BPEL Remote Objects: Integrating BPEL Processes into Object-Oriented Applications

Marvin Ferber, Thomas Rauber
Department of Computer Science
University of Bayreuth, Germany
Email: {marvin.ferber,rauber}@uni-bayreuth.de

Sascha Hunold[‡]
International Computer Science Institute
Berkeley, California, USA
Email: sascha@icsi.berkeley.edu

***Abstract*—Service-orientation and object-oriented design are common practice in the field of business application development. Business process execution languages help to facilitate the orchestration of Web services in service-oriented architectures (SOA). However, using business processes from within object-oriented and event-driven applications is difficult as asynchronous event handling is missing in workflow-based business process modeling languages. The present article presents an approach for integrating BPEL business processes into object-oriented applications. We propose *BPEL remote objects* (BPELROs) that can be accessed asynchronously in an object-oriented manner. We present a method how state-based business processes can be implemented using BPELROs. It is shown how to apply BPELROs for software modernization tasks and we also evaluate the performance of BPELROs on different BPEL engines.**

I. Introduction

Object-oriented programming languages are standard in application development. They facilitate a modular development, which is supported by a variety of tools for modeling and verification of object-oriented software designs. Languages like Java and C# are very popular for the development of business applications.

On the other hand, the service-oriented architecture (SOA) has evolved, which provides support for distributed software development. Web service technology is a prominent approach for implementing SOA. Application modules are exposed as Web services to collaborate over the Internet. The interface of a Web service is defined in the Web Service Description Language (WSDL) [1]. Message exchange between Web services is usually done using the Simple Object Access Protocol (SOAP) [2]. Business process execution languages and engines have been developed to orchestrate such Web services [3]. Prominent examples are the XML Process Definition Language (XPDL) [4] and the Business Process Execution Language (BPEL) [5].

[‡]This work was supported by a fellowship within the postdoctoral program of the German Academic Exchange Service (DAAD).

Object-oriented programming and SOA development are difficult to connect as they are based on different technologies [6]. SOA is based on message-oriented Web services and object-oriented programming is based on objects that provide attributes and methods. Although a business process can provide an internal state similar to classes in object-oriented programming, only a set of unrelated interface methods is provided by the WSDL document of the process. There is no information whether the invocation of an interface method creates a new process instance or not. Furthermore, a process has no unique reference. The corresponding WSDL document only provides a service URL, which is shared by all instantiated processes of the same type. Thus, an object-oriented access to business processes would ease their integration into object-oriented applications.

The modeling of an object-oriented access to business processes requires a business process execution language to be capable of handling asynchronous events. We focus on BPEL as target language, since it is broadly accepted and offers asynchronous event handling. Furthermore, the modeling of business processes has to be adapted to provide an object-like behavior. For this purpose, we focus on a modeling of business processes based on finite state machines (FSMs). An FSM is able to handle asynchronous events because a transition to each state is always available, whereas a workflow-like modeling would provide message exchange only on certain points in the process definition.

As an example, an object-oriented access to BPEL processes can facilitate the integration of legacy software into a SOA landscape. To achieve that, business logic and graphical user interface (GUI) have to be divided in order to execute business logic, encapsulated as BPEL processes, on a separate execution engine. The externalization of business logic has been proposed in [7] as a step in the incremental transformation process from a legacy software to a modern distributed application. One important reason for reintegrating legacy code into the modernized software is to preserve the graphical user interface in order to facilitate the transition to a new version of the program.

```
<definitions xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop" ...>
  ...
  <vprop:property name="corelid" type="xs:int" />
  <vprop:propertyAlias messageType="ns:addItemRequest" part="parameters"
    propertyName="ns:corelid"><vprop:query>ns:id</vprop:query>
  </vprop:propertyAlias>
  ...
  <types><xs:schema ... >
    <xs:element name="addItem"><xs:complexType><xs:sequence>
      <xs:element name="id"    type="xs:string" /><!-- id -->
      <xs:element name="name"  type="xs:string" />
      <xs:element name="prize" type="xs:float" />
    </xs:sequence></xs:complexType></xs:element>
  ...
  </xs:schema></types>
  <message name="addItemRequest">
    <ns:part element="ns:addItem" name="parameters" />
  </message>
  ...
  <portType name="InvoicePortType">
    <operation name="initialize">...</operation><!-- initialize -->
    <operation name="addItem">...   </operation><!-- addItem    -->
    <operation name="destroy">...   </operation><!-- destroy    -->
              ...
  </portType>
  <binding ... >...</binding>
  <service name="Invoice"><port ... >
    <soap:address location="http://servicehost:port/resource" ... />
    <!-- serviceURL -->
  </port></service>
</definitions>
```

Fig. 1: Example of an object-oriented WSDL interface.

The contribution of this article is to present a novel modeling approach for BPEL processes (BPELROs) that facilitates the integration of BPELROs into object-oriented programming languages like Java or C#. We show how to obtain a WSDL interface that supports object-like interaction with BPEL processes. The BPELROs can be used to express the behavior of state-based business processes. Thus, our approach makes it easier to embed business processes, modeled in BPEL, into object-oriented applications. In contrast to other remote object technologies, BPELROs target the integration of business processes from a SOA landscape into a object-oriented programming language.

The remainder of the article is organized as follows. An object-aware WSDL interface is presented in Section II. Section III introduces BPELROs and Section IV describes how they can be accessed from object-oriented programming languages. We show how state-based business processes can be mapped to BPELROs in Section V. The proposed method is applied in Section VI, and the performance of BPELROs is investigated for different BPEL engines. Sections VII discusses related work and Section VIII concludes the article.

II. Object-oriented Interface with WSDL

The communication interface to a BPEL process is described in a WSDL document. To provide an object-oriented access to the BPEL process, an object interface has to be modeled in WSDL. The WSDL definition only provides a set of methods which together form a service. We define a service that represents an object interface in the object-oriented WSDL document. Hence, all service methods become member methods of the corresponding object. As a convention, we define two mandatory methods of the WSDL object interface to be used as constructor (`initialize()`) and destructor (`destroy()`), although there can be additional con/destructors.

Normally, a WSDL document defines stateless Web services, but BPELROs are stateful resources that combine a set of Web service methods to a self-contained object. A common way to address such behavior is to strictly separate the Web service from the underlying object (resource). This means that a resource is referenced by a unique ID in addition to the regular Web service URL.

In the WS-Resource Framework (WSRF) specification, the corresponding object ID is normally added to the SOAP header of a method invocation [8]. However, the BPEL standard does not support the WSRF, but BPEL provides correlation sets to distinguish method invocations on different objects. Similar to WSRF, each underlying object is assigned a unique object reference ID, which is part of a specific XML namespace in the WS-BPEL v2.0 specification. We add this ID as a parameter to all service methods in the WSDL document. A object-oriented WSDL interface showing the ID parameter and the con/destructor is presented in Fig. 1.

The unique object reference ID parameter in all method definitions lets the underlying engine forward method invocations to the corresponding object. When BPELROs are used in a Web service context, an object can be uniquely referenced by its URL and its object ID. We use this combined object reference later to assign wrapper classes in object-oriented languages to BPELROs.

As a consequence, each BPELRO must own a unique ID on the particular BPEL engine. Similar to the WS-Resource factory, unique IDs can be obtained from an ID generator that is integrated into the software system. The object reference ID can be obtained from the constructor of the BPELRO and returned to the calling object that requested the instantiation of a new BPELRO. The ID generator could be a Web service, which resides on the same host as the BPEL engine. All BPEL engines provided in a software system must use the same ID generator to prevent failure.

III. BPEL Remote Objects

In order to treat BPELROs as remote objects as found in other technologies (CORBA or RMI), a mapping of object attributes to BPEL processes has to be defined. This includes a mapping of member variables and methods and a mapping of constructors and destructors. An interface definition and an object reference ID has already been specified in the object-oriented WSDL definition of the BPELRO. Additionally, the processing instructions (e. g., business logic) have to be added to

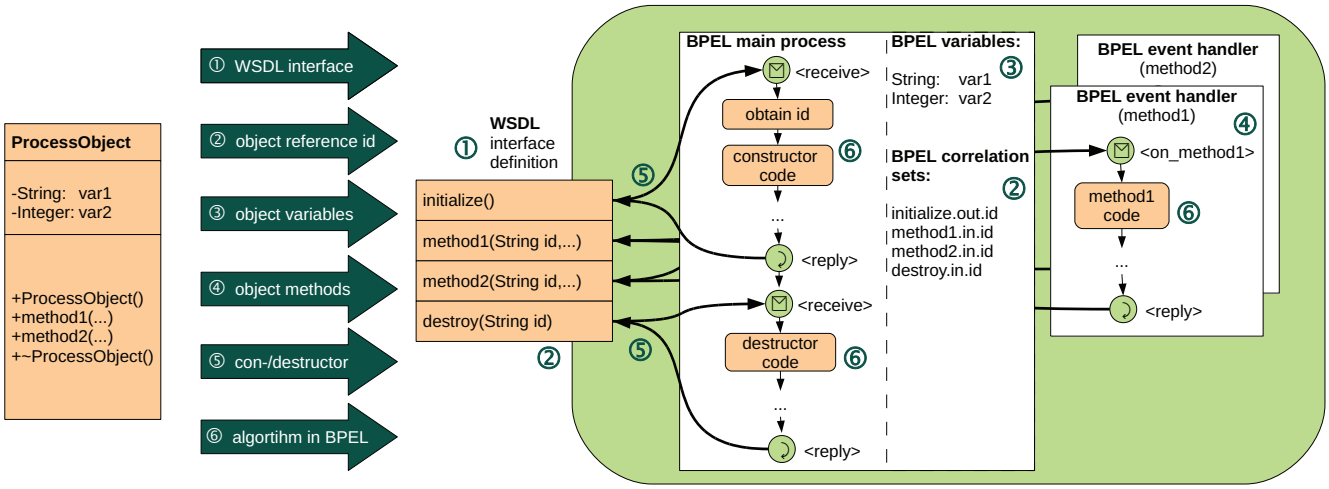

Fig. 2: Mapping of an abstract object to a BPELRO definition.

the BPELRO member methods to complete the business process. BPEL offers a comprehensive set of constructs regarding control flow, including asynchronous event and fault handler, as well as activities for document processing, message exchange, and many more.

A. Engineering BPEL Remote Objects

The mapping of the mentioned object attributes to a standard BPEL process can be performed by the following six steps. See Fig. 2 for an illustration.

(1) WSDL interface definition: As described in Section II, the BPELRO's WSDL document contains the interface definition of the BPELRO. It contains a service that exposes all member methods of the BPELRO. Furthermore, methods must always return a value, since the behavior of void functions in a (message-oriented) network context is not defined. Void functions could return before the whole execution is completed, because the receiver can close the communication after it has received the whole incoming message.

(2) Object reference mapping: On a BPEL engine all process instances share the same interface (WSDL). An object reference ID parameter is added to each member method in the object-oriented WSDL interface. Through BPEL correlation sets, this object reference ID can be used to enable the BPEL engine to distinguish between different process instances. Therefore, it has to be marked as correlating parameter in the BPEL process definition. Hence, a method invocation can be forwarded to the corresponding BPEL process instance.

(3) Member variable mapping: Member variables have to be defined as BPEL variables in the scope of the main process to make them accessible for attached subprocesses (e. g., event handlers).

(4) Member method mapping: Each member method of the BPELRO, except the con/destructor, is mapped to a separate event handler in the corresponding BPEL process definition. Event handlers can be executed decoupled from the main process. Parameters are received by the event handler's entry activity `<onEvent>` and the return value can finally be sent by a `<reply>` activity at the end of the event handler definition. Between those two communication activities, processing instructions can be defined.

(5) Con/Destructor mapping: An object lives from the time a constructor is called until a corresponding destructor is called. A BPEL object lives as long as the control flow in the main process is not terminated. Therefore, constructor and destructor of the BPELRO are mapped into the main process of the BPEL definition. Both are invoked by a `<receive>` activity (or `<pick>` for multiple con/destructors) followed by processing instructions and a trailing `<reply>` activity. A constructor invocation creates a new process instance and a new unique object reference ID is obtained and assigned to the new BPELRO instance. The object reference ID is returned to the calling object as a constructor return value. The `<receive>` activity of the destructor suspends the execution of the process and keeps it alive until the destructor is invoked. After the destructor has terminated, the main process ends and the BPELRO will be destroyed by the engine.

(6) Algorithm mapping: The definition of processing instructions in BPEL is supported by a variety of tools and graphical editors, e.g., as plugins for NetBeans or Eclipse. Processing instructions have to be added to the member methods (event handlers) and the constructor and destructor of the BPELRO. By doing this, the unrestricted set of BPEL constructs can be utilized.

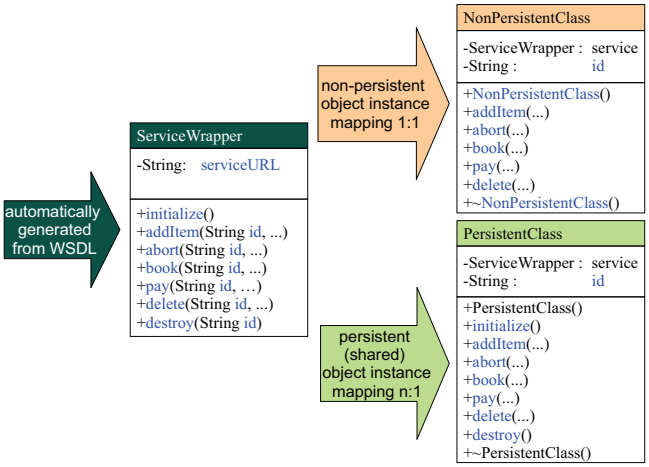

Fig. 3: Deriving wrapper classes for a 1:1 mapping and a n:1 mapping from a BPELRO's WSDL interface.

B. Restrictions of BPEL Remote Objects

The use of BPEL entails several restrictions for modeling and accessing BPELROs:

- Member variables cannot be accessed from other objects directly (private access). Access can be provided by implementing getter and setter methods.
- All member methods are public.
- A BPELRO cannot invoke member methods of itself, since it cannot invoke an event handler of itself. A workaround could be to copy and inline the desired member method into the calling method.

Furthermore, the object-oriented WSDL interface and the SOAP communication require additional restrictions when accessing BPELROs:

- Each member method must return a value due to message-orientation (no void functions).
- BPEL only accepts XML documents as parameters (SOAP). Simple types can be taken from the XML schema definition. Complex data structures must be passed to the method as XML tree.
- Member methods must not be long blocking. They must return immediately to avoid network timeouts during their invocation.

BPELRO methods and variables have to be protected from unwanted side effects through concurrent access. Therefore, BPEL offers the `<scope>` attribute `isolated="yes"`. Isolated scopes will be executed sequentially. Thus, concurrent access to variables can be avoided.

IV. Integrating BPEL Remote Objects into Object-Oriented Languages

For an easy integration of BPELROs into object-oriented languages, it is desirable to have wrapper classes that represent the BPELRO at client side. Such wrapper classes are derived in a two-step process from the corresponding object-oriented WSDL interface.

Since BPELROs are accessed through SOAP, in a first step, an object wrapper class A is generated that hides all the necessary marshaling and network protocol handling from the programmer. There is rich tool support for deriving such wrapper classes in a variety of object-oriented programming languages like Java, C++, or C# [9], [10]. Even common complex data structures like lists can be marshaled automatically.

In a second step, the generated wrapper class A is wrapped again to create an object-like class representative B for the BPELRO on client side. To hide the internal object reference ID from the programmer, the ID parameter becomes a dedicated member variable of class B. Class A also becomes a member variable of B. So, B internally forwards each method invocation to A using the corresponding object reference ID parameter. Additionally, con/destructor of the BPELRO are mapped to the con/destructor of B. The mapping of a BPELRO to an object-oriented wrapper class is illustrated in Fig. 3. The obtained wrapper class B (proxy) encapsulates the remote method calls of A and can therefore be used like any regular object.

In contrast to most other remote object technologies, BPELROs can be executed on different execution engines, which makes BPELROs more portable. Furthermore, BPELROs can be shared among different client applications, because a BPELRO can be reassigned to a client wrapper class by providing the Web service URL and the BPELRO reference ID.

When mapping con/destructor from BPELROs to a class B in an object-oriented language, there can be a non-persistent 1:1 mapping and a persistent n:1 mapping. In either case, the wrapper class B represents the BPELRO by providing an interface to its member methods. For a non-persistent mapping, constructor and destructor methods of the BPELRO are mapped 1:1 to the wrapper class B (see *NonPersistentClass* in Fig. 3). If the remote object is non-persistent, it only exists as long as the wrapper class instance of B is alive on client side. This mapping can be used to implement a session between an application and a remote business process.

BPELROs can also be modeled persistently, and thus can be shared among different wrapper classes. Therefore, con/destructor of the BPELRO have to be called separately from the con/destructor of the wrapper class B (see *PersistentClass* in Fig. 3). A new instance of

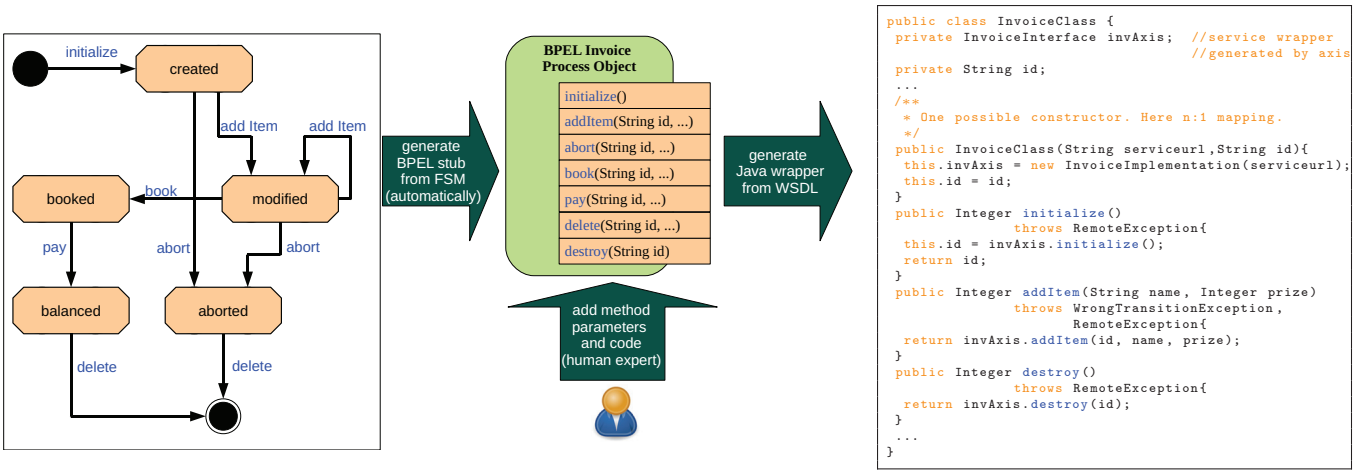

Fig. 4: Modeling state-based business processes with BPELROs. A Java wrapper class for an object-oriented integration is shown on the right.

a wrapper class B can be linked to an existing BPELRO by assigning the Web service URL and the object reference ID to the wrapper class. This n:1 behavior can be used to give a number of clients access to a remote document, e. g., if several clients want to access the status of a booking process.

V. Business Process Modeling with BPEL Remote Objects

Development of complex business processes is mostly done with graphical editors since high level business process modeling has evolved in recent years. High level modeling languages are the Business Process Modeling Notation (BPMN) [11], Event-driven Process Chains (EPC) [12], or UML 2 state machines. Such high level descriptions are rather abstract and normally not executable. Since BPELROs are BPEL processes that are modeled in a novel way, the modeling of business processes that should be expressed with BPELROs has to be adapted, too. In this section, we present how high level business process definitions, modeled as finite state machines, can be mapped to BPELROs.

A. Need for state-based business processes

Common business process execution engines require a workflow-like modeling of business processes, i. e., control flow is modeled by activities and transitions between them. Most business process execution engines support user interaction by human task activities that can be connected to a language-specific user management. User data can often be passed to running processes by a proprietary Web interface that is provided by the specific business process execution engine. Although this is an easy-to-use approach for developing new applications, it has some drawbacks when legacy code should be reused, e. g., when an existing GUI should be connected. Customized communication between a business process and other program modules can be realized by exporting and accessing standardized interfaces like SOAP Web services. However, from a programmer's point of view, such a set of unrelated remote procedure calls is often difficult to handle, because a certain message exchange can only be invoked in a certain state of the workflow-like process (synchronous communication).

A solution to avoid these difficulties can be the state-based modeling of business processes together with an object-oriented access to these processes. A state-based process is capable of offering asynchronous event handling because a transition to each state is always available. Although most business process technologies only support activity-based modeling (workflow), business processes can normally be expressed in both directions [13]. This is supported by our own experience with legacy business applications that often include state-based business processes.

B. Mapping of FSMs to BPEL Remote Objects

A state-based business process can be expressed as a finite state machine (FSM) as shown in Fig. 4. The FSM representation of the business process is used to generate source code that represents the FSM in BPEL. Abstract business process definitions (e. g., as a state machine) are not executable, because mandatory mapping information (e. g., communication port, variables, etc.) is missing. Thus, only the stub of the BPELRO can be created from a state-based business process definition. In a subsequent step, the generated BPEL stub has to be implemented to become executable on a BPEL engine.

In the following we describe how business processes,

modeled with a FSM, can be transformed into a BPELRO. The goal of the transformation is to assign each interaction that can be performed on the state machine to a member method of the BPELRO, see Fig. 4 (middle). To save the state of the state machine, a dedicated member variable *state* is added to the BPELRO. Because methods can be executed on objects in arbitrary order, we have to filter out transitions that do not exist in the FSM to preserve correctness. This is done by checking the current *state* at the beginning of each member method. In case of an incompatible input and state combination, the method throws an exception. Exceptions can also be transmitted by SOAP to signal a wrong behavior to the calling object on client side, see Fig. 4 (right). Furthermore, we have to ensure that only one member method can be executed at each point in time as multiple transitions at the same time to different states are not permitted. The destructor of the BPELRO plays a special role, because it can be called in each state of the FSM. A transition to the final state, which means the destruction of the object, is always possible (for the sake of clarity those edges are not shown in Fig. 4). Finally, the abstract object needs at least one constructor, but all constructors must set the *state* variable initially.

The resulting BPEL stub can automatically be derived from a high level FSM definition and includes a broad control flow definition of the business process. To complete the BPELRO definition, the following subsequent refinement steps have to be performed manually to add business logic:
1) Method parameters, return values, and the corresponding datatypes have to be added to the object interface definition;
2) Member methods have to be completed to process parameters and generate return values according to their interface definition.

VI. Experimental evaluation

BPELROs can be used in scenarios where processes are accessed in an unpredictable way, e.g., the synchronization of processes, the shared Web access to a document, or the connection of a user interface to a business process. In this section, we consider a scenario in which BPELROs are used to support the modernization of a monolithic object-oriented legacy application into a distributed application.

The investigated scenario is illustrated in Fig. 5. The legacy application is a stand-alone GUI application for accounting in middle sized companies. It consists of a fat client that is connected to a database. We use an incremental transformation process to divide the monolithic code into business logic and the graphical user interface. In a second step, the application modules derived should

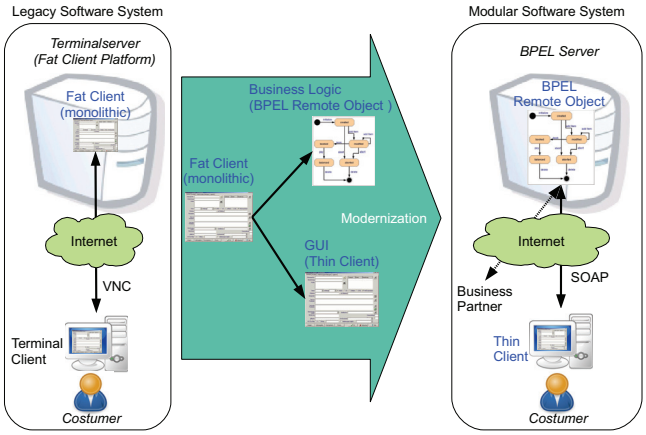

Fig. 5: Example of a software transformation into a SOA using BPELROs.

be enabled for a client/server-based utilization. The main goal of the transformation process is to increase the scalability of the application by putting all business logic into business processes and to connect a thin client to the processes. This thin client only represents the original GUI. For the purpose of a better business to business integration, BPEL is used as target language for business processes. Moreover, the object-oriented GUI should be reintegrated into the transformed application. This is the point where BPELROs come into play. BPELROs build the bridge between SOA integration and (object-oriented) legacy module reuse. Furthermore, the legacy business logic is implemented in state-based format. Hence, a straight-forward mapping to BPELROs is possible.

To examine the performance of BPELROs on open source BPEL engines, we used an invoice process, which is represented by a BPELRO. See Fig. 4 for an illustration. We used a Java application on the client side to utilize BPELROs on the BPEL engines. The wrapper classes for accessing the Web service interface on the server side were created using Apache Axis tools. The generated set of methods has been wrapped into a real object as described in Section IV. We used two BPEL engines for testing: the activeBPEL community engine (5.0.2) on an Apache Tomcat application server (5.0.28) and the BPELSE engine (2.5.1) of the openESB project on a Glassfish application server (2.1.1). Both application servers were executed on a 16 core Intel Xeon MP system with 16 GB memory. The test client was executed on an Intel Core2 Duo system with 4 GB of memory. Client and server were running a 64-bit Linux.

In a first test, an increasing number of BPELROs should be instantiated simultaneously to examine the maximum capacity of the considered BPEL engines. Therefore, a client application instantiates new BPEL-

ROs through a persistent wrapper class. After each instantiation, it calls a member method (`addItem()`) of all BPELROs that were instantiated until now. This procedure is repeated until either the instantiation of a new BPELRO or a method invocation fails due to memory restrictions on the server side. We were able to safely instantiate around 30000 invoice BPELROs on all engines. However, the sequential instantiation of 30000 objects took more than 90 minutes on the Sun BPEL engine (180 ms average) and only 13 minutes on the ActiveBPEL engine (26 ms average). Furthermore, an increasing memory consumption on the ActiveBPEL engine could be observed when invoking a BPELRO's method. This is possibly a logging issue and was not found on the Sun BPEL engine.

Performance is related to the question of how the massive utilization of correlation sets influences the response time of BPELRO member methods with an increasing number of parallel invocations. A low latency is important to use BPELROs in GUI applications. The latency mainly depends on the network used (WiFi, Ethernet, etc.) and the current network utilization. Therefore, we only measure the BPEL engine's part of the response time by ensuring a high bandwidth (1 Gbit) and a very low round trip time between the server and the client machine ($\ll 1$ ms). The tests are performed by instantiating a number of invoice processes on the engine. Afterwards a subset of 1000 processes is utilized simultaneously, but not concurrently, by a certain number of threads. Each thread processes its invoice objects sequentially, adding 10 items to each (`addItem()`) and finally performing a call to calculate the total amount of the invoice (`book()`). We measured the response time of each call to `addItem()` and `book()`, and compute the average response time over all calls.

Our results only show a very brief evaluation of the capability of BPEL engines to support BPELROs. Fig. 6 shows the average response time of method invocations on BPELROs over an increasing number of threads, i. e., parallel method invocations. As expected, the response time increases with an increasing number of processes as well as with an increasing number of parallel method invocations. With a low parallel utilization (< 4) all engines offer an acceptable response time for using BPELROs with graphical interfaces. Assuming an additional network latency of 100 ms, which is a pessimistic assumption, the graphical user interface responds within 500 ms. This would probably seem as an almost fluent reaction. However, due to possible Java garbage collection on the server side, while a method invocation is performed, some response times can be four times higher than the average. The relatively poor performance with a huge number of parallel invocations could be observed for all engines. However, this is only an issue under heavy load and when using BPELROs in GUI applications. One solution for this problem would be to use a cluster of engines to distribute BPELROs among different engines.

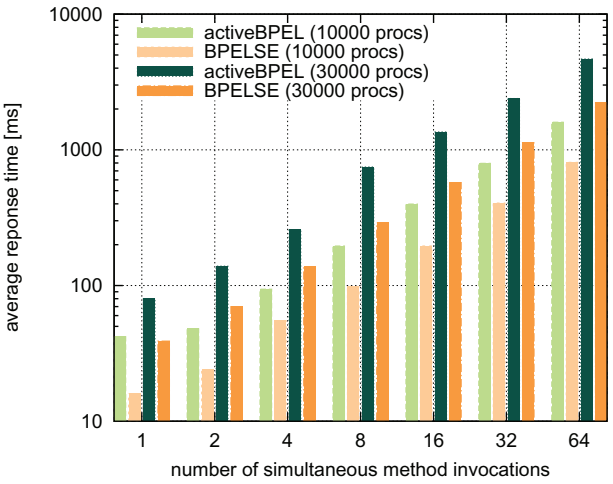

Fig. 6: Latency of simultaneous method invocations for different engines and BPELRO instances.

VII. Related work

The modeling of business processes has been studied intensively during the last years. In [14], an overview of different modeling techniques for business processes is given. This article distinguishes between graph-based modeling and block structured modeling. A comparison between BPMN, BPEL, EPC, and the Windows Workflow Foundation (WWF) is given. Business processes models can also be divided into state-based models and workflow-based models. The pros and cons of both modeling approaches have been studied in [15] and the necessity of state-based modeling is discussed in [13]. An example of bringing together state-based business processes and object-oriented programming is Microsoft's WWF [16]. They developed an embeddable workflow engine that is part of the .NET framework. Since this is more a domain specific approach, it lacks interoperability. A summary of object-oriented workflow modeling is presented in [17]. However, BPEL was not considered. How the life cycle of objects can be expressed by a finite state machines is shown in [18].

The transformation of abstract description languages into BPEL is desirable. In [19], BPEL is stated to be an expressive language that can be used to model complex behavior like common programming languages and a mapping from Workflow Nets to BPEL is presented. The Unified Modeling Language (UML) is used to define business processes in [20] and a mapping from UML 2 activity diagrams to BPEL is presented. Furthermore, it has been shown how to transform FSMs into BPEL

for the domain of event-driven processes for human interaction in [21].

An extensive overview of the modeling of user interaction in business processes is presented in [22]. Human activities can be integrated into BPEL processes through the BPEL Extensions for People [23] and Human Tasks [24]. However, those extensions do not add the capability of event-driven human interactions to BPEL. An event-driven approach was proposed in [25], where BPEL processes are divided into the actual business processes and processes dedicated to user interactions. This method introduces asynchronous communication to a workflow-based BPEL process, but it does not provide an object-oriented access to the business process.

VIII. Conclusions

In this article, we have shown how asynchronous communication can be applied to BPEL processes by defining BPELROs. BPELROs can be used like regular objects of object-oriented languages, similar to remote object technologies like RMI or CORBA. Beyond that, BPELROs can be persistent on a BPEL engine, i. e., they can be reassigned to wrapper classes on client side or can be shared among different client applications. It has been shown how these object-oriented concepts, such as constructors, destructors, or method calls, can be expressed by using a WSDL interface. The article also summarizes the necessary requirements and limitations of a language binding for accessing BPELROs from languages like Java or C#. The experimental results have shown that the latency of method calls on BPELROs, executed on an open source BPEL engine, are small enough to be used in practice. We also discuss how BPELROs can help to modernize legacy applications, e.g., by connecting service-oriented business processes with a complex legacy GUI.

References

[1] "Web Services Description Language," 2001. [Online]: http://www.w3.org/TR/wsdl/

[2] "Simple Object Access Protocol (SOAP) 1.1," 2000. [Online]: http://www.w3.org/TR/soap/

[3] J. Pasley, "How BPEL and SOA are changing Web services development," *IEEE Internet Computing*, vol. 9, no. 3, pp. 60–67, 2005.

[4] "XML Process Definition Language Specification," 2005. [Online]: http://www.wfmc.org/xpdl.html

[5] "WS-BPEL 2.0 Specification," 2007. [Online]: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[6] H. He, "What is Service-Oriented architecture," 2003. [Online]: http://www.xml.com/pub/a/ws/2003/09/30/soa.html

[7] T. Rauber and G. Rünger, "Incremental Transformation of Business Software," in *Proc. of the 9th Int. Conf. on Enterprise Information Systems (ICEIS 2007)*, 2007, pp. 81–94.

[8] "Web Service Resource Framework (WSRF) v1.2," 2006. [Online]: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[9] "Apache Axis." [Online]: http://ws.apache.org/axis/

[10] Microsoft Corporation, "Web services description language tool," 2007. [Online]: http://msdn.microsoft.com/en-us/library/7h3ystb6.aspx

[11] Object Management Group, "Business Process Modeling Notation." [Online]: http://www.bpmn.org/

[12] G. Keller, M. Nüttgens, and A. W. Scheer, "Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK)," in German, Saarbrücken, Tech. Rep. 89, 1992.

[13] P. Indurkar, "Why state machine workflows," 2005. [Online]: http://blogs.msdn.com/pravin_indurkar/archive/2005/09/25/473826.aspx

[14] O. Kopp, D. Martin, D. Wutke, and F. Leymann, "On the Choice Between Graph-Based and Block-Structured Business Process Modeling Languages," in *Modellierung betrieblicher Informationssysteme (MobIS 2008)*, ser. Springer LNI, vol. 141, 2008, pp. 59–72.

[15] D. Green, "Which Style of Workflow When?" 2005. [Online]: http://blogs.msdn.com/davegreen/archive/2005/10/20/483309.aspx

[16] D. Chappell, "The Workflow Way: Understanding Windows Workflow Foundation," 2009. [Online]: http://msdn.microsoft.com/en-us/library/dd851337.aspx

[17] J. Pamplin, S. Pellicer, and D. X. Hu, "Object-Oriented workflow modeling," 2005. [Online]: http://www.jasonpamplin.com/research/writings/Workflow.pdf

[18] J. J. Odell, *Advanced Object-Oriented Analysis and Design Using UML*. Cambridge University Press, 1998.

[19] W. M. van der Aalst and K. B. Lassen, "Translating unstructured workflow processes to readable BPEL: theory and implementation," *Information and Software Technology*, vol. 50, no. 3, pp. 131–159, 2008.

[20] M. Zhang and Z. Duan, "From Business Process Models to Web Services Orchestration: The Case of UML 2.0 Activity Diagram to BPEL," in *Proc. of the 6th Int. Conf. on Service-Oriented Computing (ICSOC 2008)*, ser. Springer LNCS, vol. 5364, 2008, pp. 505–510.

[21] W. Shi, J. Wu, and Z. Wu, "Using State Machine to Integrate Human Activity into BPEL in Dartflow," in *Proc. of the 3rd Int. Conf. on Services Computing (SCC 2006)*, 2006, p. 517.

[22] K. Harrison-Broninski, *Human Interactions: The Heart And Soul Of Business Process Management: How People Reallly Work And How They Can Be Helped To Work Better*. Meghan-Kiffer Press, Feb. 2005.

[23] A. Agrawal, M. Amend *et al.*, *WS-BPEL Extension for People (BPEL4People), version 1.0*, 2007.

[24] A. Agrawal, M. Amend *et al.*, *Web Services Human Task (WS-HumanTask), version 1.0*, 2007.

[25] W. Shi, J. Wu *et al.*, "Facilitating the Flexible Modeling of Human-Driven Workflow in BPEL," in *Proc. of the 22 Int. Conf. on Advanced Information Networking and Applications - Workshops*, 2008, pp. 1615–1624.