Facilitating Rich Data Manipulation in BPEL using E4X

Tammo van Lessen, Jörg Nitzsche, and Dimka Karastoyanova

Institute of Architecture of Application Systems
University of Stuttgart
Universitaetsstrasse 38, 70569 Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de

Abstract. The Business Process Execution Language (BPEL) uses XML to specify the data used within a process and realizes data flow via (globally) shared variables. Additionally, assign activities can be used to copy (parts of) variables to other variables using techniques like XPath or XSLT. Although BPEL's built-in functionality is sufficient for simple data manipulation tasks, it becomes very cumbersome when dealing with more sophisticated data models, such as arrays. ECMAScript for XML (E4X) extends JavaScript with support for XML-based data manipulation by introducing new XPath-like language features. In this paper we show how E4X can help to significantly ease data manipulation tasks and propose a BPEL extension that allows employing JavaScript/E4X for implementing them. As E4X allows defining custom functions in terms of scripts, reusability with respect to data manipulation is improved. To verify the conceptual framework we present a proof-of-concept implementation based on Apache ODE.

1 Introduction

Business Process Management (BPM) and the workflow technology [1,2] in particular have enjoyed a great success and have a heavy impact on industry and research. The separation of business process logic and implementation of business functions enables programming on a higher, i.e. business process-oriented level [3], and renders the workflows flexible. Currently, the language for executable business processes is the Business Process Execution Language (BPEL) [4] which is standardized by OASIS[1]. BPEL is XML based and is a part of the Web Service standard stack [5]. It uses XML as data model and specifies activity implementations using the Web Service Description Language (WSDL) [6].

Data flow in BPEL is not explicitly specified but can be realized using (globally) shared variables. Assign activities can be used to copy (parts of) variables to other variables using XML data processing techniques. Although arbitrary expression languages (e.g. XPath [7] and XSLT [8]) can be used to specify expressions to select and copy data, specifying the data manipulation is still a cumbersome task. For instance, it is not possible with the conventional use of BPEL to add elements into a node set (i.e. array operations) or to modify a certain value of a filtered node set.

[1] http://www.oasis-open.org/

ECMAScript for XML (E4X) [9,10] extends JavaScript with support for XML-based data manipulation by introducing new XPath-like language features. The resulting language provides convinient access to XML data and intuitive scripting primitives with direct support for e.g. array operations. In this paper we use this extension to improve BPEL with respect to its data manipulation capabilities and therefore we propose an extension to WS-BPEL 2.0 to allow defining variable assignments in terms of JavaScript/E4X expressions. For this we employ the extensibility features of BPEL, in particular, `<extensionActivity>` and `<extensionAssignOperation>`. This approach contributes a significant enhancement to data manipulation.

The paper is structured as follows. Section 2 provides background information about ECMAScript for XML. Subsequently, The BPEL extensions for E4X are presented in Section 3 and are explained by example in Section 4. Section 5 presents a proof-of-concept implementation based on Apache ODE. Finally, Section 6 discusses related work and Section 7 concludes the paper.

2 ECMAScript for XML (E4X)

ECMAScript for XML (E4X) is a language extension that adds native support for XML to the ECMAScript family [11] (including JavaScript, ActionScript, JScript etc.). Unlike other programming languages (like Java) that allow accessing XML data either as event stream or in terms of the W3C DOM object model, E4X allows processing of XML data directly on the language level. The XML tree can be navigated using an object-like "dot" notation and allows for addressing XML child elements, attributes and node sets. Node sets can be filtered using parentheses. The example in Listing 1 illustrates how an E4X object is created and how subsequently the values of the quantity attribute for all items with the name "SOA book" are retrieved. The example also shows how E4X can be used in *for* loops to sum up the prices of all items in the example shopping cart.

```
var items = <items>
     <item name="SOA book" price="40" quantity="2"/>
     <item name="BPM book" price="35" quantity="3"/>
     <item name="EAI book" price="30" quantity="1"/>
  </items>;

alert( items.item.(@name == "SOA book").@quantity );

for each( var thisPrice in items..@price ) {
  sum += thisPrice;
}
```

Listing 1. E4X sample code

E4X allows assigning data values not only to single XML nodes but also to node sets. This enables batch-like modifications of multiple nodes with a single assignment expression (see [12] for further details about E4X).

3 E4X Extension for BPEL

BPEL 2.0 introduces effective extensibility mechanisms that allow for defining new activity types (extension activities) as well as using different mechanisms for data manipulation (extension assign operations). Since E4X provides powerful language extensions to directly address and modify XML data, it makes a good candidate for significantly improving BPEL's data manipulation capabilities. The E4X extension for BPEL is defined in terms of an extension namespace and an extension element for both `<extensionAssignOperation>` and `<extensionActivity>` elements. The extension namespace[2] must be declared as a `mustUnderstand` extension in the preamble of the BPEL process model to ensure that BPEL engines can understand and execute E4X expressions. Subsequently, the `<js:snippet>` element can be used within assign and extension activities respectively and can contain arbitrary JavaScript code.

The E4X extension for BPEL comprises two main parts. First, it makes sure that all visible BPEL variables are injected into the JavaScript context so that they can be treated as normal E4X variables within the JavaScript code snippet. Second, it defines a number of functions that are necessary to glue both worlds together. These functions are listed in Table 1.

`load(string...)`	Allows importing reusable JavaScript libraries. That way, code snippets can be reused across JS/E4X extended activities.
`print(string...)`	Allows printing debug messages to the underlying engine's logging console.
`validate(BPELvariable)`	Makes sure that the given XML object complies with the variable declaration.
`throwFault(...)`	Creates a BPEL fault with a given QName and fault message.
`processName()`	Returns the name of the process model that is currently being executed.
`activityName()`	Returns the name of the activity that executes the JavaScript snipped that is currently being executed.
`piid()`	Returns the name of the activity that executes the JavaScript snipped that is currently being executed.

Table 1. E4X/BPEL built-in function list

[2] `http://ode.apache.org/extensions/js`

4 Example

As identified in Section 1 the most burdensome tasks are the (recurring) initialisation of variables and dealing with arrays (which always requires the use of external XSL scripts). In Listing 2 we demonstrate how E4X extension assign operations are utilized in BPEL. The first operation makes use of the string concatenation operator += and realises a typical *Hello World!* example. The second operation addresses the problems mentioned above. First, it loads an external, reusable JavaScript library, which contains helper methods to create and manipulate XML structures for our shopping cart example. Instead of manually assigning an XML skeleton to a BPEL variable and setting several values later on using XPath expressions, we can (re)use a shared method to create an empty shopping cart. In the last line we use a different JavaScript method to transform the values of the BPEL variable `item`, which was received from an external service, into the XML format prescribed by the shopping cart structure. Subsequently it is added to the virtual shopping cart (+= is the add operator on a node set).

```
<assign name="e4x-assign">
    <extensionAssignOperation>
        <js:snippet xmlns:js="http://ode.apache.org/extensions/js">
            myVar.TestPart += ' World';
        </js:snippet>
    </extensionAssignOperation>
    <extensionAssignOperation>
        <js:snippet xmlns:js="http://ode.apache.org/extensions/js">
            load('shoppingCartUtils.js');
            shoppingCart.parameters = createShoppingCartSkeleton();
            shoppingCart.parameters.items += createCartItem(item);
        </js:snippet>
    </extensionAssignOperation>
</assign>
```

Listing 2. JavaScript/E4X as extension assign operation implementation

Listing 3 demonstrates how to use JavaScript/E4X as extension activity implementation. We assume that the shopping cart has been transformed into a purchase order structure. Depending on the customer type (*gold*, *silver*, *besteffort*), we want to apply different discount ratios. After checking whether the ratios are within a reasonable range, the selected ratio is applied to all items, again by assigning values to a node set. In addition we set the shipping mode to a non-priority mode for best-effort customers.

5 Implementation

The concepts proposed above have been implemented as an extension to Apache ODE[3] and will be part of the upcoming ODE 2.0 release. It was originally intended to be a proof-of-concept implementation for the also newly introduced implementation of BPEL's

[3] `http://ode.apache.org`

```
<extensionActivity name="calculateDiscount">
  <js:snippet xmlns:js="http://ode.apache.org/extensions/js">
    if (goldRatio > 1.0 || silverRatio > 1.0) {
      throwFault('urn:myprocess', 'IllegalArgumentFault',
          'discount ratios must be <= 1.0');
    }
    if (customer.type == 'gold') {
      po.items.item.price *= goldRatio;
    } else if (customer.type == 'silver') {
      po.items.item.price *= silverRatio;
    } else if (customer.type == 'besteffort') {
      po.shippingMode = 'snailmail'
    }
  </js:snippet>
</extensionActivity>
```

Listing 3. JavaScript/E4X as extension activity implementation

extensibility mechanisms in ODE. It provides an *extension operation* implementation which integrates Apache ODE with Mozilla's Rhino[4] as the underlying JavaScript/E4X engine. Since the internal XML model of Rhino is not compatible with W3C's DOM model used by Apache ODE, it was necessary to implement a variable bridge that facilitates overlaying BPEL variables in the JavaScript context. This has been implemented in terms of a Rhino `Delegator`. The built-in functions are realized by overriding Rhino's `ImporterTopLevel`. The source code is available in Apache's Subversion repository[5].

6 Related Work

BPELJ [13] is an extension to BPEL that enables the use of Java code within BPEL activities. While the focus of BPELJ and the approach presented here is similar, BPELJ does not comply with the extensibility features of BPEL 2.0 yet. Furthermore, selecting an XML node in a DOM representation is still cumbersome since Java does not enable selecting XML nodes directly. Hence E4X for BPEL becomes a valuable extension that addresses this deficiency.

[14] proposes an extension to BPEL enabling data manipulation based on ontological knowledge. Using the semantics of data used in a process allows abstracting away the actual implementation of the data manipulation task. It is sufficient to describe which ontology concepts will be provided as input and what is expected as output to discover appropriate data mediators using ontology reasoning. Abstracting away the actual implementation of the data manipulation in processes completely frees process modellers from defining data transformation in a process model and increases reusability of data manipulation tasks. The downside of this approach is that performance decreases

[4] `http://www.mozilla.org/rhino/`
[5] `http://svn.eu.apache.org/repos/asf/ode/trunk/extensions/e4x`

because appropriate transformation/mediation implementations have to be discovered every time data has to be copied from one variable to another. Using the custom E4X functions however, enables defining highly performant and reusable data manipulation functionality.

7 Conclusion

Data manipulation in BPEL is based on XML data processing which makes it a cumbersome task. In this paper we have proposed a BPEL extension that allows employing Javascript/E4X for data manipulation tasks in BPEL and we have shown how E4X can help to significantly ease their implementation. Moreover, reusability with respect to data manipulation is improved as E4X allows defining custom functions in terms of scripts. To verify the conceptual framework we have presented a proof-of-concept implementation based on Apache ODE.

Acknowledgements

The work published in this article was partially funded by the SUPER project[6] under the EU 6^{th} Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

References

1. Leymann, F., Roller, D.: Production workflow. Prentice Hall (2000)
2. van der Aalst, W., van Hee, K.: Workflow management. MIT Press (2002)
3. Leymann, F., Roller, D.: Workflow-based applications. IBM Systems Journal **36**(1) (1997) 102–123
4. A. Alves et al.: Web Services Business Process Execution Language Version 2.0. Committee specification, OASIS (January 2007)
5. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR Upper Saddle River, NJ, USA (2005)
6. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001)
7. Clark, J., DeRose, S.J.: XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116 (November 1999)
8. Adler, S., Berglund, A., Caruso, J., Deach, S., Grosso, P., Gutentag, E., Milowski, R.A., Parnell, S., Richman, J., Zilles, S.: Extensible Stylesheet Language (XSL) Version 1.0. World Wide Web Consortium, Recommendation REC-xsl-20011015 (October 2001)
9. International Organization for Standardization: Information Technology — ECMAScript for XML (E4X) Specification. ISO/IEC 22537:2006 (February 2006)
10. Ecma International: ECMAScript for XML (E4X) Specification. Standard ECMA-357 (June 2004)

[6] `http://www.ip-super.org/`

11. Ecma International: ECMAScript Language Specification. Standard ECMA-262 (December 1999)
12. Tynjala, J.: E4X: Beginner to Advanced. `http://developer.yahoo.com/flash/articles/e4x-beginner-to-advanced.html`
13. Blow, M., Goland, Y., Kloppmann, M., Leymann, F., Pfau, G., Roller, D., Rowley, M.: BPELJ; BPEL for Java. Joint white paper by BEA and IBM (March 2004)
14. Nitzsche, J., Norton, B.: Ontology based data mediation in BPEL(4SWS). In: Proceedings of the Workshop on Semantics for Web Services (semantics4WS 2008), Milano, Italy (September 2008)