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
18051 Rostock, Germany
{hendrik.bohn, andreas.bobek, frank.golatowski}@uni-rostock.de

Abstract

Process management and workflow systems play an important role in the composition of services in business as well as automation environments. Processes are designed using tools and deployed on a process management engine which control their execution. Unfortunately, the extensive requirements of process management engines on the underlying hard- and software often exceeds the limits of the resources of embedded systems in terms of memory and processing power.

This paper proposes an approach of compiling processes to executable programs with very small footprints which can also be run on embedded systems with limited resources. This approach uses the WS-BPEL 2.0 specification for designing the process, XSL transformation and the Apache Axis2 Java architecture for the code generation being flexible and adaptable to future extensions and enhancements of WS-BPEL.

1. Introduction

The demand for flexible applications in industrial automation leads to the adaptations of approved technologies from other areas. Business process management systems are an adequate approach to compose numerous heterogeneous components from different vendors to new applications in an adaptable way. This approach is also suitable for industrial automation where a large number of hard- and software components have to be integrated into changing environments.

Business process management (BPM) [16] takes care of the planning, modeling, executing and controlling of processes and workflows, respectively. It bridges the gap between business and IT by offering an outside view which is oriented on the interaction process of components rather than the performance of participating components. Participating components are ”black boxes” for the process designer. BPM reacts on the adaptability requirements of constantly changing environments. BPM requires a standardized communication and standardized interfaces to involved components which are often very heterogeneous and from different vendors. Process definitions in the area of IT have to be machine-readable to enable automation.

The design paradigm of *Service-Oriented Architectures* (SOA) [10] provide an ideal approach for the modeling of components taking part in a BPM environment. SOA concentrates on the autonomy and interoperability of components whose self-described, standardized interfaces to their functionality are called *services*. The SOA implementation with the highest market penetration today are Web services. *Web services* [8] are a set of modular protocol building blocks which can be composed in varying ways to suit a certain application. Web service protocols are XML-based and define the description of services (WSDL), data types (XML Schema), transport issues as well as the representation of meta data about the services and their interactions.

Since 2003 the Organization for the Advancement of Structured Information Standards (OASIS) governs the standard *Web Service Business Process Execution Language* (WS-BPEL) which is currently available as version 2.0. WS-BPEL [3] is an XML format to describe machine-readable processes which is based on Web services standards. WS-BPEL processes can be designed using graphical tools and can be executed on a WS-BPEL engine. Unfortunately, available WS-BPEL engines have strong requirements on underlying hard- and software [11]. The operation on embedded systems with constrained resources is very limited.

This paper proposes a compiler for WS-BPEL processes generating stand-alone process programs which execute and control corresponding processes. The process execution can be started manually or due to a WS-BPEL process service invocation. The compiler is designed using a DOM parser, XSLT and the Apache Axis2 architecture.

The paper is organized as follows. Section 2 describes

the basics and related work for this paper. The design of the compiler is presented in section 3. A prototypical implementation is discussed in section 4 together with a comparison to the requirements of currently available WS-BPEL engines. Section 5 briefly presents the application of proposed compiler in the ongoing European ITEA2 research project LOMS. In section 6 conclusion is drawn and future work in this area is presented.

2. Preliminaries and related work

2.1. Web Services Business Process Execution Language (WS-BPEL)

The *Web Services Business Process Execution Language* (WS-BPEL) [3] is an OASIS standard and specifies an XML-based language based on Web services standards which is used to describe Web service processes. A WS-BPEL process is a composed Web service reflecting the interactions (called *activities*) between other Web services. It is stored in an XML format and also possesses a separate WSDL description such as any other Web service. The WSDL document contains all relevant information for the execution of a process such as the description of the service interface of the process and relevant interface descriptions of the Web services participating in the process execution.

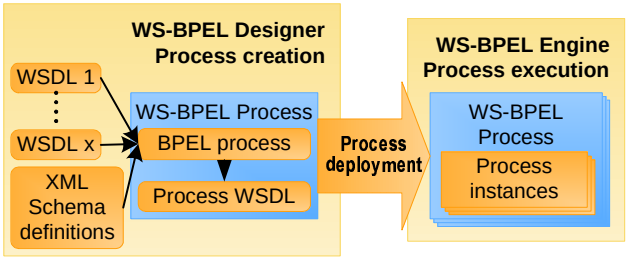

Figure 1. Process interpreted by a WS-BPEL engine.

Figure 1 shows the creation, deployment and execution of a WS-BPEL process. WS-BPEL process can be designed using visual tools and thereby allow the ”Programming in the large” [21] also for non-technical personal. The WSDL description of participating Web services are used to define the interaction interfaces and the XML Schema definitions specify the used data types. At the end of the process creation the WS-BPEL process as well as the WSDL description for the process are deployed on a WS-BPEL engine. Every time the process is requested by a Web service invocation a new instance of the corresponding process is started on the WS-BPEL engine. The WS-BPEL engine takes care of the execution and control of deployed WS-BPEL processes.

A WS-BPEL process supports sequential and parallel activities, loops, validity scopes, event and fault handling. WS-BPEL also supports the compensation of all previously completed activities in case an error occurs.

2.2. Apache Axis2 architecture

Apache Axis2 [24] is an open source project being the successor of Axis – a Java-based SOAP processor for Web services. Axis2 is a modularly built SOAP stack in which additional functionality can be plugged in via modules containing handlers. When SOAP messages arrive and leave these handlers are called sequentially which in turn may read and write SOAP headers and inject values in a context object. This object is visible to other handlers which may adapt their behavior according to the current context state.

The main idea behind Axis2 extension architecture is to provide a framework in which other Web services protocol implementations can be easily integrated. Since we are currently working on extending BPEL by dynamic discovery capabilities we chose Axis2 as the underlying platform.

Further Axis2 has a built-in code generation engine (also available as Eclipse plug-in) which is highly adaptable and generates either Java code from a WSDL document or vice versa.

2.3. Model transformation

The presented work is following a *Model-Driven Engineering* (MDE) approach. MDE technologies combine domain-specific modeling languages (in this case WS-BPEL) with transformation engines and generators (WS-BPEL to Java code generator) and is often referred to as ”correct-by-construction” [22]. A variant of MDE proposed by the Object Management Group (OMG) is the *Model Driven Architecture* (MDA) [17]. The aim of MDA is to bridge the gap between models and source code through the definition and implementation of *model transformations* [6] between Meta Object Facility (MOF) compliant languages. However, instead of focusing on fully automated code generation from models to a complete source code as suggested by CASE tools, MDA targets to automated reasonable parts of it only. Model transformation improves productivity, portability, interoperability, maintenance and reusability in software engineering. The OMG has developed the *Queries, Views and Transformations* (QVT) standard [19] to specify transformations between a source and a target model. QVT is a very complex standard and compliant model transformations are rare. A good example of a programming language for model transformation is *Atlas*

Transformation Language (ATL) [4] which is very close to the QVT standard.

There are several works for transforming BPEL models into other models. Among those, Reiter has presented an approach for the transformation of BPEL into UML Activity Diagrams [20] and Hinz, Schmidt and Stahl have proposed BPEL transformation into Petri Nets for verification purposes [12].

2.4. XML parser

There are two widely used interface for accessing and parsing an XML document. The *Document Object Model* (DOM) builds an internal tree representation of the XML document [13]. The DOM API offers functionality to read and manipulate the elements of the tree structure. The *Simple API for XML* (SAX) follows an event based approach [9]. There are event handlers for each information set of an XML document. Whenever an information set is parsed the corresponding event handler is called.

SAX is fast and does not require an internal representation. However, every information set is only parsed once whereas DOM can be used to traverse the tree in any direction any time. As specific information in a WS-BPEL document is interlinked with other in the same document or associated WSDLs, DOM parsing is preferred in this work.

2.5. XSL Transformation

The *Extensible Stylesheet Language Transformation* (XSLT) [15] is a Turing-complete language for the transformation of XML models/documents into other (mostly XML) documents according to defined transformation rules. The transformation is based on the tree-structure of an XML document. XSLT documents are also XML conform. *XSLT processors* apply the transformation rules defined in an XSLT document and generate the new document. XSLT uses *XPath* [5] to address portions of an XML document (element, attribute and text nodes) and offers mechanisms (*predicates*) to manipulate values based on the content of an XML document (strings, numbers and boolean values).

As WS-BPEL processes are XML structures, XSLT and DOM parsing are better approaches for transforming WS-BPEL documents into Java programs than QVT or ATL.

2.6. BPEL to Java transformation

The *BPEL to Java* project is a subproject of the SOA Tools Platform Project for Eclipse (originally a subproject of the Eclipse Test and Performance Tools Platform) [18]. It generates Java code from WS-BPEL and creates engines which can be executed locally or in a distributed manner. The underlying engines can be replaced by different implementations. BPEL2Java is based on its own Java binding. Operations, messages, port types etc. of involved Web services are bound to Java as defined by BPEL2Java.

Tai et al. have introduced a middleware prototype for policy-based transactions using BPEL (called T-BPEL) as part of their research which is also able to execute BPEL processes [23]. It is based on an earlier version of WS-BPEL (called BPEL4WS) and Apache Axis (the predecessor of Apache Axis2). The way the BPEL code is transformed into Java is not known to the authors.

2.7. Comparison of BPEL environments

Ruf and Strotbek (2006) [11] have compared numerous BPEL engines in terms of installation and configuration, support, portability, hard- and software requirements in a survey. Figure 2 shows an excerpt of the survey illustrating the requirements of currently available BPEL engines on the underlying hardware.

Engine	Processor	RAM	HDD
Oracle, *BPEL Process Manager*	300 MHz	256 MB	600 MB
Cape Clear, *Enterprise Service Bus*	n/a	512 MB	400 MB
JBoss, *JBPM BPEL Extension*	400 MHz	512 MB	20 MB
Telelogic, *System Architect*	500 MHz Pentium	256 MB	350 MB
IBM, *Web Sphere*	500 MHz Pentium 3	512 MB	2.8 GB

Figure 2. Hardware requirements of a subset of available BPEL engines.

It should be noted that most BPEL engines also rely on specific underlying software such as Web servers, runtime engines or databases. From the survey can be derived that the application of BPEL engines on resource-constrained embedded systems is rather limited [7].

3. Design of the WS-BPEL compiler

The way a compiled WS-BPEL process is created is similar to the creation of a WS-BPEL process executed on an engine. The difference starts after the process creation. Instead of deploying the process onto the engine a stand-alone program is created. The compiler can be adjusted to either create a stand-alone Web service for the process or a program executing the process immediately when it is started.

The compilation process involves three stages (shown in figure 3): Integrating all information about the process into

one intermediate XML process description, compilation of the BPEL process to Java and the code generation of the involved stubs and process skeleton for the marshaling and de-marshaling of corresponding SOAP messages.

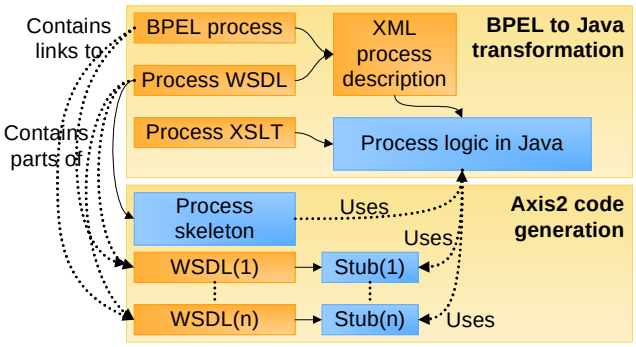

Figure 3. Transformation of a BPEL process.

3.1. Generation of the intermediate XML process description

In this step, an intermediate XML document (*XML process description*) is created which includes all information required for the process as some information are defined in associated WSDL documents (e.g. variable definitions and Web service invocation bindings). A WS-BPEL process has links (imports) of all WSDL documents it uses for Web service invocations. The process WSDL contains information about invoking the process and replying results when the process completes.

The structure of the XML process description is oriented on the structure of the WS-BPEL process description. Namespace definitions and all declarations of variables (used in the process) are added by parsing through the corresponding WSDL documents and extracting required information. Variables defined in XML Schema are resolved and represented as XML elements describing the individual variables. The intermediate process description is formed using a DOM parser. Thereby, the semantic correctness can be validated, namespace definitions and variable definitions are resolved.

The XML process description can be used to compile the process into different kind of programming languages and for different SOAP engines. This work deals with the compilation for Java based on Axis2 only.

3.2. Compilation of the BPEL process

This step is generating a Java representation of the computation logic of the BPEL process compliant to Apache Axis2. XSLT is an adequate and simple way to transform the intermediate XML process document into Java. It is very flexible, extensible and can be easily adapted to other programming languages. The Java computation logic implements the Java method which will be called by Apache Axis2 when the process is invoked.

All transformation rules are defined in the *Process XSLT* document which are applied by the XSLT processor. Compilers for different programming languages and SOAP engines have to adapt the transformation rules accordingly. Firstly, all global variables for the Java class and method names are generated from the service, operation and message names of the XML process description. The naming rules follow the syntax defined by the Apache Axis2 code generator. These names are used by the XSLT processor in subsequent steps. Secondly, all process variables and their data types are transformed into corresponding Java variables and data types. Each process variable is reflected by local Java variables. Finally, all activities are transformed into Java methods reflecting the execution rules of the process. Web service invocation activities are transformed into statements calling invocation methods of corresponding service stubs.

```
<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;
```

Figure 4. Variable transformation example: Variable declaration in WS-BPEL, corresponding WSDL, intermediate XML document and Java.

Figure 4 shows the transformation of a variable declaration. The variable `DataRequest` is declared in WS-BPEL and defined in the corresponding WSDL as an ele-

ment called `Data`. `Data` is a XML Schema complex type. The namespace of `Data` is added to the intermediate XML process description as well as the variable declarations for `Data` and its embedded elements. Thereby, the XML process description include all details which are needed for the code generation using XSLT. The rules for the XSL transformation are defined by Apache Axis2: The class `Data` is generated for the complex type, `String` and `int` for the simple types. The Java variable definitions are generated using XSLT and injected into the skeleton in the next step.

The resulting logic is imported into the corresponding Java method of the process skeleton.

3.3. Generation of stubs and skeleton

Apache Axis2 deals with the marshaling and de-marshaling of SOAP messages between the process and contributing Web services. It generates an empty skeleton for the process service from its process WSDL description. This skeleton includes a method (*process invocation method*) which is called when the service is invoked. For a manual startup the process invocation method is explicitly called by a starting program. The logic behind the process invocation method is generated as described in the previous subsection and entered into the process invocation method. Apache Axis2 also generates the stubs for the Web services contributing to the process. The desired stub is called from the process invocation method whenever an invocation activity is performed.

4. Prototyping and comparison to previous approaches

A first prototype was implemented being able to transform processes consisting of the activities `Receive`, `Reply`, `Assign` and `Invoke`. These four activities are sufficient to design simple processes. The transformation rules are based on XSLT 1.0, and Saxon SA 8.9 for Java [14] was used as XSLT processor. Saxon SA is a commercial XSLT 2.0 processor. It was selected due to its support of XML Schema which might be important in future versions of the prototype.

The implementation of a all four activities reached a size of about 20 MB (HDD and RAM) only requiring a Java runtime environment. It is important to note that almost the entire size is used by the Axis2 SOAP engine for the marshaling and de-marshaling of SOAP messages. The size marginally grows with the number of activities in a BPEL process. Whenever a new instance of the process is created a new thread is started by the Axis2 SOAP engine which handles the process instance. This increases the required RAM by a few KB for each additional process instance depending on the called activities. Invocations also compile the require stubs of invoked Web services into the same skeleton (at design-time) and thereby use the same engine. Each invocation increases the required RAM and HDD by a few KB depending on the functionality.

In comparison to currently available WS-BPEL engines this is very small footprint. Although a large number of process instances are possible, only a small number of different executed processes are possible on an embedded system. This is due to the fact that every Web service – and thereby also processes – based on Axis2 needs its own SOAP engine. Long living transaction are also supported by Axis2 and thus also by the BPEL compiler.

Furthermore, the transformation rules in the XSLT document can be easily adapted to other SOAP environments which might be even smaller than Apache Axis2.

5. Demonstration of the prototype

In the European ITEA2 project LOMS presented process compilation is used for a maintenance scenario [1]. A robot in a factory reports its failure to the factory control center which relays it to the responsible robot maintenance provider. The Customer Relationship Management (CRM) of the maintenance provider creates a service case and calls a service technician. The service technician retrieves all relevant information about the service case such as manuals, location of the robot and forms. Also routing information can be provided to the service technician in order to repair the faulty robot.

Processes in LOMS have been designed using the ActiveBPEL Designer – a graphical tools for the creation and simulation of processes based on WS-BPEL [2]. ActiveBPEL generates the process WSDL and the WS-BPEL code for the process which is injected into presented BPEL code compiler.

Without process management the interactions between participating components (e.g. CRM, factory control center, service technician) have to be implemented for each component separately, although Web service interfaces provide a common basis for interactions between them. Available process management engines can be used to easily design interactions and processes but they are often expensive (especially for small and medium enterprises) and require additional hard- and software. The presented approach offers the most functionality of dedicated engines, can also be used for process management in industrial automation where size matters and can be easily adapted to suit other (existing) hard- and software environments. Furthermore, the BPEL code can be extended in such a way that other Web service specifications are supported. This is facilitated by the code generator as new transformation rules can be added easily.

6. Conclusion and future work

The paper presented an ongoing research work on a compiler based on WS-BPEL. A simple, flexible and extensible approach of using XSL transformation for the compiling process was described and an early prototype of the compiler was presented. BPEL processes can be started by a Web service invocation such as supported by any WS-BPEL engine but also manually which offers new application areas for WS-BPEL in automation environments. The presented compiler will be extended to support the Devices Profile for Web Services (DPWS) in order to address the needs of Web service enabled embedded systems. Currently, further activities are designed and implemented for XSL transformation in order to develop a prototype being fully compliant with WS-BPEL 2.0. The implementation of presented work will be published on the website http://www.ws4d.org.

The simplicity of presented approach is very useful for testing and demonstration purposes of extensions applied to WS-BPEL. Additionally, there is great potential that this approach will replace WS-BPEL engines in smaller process environments.

Future research will deal with extensions to WS-BPEL and their implementations using the BPEL compiler.

7. Acknowledgment

This work has been funded by German Federal Ministry of Education and Research (BMBF) under reference number 01SF11H as part of the ITEA2 project LOMS.

References

[1] LOMS: Local Mobile Services. http://www.loms-itea.org/, 2007.

[2] Active Endpoints. ActiveBPEL Designer 4.1. http://www.active-endpoints.com/active-bpel-designer.htm, 2007.

[3] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. *Web Services Business Process Execution Language Version 2.0*. OASIS Standard, 2007.

[4] ATLAS group–LINA & INRIA. ATL: Atlas Transformation Language – ATL Starter's Guide – version 0.1, 2007.

[5] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. *XML Path Language (XPath) 2.0*. W3C Recommendation, 2007.

[6] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow. Model Transformations? Transformation Models! In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006*, Genova, Italy, 2006.

[7] H. Bohn, F. Golatowski, and D. Timmermann. Sind Prozessmanagement-Systeme auch auf eingebetteten Systemen einsetzbar? *12. Symposium Maritime Elektrotechnik, Elektronik und Informationstechnik, Rostock, Germany*, October 2007.

[8] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. *Web Services Architecture*. W3C, 2004.

[9] D. Brownell. *SAX2*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[10] T. Erl. *Service Oriented Architecture: Concepts, Technology, and Design*. Pearson Education Inc., Upper Saddle River, New Jersey, USA, 2005.

[11] R. Hantschel, F. Ruf, and H. Strotbek. Vergleich von BPEL Laufzeitumgebungen. *Fachstudie Nr. 54, Institut für Architektur von Anwendungssystemen, Universität Stuttgart*, January 2006.

[12] S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In *Third International Conference on Business Process Management (BPM 2005)*, Nancy, France, 2005.

[13] A. L. Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. *Document Object Model (DOM) Level 3 Core Specification*. W3C Recommendation, 2004.

[14] M. Kay. Saxon-SA 8.9.0.4 for Java. http://saxon.sourceforge.net/, 2007.

[15] M. Kay. *XSL Transformations (XSLT) Version 2.0*. W3C Recommendation, 2007.

[16] M. Keen, G. Ackerman, I. Azaz, M. Haas, R. Johnson, J. Kim, and P. Robertson. *Patterns: SOA Foundation – Business Process Management Scenario*. IBM International Technical Support Organization, Armonk, New York, USA, 2006.

[17] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture–Practice and Promise*. Addison-Wesley Professional, Boston, MA, USA, 2003.

[18] A. Miguel. Developer Tutorial (How to build on B2J), 2006.

[19] OMG. MOF QVT Final Adopted Specification, 2005.

[20] T. J. Reiter. Transformation of Web Service Specification Languages into UML Activity Diagrams. Master's thesis, Johannes Kepler University of Linz, Linz, Austria, 2005.

[21] F. Ryan, D. König, and C. Barreto. WS-BPEL Technical Overview for Developers and Architects, Presentation, 2007.

[22] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer*, 2006.

[23] S. Tai, T. Mikalsen, E. Wohlstadter, N. Desai, and I. Rouvellou. Transaction policies for service-oriented computing. *Data & Knowledge Engineering 51, Elsevier B.V.*, 2004.

[24] The Apache Software Foundation. Apache Axis2. http://ws.apache.org/axis2/, 2007.