Composing Aggregate Web Services in BPEL

Technical Report FIU-SCIS-2005-10-01

October 2005

Onyeka Ezenwoye and S. Masoud Sadjadi

Autonomic Computing Research Laboratory
School of Computing and Information Sciences
Florida International University
11200 SW 8th St., Miami, FL 33199
{oezen001,sadjadi}@cs.fiu.edu

Abstract

Web services are increasingly being used to expose applications over the Internet. These Web services are being integrated within and across enterprises to create higher function services. BPEL is a workflow language that facilitates this integration. Although both academia and industry acknowledge the need for workflow languages, there are few technical papers focused on BPEL. In this paper, we provide an overview of BPEL and discuss its promises, limitations and challenges.

Keywords: Web services, workflow language, BPEL, business processes, application-to-application integration, and business-to-business integration.

1 Introduction

The rapid growth of the *electronic* commerce (e-commerce) has forced business organizations to offer their services electronically, not only to their end customers, but also to their business partners. In order to quickly respond to the ever changing business needs, enterprises need to electronically interact with one another, effectively integrating their fine-grained business functions (*e.g.*, accounting, production, inventory and delivery services) into more coarse-grained business processes (*e.g.*, a sales service) [9, 14]. Business functions are typically developed separately as software applications, possibly implemented in different programming languages and targeted to run on different platforms. Therefore, one key challenge that enterprises face when trying to interact electronically is how to convert the data and commands among their heterogeneous software applications.

The advent of middleware – which hides differences among programming languages, computing platforms, and network protocols [3, 5, 8] – in the 1990's, mitigated the difficulty of heterogeneous application integration. Indeed, the maturity of middleware technologies has produced several successful approaches to *corporate-wide* application integration [10,20], where applications developed and managed by the same corporation are able to interoperate with one another. The efficient integration of enterprise applications entails development that is rapid, utilizes existing applications and saves cost. Successful middleware technologies such as Java RMI [12], CORBA [17], and DCOM/.NET Remoting [7, 16] have been able to integrate corporate-wide applications.

Traditional middleware technologies are often unable to integrate applications managed by different corporations connected through the Internet. The reasons are twofold: (1) different corporations select different middleware technologies, which are more appropriate to integrate their own applications; and (2) middleware packets often cannot pass through Internet firewalls (see Figure 1). Also, the tightly coupled integration model of traditional middleware technologies creates inflexible application connections, making the development of *dynamic* cross-enterprise applications with such technologies arduous and expensive. Moreover, the difficulty of application integration, once alleviated by middleware, has reappeared with the proliferation of *heterogeneous* middleware technologies. As a result, there is a need for a "middleware for middleware" to enable Internet-wide and business-to-business application integration [21].

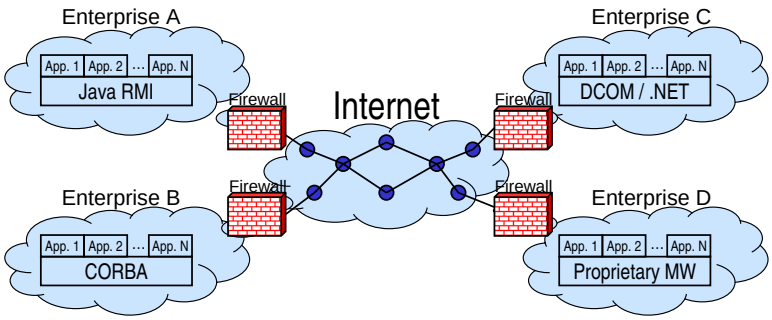

Figure 1: Middleware technologies used for corporate-wide application integration failed the Internet-wide application integration, because of heterogeneity of the middleware technologies and the presence of Internet firewall.

To simplify application-to-application communication, the Web service paradigm was invented. [4]. A *Web service* is a program delivered over the Internet that provides a service described in the Web Service Description Language (WSDL) [6] and communicates with other programs, typically through SOAP messages [11]. Web services provide the desired abstraction uniformity that is needed to bridge applications regardless of the heterogeneity of platforms and implementation languages. They provide a middleware layer that is relatively lightweight and have neither the object model nor programming language restrictions imposed by other traditional middleware systems. WSDL and SOAP are both independent of specific platforms, programming languages, and middleware technologies. Moreover, SOAP leverages the optional use of the HTTP protocol, which can bypass firewalls, thereby enabling Internet-wide application integration.

Web services, which are typically used to represent reusable business functions (*e.g.*, flight reservation), can be the building blocks of more complex business processes. Although complex business processes can be developed in general-purpose languages such as Java and C++, such languages do not provide high-level constructs to readily define workflow processes that represent composite Web services. Business Process Execution Language (BPEL) is a high-level workflow language that can be used to create coarse-grained business processes that constitute a number of related business functions [15, 19, 22]. By representing a workflow that coordinates activities among other Web services, BPEL allows for the creation of coarse-grained Web services by wiring together activities that can invoke other Web services, manipulate data and handle exceptions. BPEL and Web services make it possible for organizations to deploy flexible Service Oriented Applications (SOA) [4, 13]. SOA-based integration permits the discovery and use of existing resources and thereby reducing the cost and speed with which applications can be developed.

In this paper, we introduce BPEL, a language that facilitates the development of composite Web services. This paper is not meant to be a guide to the usage and syntax of BPEL but rather a high-level executive overview of the language, its challenges and some of the limitations we have encountered in its use. The rest of this paper is structured as follows. Section 2 discusses the integration of Web services and shows how BPEL facilitates this integration using a commonly-used example. In section 3, we present an overview

of BPEL activities and constructs in the context of the example. Section 4 provides a layered architecture for BPEL processes and shows how supporting technologies such as WSDL, SOAP, and HTTP are used in the context of the running example. A discussion of some of the limitation of the language is provided in Section 5. Finally, some concluding remarks are presented in Section 6.

2 Web Service Integration: A Motivating Example

To help illustrate the need for integration of services, consider a fictitious company, Bloggs Financial Services (BFS), that provides loan services to car dealerships. The car dealerships make loan requests (on behalf of their respective customers) to BFS. As illustrated in Figure 2, BFS has a loan assessor service that electronically receives requests for loan, assesses the risk involved, and if the risk associated with a loan request is low, then issues a loan. In case the risk is high, BFS forwards the corresponding requests to its partner bank, the XYZ bank.

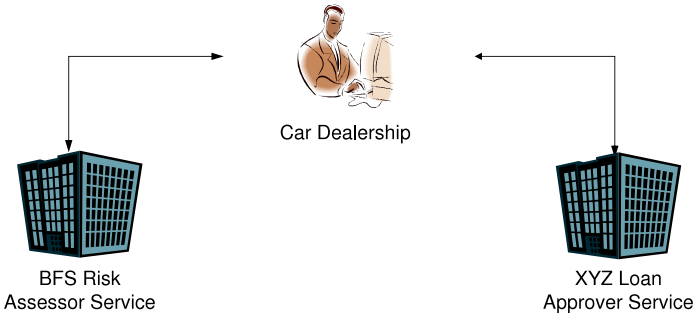

Figure 2: BFS forwards high-risk loan requests to XYZ.

We argue that it is in the best interest of BFS (and its partners) to have a business process that encapsulates the logic of this transaction by aggregating the BFS risk assessor and XYZ loan approver services: first, car dealerships enjoy the customized and unified loan service provided by BFS and do not need to interact with potentially different loan approver services provided by different lenders; second, banks do not need to provide customized services for each business domain and do not need to interact directly with each end customer; and third, BFS do not lose customers to other lenders as it will be in the loop of all transactions by providing a one-stop-shop for its customers. Moreover, this business process would not only allow BFS to expedite loan approvals but also provides the flexibly of integrating with new partners without the need to modify the logic of its own business.

As illustrated in Figure 3, the BPEL process itself is a Web service that specifies the logic of the interaction with its partners. The partner Web services are the BFS risk assessor service and the XYZ loan approver services. This model allows for changes to be made to those business functions without necessarily affecting that of the BPEL process. For instance, there might be changes to the policies for evaluating loan requests, as well as changes to the actual banking partners.

Because BPEL is a very high-level language, business analysts and managers (non-technical personnel) are able to easily compose business processes *graphically* with tools such as ActiveWebflow Professional [1] and Oracle BPEL Manager [2]. These development tools allow for the logic of the business process to be specified by graphically linking together desired activities. The flowchart in Figure 4 is a screen dump from ActiveWebflow Professional [1] [1]. It is a graphical representation of the BPEL process implementing the BFS loan service (see Figure 3).

[1]We have added some annotation to this image for clarity.

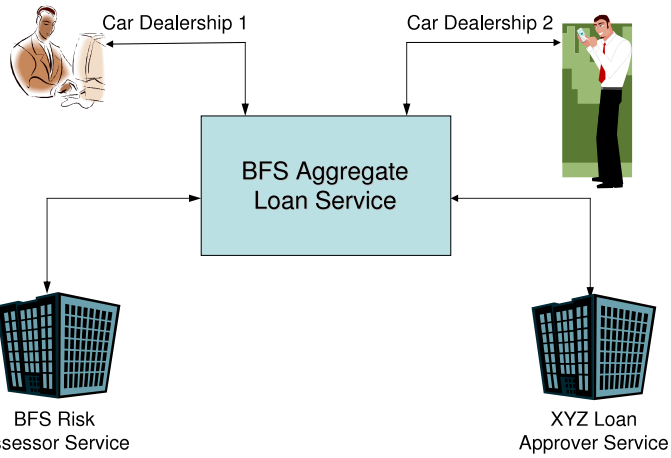

Figure 3: BFS provides an aggregate service to its customers.

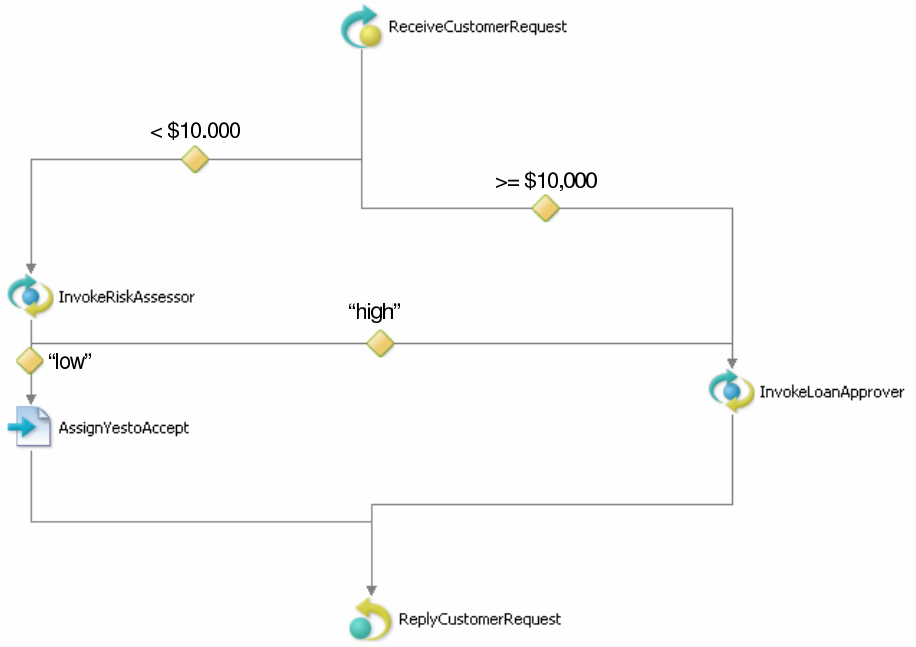

Figure 4: A screen dump of the graphical representation of the loan service BPEL process from ActiveWebflow Professional [1].

As shown in the flowchart, the BPEL process receives as input a loan request (`ReceiveCustomerRequest`) from a car dealership. The loan request message comprises two variables: the name of the customer and the loan amount (not shown in the figure). If the loan amount is less than $10,000, then the BFS risk assessor service is invoked (`InvokeRiskAssessor`), otherwise the XYZ loan approver service is invoked (`InvokeLoanApprover`). After the risk assessor is invoked, the BPEL process expects to receive as reply a value of either "high" or "low". When the risk assessment is "low", this means the loan is approved and the BPEL process sends an approval message (with "yes" value, `AssignYestoAccept`) to the car dealership and terminates (`ReplyCustomerRequest`). If the risk assessment message is "high", the XYZ loan approver service is invoked (`InvokeLoanApprover`). The loan approver service returns either

"yes" or "no", which is then sent as reply to the car dealership.

Both the risk assessor service and the loan approver service can also return a predefined fault message to the BPEL process. When any of these services reply with a fault message, the BPEL process sends an error message to its user and terminates (not shown in the figure).

3 BPEL Overview: Activities and Constructs

Any workflow language that is used to compose and manage business processes involving multiple Web services must meet some key requirements: (1) The ability to adequately represent the business logic of the process; (2) The ability to provide asynchronous as well as synchronous invocations of Web services; (3) The ability to support long-running transaction; and (4) The ability to manage failures, exceptions and recovery. The BPEL specification attempts to meet these requirements, although some of the available features have certain limitations, as we will cover in a later section. BPEL is the result of collaborative effort by IBM, Microsoft and BEA. Earlier work to create a business process workflow language includes, amongst others, Microsoft's XML business process language (XLANG) and IBM's Web Services Flow Language (WSFL) [18]. BPEL combines the best features of XLANG and WSFL as well as additional functionality and flexibility. In this section, we present some key BPEL constructs. For some of the constructs, we will use code snippets from the Loan Approval BPEL process described in Section 2.

BPEL is simply a flow language that weaves together *basic* and *structured* activities to create the logic of a business process. A *basic* activity is a primitive BPEL activity that performs an atomic action, while a *structured* activity is derived from a combination of several activities (either basic or other structured activities). For example, the `invoke` activity is a basic activity that performs an operation on a partner Web service. The XML code in Figure 5 is an example of a service invocation. The code instructs the BPEL engine (a virtual machine that interpretes and executes BPEL processes) to invoke a partner web service. The actual web service partner is defined by the `partnerLink` (line 6). Lines 7 and 3 identify the interface (`portType`) of the partner and what method (`operation`) the invocation wishes to call. The input and expected output variables are specified in lines 4 and 5.

```
1. <invoke
2.    name="InvokeLoanApprover"
3.    operation="approve"
4.    inputVariable="request"
5.    outputVariable="approval"
6.    partnerLink="approver"
7.    portType="loanApprovalPT">
8.    <target linkName="receive-to-approve"/>
9.    <target linkName="assess-to-approve"/>
10.   <source linkName="approver-to-reply"/>
11.</invoke>
```

Figure 5: The invocation of the XYZ loan approver service.

Some other examples of basic activities include the `receive` and `reply` activities. The `receive` activity waits for external input from a partner via some predefined operation, while the `reply` activity sends back a message to the partner that invoked a `receive` operation. Figure 6 shows the code that defines the receive activity (*ReceiveCustomerRequest*) from Figure 4. The `receive` activity contains properties that specify what partner makes the input, the operation and the required variable (lines 4, 5 and 7 respectively). The transition conditions from the `receive` activity to the other activities are defined in the `source` properties on lines 8-10 and 11-13.

```
1. <receive
2.   createInstance="yes"
3.   name="ReceiveCustomerRequest"
4.   partnerLink="customer"
5.   operation="approve"
6.   portType="loanApprovalPT"
7.   variable="request">
8.   <source
9.     linkName="receive-to-assess"
10.    transitionCondition="bpws:getVariableData('request', 'amount') &lt; 10000"/>
11.  <source
12.    linkName="receive-to-approve"
13.    transitionCondition="bpws:getVariableData('request', 'amount') &gt;= 10000"/>
14.</receive>
```

Figure 6: The receive activity of the BFS aggregate loan service (the BPEL process).

Structured activities specify the order in which combined activities execute. They provide support for asynchronous interactions which is important for efficiency and scalability. For instance, activities that are meant to execute concurrently or sequentially are enclosed in `flow` and `sequence` tags, respectively. Other structured activities include `switch` for conditional branching, `pick` for alternative choices and `while` for looping activities. Structured activities can be nested and the execution order between blocks of activities can be defined with the use of links.

According to Peltz, about 80 percent of the total amount of time spent in developing business processes is spent in exception management [18]. Therefore the management of faults is a key feature in any business processes. BPEL processes can manage exceptions generated from a service invocations. Faults can be generated through the `throw` construct. Faults are caught with `catch` or `catchAll` handlers defined inside a `faultHandlers` tag. The `catch` elements specify custom fault-handling activities that execute on a given fault name or fault variable, while the `catchAll` element specify fault-handling activities that execute when a fault is not caught by a `catch` fault handler. The code in Figure 7 shows a fault handling mechanism for the process. Inside the fault handler is a catch clause (lines 2-12) for a predefined fault (*loanProcessFault* at line 3). The action to take upon this fault is to send a reply (with an error message) back to customer as specified by lines 5 to 11.

```
1. <faultHandlers>
2.   <catch
3.     faultName="loanProcessFault"
4.     faultVariable="error">
5.    <reply
6.       faultName="loanProcessFault"
7        name="ReplyToFault"
8.       operation="approve"
9.       partnerLink="customer"
10.      portType="loanApprovalPT"
11.      variable="error"/>
12. </catch>
13.</faultHandlers>
```

Figure 7: Fault handler from the BPEL process

The BPEL specification also comes with other constructs including the `scope`, `correlation`, and `compensation` activities. Using the `scope` tag, activities can be grouped into a single transaction. The `scope` container provides a context for a subset of activities. It can contain fault and event handler for activities nested within it. The `scope` allows a group of activities to be managed as one logical unit.

The `correlation` and `compensation` provide support for long-running transactions. In the BPEL process described in Section 2, for instance, multiple customers could be making requests for loan at the same time, meaning that several of the same type of messages are flowing to the business process and its partner services. It is thus important that there is some sort of message correlation to ensure that messages are being routed properly. This feature is especially important in the case of asynchronous interactions. BPEL utilizes a correlation construct for managing groups of messages that belong to a specific business partner interaction. `Compensation` enables a BPEL process to specify an activity at the `scope` or process level whose execution serves to undo some application logic that has already successfully completed. Compensation is important to ensure that activity groups do not complete partially, for instance in the event of a fault. An example of a compensation activity could be to cancel a reservation.

4 Supporting Technologies: WSDL, SOAP, HTTP

As illustrated in Figure 8, the key to achieving interoperability in BPEL is its layered architecture. The dependency relation between each two adjacent layers is top-down. While the process layer (BPEL) takes care of the composition of Web services, the service layer (WSDL) provides a standard for describing the service interfaces. At the messaging layer (typically SOAP), the operations defined in the service layer are realized as two related output and input messages, which serialize the operation and its parameters. At the bottom of the stack is the transport layer (typically HTTP) that facilitates the physical interaction between the Web Services. Although Web services are independent of transport protocols, HTTP is the most commonly used protocol for Web service interaction. Except for the transport layer, all the protocols in the other layers are typically based on XML. In the rest of this section, we presents a brief description of WSDL, SOAP, and HTTP in the context of the running example.

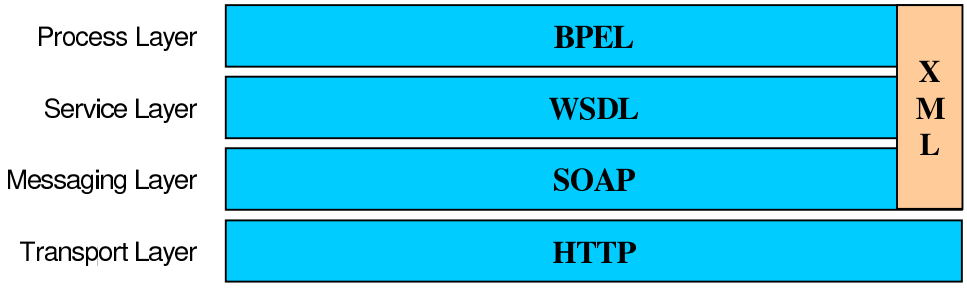

Figure 8: The BPEL protocol stack.

WSDL. Web Services Description Language (WSDL) is an XML-based standard for describing a Web service [6]. A WSDL definition is divided into two parts: abstract and concrete. The *abstract* part describes the service interface, the operations it performs and the messages involved; the *concrete* part describes the location of the service and how to access and bind to a Web service.

Figure 9 shows the abstract portion of the BPEL process described in Section 2, while Figure 10 shows the concrete portions of this process[2]. Recall that a BPEL process itself is a Web service and its interface is described by WSDL (similar to its partners). The abstract portion of the WSDL (Figure 9) contains the

[2]Some detail has been omitted from the WSDL portions for clarity.

```
1. <message name="creditInformationMessage">
2.   <part name="firstName" type="string"/>
3.   <part name="lastname" type="string"/>
4.   <part name="amount" type="integer"/>
5. </message>
6.
7. <message name="approvalMessage">
8.   <part name="accept" type="string"/>
9. </message>
10.
11.<portType name="loanApprovalPT">
12.  <operation name="approve">
13.    <input message="creditInformationMessage"/>
14.    <output message="approvalMessage"/>
15.  </operation>
16.</portType>
```

Figure 9: The abstract portion of the WSDL description for the BFS aggregate loan service.

description of the input message (`creditInformationMessage`, lines 1-5), the expected output message (`approvalMessage`, lines 7-9). The service port type (`loanApprovalPT`, lines 11-16) describes the interface of the BPEL process, that is, what operations it exposes and the input and output messages involved. In the concrete portion of the WSDL (Figure 10), the binding to the port type (`loanApprovalPT`, lines 1-17) and the physical location of the service (lines 19-23) are defined. An abstract definition can be mapped to multiple concrete implementations. Both abstract and concrete WSDL definitions are independent of the service implementation. In other words, the description of service endpoints, the operations and messages, the message formats and the network protocols used in the communication are independent of how the services are implemented.

SOAP. SOAP is an XML-based messaging protocol designed to be independent of specific platforms, programming languages, middleware technologies, and transport protocols [11]. SOAP messages are used for interactions among Web services. Unlike object-oriented middleware such as CORBA, which requires an object-oriented model of interaction, SOAP provides a simple message exchange among interacting parties.

A SOAP message is an XML document with one element, called an envelope, and two children elements, called header and body. The contents of the header and body elements are arbitrary XML. The header is an optional element, whereas the body is not; there must be exactly one body defined in each SOAP message. To provide the developers with the convenience of a procedure-call abstraction, a pair of related SOAP messages can be used to realize a request and its corresponding response. SOAP messaging is *asynchronous*, that is, after sending a request message, the service requester will not be blocked waiting for the response message to arrive.

Figure 11 shows the structure of a HTTP request containing an output SOAP message that corresponds to calling the `approve` method of the XYZ loan approver service[3]. A HTTP post request contains a header (lines 1-4) and a body (lines 6-18). The body of this post request is a SOAP message. The body of the SOAP message contains the serialized method call (lines 10-17) to the `approve` operation with the required parameters.

[3]The SOAP message is simplified for clarity. Details of namespaces are not present.

```
<binding name="SOAPBinding" type="loanApprovalPT">
  <binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="approve">
    <operation soapAction="" style="rpc"/>
      <input>
        <body use="encoded"
          namespace="http://www.bfs.com/wsdl/loanapprover"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <body use="encoded"
          namespace="http://www.bfs.com/wsdl/loanapprover"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
  </operation>
</binding>

<service name="LoanApprover">
  <port name="SOAPPort" binding="SOAPBinding">
    <address location="http://www.bfs.com/services/approver"/>
  </port>
</service>
```

Figure 10: The concrete portion of the WSDL description for the BFS aggregate loan service.

```
POST /LoanApprover
SOAPAction: "http://www.bfs.com/services/approver"
Content-Type: text/xml; charset=‘‘utf-8"
...

<soap:Envelope xmlns:soap= "http://schemas.xmlsoap.org/soap/">
  <soap:Header>
    <!-- Optional header contents.>
  </soap:Header>
  <soap:Body>
    <approve>
      <creditInformationMessage
        firstName = "Jane"
        lastName = "Doe"
        amount = "5000"/>
    </approve>
  </soap:Body>
<soap:Envelope>
```

Figure 11: The HTTP request containing the SOAP message for calling the `approve` method.

5 Some Shortcomings: Sharing Experience

BPEL is not a general-purpose language and as such lacks many traditional programming language constructs. This amongst other factors have created several limitations for the language. In this section we

present some of those limitations we have encountered in our use of this language.

Fault handling: As stated in Section 3, the BPEL specification provides two constructs for handling faults; `catch` and `catchAll`. The `catch` clause is used to handle custom predefined programmatic faults and the `catchAll` handler used for all other undefined faults. Before a fault can be caught within the BPEL process by the `catch` handler, that fault would have had to be defined in the WSDL description of the service partner as part of the possible messages of that interface. The specification makes no provision for system faults that are common in this type of architecture. For example, the fault generated when a BPEL process tries to invoke an unavailable Web service. Currently these type of faults can be caught only by the `catchAll` construct. Since `catchAll` has no way of distinguishing between faults, this approach is not desirable because the course of action to take in the event of a fault is often determined by the type of fault that was generated. This inadequacy limits the quality of service of BPEL business processes.

Data manipulation: BPEL offers limited data manipulation. The main purpose of BPEL is to orchestrate the interaction among Web services which includes moving data from one service to another. The language thus has very little facilities for manipulating the data. It is currently not possible to dynamically create or destroy variables. Each variable used has to be declared at design time and at the process level. Local variables cannot be declared within activity blocks (*e.g.*, `scope`). Also BPEL is not able to recognize whether two variables are composed of the same primitive types. Therefore, a message can be passed from one Web service to another, only if the message have the exact same definition from the exact same XML namespace. If this is not the case, a BPEL `copy` operation would be needed to copy the contents of one variable to the other before a message can be sent between services. This severely hampers the flexibility of BPEL process because it often means that the Web service partners of a process have to share the same WSDL interface descriptions for operations that exchange messages. Robust data manipulation often has to be delegated to a Web service which sometime often complicates the process and limits its dynamism.

6 Conclusion

In this paper we have presented the need for organizations to electronically interact with one another. We discussed the role BPEL plays in enabling the required application-to-application (A2A) and business-to-business (B2B) integration. In the context of an aggregate loan service example, we have presented a high-level overview of the language constructs and some of its limitations as we have experienced them. Although it is not a traditional general-purpose programming language, some of its inadequacies do limit its effectiveness as a dynamic and flexible language for Web service integration. Some factions may argue that its not necessary to extend the language, since it is purely intended for orchestration and does not actually implement business logic components, thereby making it user friendly for business managers. But we believe the language can be improved in a way that still allows business managers to compose business process while permitting more experienced programmers to transform such processes into more robust and efficient solutions.

Further Information. A number of related papers, technical reports, and a download of the software developed for this paper can be found at the following URL: `http://www.cs.fiu.edu/~sadjadi/`.

Acknowledgements. This work was supported in part by IBM SUR grant.

References

[1] Active Webflow. Available at URL: `http://www.active-endpoints.com/products/`.

[2] Oracle BPEL Process Manager. Available at URL: http://www.oracle.com/technology/products/ias/bpel/index.html.

[3] D. E. Bakken. *Middleware*. Kluwer Academic Press, 2001.

[4] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. *Web Services Architecture*. W3C, 2004. Available at URL: http://www.w3.org/TR/ws-arch/.

[5] A. T. Campbell, G. Coulson, and M. E. Kounavis. Managing complexity: Middleware explained. *IT Professional, IEEE Computer Society*, (5):22–28, September/October 1999.

[6] R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer, and S. Weerawarana. *Web Services Description Language (WSDL) Version 2.0*. W3C, 2.0 edition, March 2004. Available at URL: http://www.w3.org/TR/wsdl20/.

[7] D. Conger. *Remoting with C# and .NET*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

[8] W. Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the Conference on The future of Software engineering*, pages 117–129, 2000.

[9] Gartner. *Application Integration & Web Services Summit 2004*, May 2004.

[10] A. Gokhale, B. Kumar, and A. Sahuguet. Reinventing the wheel? CORBA vs. Web services. In *Proceedings of International World Wide Web Conference*, Honolulu, Hawaii, 2002.

[11] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. *SOAP Version 1.2*. W3C, 1.2 edition, 2003. Available at URL: http://www.w3.org/TR/soap12.

[12] Java Soft. *Java Remote Method Invocation Specification, revision 1.5, JDK 1.2*, Oct. 1998.

[13] H. Kreger. *Web Services Conceptual Architecture (WSCA 1.0)*. IBM Software Group, May 2001. Available at URL: http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[14] J. Lee, K. Siau, and S. Hong. Enterprise integration with erp and eai. *Communications of the ACM*, 46(2), February 2003.

[15] F. Leymann, D. Roller, and M.-T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.

[16] Microsoft Corporation. *Microsoft COM Technologies - DCOM*, 2000.

[17] Object Management Group, Framingham, Massachusett. *The Common Object Request Broker: Architecture and Specification Version 3.0*, July 2003.

[18] C. Peltz. Web services orchestration a review of emerging technologies, tools and standards. *Technical Paper*, January 2003.

[19] D. Sherman. Business flows with bpel4ws. *Online article*, 2005. Available at URL: http://xml.sys-con.com/read/39780.htm.

[20] S. Vinoski. Where is middleware? *IEEE Internet Computing*, March-April 2002.

[21] S. Vinoski. Integration with Web services. *IEEE Internet Computing*, November-December 2003.

[22] S. Weerawarana and F. Curbera. Business process with bpel4ws: Understanding. *Online article*, 2002. Available at URL: http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol1/.