-alone program.
2. State of the Art – Evaluation

(1) WS-BPEL Engines

• Several engines available – e.g. Oracle BPEL Process Manager, IBM WebSphere, ActiveBPEL (open source)

• Strong requirements on underlying hardware: Pentium 300 MHz – Pentium III 500 MHz, 256 – 512 MB RAM, 20 MB – 2.8 GB HDD

(2) Toolkits for the Devices Profile for Web Services

• Two toolkits: Java/Apache Axis 2, C/gSOAP

• Require transformation of WS-BPEL either in Java or C

(3) BPEL to Java Engines

• Model transformation vs. XML parsing:
  • XML parsing simpler
  • Document Object Model (DOM) better choice than Simple API for XML (SAX) as XML documents have to be traversed in any direction
2. State of the Art – Evaluation

(3) BPEL to Java Engines

- Model transformation vs. XML parsing
- BPEL2Java – subproject of SOA Tools Platform Project (Eclipse)
  - BPEL is bound to Java and not transformed
  - Underlying engine required which supports the binding
- T-BPEL – based on BPEL4WS and Apache Axis 2
  - Policy-based transactions
  - Transformation not known

Conclusion:

**Own engine** based on the Java/Apache Axis2 stack for the Devices Profile for Web Services (DPWS)
3. WS-BPEL Process Compiler

- Two stage transformation for WS-BPEL to Java
  - Intermediate XML process description
  - Generation of process logic in Java
- Apache Axis2 code generation
  - Stubs of invoked Web services
  - Skeleton of the process to be invoked

- Incorporates all process relevant information into one XML document (e.g. data type definitions, variable definitions, namespace resolution)
- Extracts only the information being relevant to the process (e.g. other portTypes are not transferred, concrete binding is ignored)
- Avoids parsing through different documents at a later stages

- Document structure is oriented on WS-BPEL process description
- DOM parser is used to build the document structure
- Intermediate XML Process Description can also be used for other transformations
- Example describes the transformation of an variable

```xml
<namespaces>
    <namespace prefix="ns1" uri="http://ws4d.org/Service/" />
</namespaces>
<variables>
    <variable namespace="ns1" name="Data" varType="element" complexType="sequence">
        <variable namespace="ns1" name="Id" type="xsd:string"/>
        <variable namespace="ns1" name="Sensor" type="xsd:int"/>
    </variable>
</variables>
```
3. Process Compiler – Process logic in Java

- Transformation rules defined in XSLT and naming rules by Apache Axis2
- XSLT – simple way to transform XML into Java, easily adaptable to other programming languages
- Process logic implements the method being called when the process is invoked
- Transformation procedure:
  - Global Java variables for names of classes, packages, methods etc.
  - Definition of process variables and their data types
  - Process is transformed activity by activity
3. Process Compiler – Generation of the skeleton and the stubs

- Apache Axis2 responsible for de-/marshalling of SOAP messages
- Generates empty process skeleton and the stubs of the Web services being invoked by the process
- Invocation activities are transformed into calling corresponding stub method
3. Process Compiler – Example

```xml
<bpel:variable element="ns1:Data" name="DataRequest"/>

<xsd:element name="Data">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Id" type="xsd:string"/>
      <xsd:element name="Sensor" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<namespaces>
  <namespace prefix="ns1" uri="http://ws4d.org/Service/" />
</namespaces>

<variables>
  <variable namespace="ns1" name="Data" varType="element" complexType="sequence">
    <variable namespace="ns1" name="Id" varType="element" type="string" />
    <variable namespace="ns1" name="Sensor" varType="element" type="int" />
  </variable>
</variables>

Data data = new Data();
String id;
Int sensor;```
4. Prototyping

- First prototypes were implemented with the activities Receive, Reply, Assign and Invoke.
- Transformation is based on JDOM for DOM parsing as well as XSLT 1.0 and SAXON SA 8.9 for Java for XSL Transformations.
- All requirements satisfied:
  - Used existing technologies
  - Changed only the Engine layer
  - Targeted resource-constrained devices by building on top of DPWS – low footprint (about 20 MB HDD & RAM)
  - Easily extensible by enhancing the transformation rules in XSLT.
5. Conclusions and future work

- WS-BPEL does not support following properties of DPWS:
  - Integration of devices – solved
  - Discovery of devices and their services (WS-Discovery) – will be presented at the ICSSSM’08, Melbourne, Australia
  - Publish/Subscribe mechanism (WS-Eventing) – will be finished in April
  - Protocol independent addressing (WS-Addressing) – BPEL uses an own mechanism (Correlation Sets) – will be finished in April
  - Validating QoS requirements (WS-Policy) – is planned

- DPWS does not support following properties of WS-BPEL
  - Asynchronous messaging – nevertheless support has been integrated by using basic functionality of Apache Axis2
Which questions are arising?

Thank you for your attention!

Hendrik Bohn
University of Rostock
Institute for Applied Microelectronics and Computer Engineering
hendrik.bohn@uni-rostock.de