Integrating Transactions into BPEL Service Compositions: An Aspect-Based Approach

CHANG-AI SUN, XIN ZHANG, and YAN SHANG, University of Science and Technology Beijing, China
MARCO AIELLO, University of Groningen, The Netherlands

The concept of software as a service has been increasingly adopted to develop distributed applications. Ensuring the reliability of loosely coupled compositions is a challenging task because of the open, dynamic, and independent nature of composable services; this is especially true when the execution of a service-based process relies on independent but correlated services. Transactions are the prototypical case of compositions spanning across multiple services and needing properties to be valid throughout the whole execution. Although transaction protocols and service composition languages have been proposed in the past decade, a true viable and effective solution is still missing. In this article, we propose a systematic aspect-based approach to integrating transactions into service compositions, taking into account the well-known protocols: Web Service Transaction and Business Process Execution Language (BPEL). In our approach, transaction policies are first defined as a set of aspects. They are then converted to standard BPEL elements. Finally, these transaction-related elements and the original BPEL process are weaved together, resulting in a transactional executable BPEL process. At runtime, transaction management is the responsibility of a middleware, which implements the coordination framework and transaction protocols followed by the transactional BPEL process and transaction-aware Web services. To automate the proposed approach, we developed a supporting platform called *Salan* to aid the tasks of defining, validating, and weaving aspect-based transaction policies, and of deploying the transactional BPEL processes. By means of a case study, we demonstrate the proposed approach and evaluate the performance of the supporting platform. Experimental results show that this approach is effective in producing reliable business processes while reducing the need for direct human involvement.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architecture; D.2.2 [**Software Engineering**]: Design Tools and Techniques; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Reliability, Transaction, Performance, Measurement

Additional Key Words and Phrases: Web services, transaction management, Business Process Execution Language, aspect-oriented programming

This research was supported in part by the National Natural Science Foundation of China (grant no. 61370061), the Beijing Natural Science Foundation of China (grant no. 4112037), the Fundamental Research Funds for the Central Universities (grant no. FRF-SD-12-015A), and the Beijing Municipal Training Program for Excellent Talents (grant no. 2012D009006000002).
Authors' addresses: C.-A. Sun, X. Zhang, and Y. Shang, School of Computer and Communication Engineering, University of Science and Technology Beijing, 30 Xueyuan Road, Haidian District, Beijing 100083, China; emails: casun@ustb.edu.cn, s20131068@xs.ustb.edu.cn, shangyan@live.com; M. Aiello, Johann Bernoulli Institute, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands; email: aiellom@cs.rug.nl.

DOI: http://dx.doi.org/10.1145/2757288

ACM Reference Format:
Chang-Ai Sun, Xin Zhang, Yan Shang, and Marco Aiello. 2015. Integrating transactions into BPEL service compositions: An aspect-based approach. ACM Trans. Web 9, 2, Article 9 (May 2015), 31 pages.
DOI: http://dx.doi.org/10.1145/2757288

1. INTRODUCTION

Service-oriented architecture (SOA) is a style for building highly scalable and dynamic systems that increasingly has been adopted to develop applications in the context of the Internet [Papazoglou et al. 2008]. Web services, a technology enabling the creation of SOAs, provide their functionalities by exposing a set of interfaces described by the standard Web Services Description Language (WSDL) [WSDL 2007]. Considering that functionalities of individual Web services are often very limited, a bundle of Web services are coordinated to realize complex business processes. Such orchestration of independent services can be described via the Web Services Business Process Execution Language (BPEL) [OASIS 2007], which is a widely recognized process-oriented executable service composition language. All of these standards and protocols are based on XML, an independent and exchangeable data representation markup language. Thus, SOA can be used to address some important and challenging issues in a distributed and heterogeneous environment, such as data exchangeability and application interoperability [Papazoglou et al. 2008]. Moreover, SOA is increasingly adopted to cater to dynamic and quickly changing requirements due to its provisions for modularity and standardization. On the other hand, ensuring the reliability of loosely coupled service compositions becomes a key issue, especially when Web services participating in such compositions may come from several application domains and are owned by different organizations.

The area of transaction management was initially proposed in the context of databases with the goal of addressing issues of consistency and reliability of database-centric applications [Gray 1981]. Traditionally, a transaction is a set of operations that logically constitute an atomic execution unit. The existing transaction models usually guarantee the ACID properties, namely atomicity, consistency, isolation, and durability. In controlled environments, these properties can be guaranteed, but in decentralized systems, such as SOA, a new set of requirements emerges and some traditional ones need to be loosened [Papazoglou 2003]. Most notably, the loose connectivity among interacting services may render the communication slow, coordination protocols may fail, and rolling-back mechanisms may not be available [Little 2003]. In recent years, various transaction models and protocols for Web services have been proposed by relaxing some ACID properties. The Business Transaction Protocol (BTP) [OASIS 2004] and WS-Transaction [WS-C 2007; WS-AT 2007; WS-BA 2007] are two groups of well-known transaction protocols for Web services.

If individual transaction protocols exist, their inclusions into compositions, as described by BPEL, are still an open problem [Tai et al. 2004; Sun and Aiello 2007] . No specifications or mechanisms are available on how the transactions can be supported in BPEL processes. Some efforts have been made to address this gap [Charfi et al. 2007; Fletcher et al. 2003; Sun 2009; Sun et al. 2011a; Tai et al. 2004]. One approach has been to extend BPEL to provide transactional activities. For instance, Fletcher et al. [2003] used BPEL's variables to express the coordination context and extend BPEL's activities to support the transaction operations, such as *new*, *confirm*, and *cancel*. A limitation of this approach is that the BPEL engine must be modified to support the extension of BPEL's activities; this restricts the applicability of this method. Another effort is to turn to AOBPEL [Charfi and Mezini 2007], an aspect-oriented extension to BPEL, for integrating transaction into BPEL processes. Similarly, this approach relies on the nonstandard BPEL; thus, it is not applicable for existing BPEL engines. Furthermore, the current implementation only supports the WS-AtomicTransaction (*WS-AT*) specification.

The other way is to keep BPEL as it is and mix the coordination logic and transaction logic. In our previous work [Sun 2009], transaction logic is explicitly implemented in the service composition specification. The resulting BPEL process not only contains the business logic but also the transaction logic. This means that one has to do hard coding of the transaction logic for each BPEL process. Furthermore, this approach results in maintenance problems—that is, one has to recode the transaction logic once the business logic changes. To overcome this, we proposed a framework for the integration of BPEL and WS-Transaction specification [Sun et al. 2011a]. We used a middleware to encapsulate the transaction logic and govern transaction management there. The approach relied on the Service-Centric System Engineering (SeCSE) platform [SeCSE Team 2007], which is a very complex service composition platform and in turn an overkill for simply integrating transaction management into compositions while using ordinary BPEL engines. Similarly, Tai et al. [2004] presented a WS-Policy–based approach to implement transactions into BPEL. The coordination policy is attached to BPEL partner links and scopes.

In this article, we go one step further by proposing an aspect-based approach to integrating transactions into BPEL service compositions. This approach partially leverages our previous work on integrating transactions into BPEL and further addresses the key open issues. Aspects are used to define transaction policies that designers wish to declare on the business process. These transaction policies are then converted into standard BPEL elements by a preprocessing step and weaved into BPEL processes to generate a transactional BPEL process. The transactional BPEL process is deployed as normal BPEL processes and executed on any standard BPEL engine. Maintenance is achieved by automation of the translation. In fact, we implemented a supporting platform that provides the aids to edit the aspect-based transaction policies, validate, and weave the transaction-related elements into BPEL processes. With such a platform, designers are able to effectively integrate transactions into BPEL business processes with very limited efforts. Finally, we conducted a case study to demonstrate and validate the proposed approach and the supporting platform. Experimental results not only show the feasibility of our approach but also indicate that the performance and transaction integration efforts are satisfactory.

Original contributions of this article are as follows:

(1) A systematic aspect-based approach to integrating transactions into BPEL service compositions that is “light-weight” (no changes on the standard BPEL specification and a little effort required for transaction integration) and “generic” (not relying on any specific BPEL engine).
(2) An implemented supporting platform called *Salan*, which consists of components that are expected to automate the approach to the maximum and hide the details of transaction integration. It also includes the middleware for transaction protocols, such as *WS-AT* and *WS-BusinessActivity* (*WS-BA*).
(3) A comprehensive case study that was conducted to validate the proposed approach and the developed platform. In this case study, we examined *WS-AT* and *WS-BA*. To the best of our knowledge, the present proposal is the first methodical attempt to integrate *WS-BA* into BPEL service compositions.

The remainder of this article has the following organization. Section 2 introduces the underlying concepts related to WS-Transaction, WS-BusinessActivity-Initiator, BPEL, and aspect-oriented programming (AOP). Section 3 proposes an aspect-based transaction integration approach for BPEL service compositions. Section 4 describes the supporting software platform. Section 5 reports on a case study where the proposed approach and platform were employed to integrate transactions into BPEL service compositions. Section 6 reports the performance evaluation of the proposed approach in terms of time and workload introduced by transaction integration and compares the performance of the proposed approach with that of an existing transaction integration approach. Section 7 discusses related work. Section 8 concludes the article and discusses possibilities for future work.

2. BACKGROUND

The Web service stack describes the relationship of the most relevant protocols in a layered fashion. WS-Transaction (including *WS-Coordination* (*WS-C*), *WS-AT*, and *WS-BA*) is part of the stack and deals with transactionality. *WS-BusinessActivity-Initiator* is an extension of *WS-BA* that specifies interactions between the transaction initiator and the coordinator. BPEL is an executable business process description language that describes the compositions of Web services. In addition to these Web service stack protocols, we overview AOP, a programming paradigm that aims to increase modularity by separating the cross-cutting concerns in designing and realizing large-scale software systems.

2.1. WS-Transaction

The *WS-Transaction* specifications define a mechanism for transactional interoperability among Web services. They include an extensible coordination framework—the *WS-C* specification [WS-C 2007]—and two specific coordination types—atomic transaction with the *WS-AT* specification [WS-AT 2007] and business activity with the *WS-BA* specification [WS-AT 2007].

The *WS-C* specification defines a generic and extensible coordination framework to provide the foundation for implementing transaction interactions among Web services. Through the framework, all participants in a transaction can achieve a consistent agreement on the outcome.

The *WS-AT* specification defines an atomic transaction coordination type that has an "all or nothing" property. *WS-AT* is used for simple short-lived and small-scale interactions made up of Web services. *WS-AT* is intended to satisfy the ACID properties and guarantee that all participants in a transaction achieve the consistent outcome.

The *WS-BA* specification defines a business activity coordination type that is used for complex long-lived activities made up of Web services. Therefore, *WS-BA* no longer uses the "lock" policy to process transactions; instead, it provides the "compensation" mechanism. *WS-BA* assumes that all updates are committed immediately; thus, it does not lock resources or delay the data updates. When a transaction fails, *WS-BA* provides the compensation mechanism to abort the effect of committed operations. *WS-BA* supports two coordination types—atomic and mixed—and two protocol types—*Business-Agreement-With-Participant-Completion* (*BAwPC*) and *BusinessAgreementWithCoordinatorCompletion* (*BAwCC*).

2.2. WS-BusinessActivity-Initiator

A transaction usually consists of an initiator, a coordinator, and one or more participants. The *WS-BA* specification defines how a coordinator communicates with participants in a business activity transaction. Unfortunately, the *WS-BA* specification does not specify how the initiator communicates with the coordinator. This forces vendors to integrate *WS-BA* coordination function into their business process engine. However, it is difficult to guarantee interoperability between the initiator and coordinator when they are implemented by different parties. To address this problem, Erven et al. [2007] proposed an extension of the *WS-BA* specification that is known as the *Web Service-BusinessActivity-Initiator* (*WS-BA-I*) protocol. *WS-BA-I* separates the transaction logic from the business logic and standardizes the communication interface between initiator and coordinator.

2.3. Business Process Execution Language

To realize complex business goals, many activities need to be executed according to predetermined flows of execution and invocation. BPEL is an executable service composition language that orchestrates Web services to form a business process [OASIS 2007]. BPEL process definitions may be exported and reused. In this way, a BPEL process can be used as a basic service unit to participate in higher-level business processes. BPEL processes consist of four sections: *partner link statements*, *variable statements*, *handler statements*, and *interaction* steps. Activities are basic interaction units of a BPEL process and are further divided into *basic* activities and *structural* activities. Basic activities is an atomic execution step in the BPEL process, including *assign*, *invoke*, *receive*, *reply*, *throw*, *wait*, *empty*, and so on. Structural activities are compositions of basic activities and/or structural activities, including *sequence*, *switch*, *while*, *flow*, *pick*, and so on.

Handlers provide a mechanism for performing predefined actions when a fault or other event is caught at runtime. A *compensation handler* provides a way to abort the impact of the completed activities. If a compensation is necessary, the hand-coded operations specified in the compensation handler are executed. In this context, there is no concept of transaction, and there is not a mechanism for coordinating multiple participants to reach an agreed outcome [Greenfield et al. 2003]. On the other hand, BPEL lacks the support of a transaction coordination mechanism to ensure its reliability.

2.4. Aspect-Oriented Programming

In many situations, functionalities scatter in several modules of an application, such as *log*, *transaction*, *security*, *auditing*, *authority*, and *cache*. These functionalities are called *cross-cutting concerns*. Although various aspects, such as business logic and transaction logic, can be thought of as being relatively independent of each other, they have to be combined together at the implementation time. Current practice is to hand code the functionalities at the place where the application needs them. Thus, the implementation of cross-cutting concerns likely results in intertangled executable code, which makes such applications cumbersome to implement and maintain.

AOP [Kiczales et al. 1997] is a programming paradigm proposed to address the preceding challenges by allowing the programmer to express cross-cutting concerns in stand-alone modules called *aspects*. An *aspect* is a combination of the pointcut and the advice. A *pointcut* refers to the point of execution in the application at which cross-cutting concerns need to be applied, whereas an *advice* refers to the additional code to apply to an existing model (i.e., code joined to specified points in the program). Aspects and their corresponding treatments are automatically "weaved" into the implementation. In this way, cross-cutting concerns can be easily modularized, and developers just need to specify where and in what condition these aspects should take place in a declarative way.

3. ASPECT-BASED TRANSACTION INTEGRATION IN BPEL

3.1. An Overview of the Approach

Transactions are typical nonfunctional requirements in business processes, and they may occur at multiple places within BPEL processes. In this context, AOP presents an appropriate choice that can be employed to address the problem of integrating transactions into BPEL service compositions. We propose an aspect-based transaction integration framework for BPEL service compositions, as illustrated in Figure 1. The framework is composed of three parts. At the end of BPEL service compositions, *aspect-based transaction policies* are defined, then validated, and finally weaved into the original BPEL process. At the other end of Web services, necessary extensions are required

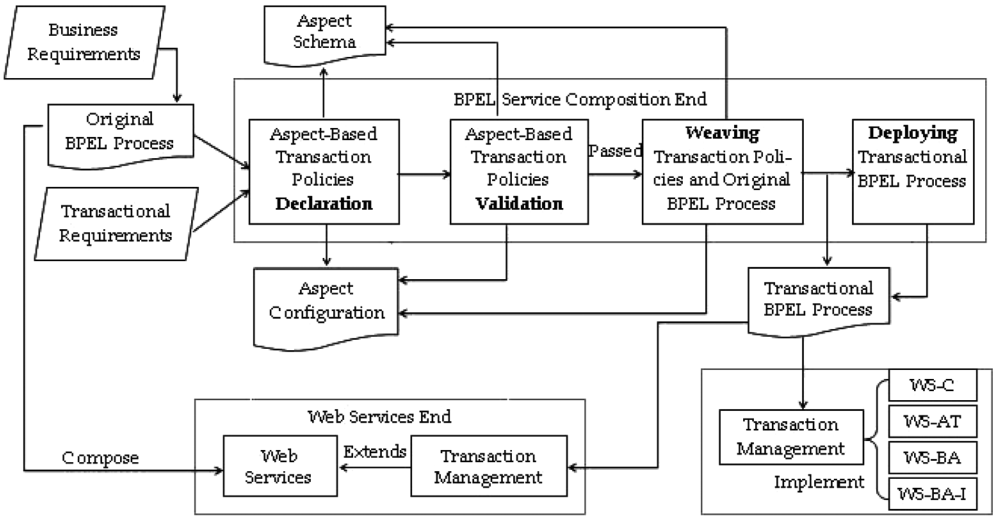

Business Requirements
Original BPEL Process
Transactional Requirements
Aspect Schema
BPEL Service Composition End
Aspect-Based Transaction Policies **Declaration**
Aspect-Based Transaction Policies **Validation**
Passed
Weaving Transaction Policies and Original BPEL Process
Deploying Transactional BPEL Process
Aspect Configuration
Transactional BPEL Process
Web Services End
Compose
Web Services
Extends
Transaction Management
Transaction Management
Implement
WS-C
WS-AT
WS-BA
WS-BA-I

Fig. 1. The aspect-based transaction integration framework for BPEL dervice vompositions.

to support transaction integration at the service composition level. *Transaction management* is left to a middleware that implements the coordination infrastructure and transaction protocols and is responsible for transaction coordination.

An original BPEL process only contains business logic without the concept of transactions. Aspect-based transaction policies are used to address the transactional requirements. First, we declare aspect-based transaction policies with respect to the given BPEL process, and such a declaration results in an *aspect configuration*. Then, we *validate* declared transaction policies using an *Aspect Schema*, which defines the metamodel of aspect-based transaction integration. Before weaving, it is necessary to validate transaction policies and make sure that the declared transaction policies are acceptable and valid. After the validation passes, the weaving is responsible for converting aspect-based transaction policies to standard BPEL elements, which will thereafter invoke transaction management to execute the transaction protocols, and weaving these BPEL elements into the original BPEL process. After the weaving, the original BPEL process and aspect-based transaction policies are merged together. As a consequence, a transactional BPEL process is synthesized, which contains not only the business logic but also the transaction logic. Such a resulting transactional BPEL process is still a standard BPEL process—that is, no extra BPEL elements are introduced; so, it can be interpreted by any standard BPEL engine.

Four steps are necessary to integrate transactions into BPEL processes based on AOP: aspect-based transaction policies declaration, aspect-based transaction policies validation, weaving transaction policies and the original BPEL process, and deploying the transactional BPEL process. Among them, only the step of declaring aspect-transaction policies needs direct human intervention; the other steps are automated as we show later in the article.

Transaction integration implemented using such a preprocessing step has the following benefits. First, the BPEL engine does not need to be modified to support transaction integration. This is because the resulting BPEL process, after the weaving, only consists of standard BPEL elements, and it can be executed by existing BPEL engines. Second, details on transaction integration are completely hidden from users during the weaving, and transactions are managed by a specifically designed middleware. This means that the designer can focus on the declaration of transaction policies and disregard further implementation details. Third, transaction integration maintenance and evolution becomes extremely easy and efficient with our approach. When one wants to change transaction policies on any given BPEL process, one simply changes aspect-based transaction policies via the *Aspect Generator*, a component of the supporting platform, and the supporting platform will then take over the remaining tasks.

On the other hand, the approach may have some limitations. For example, it is not involved in monitoring the process, particularly changes that take place in the process definition. When the original BPEL process undergoes significant changes, one has to repeat the whole procedure. However, the efforts involved in such a repetition can be reduced significantly with the aid of a tool, as we describe later.

3.2. Aspect-Based Transaction Integration Process

3.2.1. Aspect-Based Transaction Policy Declaration. Transaction policies are declared according to AOP principles, which contain two important parts: pointcut and advice. The *pointcut* described by an XPath expression refers to the position where transaction operations should occur, whereas the *advice* refers to the expected transaction operations. Accordingly, we propose an aspect-based transaction policy declaration language, and its definition is illustrated later in Figure 14 in Appendix A. Generally speaking, these aspect-related elements are defined in an XML file, and their syntax follows the AOP framework.

We further explain their semantics in the context of transaction policies:

—The <*aspect-config*> element is the root of a transaction policy declaration. It includes one <*variables*> and one or more <*aspect*> elements.
—The <*variables*> element is a set of input and output variables of operations, and its child element is <*variable*>. The <*variable*> element refers to the input or output variable of a specific operation. It has two attributes: *name* refers to the name of the variable, and *type* refers to the type of variable.
—The <*aspect*> element declares an aspect of transaction concern. It has an attribute "name," which refers to the name of the aspect, and its child elements are one or more <*pointcut*> and <*advice*> elements.
—The <*pointcut*> element specifies a target position of transaction operation. This element has two attributes: *name* refers to the name of pointcut, and *expression* specifies the target position in the BPEL process that is represented as an XPath expression.
—The <*advice*> element specifies what should be added at the target position that is described by the *bind-to* attribute, whose value actually is a pointcut name. This element has three attributes: *name* refers to the name of advice, *type* refers to the weaving policy, and *bind-to* refers to a pointcut's name, and one <*transaction*> element.
—The <*transaction*> element specifies attributes that are related to a transaction operation. This element has five attributes: *name* refers to the name of transaction, *type* refers to the transaction type (namely transaction protocol), *operation* refers to the transaction operation, *inputVariable* refers to the input variable of transaction operation, and *outputVariable* refers to the output variables of transaction operation. The value of *inputVariable* and *outputVariable* should be declared by the <*variable*> element. It also has one or more <*param*> elements.
—The <*param*> element specifies the parameters of a transaction operation. This element has two attributes: *name* refers to the name of the parameter, and *value* refers to the value of the parameter.

When the aspect-based transaction declaration language is used, the value for some of the attributes must be fixed according to the transaction type—that is,

- For the *type* attribute of the *<advice>* element, the value must be *before*, *after*, or *child*.
- For the *type* attribute of the *<transaction>* element, the value has to be *wsat*, *wsba-atomic*, or *wsba-mixed*.
- For the *operation* attribute of the *<transaction>* element, the value is assigned in the following way:
  - For the *wsat* transaction type, the value is to be *begin*, *commit*, or *rollback*.
  - For the *wsba-atomic* and *wsba-mixed* transaction types, the value must be *createNewContextWithWSBAI*, *getCoordinationContextWithMatchcode*, or *complete Participants*.
  - For the *wsba-atomic* transaction type, the value has to be *cancelOrCompensateAllParticipants*, *closeAllParticipants*, or *assignTransaction*.
  - For the *wsba-mixed* transaction type, the value has to be *cancelParticipants*, *compensateParticipants*, *closeParticipants*, or *assignTransaction*.
- For the *type* attribute of variable, the value is assigned in the following way:
  - For the *wsat* transaction type, the value must be *BeginRequestType*, *Begin ResponseType*, *CommitRequestType*, *CommitResponseType*, *RollbackRequestType*, or *RollbackResponseType*.
  - For the *wsba-atomic* and *wsba-mixed* transaction types, the value is to be *createNewContextWithWSBAIRequestType*, *createNewContextWithWSBAI ResponsetType*, *getCoordinationContextWithMatchcodeRequestType*, *getCoordinationContextWithMatchcodeResponseType*, *completeParticipantsRequestType*, or *completeParticipantsResponseType*.
  - For the *wsba-atomic* transaction type, the value can be *cancelOrCompensateAll ParticipantsRequestType*, *cancelOrCompensateAllParticipantsResponseType*, *close AllParticipantsRequestType*, or *closeAllParticipantsResponseType*.
  - For the *wsba-mixed* transaction type, the value can be *cancelParticipantsRequestType*, *cancelParticipantsResponseType*, *compensateParticipantsRequestType*, *compensateParticipantsResponseType*, *closeParticipantsRequestType*, or *closeParticipantsResponseType*.

In addition to the preceding operations, we provide an operation *assignTransaction* in the *wsba-atomic* and *wsba-mixed* transaction type. For the *wsba* transaction implementation, the coordination context should be sent to participant Web services as a request parameter. Thus, we designed the *assignTransaction* operation to copy the coordination context object from the response of *getCoordinationContextWithMatchcode* operation to the request of business operation when invoking a participant in the business activity. The *assignTransaction* operation takes two parameters—*from* and *to*—and uses an XPath expression to specify the request and response variables, respectively.

With this, one can declare transaction policies for a BPEL process. The resulting transaction policy declaration is an XML file (aspect-config.xml). To aid the user to declare transaction policies (or actually edit such an XML file), we developed a GUI style editor, *Aspect Generator*, which is part of the supporting platform that will be described later.

3.2.2. Validation. During the transaction policy declaration, some errors may be present. For instance, some typos may occur when one edits the aspect-based transaction policies manually; some mismatched operations may be introduced for a specific transaction type. Therefore, it is necessary to validate the resulting transaction policy

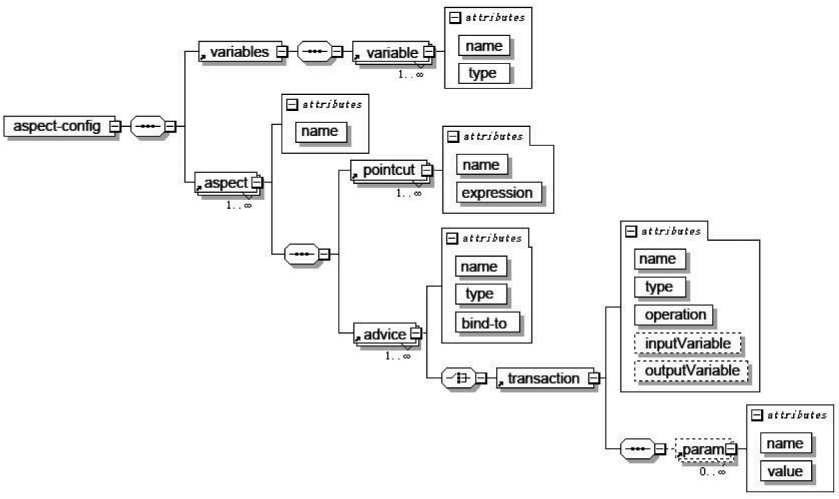


Fig. 2. The structure of aspect-config.xsd.

declaration (namely aspect-config.xml), before weaving them into the original BPEL process. Otherwise, it will result in a malicious or nonexecutable BPEL process. Four kinds of validation are involved. These include syntax validation, variable validation, transaction validation, and hierarchy validation.

(1) *Syntax validation* involves checking whether the aspect-config.xml file follows the syntax defined later in Figure 14. To formalize the validation, we provide a formal syntax description by the *aspect-config.xsd*; this is illustrated in Figure 2. This XML schema shows the structure and relationships among elements in aspect-config.xml. When a syntax error exists in the aspect-config.xml file, the *validator* will report it and provide detailed information about the detected error.
(2) *Variable validation* involves checking whether the variable types in the aspect-config.xml file are correctly matched. For instance, the input and output variable of the *begin* operation in the *wsat* transaction type should be *BeginRequestType* and *BeginResponseType*, respectively. An error of this kind is very common, irrespective of whether the aspect-config.xml is edited manually or through our GUI tool, *Aspect Generator*.
(3) *Transaction validation* involves checking whether a transaction is correctly opened and closed. Each transaction has a boundary, which is usually specified by transaction opening and closing operations. For instance, for a *wsat* transaction type, the transaction is opened by the *begin* operation and closed by the *commit* operation or the *rollback* operation; for a *wsba-atomic* transaction type, the transaction is opened by the *createNewContextWithWSBAI* operation and closed by the *closeAll-Participants* operation; for a *wsba-mixed* transaction type, the transaction is opened by the *createNewContextWithWSBAI* operation and closed by the *closeParticipants* operation. If a transaction opening does not match with the transaction closing, the *validator* must report such an error.
(4) *Hierarchy validation* involves checking whether the transaction operation is correctly declared to be weaved into the hierarchy of the BPEL process. A BPEL process is composed of a set of statement blocks. A statement block is represented as an XML element with specific semantics and may have child elements. All statement blocks in the BPEL process form a hierarchical structure. Transaction operations

in the aspect-config.xml file should be inserted at the appropriate level in the BPEL hierarchy. For instance, the opening and closing transaction operations should be at the same level of statement blocks; otherwise, the *validator* must report such an error.

The validation is only passed when all four kinds of validation return as true.

3.2.3. Weaving. Once aspect-based transaction policies pass the validation, weaving takes place. Weaving is a synthesis process that involves reading aspect-based transaction policy declarations, converting the transaction policy into BPEL elements, and merging the converted elements into the original BPEL specification; this results in a transactional BPEL process.

A key task during the weaving process is to convert aspect-based transaction policies into a set of standard BPEL elements, which activate predefined transaction operations that are implemented by transaction management middleware. To do this, we define a set of conversion rules:

—*Rule I*: A *variable* element in aspect configuration is mapped onto a *variable* element in BPEL, and the *name* and *type* attributes of the variable element in *aspect configuration* are mapped to the *name* and *messageType* attributes of the variable element.

—*Rule II*: A *transaction* element in *aspect configuration* is mapped onto an invoke activity in BPEL, except when its type attribute is *assignTransaction*. The *name*, *operation*, *inputVariable*, and *outputVariable* attributes of the *transaction* element are mapped to the *name*, *operation*, *inputVariable*, and *outputVariable* attributes of the invoke activity. The *partnerLink* and *portType* attributes of the *invoke* element are generated according to the *type* attribute of the *transaction* element.

—*Rule III*: When the *operation* attribute of a *transaction* element is *assignTransaction*, the *transaction* element is mapped in *aspect configuration* to an *assign* element in BPEL to copy coordination context to the request parameter of Web services' operation.

—*Rule IV*: A *pointcut* element in *aspect configuration* has no direct mapping element in BPEL. It is used to locate the target position described by the xpath *expression* attributed in the *pointcut* element.

—*Rule V*: A *param* element in *aspect configuration* is mapped onto an *assign* element in BPEL to assign the parameter value of a transaction operation.

—*Rule VI*: A *namespace*, *import* tag, *partnerLink* tag, and *partnerLinkType* tag are generated according to the *transaction* type declared in *type* attribute of the *transaction* element in *aspect configuration*.

During the weaving process, the elements in aspect-based transaction policy declaration are going to be converted into BPEL elements, following the preceding conversion rules. These elements are then added to the BPEL specification. For example, a *begin* operation and its request and response variables specified in the aspect-based transaction policy declaration are correspondingly converted into the variable and the invoke activity of the BPEL specification (Figure 3). Note that the prefix "ns" refers to the prefix of the namespace of the transaction middleware Web service. To employ middleware to take over transaction management (i.e., *beginTransaction*), some extra XML tags must be added to the BPEL specification. As a result, the namespace of the transaction middleware must be added to the converted BPEL specification, and the *import* tag and *partnerLink* tag need to be added to the BPEL specification as well.

The converted elements including *namespace*, *variable*, and *invoke* activities are then merged into the resulting BPEL specification. During the merging process, several XML files need to be read or written; examples of these files include the aspect-based

Aspect-Based Transaction Policy Declaration (in aspect-config.xml)	`<variable name="beginRequest" type="beginRequestType"/> <variable name="beginResponse" type="beginResponseType"/>`
	`<transaction name="beginTransaction" type="wsat" operation="begin" inputVariable="beginRequest" outputVariable="beginResponse"/>`
Converted BPEL Specification	`<bpel:variable name="beginRequest" messageType="ns:beginRequest"/> <bpel:variable name="beginResponse" messageType="ns:beginResponse" />`
	`<bpel:invoke name="beginTransaction" partnerLink="TransactionManagerPL" operation="begin" portType="ns:TransactionManagerImpl" inputVariable="beginRequest" outputVariable="beginResponse" />`

Fig. 3. An illustration of the conversion of transaction policies into BPEL elements.

transaction policy file (aspect-config.xml), the original BPEL specification (*.bpel), the Web Service specification of the BPEL process (*Artifacts.wsdl), the transaction middleware Web services specification (TransactionManager*.wsdl), and the deploy file (deploy.xml). Among these files, the BPEL specification, its Web Service specification, and the deploy file are from the original BPEL process artifacts. On the other hand, the transaction middleware Web service is newly created to support the transaction type as specified in the transaction policy (namely, aspect-config.xml).

To avoid faults in such a complex process, we opt for creating a supporting tool; in particular, we developed *Weaver*, which will be described in Section 4.

In our approach, the weaving process is completed according to a set of conversion rules that convert transaction policies into BPEL elements. These elements are mainly relevant to transaction contexts and transaction operations invocations. It is hard to theoretically prove the correctness of the conversion rules due to the lack of a general BPEL formal semantics. Instead, we turn to testing the implementation of the conversion rules in various transaction protocol settings.

3.2.4. Deploying. The deployment of a transactional BPEL process, as described with the previous steps, is the same as the one for any BPEL process. One simply replaces the original process specification and its Web service description in the deploy directory of the BPEL execution engine with the transactional BPEL process specification and its Web service description. Transaction middleware Web service specification and aspect-based transaction policy declaration schema (aspect-config.xml) are then added.

When the replacement is finished, the transactional BPEL process is ready for execution. Conveniently, any existing BPEL engine can be used for this task, and no modifications or extensions for such an engine are needed.

3.3. Transaction Management

Next we answer the following questions: which component will be responsible for transaction management, and how do Web services in BPEL compositions participate in declared transactions?

3.3.1. Transaction Management for BPEL Service Compositions. Three transaction protocols (or types) are examined in our approach as an illustration. They include *wsat*, which is defined by the *WS-AT* specification [WS-AT 2007], *wsba-atomic*, which is defined by the *AtomicOutcome* of *WS-BA* specification [WS-BA 2007], and *wsba-mixed*, which is defined by the *MixedOutcome* of *WS-BA* specification [WS-BA 2007]. Accordingly, we developed three Web services to provide transaction management: *TransactionManager* for *wsat*, *TransactionManagerAtomic* for *wsba-atomic*, and *TransactionManagerMixed* for *wsba-mixed*. These services provide necessary transaction management operations, as defined by the corresponding specifications. For *TransactionManager*, these are the following:

—*begin()*, which is responsible for creating the coordination context and is called when opening an atomic transaction;
—*commit()*, which is responsible for making sure that the business work is performed and persistent; and
—*rollback()*, which is responsible for cancelling the performed work.

The *TransactionManagerAtomic* service provides the following five operations:

—*createNewContextWithWSBAI()*, which generates a coordination context;
—*getCoordinationContextWithMatchcode(String match-code),* which returns a participant-specific coordination context containing a matchcode as an invitation ticket—this coordination context will be passed to the participant's business operation as a parameter, and participant registers with the coordinator using the invitation ticket;
—*completeParticipants(List<String> participants)*, which directs participants specified by the matchcode list to complete their business work—for participants registered as *BAwCC*, the coordinator needs to tell them to complete their work;
—*cancelOrCompensateAllParticipants()*, which directs all participants to cancel or compensate their work; and
—*closeAllParticipants()*, which directs all participants to close.

In the *WS-BA-I* specification, each participant has a unique identification *matchcode*, which is defined by developers. For the *mixed* coordination type, each participant can reach its individual outcome; thus, one must use the matchcode list to specify on which participant the *cancel*, *compensate*, and *close* operations should execute. Similarly, the *TransactionManagerMixed* service provides the following six operations:

—*createNewContextWithWSBAI()*;
—*getCoordinationContextWithMatchcode(String match-code)*;
—*completeParticipants(List<String>participants)*;
—*cancelParticipants(List<String> participants)*;
—*compensateParticipants(List<String> participants)*; and
—*closeParticipants(List<String> participants)*.

3.3.2. Development of Transaction-Aware Web Services. When a Web service part of a BPEL service composition involves a transaction as a participant, some transaction-related behavior must be supported. A participant Web service should implement not only its common business operations but also transaction interfaces. The latter con-tain operations related to the specific transaction protocol. For instance, when the *WS-AT* is considered, transaction operations include *prepare*, *commit*, and *rollback*. With these transaction operators, transaction management in the form of the coordinator can then orchestrate the participant Web services to finish a job in the specified transactional way. This means that to support transactions, extensions to the common Web services are necessary.

In our previous work [Sun et al. 2011b], we proposed a framework that can extend a Web service to support transaction operations. The framework is presented by extending Web services with transaction interfaces for *WS-AT*. The basic idea lies in the separation of the transaction logic from the business logic. Business operations related to the transaction are encapsulated within transaction operators, which are used to follow the state transition during the coordination.

Next we propose two ways to extend Web services for supporting transaction behaviors. Such extensions are illustrated using Apache Kandula 1 [Apache 2013a]. This is because Apache Kandula is widely adopted and available as open source software providing basic infrastructure for transaction management.

Atomic transaction, namely *WS-AT*, can be supported in the following ways:

—One way is to implement the *NamedXAResource* interface within Web services. *NamedXAResource* is a JTA interface from Apache Geronimos (http://geronimo.apache.org), which is an open source implementation of J2EE and provides operations such as *prepare*, *commit*, and *rollback* [Maple 2004]. Web services can be directly incorporated into Apache Kandula if it implements a J2EE transaction using the *NamedXAResouce* interface.
—The other way is to implement the *AbstractParticipant* class. *AbstractParticipant* is an abstract class that defines a set of transaction operations such as *prepareOperation*, *commitOperation*, and *rollbackOperation*. In this way, Apache Kandula allows those Web services that have not implemented J2EE transactions to join an atomic transaction.

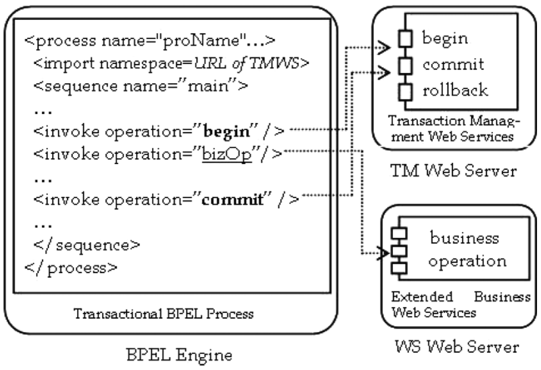
Similarly, to support a business activity, namely *WS-BA*, one can work in the following way. First, the business operations are extended. During the extension, the coordination context must be added as a parameter of the business operation. For instance, for the orderFlight business operation, *wscoor:CoordinationContext* is newly introduced. The extended WSDL segment aware of the coordination context is shown in Figure 4. Second, the Web service needs to extend one of the following abstract classes provided by Apache Kandula according to the specific coordination protocols of *WS-BA*:

—*org.apache.kandula.coordinator.ba.participant.BAwPCParticipant*, which refers to *BAwPC*, and
—*org.apache.kandula.coordinator.ba.participant.BAwCCParticipant*, which refers to *BAwCC*.

In summary, if a Web service is going to participate in a transaction, the corresponding transaction interface needs to be implemented. Only when these transaction interfaces are implemented can the coordinator then communicate with the participants. In other words, when the coordinator sends out a transaction-related message, one of the transaction operations of a participant Web service will be invoked. At runtime, the transaction management middleware, which has realized the *WS-Coordination*

```
<wsdl:definitions>
  <wsdl:types>
    <xsd:schema xmlns:xsd=
        "http://www.w3.org/2001/XMLSchema">
    <xsd:element name="applicationRequest">
      <!--original variables -->
      <xsd: element name="flightNo" type="..." .../>
      <!--newly added below-->
      <xsd:element ref="wscoor:CoordinationContext" />
    </xsd:element>
    </xsd:schema>
  </wsdl:types>
</wsdl:definitions>
```

Fig. 4. An illustration of a WSDL segment that extends business operations with coordination context.



```
<process name="proName"...>
  <import namespace=URL of TMWS>
  <sequence name="main">
  ...
  <invoke operation="begin" />
  <invoke operation="bizOp"/>
  ...
  <invoke operation="commit" />
  ...
  </ sequence>
</ process>
```

begin
commit
rollback
Transaction Management Web Services
TM Web Server
business operation
Extended Business Web Services
WS Web Server
Transactional BPEL Process
BPEL Engine

Fig. 5. The interactions of the BPEL engine and Web servers.

and *WS-Transaction* protocols (*WS-AT* or *WS-BA*), will take over the communication. We further argue that the transaction-aware extensions discussed here should be left for the developer of Web services because only she knows the details of the specific business logic. Practically, only those business operations that require transactional properties to be guaranteed should be extended with transactional operators, and there is no restriction on how many business operations one service should support.

3.3.3. Interactions. When a transactional BPEL process, extended business Web services, and transaction management middleware are properly deployed, they are then ready for execution. Figure 5 demonstrates the interactions between the *BPEL Engine*, where the transactional BPEL process is deployed, and *Web Servers*, where the extended business Web services and transaction management Web services are deployed. Considering that the transactional BPEL process and extended business Web services come from the original BPEL project, they are already configured to run. To invoke the transaction management Web services, we have to configure their URLs to the endpoints. During the execution of the transactional BPEL process, if an operation (usually an *invoke* activity) that is associated with the transaction management middleware is invoked, the BPEL engine will then send a message to *TM Web Server*, where transaction management Web services are deployed. Similarly, when an operation provided by extended business Web services is invoked, the engine will send a message to *WS Web Server*, where extended business Web services are deployed.

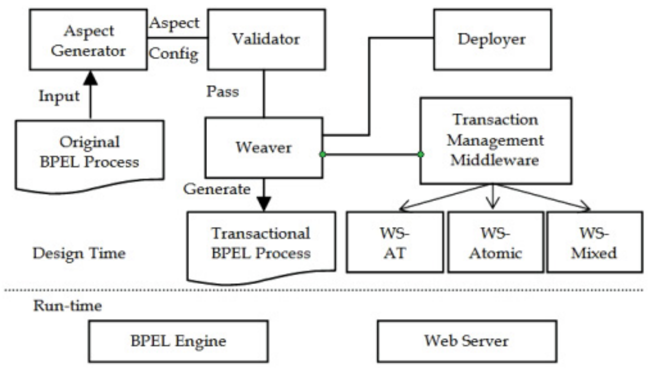


Fig. 6. The architecture of the supporting platform.

4. *SALAN*: A SUPPORTING PLATFORM OF ASPECT-ORIENTED TRANSACTION INTEGRATION FOR BPEL SERVICE COMPOSITIONS

Salan is a supporting platform, providing a set of components for aspect-based transaction policy declaration, validation, weaving, and deploying, and a transaction management middleware that is composed of a set of Web services for the coordination infrastructure (*WS-C*) and transaction protocols (including *WS-AT*, *WS-BA*, and *WS-BA-I*). *Salan* automates most of the steps in the approach proposed in Section 3.

Figure 6 illustrates the architecture of *Salan*. *Aspect Generator* is used to edit aspect-based transaction policies with respect to a given BPEL process, and declared transaction policies are exported to an XML file (i.e., aspect-config.xml). *Validator* validates the declared aspect-based transaction policies, following the given rules. Transaction policies are only passed to *Weaver* for weaving when no rule is violated. *Weaver* first converts transaction policies into the aspect-config.xml to standard BPEL elements according to the AOP semantics and BPEL's syntax and then merges these converted elements into the original BPEL process. The resulting BPEL process consists of transaction operation stubs that will be implemented by *Transaction Management Middleware*. *Deployer* deploys the resulting transaction BPEL process artifacts to the specified site, such as its service composition specification (*.bpel), Web service description (*.wsdl), and deploy file (deploy.xml). *BPEL Engine* is used to interpret the transactional BPEL process. Any existing BPEL engine can be used for this purpose. *Transaction Management Middleware* is used to implement the transaction infrastructure and protocols. Some tools are available that partially implement *WS-Transaction* protocols [Schäfer et al. 2008], such as Apache Kandula [Apache 2013a], WS-AT for WAS developed by IBM [2006], JBoss Transaction [2013], and Open WS-Transaction [Vasquez et al. 2005]. In our study, Apache Kandula is selected for this task and extended as a set of Web services.

With the aid of *Salan*, one can significantly save his or her efforts when integrating transactions into BPEL processes. In summary, *Salan* supports the following features:

—Visually *declaring* aspect-based transaction policies with respect to a given BPEL process and exporting the resulting transaction policy declaration to an XML file (aspect-config.xml).
—*Validation* of aspect-based transaction policies (aspect-config.xml) by the validation rules.
—*Weaving* aspect-based transaction policies into BPEL processes, following the conversion and merging principles.
—*Deploying* the transactional BPEL process for execution.

—An integrated transaction management middleware that implements *WS-Coordination* and *WS-Transaction* as a set of Web services. Note that *Salan* by itself only provides the implementations of transaction infrastructure and protocols on the service composition side; however, implementing transactional interfaces on the Web service side is left for service developers.

5. CASE STUDY

The process of organizing a holiday is something that typically requires transactional behavior on a number of steps. We use this recurrent example as a case study to show the working and advantages of the proposed approach. We first describe a *Holiday* process, then discuss its transaction requirements, followed by the transactional implementation using the proposed approach, and finally test and demonstrate the running results. We use the *payment* subprocess to illustrate the integration of the *wsat* transaction and use *car booking* and *hotel booking* subprocesses to illustrate the integration of the *wsba* transaction. A video demonstrating the use of the tool on the case study is available at http://www.cs.rug.nl/∼aiellom/videos/index.html. All coordination and protocol types defined in *WS-AT* and *WS-BA* specifications have been supported in the current version of *Salan*. All Web services are implemented in Java language.

5.1. The *Holiday* Process

The *Holiday* process describes a typical traveling process scenario involving booking a car and hotel, then making a payment after the successful bookings. Three distributed Web services are assumed to provide the functionalities. The Web service *rentACar* provides two operations: *getOffers* and *doBooking*. The *getOffers* operation will build a list of available cars, such as Car-1003, Car-1004, and Car-1005. The *doBooking* operation books the specific car by a given car identification. The Web service *rentARoom* provides two operations: *getOffers* and *doBooking*. The *getOffers* operation builds a list of available rooms, such as Room 9999, Room 10000, Room 10001, and Room 10002. The *doBooking* operation books the specific room by a given room identification. The Web service *BankOne* fulfills the functionality of payment, and two operations are provided. The debit operation of *BankOne* is first invoked to decrease the money from *account0*, and the credit operation of *BankOne* is invoked to increase the money in *account1*.

The *Holiday* process is composed of two flow activities and one payment sequence activity. The first flow activity consists of two branches to invoke the *getOffers* operation of the *rentACar* service and the *rentARoom* service, respectively. The second flow activity consists of two branches to invoke the *doBooking* operation of the *rentACar* service and the *rentARoom* service, respectively. The *payment* process receives an input message composed of three fields—*account0*, *account1*, and *amount*—and transfers the given amount of money specified by *amount* from sender account *account0* to receiver account *account1*. To make the main process simple (for the purpose of illustration), we assume that Car-1004 and Room 10000 are booked, the input message is empty, and the judgment logic is skipped. Figure 7 illustrates such a simplified *Holiday* process.

5.2. Declaring Aspect-Based Transactional Policies for the *Holiday* Process

The BPEL sequence activity *payment* describes the transfer of funds between two accounts. During the execution, the subprocess invokes the operations provided by the Web service *BankOne* to fulfill this functionality. Obviously, the transfer business should behave as a traditional atomic transaction. In other words, the debit and credit operations must be executed successfully all or unsuccessfully all, and the outcome must be consistent to ensure the safety of the two accounts. Before we integrate the atomic transaction to the *payment* subprocess, the *BankOne* Web service is extended

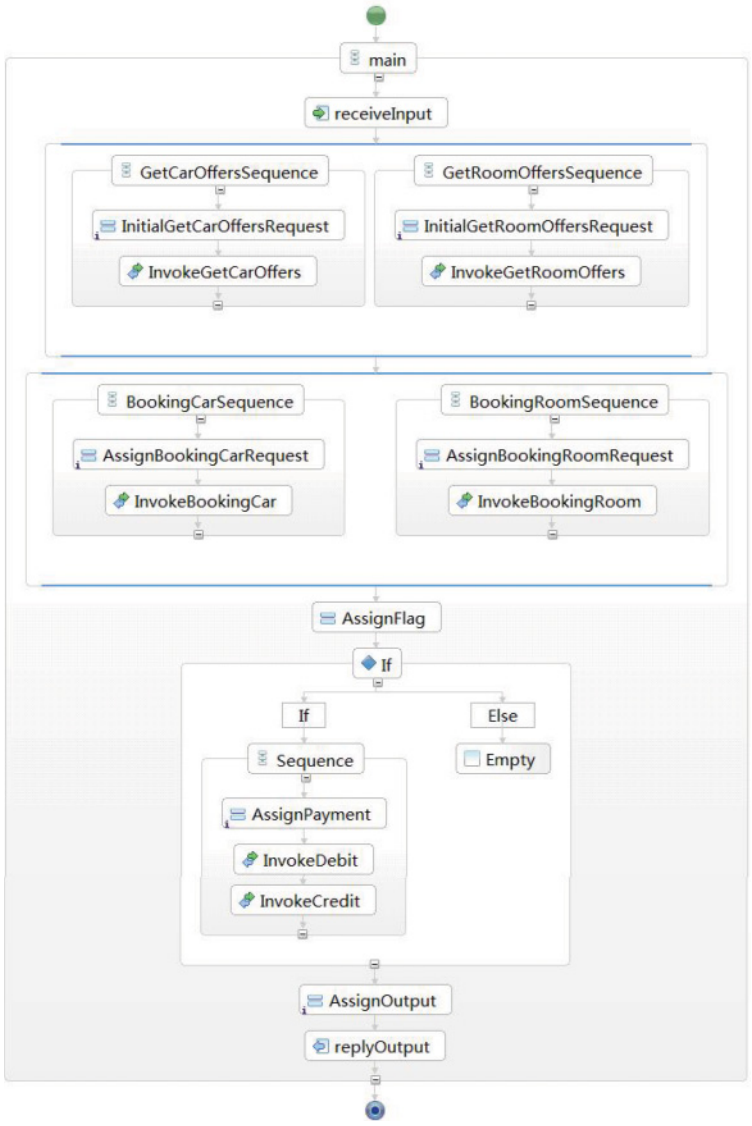
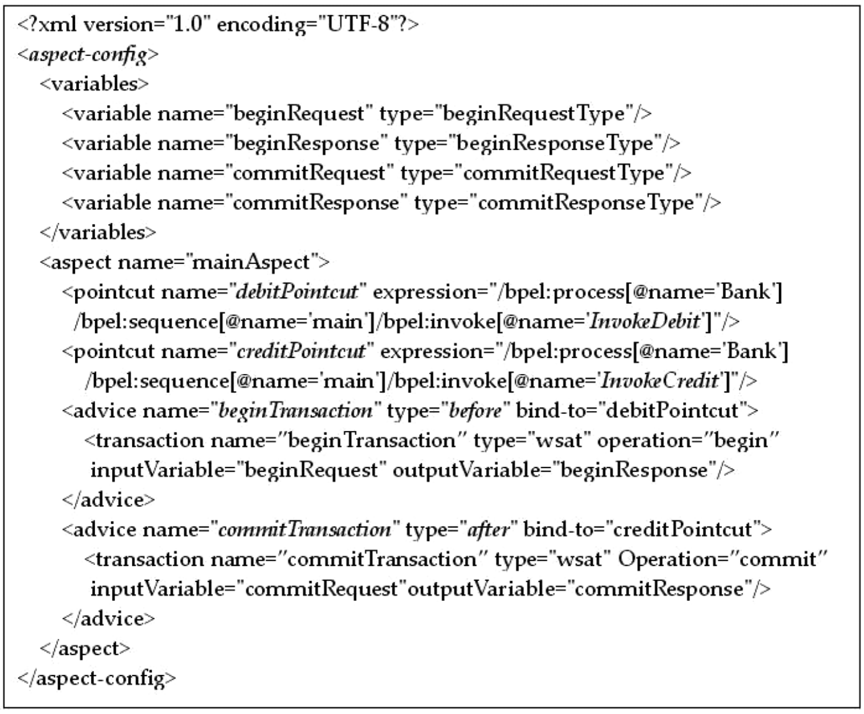


Fig. 7. An illustration of the simplified *Holiday* process.

to support transaction management. To do that, we extend the *BankOne* to implement the *NamedXAResource* interface [Sun et al. 2011b].

To add the transaction features into the *payment* subprocess, we first need to declare aspect-based transaction policies with respect to the preceding process. In this context, the transaction is going to be opened *before* transfer and closed *after* transfer. Figure 8 illustrates such aspect-based transaction policies. In the *payment* subprocess, the *begin* operation is added before the *InvokeDebit* activity, and the *commit* operation is added after the *InvokeCredit* activity. The resulting *payment* subprocess is depicted in the bottom part of Figure 9. One may observe that the transaction operations are added



```
<?xml version="1.0" encoding="UTF-8"?>
<aspect-config>
  <variables>
    <variable name="beginRequest" type="beginRequestType"/>
    <variable name="beginResponse" type="beginResponseType"/>
    <variable name="commitRequest" type="commitRequestType"/>
    <variable name="commitResponse" type="commitResponseType"/>
  </variables>
  <aspect name="mainAspect">
    <pointcut name="debitPointcut" expression="/bpel:process[@name='Bank']
     /bpel:sequence[@name='main']/bpel:invoke[@name='InvokeDebit']"/>
    <pointcut name="creditPointcut" expression="/bpel:process[@name='Bank']
      /bpel:sequence[@name='main']/bpel:invoke[@name='InvokeCredit']"/>
    <advice name="beginTransaction" type="before" bind-to="debitPointcut">
      <transaction name="beginTransaction" type="wsat" operation="begin"
       inputVariable="beginRequest" outputVariable="beginResponse"/>
    </advice>
    <advice name="commitTransaction" type="after" bind-to="creditPointcut">
      <transaction name="commitTransaction" type="wsat" Operation="commit"
       inputVariable="commitRequest"outputVariable="commitResponse"/>
    </advice>
  </aspect>
</aspect-config>
```

Fig. 8. Aspect-based transaction policies for the *payment* subprocess.

as expected. In other words, the *beginTransaction* and *commitTransaction* confine the boundary of the atomic transaction.

In the *Holiday* process as illustrated in Figure 7, *car booking* and *hotel booking* are two parallel flow activities. In this case, we want to guarantee the reliability of the booking flow activities. Next, we consider adding the business activity transaction features to the *Holiday* process—that is, the transaction should be opened before the booking flow activity and closed after the booking flow activity. The *wsba* transaction type has two variants: *wsba-atomic* and *wsba-mixed*. The two variants are similar, with exception of some transaction operations with different parameters. For brevity, we select the *wsba-atomic* transaction for *car booking* and *hotel booking* BPEL subprocesses.

Before we integrate the *wsba-atomic* transaction to the *Holiday* process, the *rentACar* service and the *rentARoom* service are extended to support transaction interfaces as discussed in Sun et al. [2011b]. First, we add an additional *Coordination Context* object besides the existing parameters. Then, to illustrate the difference between *BAwPC* and *BAwCC*, we let the *rentACar* service extend *BAwPCParticipant* and let the *rentARoom* service extend *BAwCCParticipant*. Next, the transaction operations are implemented according to the business logic.

In the *Holiday* process, the *createNewContextWithWSBAI* operation is added before the *booking* flow activity, the *closeAllParticipants* operation is added after the *booking* flow activity, and the *completeParticipants* operation is added after finishing the *rentARoom* business, according to the *BAwCC* feature. Further, we have to get a specific coordination context with the participant's matchcode by invoking the *get CoordinationContextWithMatchcode* operation, then pass this coordination context object as a parameter to the business operation by invoking the *assignTransaction* operation. Similarly, such aspect-based transaction policies, as illustrated in Figure 10, can

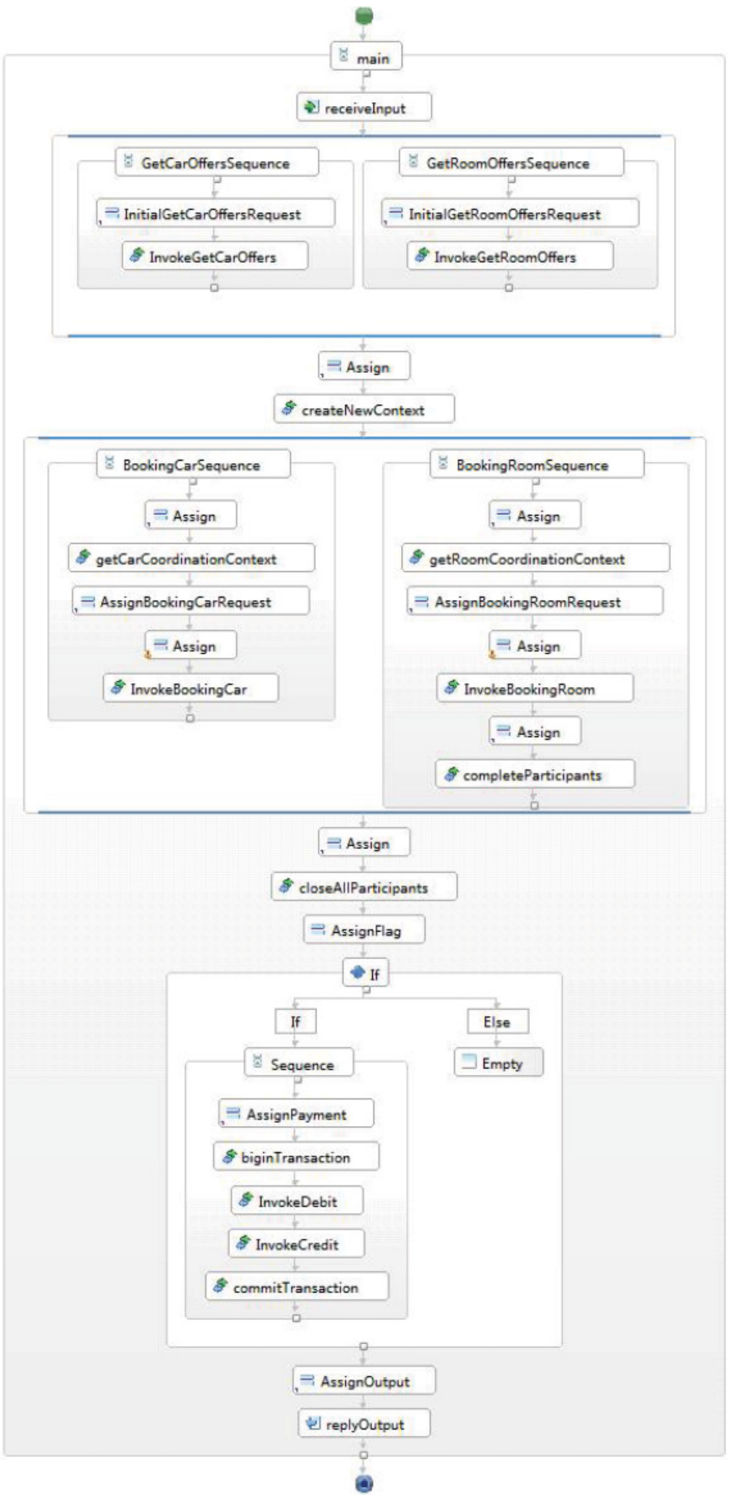
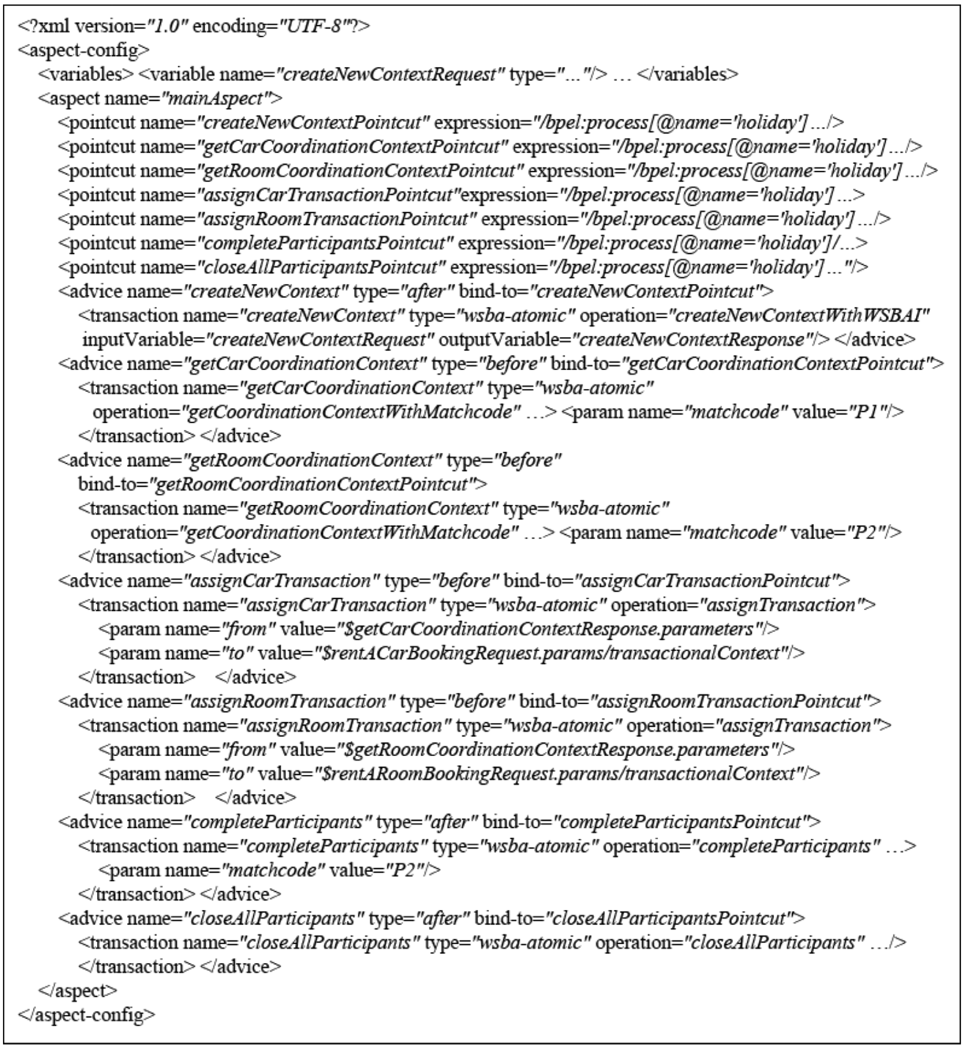


Fig. 9. An illustration of the transactional *Holiday* process.



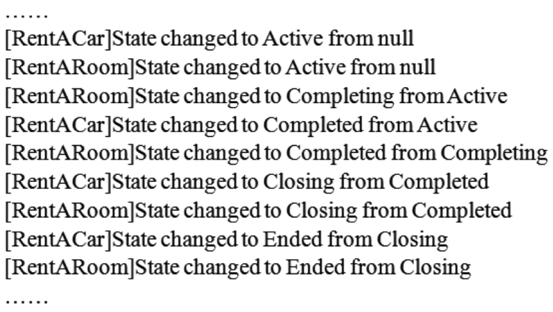
```
<?xml version="1.0" encoding="UTF-8"?>
<aspect-config>
  <variables> <variable name="createNewContextRequest" type="..."/> … </variables>
  <aspect name="mainAspect">
    <pointcut name="createNewContextPointcut" expression="/bpel:process[@name='holiday'] .../>
    <pointcut name="getCarCoordinationContextPointcut" expression="/bpel:process[@name='holiday'] .../>
    <pointcut name="getRoomCoordinationContextPointcut" expression="/bpel:process[@name='holiday'] .../>
    <pointcut name="assignCarTransactionPointcut"expression="/bpel:process[@name='holiday'] ...>
    <pointcut name="assignRoomTransactionPointcut" expression="/bpel:process[@name='holiday'] .../>
    <pointcut name="completeParticipantsPointcut" expression="/bpel:process[@name='holiday']/...>
    <pointcut name="closeAllParticipantsPointcut" expression="/bpel:process[@name='holiday'] ... "/>
    <advice name="createNewContext" type="after" bind-to="createNewContextPointcut">
      <transaction name="createNewContext" type="wsba-atomic" operation="createNewContextWithWSBAI"
       inputVariable="createNewContextRequest" outputVariable="createNewContextResponse"/> </advice>
    <advice name="getCarCoordinationContext" type="before" bind-to="getCarCoordinationContextPointcut">
      <transaction name="getCarCoordinationContext" type="wsba-atomic"
        operation="getCoordinationContextWithMatchcode" …> <param name="matchcode" value="P1"/>
      </transaction> </advice>
    <advice name="getRoomCoordinationContext" type="before"
      bind-to="getRoomCoordinationContextPointcut">
      <transaction name="getRoomCoordinationContext" type="wsba-atomic"
        operation="getCoordinationContextWithMatchcode" …> <param name="matchcode" value="P2"/>
      </transaction> </advice>
    <advice name="assignCarTransaction" type="before" bind-to="assignCarTransactionPointcut">
      <transaction name="assignCarTransaction" type="wsba-atomic" operation="assignTransaction">
        <param name="from" value="$getCarCoordinationContextResponse.parameters"/>
        <param name="to" value="$rentACarBookingRequest.params/transactionalContext"/>
      </transaction>   </advice>
    <advice name="assignRoomTransaction" type="before" bind-to="assignRoomTransactionPointcut">
      <transaction name="assignRoomTransaction" type="wsba-atomic" operation="assignTransaction">
        <param name="from" value="$getRoomCoordinationContextResponse.parameters"/>
        <param name="to" value="$rentARoomBookingRequest.params/transactionalContext"/>
      </transaction>   </advice>
    <advice name="completeParticipants" type="after" bind-to="completeParticipantsPointcut">
      <transaction name="completeParticipants" type="wsba-atomic" operation="completeParticipants" …>
        <param name="matchcode" value="P2"/>
      </transaction> </advice>
    <advice name="closeAllParticipants" type="after" bind-to="closeAllParticipantsPointcut">
      <transaction name="closeAllParticipants" type="wsba-atomic" operation="closeAllParticipants" …/>
      </transaction> </advice>
  </aspect>
</aspect-config>
```

Fig. 10. Aspect-based transaction policies for *car booking* and *hotel booking* subprocesses.

be declared as we have done for the *payment* subprocess. The resulting transactional *car booking* and *hotel booking* subprocesses are shown in the top part of Figure 9. One may observe that the transaction operations are added. The *createNewContext* and *closeAllParticipants* confine the transaction boundary of business activity.

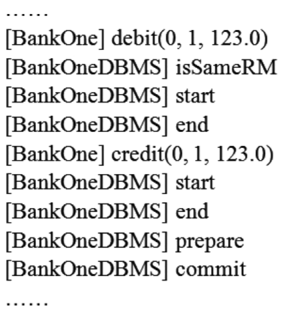
5.3. Running the Transactional *Holiday* Process

Next, the transactional *Holiday* process is deployed on Apache ODE 1.3.5 [Apache 2013b], which is an open source BPEL execution engine. We run three instances deployed on Apache Tomcat 5.5.26 [Apache 2013c], which is an open source Web server developed by the Apache Software Foundation, to demonstrate this case. The first instance is responsible for running the transaction coordinator, the Web service *Transaction ManagerAtomic* in the transaction management middleware is used to support the *wsba* transaction, and the Web service *TransactionManager* in the transaction



```
......
[RentACar]State changed to Active from null
[RentARoom]State changed to Active from null
[RentARoom]State changed to Completing from Active
[RentACar]State changed to Completed from Active
[RentARoom]State changed to Completed from Completing
[RentACar]State changed to Closing from Completed
[RentARoom]State changed to Closing from Completed
[RentACar]State changed to Ended from Closing
[RentARoom]State changed to Ended from Closing
......
```

Fig. 11. Snapshot of the execution log of *Participant* of *car booking* and *hotel booking* subprocesses in the *wsba-atomic* transaction.



```
......
[BankOne] debit(0, 1, 123.0)
[BankOneDBMS] isSameRM
[BankOneDBMS] start
[BankOneDBMS] end
[BankOne] credit(0, 1, 123.0)
[BankOneDBMS] start
[BankOneDBMS] end
[BankOneDBMS] prepare
[BankOneDBMS] commit
......
```

Fig. 12. The execution log of the *Committed Payment* subprocess in the *wsat* transaction.

management middleware is used to support the *wsat* transaction. The second instance is responsible for running the transaction participant. The third instance is responsible for running Apache ODE, where the transactional *Holiday* process is deployed. Now, we run the deployed transactional *Holiday* process. To trace communication messages and states changing among coordinator and participants, we show the printout execution logs on the console of Apache Tomcat. For simplicity, we just show a subset of the printout message of Apache Tomcat, indicating which transaction operations are invoked.

Figure 11 shows a log fragment of the *Participant* instance. Here, the *rentARoom* service is extended from *BAwCCParticipant* and relies on the coordinator to tell it when to complete. Therefore, the *rentARoom* service moves into the *completing* status after finishing its work, then moves into *completed* from *completing*. The transaction behaviors are clearly followed during the *car booking* and *hotel booking* subprocesses.

For the *payment* subprocess, we execute the deployed *Holiday* process with the input message (0, 1, 123), which means that the subprocess is expected to transfer $123 from *account0* to *account1*. The execution logs are printed on the console of Apache Tomcat, as illustrated in Figure 12. From the log, one can observe that the two-phase commit of the atomic transaction is executed, and such an execution can guarantee a consistent outcome. In this case, the initial value of the two accounts was $1,000; after execution, the amount in *account0* is $877 and the amount in *account1* is $1,123.

It is possible for the transfer operation in a BPEL process to fail, especially when a large number of transactions are present. This may further result in a transaction failure. Figure 13 illustrates the log of a simulated transaction with failure. In this case, the rollback is used to ensure the consistency of the two accounts, and the value of the two accounts remains the initial value (namely $1,000).

```
......
[BankOne] debit(0, 1, 123.0)
[BankOneDBMS] isSameRM
[BankOneDBMS] start
[BankOneDBMS] end
[BankOne] credit(0, 1, 123.0)
[BankOneDBMS] start
[BankOneDBMS] end
[BankOneDBMS] rollback
......
```

Fig. 13. The execution log of the *Rollback Payment* subprocess in the *wsat* transaction.

We have conducted a case study to illustrate how the proposed approach can introduce transaction management into service compositions without significant effort. Although not so complex, the process is used for the purpose of illustration and captures the essence of our proposal involving parallel flow of activities and sequence of activities. Additionally, it demonstrates the different types of transaction requirements for subprocesses. The proposed approach is applicable to larger-scale processes with more complex business logics. To do that, one needs to mainly declare aspect-based transaction policies for some activities of a BPEL process with respect to specific transactional requirements, as we have illustrated in the preceding case.

5.4. Discussion

We illustrated our approach and supporting platform using WS-Transaction [WS-C 2007; WS-AT 2007; WS-BA 2007]. However, it is easy to extend the approach to support other transaction protocols, such as BTP [OASIS 2004]. To do that, we need to extend the setting of elements and attributes in aspect-oriented transaction policies and then weave them into BPEL processes.

In many situations, transactions may be composed in a nested way. This means that if an error occurs in the inner transaction, the inner transaction should roll back. To the best of our knowledge, most available middleware for WS-Transaction ignores this issue. Most notably, Apache Kandula does not clearly handle nested transactions.

6. PERFORMANCE AND COST EVALUATION

Transaction integration enhances the reliability of BPEL processes, although one may wonder about the necessary overheads to manage this. In other words, how will transaction management impact the performance of BPEL process execution? What will be the extra workload for such management, particularly for the developers? In this section, we answer these questions by looking at time overhead, line of code (LOC) [Park 1992], and development time as the metrics for evaluating the performance and workload of transactional BPEL processes.

Furthermore, one may consider the difference in costs of the proposed approach with the existing transaction integration approaches. In this study, the approach of manually and explicitly mixing transaction logic with business logic (briefly *MML*) was selected for comparison because of its wide use and simplicity. This approach simulates transactions in a BPEL by using BPEL variables to record the transaction's status and using links and assign activities to simulate the state transitions within a transaction [Sun 2009]. We report and compare the costs of the proposed approach with those of the *MML* approach by extending the previous case study reported in Section 5.

Table I. Summary of Average Execution Time of the *Holiday* and Transactional *Holiday* Processes

	Original BPEL Process	Transactional BPEL Process Using the Proposed Approach			Transactional BPEL Process Using the *MML* Approach		
Repeated Times	Execution Time (ms)	Execution Time (ms)	Absolute Increment (ms)	Relative Increment (%)	Execution Time (ms)	Absolute Increment (ms)	Relative Increment (%)
5	2,248.0	2,562.4	314.4	14.0	2,447.4	199.4	8.9
10	2,216.2	2,531.9	315.7	14.2	2,426.9	210.7	9.5
50	2,167.7	2,572.1	404.4	18.6	2,406.4	238.7	11.0
100	2,165.0	2,552.0	387.0	17.9	2,367.8	202.8	9.4

6.1. Performance

To evaluate the impact of transaction integration on the performance of BPEL processes, we compare execution time and use the increment of execution time as a metric. Considering that a BPEL process provides the function of invoking Web services, we calculate the period of each invocation (by recording the time before invocation and after the invocation) and then get the total execution time of a BPEL process by adding all invocation times (thus, the measured performance includes the execution time of transaction management middleware and invoked transactional Web services). To make the evaluation results statistically relevant and less biased by contextual conditions, we repeat the evaluation 5, 10, 50, and 100 times, and for each we calculate the average execution time because the performance of each execution varies randomly due to different operational environments (e.g., the CPU's and memory's utility and availability). The experiment was conducted on a standard laptop with the MS Windows 7 operating system on a 64-bit quad processor of 2.4GHz and 8GB memory.

Table I summarizes the average execution time of the original BPEL process and the transactional BPEL process using the two approaches. Here, the "Original BPEL Process" column shows the execution time in milliseconds of the original BPEL process, the Transactional BPEL Process Using the Proposed Approach column shows the execution time in milliseconds of the transactional BPEL process using our approach, and the Transactional BPEL Process Using the *MML* Approach column shows the execution time in milliseconds of the transactional BPEL process using the *MML* approach. The execution time overhead is evaluated in terms of absolute increment (ms) and relative increment (%). We note the following:

—Integrating transactions into a BPEL process evidently introduces execution time overhead. This is because transactional BPEL processes introduced additional transaction interactions with transaction management middleware and transaction Web services.

—The proposed approach is slightly less efficient than the *MML* approach. For the *Holiday* process, the execution time overhead increases from 314.4ms to 404.4ms (i.e., relatively increases by 14.0% to 18.6%) when the proposed approach was used, and increases from 199.4ms to 238.7ms (i.e., relatively increases by 8.9% to 11.0%) when the *MML* approach was used. After carefully investigating the differences, we find that more message exchanges between the BPEL engine and Web service servers are incurred in the proposed approach, whereas all transaction states and transitions are simulated by BPEL activities or variables in the *MML* approach.

6.2. Workload

To evaluate the additional workload of a BPEL process that is due to the transaction integration, two aspects are considered: the extra code size and development time. In this study, the LOC and development time were selected as the metrics. The latter very

Table II. Summary of the LOC of Web Service Implementation and XML Tags of the Holiday BPEL Process

	Original BPEL Process	Transactional BPEL Process Using the Proposed Approach		Transactional BPEL Process Using the *MML* Approach	
	Code Size	Code Size	Absolute Increment	Code Size	Absolute Increment
LOC of WS	772	1,068	296	844	72
XML Tags of BPEL	171	264	93	428	257

much depends on the experience of the designer and thus is rather subjective. Next, we report and compare the LOC and development time of the original and transactional BPEL processes using the proposed approach and the *MML* approach.

A BPEL process is defined as an XML file consisting of a set of tags, and an XML tag can span multiple lines; therefore, we use the number of XML tags to represent the size of the BPEL process. Besides, in a BPEL project, only BPEL process specification (*.bpel), the WSDL specification of BPEL process (*Artifacts.wsdl), and deploy description file (deploy.xml) are extended. Therefore, in our experiments, we count the LOC of the core implementation of all Web services and the XML tags of the three XML files related to BPEL processes. We do not take into account the size of the transaction management middleware because it can be applicable to various transactional BPEL processes and Web services.

Table II summarizes the code size of the BPEL process and Web services. Here, the Original BPEL Process column shows the number of LOCs for the original Web services and XML tags for the original BPEL process, the Transactional BPEL Process Using the Proposed Approach column shows the number of LOCs for transactional Web services and XML tags for the transactional BPEL process using the proposed approach, the Transactional BPEL Process Using the *MML* Approach column shows the number of LOCs for the Web services and XML tags for the BPEL process using the *MML* approach, and the Absolute Increment column shows the size increment of the transactional implementation against the original implementation. We note the following:

—Integrating transactions into a BPEL process evidently increases the workload costs in terms of extra code size when both approaches are used. The workload does not greatly increase in terms of absolute size.
—The increments of XML tags in the proposed approach are significantly less than the ones in the *MML* approach. In the case of *Holiday*, the XML tags increase by 93 in the proposed approach, and the XML tags of BPEL increase by 257 in the *MML* approach.
—The LOC of Web services in the proposed approach increases more than the one in the *MML* approach. In the case of *Holiday*, the LOC of the core implementation of Web services increases by 296, whereas core implementation of Web services increases by 72 LOCs in the *MML* approach. This difference can be further explained as follows. Generally speaking, the transaction logic is distributed at the Web service layer and the service composition layer. In the proposed approach, most of the transaction logic is implemented in the transaction management Web services, whereas in the *MML* approach, most of the transaction logic is implemented in the service composition layer. Although the core implementation of Web services in the *MML* approach increased less, it suffers from a heavy increase provided that the declared transaction logic policy is changed. This is because extending Web services in the *MML* approach only considers specific transaction interfaces for the current requirements. On the other hand, Web services that are extended by using the approach proposed here are able to cater to various transaction protocols without any further changes.

Table III. Summary of Development Time for Transactional Support Using Two Approaches

	Transactional BPEL Process Using the Proposed Approach			Transactional BPEL Process Using the *MML* Approach		
	Extending WS (min)	Integrating Transaction (min)	Total (min)	Extending WS (min)	Integrating Transaction (min)	Total (min)
Holiday	120	25	145	125	143	268

—In our approach, the LOC of the core implementation of Web services increases more than that of XML tags of BPEL processes. This is because in the original implementation of Web services, no transaction operations are implemented; on the other hand, only a few elements of invoking transaction operations are inserted into BPEL processes.

Table III summarizes the development time to support transactional behavior based on the original BPEL process using two approaches. Obviously, such an evaluation depends more on the practitioner's experience. To make the comparison fair, two master-level participants involved in the experiments were provided with the same training for the experiments. They were asked to practice the two approaches for one subject program. The Transactional BPEL Process Using the Proposed Approach column shows the average development time when the proposed approach is used, and the Transactional BPEL Process Using the *MML* Approach column shows the average development time when the *MML* approach is used. The Extending WS (min) column shows the average development time of extending an existing Web services that will participate in transactions. The Integrating Transaction (min) column shows the average development time of integrating transactions into an existing BPEL process. The Total (min) column shows the average entire development time, which is a sum of average development time for extending Web services and integrating transactions.

We remark that the entire development time using the approach that we propose is lower than the *MML* approach. The time of extending Web services is about 120 minutes and the time of integrating transactions is 25 minutes using our proposed approach, whereas the time of extending Web services is about 125 minutes and the time integrating transactions is about 143 minutes using the *MML* approach. The development time using the proposed approach is much less than that using the *MML* approach. This is because *Salan* significantly reduces efforts when integrating transactions.

7. RELATED WORK

The study of transactions was initially proposed in database studies, but as workflows become a popular way to build information systems, the issue of how to handle exceptions and transactions has become key [Alonso et al. 1996]. Zimmermann et al. [2007] described and modeled typical transaction patterns in BPEL processes. Eshuis et al. [2011] described a taxonomy of transactional workflow models and proposed a transactional process view. Eder and Liebhart [1996] examined the exception handling issue of transactional workflows and analyzed various failure sources and classes, which are used to define specific rollbacks. Schäfer et al. [2008] proposed a forward recovery–based approach to deal with compensations. The basic idea is to prevent a rollback when an error occurs at some point in the transaction. Several efforts have been reported to manage exceptions for Web services or Web service compositions in declarative ways [Erradi et al. 2006; Grefen 1993; Zeng et al. 2005]. These approaches are similar to ours because they are declarative methods.

For a complex workflow system that integrates several independent and distributed components, an important issue is how to tackle the problem of isolation and atomicity of transaction processes. To address this, Schuldt et al. [2002] presented a unified model for concurrency control and recovery in transactional processes. BPEL processes are often composed of long executions of coordinated Web services, and these distributed Web services are out of control; therefore, transactional BPEL processes also face atomicity and isolation issues. Fortunately, these issues have been addressed by various *WS-Transaction* protocols. The motivation of this work is to propose a framework to integrate transactions into service compositions, and in the proposed framework, such issues are left for *WS-Transaction* protocols, which are implemented as transactional management middleware.

Several approaches to transaction integration for BPEL processes exist. Tai et al. [2004] presented a *WS-Policy*–based approach. In their solution, a middleware implements the coordination services described by *WS-Coordination*, and the coordination specifying the transaction types and protocols is described by *WS-Policy*. In the BPEL specification, the coordination policy is attached to BPEL partner links and scopes. The method is very similar to our approach, as both are declarative. However, our approach employs AOP to declare transaction policies, converts aspect-based transaction policies to standard BPEL elements, and finally merges them into the original BPEL process to achieve the transaction property. Further, our approach is highly automated, a supporting platform is available, and the approach is validated with a case study.

Fletcher et al. [2003] proposed an approach to implement transaction management by extending BPEL. The transaction coordination context is introduced as variables of BPEL processes, and a set of transaction activities including "new," "confirm," and "cancel" are added to the BPEL specification. The compensation handler is extended to derive the *confirmHandler* and *cancelHandler*. Unfortunately, this method needs to extend or modify BPEL execution engines, as new types of BPEL elements are introduced. Up to now, no actual extensions have been reported. Unlike this method, our approach integrates transactions into BPEL processes using a preprocessing step, which does not extend BPEL and hence does not need special-purpose BPEL engines.

Charfi et al. [2006, 2007] proposed a method to support transaction behaviors in BPEL processes by using the AO4BPEL process container framework. In this method, the transaction requirements of a BPEL process are specified in a deployment descriptor. An aspect-based container is generated to integrate the process execution with the transaction middleware that is also implemented with Apache Kandula. However, this method relies on a nonstandard BPEL and thus requires extended engines. Further, the implementation only supports the *wsat* specification. This method is similar to our approach in that both make use of the AOP concept to separate the transaction logic from the business logic, and transaction policies are declared in an abstract way. Unlike their method, our approach first converts aspect-based transaction policies into standard BPEL elements and then automatically weaves these BPEL elements into final BPEL processes. Therefore, our approach does not make any modifications to standard BPEL. Additionally, our approach has a comprehensive tooling support and is able to fully support the transaction types and protocols in the standard stack for Web services, including *wsat*, *wsba-atomic*, and *wsba-mixed*, as illustrated in the case study.

Braem et al. [2006] proposed a similar approach to support the separation of concerns in BPEL programs based on AOP. An aspect-oriented extension to WS-BPEL—named *Padus*—was used to overcome the lack of support for modularization of cross-cutting concerns. The cross-cutting concerns defined in Padus are weaved into the BPEL process through static weaving. The approach is similar to ours in that both of them are based on AOP, and the cross-cutting concerns are represented in a declarative way. However, this approach is much more generic than ours and does not deal with specific issues related to transaction management. Therefore, extra efforts are needed to support various transaction types and protocols.

Limthanmaphon and Zhang [2004] proposed a transaction management model for Web service compositions that is based on the concepts of tentative hold and compensation. The objective of the tentative hold is to facilitate the automated coordination of multibusiness transactions. Unlike atomic transaction using the locking strategy, a participant within a transaction scope enters a “tentative hold” state after it finishes its business work, and in this state, services are still available for invocation. If the transaction fails, the compensate action is performed to undo the effect. Finally, all services that are either committed or compensated will move into the ended state. The proposed transaction model is similar to the mixing of *WS-AT* and *WS-BA* protocols. Unfortunately, their work only proposes a transaction management model, and no implementation or supporting tools are reported.

Curbera et al. [2003] pointed out that Web services are moving from the simple “describe, publish, and interact” phase to a new phase of ensuring the robustness of business processes. Similarly, Portilla [2006] proposed the idea of providing transactional behaviors for service coordination. These visions match very well with our development of transaction-aware Web services [Sun et al. 2011b]. To support transactions at the service composition level, transaction extensions to Web services under compositions are necessary. However, this extension has been neglected in most of the transaction integration approaches mentioned earlier. In fact, we took the first step and proposed a transaction-aware Web service development framework [Sun et al. 2011b]. Such a framework is only illustrated by the *wsat* transaction. Following up on our previous work in Sun et al. [2011b], we take a further step and extend the framework to cover *wsat*, *wsba-atomic*, and *wsba-mixed*.

In our previous work [Sun 2009; Sun and Aiello 2007; Sun et al. 2011a], we looked into requirements for transaction management for service compositions and explored some transaction integration solutions. We provided a detailed survey on transaction models, protocols, and implementation tools [Sun and Aiello 2007]. In Sun [2009], we proposed a transaction integration approach by implementing transaction logic directly into business logic specified in BPEL processes. Such an approach has advantages, such as the fact that no changes are necessary for the BPEL engine. However, it suffers from low efficiency and maintainability issues. This is because the transaction logic has to be repeatedly implemented for individual BPEL processes, and transaction logic and business logic are intertangled. To address the low efficiency issue, we proposed to encapsulate the transaction logic using a transaction management middleware; in this way, all BPEL processes can be supported by efficient transaction management [Sun et al. 2011a]. However, the implementation of this approach relies on the complex SeCSE platform [SeCSE Team 2007], which is an integrated project whose primary goal is to create methods, tools, and techniques to support the cost-effective development of service-centric applications. In this work, we proposed a light-weight way to efficient transaction integration for BPEL processes and employ the AOP technique to further address the low maintainability issue of our previous approaches.

A comparison of our approach with major related work is summarized in Table IV. From the comparison, our approach shows a number of unique features, as it encapsulates transaction logic via middleware, does not modify existing standards and engines, and has tooling support. Its feasibility and performance are also validated by a case study.

8. CONCLUSIONS

The reliability of loosely coupled service compositions is a key issue when they are used as infrastructure for critical business processes. The integration of transactions is an essential step in this direction.

Table IV. Summary of Average Execution Time of Bank Process and Transactional Bank Processes

	Methodology	Transaction Logic	Modification of Existing Standards and Engine	Tooling Support	Empirical Study
Tai et al. [2004]	Using WS-Policy	Encapsulate by middleware	No	No	No
Fletcher et al. [2003]	Extending BPEL	Mixed in BPEL	Yes	No	No
Charfi et al. [2007, 2006]	Using AO4BPEL	Defined in separated file	Yes	Yes	Yes
Braem et al. [2006]	Using Padus	Defined in separated file	No	Yes	No
Sun [2009]	Directly Implementing in BPEL	Mixed in BPEL	No	No	No
Sun et al. [2011a]	Using SeCSE	Encapsulate by middleware	Yes	Yes	No
Our Work	Using AOP	Encapsulate by middleware	No	Yes	Yes

Although transaction models and protocols have been proposed for Web services, they are separate from existing service composition languages. We proposed a systematic aspect-based transaction integration approach for service compositions. Our approach relies on the widely recognized service composition language, BPEL, and the Web service transactions protocol, WS-Transaction. In our approach, transactional requirements for BPEL processes are first declared as aspect-based policies. After validation, policies are converted into a set of transaction-related operations, which are finally weaved into the original BPEL process; this results in a transactional BPEL process. The actual runtime transaction management is left to a middleware that implements the generic coordination framework and transaction protocols. In addition, we considered the extension of Web services participating in the transactional compositions.

We have developed a comprehensive supporting platform to make the approach practical and deployable. The platform provides a helpful aid to define aspect-based transaction policies and executes the tasks of validating aspect-based transaction policies, weaving, and deploying transactional BPEL processes. The middleware itself is a set of Web services extending Apache Kandula.

To further demonstrate our approach and validate its feasibility and performance, we have conducted a case study. This case study demonstrates two types of transaction requirements for the *Holiday* BPEL process: one is with the *wsat* transaction and the other with the *wsba* transaction protocol. Results show that our approach is able to include transactionality while increasing the reliability of business process execution with little human involvement. We also evaluated the costs of our approach in terms of time and workload overhead due to the transaction integration. From the evaluations, we observe that transaction integration introduces extra costs, although it enhances the reliability of BPEL service compositions. Such costs are greatly reduced with the use of the supporting platform, and they appear absolutely acceptable in any case where reliability is a high priority. The work reported in this article provides an efficient and practical way to address the reliability issue of loosely coupled service-oriented systems.

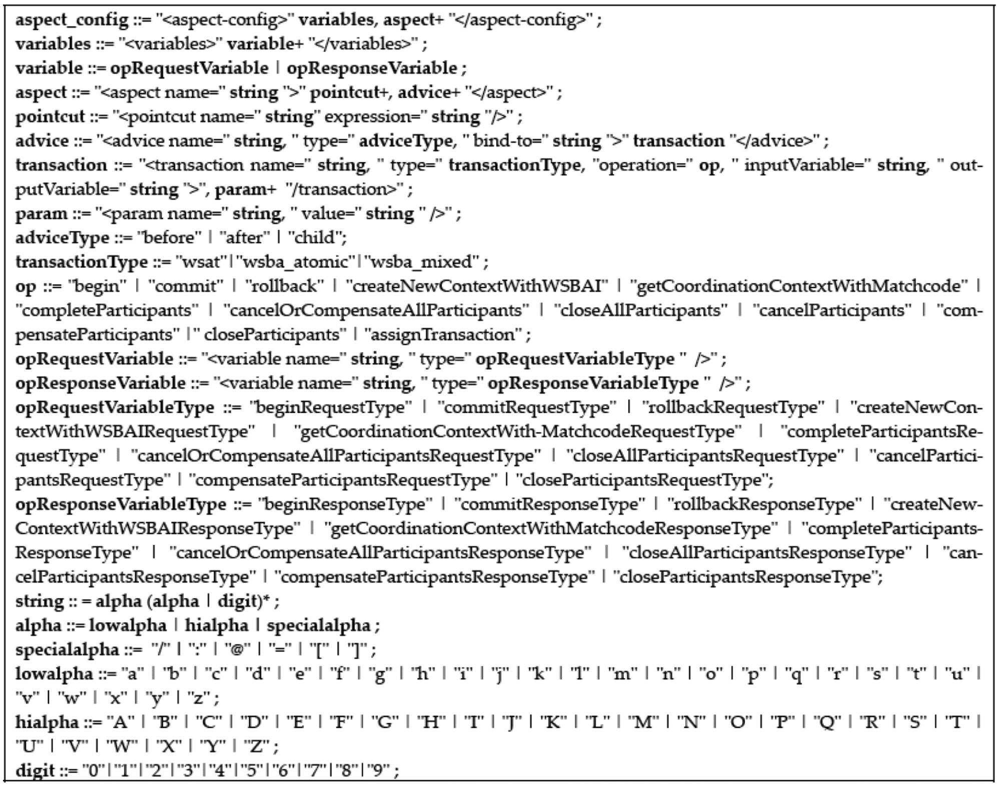
Our evaluation shows the technical feasibility of the proposed approach. Naturally, there is also a practical feasibility issue. In other words, will service providers agree to use the required protocol? This appears to be easy to realize in the case of services running within a possibly very large organization, as to some extent there is full ownership of the components involved. However, it will prove challenging to realize in large decentralized open networks such as the Internet.

In the future, we plan to investigate the extension of existing open source middleware to support nested transactions in the context of service compositions while continually improving the supporting platform and extending the set of case studies for validation purposes. We are also interested in improving the proposed approach by partially automating the transactional extension procedure for Web services using a declarative approach, as our approach currently requires one to adapt the participant Web services to support the required transactional behavior, which requires a kind of arrangement between the participants.

APPENDIX

A. SYNTAX OF ASPECT-BASED TRANSACTION POLICY DECLARATION LANGUAGE

Figure 14 illustrates an Extended Backus-Naur Form (EBNF) definition of the aspect-based transaction policy declaration language.



```
aspect_config ::= "<aspect-config>" variables, aspect+ "</aspect-config>" ;
variables ::= "<variables>" variable+ "</variables>" ;
variable ::= opRequestVariable | opResponseVariable ;
aspect ::= "<aspect name=" string ">" pointcut+, advice+ "</aspect>" ;
pointcut ::= "<pointcut name=" string" expression=" string "/>" ;
advice ::= "<advice name=" string, " type=" adviceType, " bind-to=" string ">" transaction "</advice>" ;
transaction ::= "<transaction name=" string, " type=" transactionType, "operation=" op, " inputVariable=" string, " out-
putVariable=" string ">", param+  "/transaction>" ;
param ::= "<param name=" string, " value=" string " />" ;
adviceType ::= "before" | "after" | "child";
transactionType ::= "wsat"|"wsba_atomic"|"wsba_mixed" ;
op ::= "begin" | "commit" | "rollback" | "createNewContextWithWSBAI" | "getCoordinationContextWithMatchcode" |
"completeParticipants" | "cancelOrCompensateAllParticipants" | "closeAllParticipants" | "cancelParticipants" | "com-
pensateParticipants" |" closeParticipants" | "assignTransaction" ;
opRequestVariable ::= "<variable name=" string, " type=" opRequestVariableType "  />" ;
opResponseVariable ::= "<variable name=" string, " type=" opResponseVariableType "  />" ;
opRequestVariableType ::= "beginRequestType" | "commitRequestType" | "rollbackRequestType" | "createNewCon-
textWithWSBAIRequestType" | "getCoordinationContextWith-MatchcodeRequestType" | "completeParticipantsRe-
questType" | "cancelOrCompensateAllParticipantsRequestType" | "closeAllParticipantsRequestType" | "cancelPartici-
pantsRequestType" | "compensateParticipantsRequestType" | "closeParticipantsRequestType";
opResponseVariableType ::= "beginResponseType" | "commitResponseType" | "rollbackResponseType" | "createNew-
ContextWithWSBAIResponseType" | "getCoordinationContextWithMatchcodeResponseType" | "completeParticipants-
ResponseType" | "cancelOrCompensateAllParticipantsResponseType" | "closeAllParticipantsResponseType" | "can-
celParticipantsResponseType" | "compensateParticipantsResponseType" | "closeParticipantsResponseType";
string :: = alpha (alpha | digit)* ;
alpha ::= lowalpha | hialpha | specialalpha ;
specialalpha ::=  "/" | ":" | "@" | "=" | "[" | "]" ;
lowalpha ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" |
"v" | "w" | "x" | "y" | "z" ;
hialpha ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" |
"U" | "V" | "W" | "X" | "Y" | "Z" ;
digit ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9" ;
```

Fig. 14. The EBNF definition of the aspect-based transaction policy declaration language.

ACKNOWLEDGMENTS

The authors would like to thank master students from the University of Science and Technology Beijing and the University of Groningen for participating in the case study. The authors are grateful to the anonymous reviewers who provided useful suggestions on earlier versions of this article.

REFERENCES

G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, and C. Mohan. 1996. Advanced transaction models in workflow contexts. In *Proceedings of the 12th International Conference on Data Engineering*. IEEE, Los Alamitos, CA, 574–581.

Apache. 2013a. Apache Kandul. Retrieved April 24, 2015, from http://wso2.com/projects/kandula/java.

Apache. 2013b. Apache ODE. Retrieved April 24, 2015, from http://ode.apache.org.

Apache. 2013c. Apache Tomcat. Retrieved April 24, 2015, from http://tomcat.apache.org.

M. Braem, K. Verlaenen, N. Joncheere, W. Vanderperren, R. Van Der Straeten, E. Truyen, W. Joosen, and V. Jonckers. 2006. Isolating process-level concerns using Padus. In *Proceedings of the 4th International Conference of Business Process Management*. 113–128.

A. Charfi and M. Mezini. 2007. AO4BPEL: An aspect-oriented extension to BPEL. *World Wide Web* 10, 1, 309–344.

A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. 2006. Reliable, secure, and transacted Web service compositions with AO4BPEL. In *Proceedings of the 4th European Conference on Web Services (ECOWS'06)*. 23–34.

A. Charfi, B. Schmeling, and M. Mezini. 2007. Transactional BPEL processes with AO4BPEL aspects. In *Proceedings of the 5th European Conference on Web Services (ECOWS'07)*. 149–158.

F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. 2003. The next step in Web services. *Communications of the ACM* 46, 10, 29–34.

J. Eder and W. Liebhart. 1996. Workflow recovery. In *Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems (COOPIS'96)*. 124–134.

A. Erradi, P. Maheshwari, and V. Tosic. 2006. Recovery policies for enhancing Web services reliability. In *Proceedings of the 4th IEEE International Conference on Web Services (ICWS'06)*. 189–196.

H. Erven, G. Hicker, C. Huemer, and M. Zapletal. 2007. The Web services-business activity-initiator (WS-BA-I) protocol: An extension to the Web Services-Business Activity specification. In *Proceedings of the 5th IEEE International Conference on Web Services (ICWS'07)*. 216–224.

R. Eshuis, J. Vonk, and P. Grefen. 2011. Transactional process views. In *On the Move to Meaningful Internet Systems: OTM 2011*. Lecture Notes in Computer Science, Vol. 7044. Springer, 119–136.

T. Fletcher, P. Furniss, A. Green, and R. Haugen. 2003. BPEL and business transaction management. In *Choreology Submission to OASIS WS-BPEL Technical Committee*.

J. Gray. 1981. The transaction concept: Virtues and limitations. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'81)*. 144–154.

P. Greenfield, A. Fekete, J. Jang, and D. Kuo. 2003. Compensation is not enough. In *Proceedings of the 7th International Conference on Enterprise Distributed Object Computing (EDOC'03)*. 232–239.

P. Grefen. 1993. Combining theory and practice in integrity control: A declarative approach to the specification of a transaction modification subsystem. In *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB'93)*. 581–591.

IBM. 2006. Web Services Atomic Transaction for WebSphere Application Server. Available at http://www.alphaworks.ibm.com/tech/wsat.

G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. 1997. Aspect-oriented programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming*. 220–242.

B. Limthanmaphon and Y. Zhang. 2004. Web service composition transaction management. In *Proceedings of the 15th Australasian Database Conference (ADC'04)*.

M. Little. 2003. Transactions and Web services. *Communications of the ACM* 46, 10, 49–54.

S. Maple. 2004. *Distributed Transaction with WS-AtomicTransaction and JTA*. Technical Report. IBM.

OASIS. 2004. Business Transaction Protocol Version 1.0. Retrieved April 24, 2015, from https://www.oasis-open.org/committees/business-transaction/documents/.

OASIS. 2007. Web Services Business Process Execution Language Version 2.0. Retrieved April 24, 2015, from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

M. Papazoglou. 2003. Web services and business transactions. *World Wide Web* 6, 1, 49–91.

M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. 2008. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems* 17, 2, 223–255.

R. Park. 1992. *Software Size Measurement: A Framework for Counting Source Statements*. Technical Report CMU/SEI-92-TR-20.

A. Portilla. 2006. Providing transactional behavior to services coordination. In *Proceedings of the Very Large Data Bases (VLDB) PhD Workshop*.

M. Schäfer, P. Dolog, and W. Nejdl. 2008. An environment for flexible advanced compensations of Web service transactions. *ACM Transactions on the Web* 2, 2, 1–36.

H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. 2002. Atomicity and isolation for transactional processes. *ACM Transactions on Database Systems* 27, 1, 63–116.

C. Sun. 2009. Towards transaction-based reliable service compositions. In *Proceedings of the 33rd Annual IEEE International Computer Software and Application Conference (COMPSAC'09)*. IEEE, Los Alamitos, CA, 216–221.

C. Sun and M. Aiello. 2007. Requirements and evaluation of protocols and tools for transaction management in service centric systems. In *Proceedings of the International Workshop on Requirements Engineering for Services (REFS'07), in conjunction with COMPSAC 2007*. IEEE, Los Alamitos, CA, 461–466.

C. Sun, E. Khoury, and M. Aiello. 2011a. Transaction management in service-oriented systems: Requirements and a proposal. *IEEE Transactions on Services Computing* 4, 2, 167–180.

C. Sun, Y. Shang, and F. Li. 2011b. A transaction-aware Web service development framework. *Chinese Journal on Computer Science* 38, 10, 6–11.

S. Tai, R. Khalaf, and T. Mikalsen. 2004. Composition of coordinated Web services. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*. 294–310.

SeCSE Team. 2007. Designing and deploying service-centric systems: The SeCSE way. In *Proceedings of Service Oriented Computing: A Look at the Inside (SOC@Inside'07).*

JBoss Transactions. 2013. JBoss Transactions Home Page. Retrieved April 24, 2015, from http://labs.jboss.com/portal/jbosstm.

I. Vasquez, J. Miller, K. Verma, and A. Sheth. 2005. OpenWS-Transaction: Enabling reliable Web service transactions. In *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05)*. 490–494.

WS-AT. 2007. *Web Services Atomic Transaction (WS-AtomicTransaction), Version 1.1*. Technical Report. Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies, and Microsoft.

WS-BA. 2007. *Web Services Atomic Transaction (WS-AtomicTransaction), Version 1.1*. Technical Report. Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies, and Microsoft.

WS-C. 2007. *Web Services Coordination (WS-Coordination)*. Technical Report. Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies, and Microsoft.

WSDL. 2007. Web Services Description Language Version (WSDL) Version 2.0 Part 1: Core Language. Retrieved April 24, 2015, from http://www.w3.org/TR/wsdl20/.

L. Zeng, H. Lei, J. Jeng, J. Chung, and B. Benatallah. 2005. Policy-driven exception-management for composite Web services. In *Proceedings of the 17th IEEE International Conference on E-Commerce Technology (CEC'05)*. 355–363.

O. Zimmermann, J. Grundler, S. Tai, and F. Leymann. 2007. Architectural decisions and patterns for transactional workflows in SOA. In *Proceedings of the 5th International Conference on Service Oriented Computing (ICSOC'07)*. 81–93.

Received December 2013; revised November 2014; accepted February 2015