Transaction management in Service-Oriented Systems: requirements and a proposal

Chang-ai Sun, Elie el Khoury, and Marco Aiello

Abstract—Service-Oriented Computing (SOC) is becoming the mainstream development paradigm of applications over the Internet, taking advantage of remote independent functionalities. The cornerstone of SOC's success lies in the potential advantage of composing services on the fly. When the control over the communication and the elements of the information system is low, developing solid systems is challenging. In particular, developing reliable Web service compositions usually requires the integration of both composition languages, such as the Business Process Execution Language (BPEL), and of coordination protocols, such as WS-AtomicTransaction and WS-BusinessActivity. Unfortunately, the composition and coordination of Web services currently have separate languages and specifications. The goal of this paper is twofold. First, we identify the major requirements of transaction management in Service-oriented systems and survey the relevant standards. Second, we propose a semi-automatic approach to integrate BPEL specifications and Web service coordination protocols, that is, implementing transaction management within service composition processes, and thus overcoming the limitations of current technologies.

Index Terms—Web services, Transaction Management, Business Process Execution Language.

1 Introduction

Standardized Web service technologies are enabling a new generation of software that relies on external services to accomplish its tasks. The remote services are usually invoked in an asynchronous manner. They are known by their published interfaces, and await invocations over a possibly open network. Single remote operation invocation is not the revolution brought by Service-Oriented Computing (SOC), though. Rather it is the possibility of having programs that perform complex tasks coordinating and reusing many loosely coupled independent services. It is the possibility of having programs manage business processes which span over different organizations, people and information systems. The scenarios opened by SOC are therefore unprecedented, for instance, (i) supply chains can become ever more dynamic, efficient and cost effective, (ii) vertical marketplaces can become open and dynamic both in terms of access and reaction to changes, (iii) virtual enterprises can be created based on the functional properties (the governing processes), rather than being confined by geographical or bureaucratic constraints.

A new approach to software, such as that brought by SOC, calls for new ways of engineering software and for new problems to be solved. The central role of these systems is played by services which are beyond a centralized control and whose functional and, possibly, non-functional properties are discovered at run-time. The key problems are related to the issue of discovering services and deciding how to coordinate them. For instance, while planning to drive to a remote city, one might discover that it is heavily snowing there, and may want to obtain snow tyres. Therefore, one needs to find a supplier and a transport service to have the appropriate tyres in a specific location by a specific deadline. That is, various independent services are composed into the form of a process, called the 'get winter tyres while traveling' with the requirement that we order the tyres if and only if we find also a transport service for them. In other words, we require the services of tyre ordering and tyre delivery to be composed in a transactional manner.

In the present treatment, a *service* is a standard XML description of an autonomous software entity, it executes in a standalone container, it may have one or more active instantiations, and it is made of possibly many operations that are invoked asynchronously. A *service composition* is a set of operations belonging to possibly many services, and a partial order relation defining the sequencing in which operations are to be invoked. Such a partial order is adequately represented as a direct graph. A *service transaction* is a unit of work comprehending two or more operations that need to be invoked according to a specific *transaction policy*. The *coordination* of a service transaction is the management of the transaction according to a given policy. A service transaction can span over operations of one service or, more interestingly, of several services.

- *M. Aiello and E. el Khoury are with Johann Bernoulli Institute, University of Groningen, The Netherlands.*
  E-mails: m.aiello@rug.nl, e.el.khoury@rug.nl
- *C. Sun is with School of Information Engineering, University of Science and Technology Beijing, China.*
  E-mail: casun@ustb.edu.cn

This work is supported by the EU Integrated Project SeCSE (IST Contract No. 511680): Service Centric System Engineering and the Research Fund for the Doctoral Program of Higher Education of China (Grant No. 2008000401051) and Natural Science Foundation of China (Grant No. 60903003). We thank Dieter Hammer, Oliver Kopp and Michele Mancioppi for fruitful discussion, Gerard Biemolt and Heerko Groefsema for work on the related implementation, and the anonymous reviewers for their suggestions.

One may argue that transaction management is a well-known technique that has been around for ages but, as anticipated by Gray [1] more than fifteen years ago, nested, long-lived transactions demand for different techniques, and in fact they do. To cater for the new features of transactions executed by Web services, various Web transaction specifications have been developed. WS-Coordination [2] specification describes an extensive framework for providing various coordination protocols. The WS-AtomicTransaction and WS-BusinessActivity specifications [3], [4] are two typical Web transaction protocols. They leverage WS-Coordination by extending it to define specific coordination protocols for transaction processing. The former is developed for simple and short-lived Web transactions, while the latter for complex and long-lived business activities. Finally, the Business Process Execution Language (BPEL) [5], [6], [7] is a process-based composition specification language. In order to develop reliable Web services compositions, one needs the integration of transaction standards with composition language standards such as BPEL [8], [9]. Unfortunately, these are currently separate specifications.

This paper has a double goal: The first one is to look at the requirements of transaction management for Service-oriented systems. The systematization of requirements is the starting point for an analysis of current standards and technologies in the field of Web services. The second goal of the paper is to propose a framework for the integration of BPEL with transaction protocols such as WS-AtomicTransaction and WS-BusinessActivity. We use a simple but representative example across the paper, the drop dead order one, to illustrate requirements and the proposed approach.

The need for filling the gap regarding transaction management for BPEL in a declarative way is testified also by other proposals in the same line. E.g., independently and in the same time window, Tai et al. [10] have worked out a declarative approach to Web service transaction management. Their approach is very similar to ours with respect to the execution framework and the use of a policy-driven approach to extend BPEL definitions with coordination behavior. However, they do not consider the semi-automatic identification of transactions and consequent process restructuring as we do. Earlier, Loecher proposed a framework for a model-based transaction service configuration, though it was never implemented [11]. Even before the birth of Web services, declarative approaches to automate transaction management have been proposed, most notably [12].

The present work extends our survey and requirement analysis for service transactional systems [13] and our proposal of the XSRL language for handling requests against service compositions [14], [15]. In XSRL a construct is defined to express atomicity of services execution, though no means for recovering from failures is provided.

The rest of the paper is organized as follows. First, we introduce the drop dead order example in Section 2.

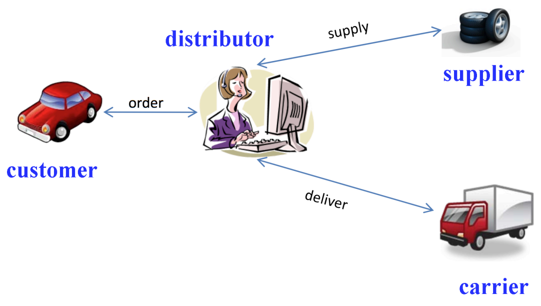

Fig. 1. The drop dead order example.

We continue by looking at transaction requirements in Section 3 and considering Transaction standards in Section 4. The proposed approach to transaction management is presented in Section 5 and illustrated using the example in Section 6. Section 7 contains a discussion of related work. Conclusions and remarking discussions regarding transaction management in Service-oriented systems are presented in Section 8.

2 The Drop Dead Order Example

The drop-dead order example (DDO) describes a scenario where a customer wants to order products from a distributor with the requirement that these must be delivered before the drop-dead date. To satisfy such a request, the distributor will try to find a supplier that has the products available. If found, he will search for a carrier that is able to deliver the products before the drop-dead date. If both the supplier and the carrier are able to fulfill the demands of the customer, the distributor will report to the customer that he can fulfill the order. After the customer has sent a confirmation of the order to the distributor, the latter sends a confirmation to the supplier and the carrier. We consider the drop-dead example in the context of the automotive industry as depicted in Figure 1.

The example is due to Haugen and Fletcher [16] who first introduced the drop-dead order to illustrate multi-party electronic business transactions. We extended this case study to demonstrate the various faults and exceptions of transactions. Though simple, we argue that the example captures all relevant aspects of a transaction in the context of service oriented systems, since it may be built from several Web services and it requires transaction management, especially nested and long-lived transactions.

One of the trends in the car industry is providing ever more sophisticated services to the driver. We have recently witnessed the blooming of GPS-based navigation equipments. The next step is to provide value-added services. For instance, the driver could desire to have traffic information and, on the basis of this, book a hotel and a parking space. Or, by checking the weather report, he may decide to acquire a set of winter tires while on the way to a ski resort. That is, the driver

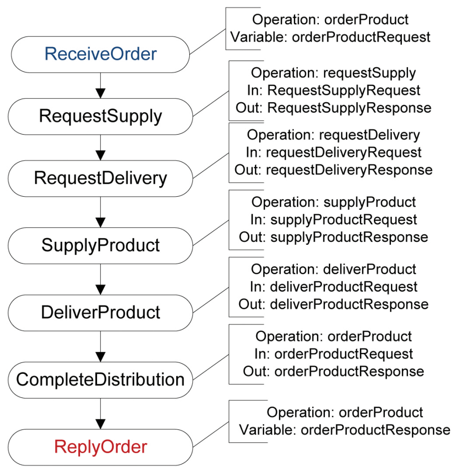
ReceiveOrder
Operation: orderProduct
Variable: orderProductRequest
RequestSupply
Operation: requestSupply
In: RequestSupplyRequest
Out: RequestSupplyResponse
RequestDelivery
Operation: requestDelivery
In: requestDeliveryRequest
Out: requestDeliveryResponse
SupplyProduct
Operation: supplyProduct
In: supplyProductRequest
Out: supplyProductResponse
DeliverProduct
Operation: deliverProduct
In: deliverProductRequest
Out: deliverProductResponse
CompleteDistribution
Operation: orderProduct
In: orderProductRequest
Out: orderProductResponse
ReplyOrder
Operation: orderProduct
Variable: orderProductResponse

Fig. 2. A simplified representation of the drop-dead order example.

may request his car information system to execute the process of ordering winter tires in a specific area in a given time frame. The car information system then has to find a tire supplier and a carrier that will deliver the tires in the appropriate location with a certain deadline. Of course, the driver is interested in tires only if these are delivered before going to the ski resort, therefore the car's information system has to perform the required process in a transactional manner and take into account the non-functional requirement of drop-dead time.

Let us now bring the example to the realm of Web services. To construct such a system, we need to develop three web services each fulfilling the responsibility on behalf of a *distributor*, *supplier* and *carrier*, respectively. Then, a business process describing the drop-dead order scenario can be built by orchestrating these Web services.

Figure 2 illustrates the major segments of the business process represented as a BPEL specification. First, the BPEL process receives an order for requesting a distributor to distribute some products via the operation *orderProduct*. If the order is valid and the distributor provides the service required by this order, the process will respond positively. Otherwise, it will reject the order request.

When the distributor accepts the order, the BPEL process seeks a suitable supplier to ask for supplying the products via the operation *RequestSupply*. If the request result from the supplier is positive, the BPEL process seeks a suitable carrier to deliver the products via the operation *RequestDelivery*. Otherwise, the process will return a fault message indicating no suitable supplier is available for the supply.

Similarly, if a carrier is found, the process will ask the supplier to dispatch the products via the operation *SupplyProduct*, and ask the carrier to deliver the products via the operation *deliverProduct*. Otherwise, the process will return a fault message indicating no suitable carrier is available for the delivery. After the successful delivery of the products to the customer, the process is informed by the distributor with the positive result, and it returns a success message.

3 Transaction Requirements

In the field of databases, transactions are required to satisfy the so called ACID properties, that is, the set of operations involved in a transaction should occur atomically, should be consistent, should be isolated from other operations, and their effects should be durable in time. Given the nature of service oriented systems, satisfying these properties is often not possible and, in the end, not necessarily desirable [17]. In fact, some features are unique to service oriented systems:

- Long-lived and concurrent transactions, not only traditional transactions which are usually short and sequential.
- Distributed over heterogeneous environments.
- Greater range of transaction types due to different types of business processes, service types, information types, or product flows.
- Unpredictable number of participants.
- Unpredictable execution length. E.g., information query and flight payment needs 5 minutes; while e-shopping an hour; and a complex business transaction like contracting may take days.
- Greater dynamism. Computation and communication resources may change at run-time.
- Unavailability of undo operations, most often only compensating actions that return the system to a state that is close to the initial state are available.

Furthermore transactions may act differently when exposed to certain *conditions* such as logical expressions, *events* expressed in deadlines and even errors in case of a *faulty Web service*. To make sure that the integrity of data is persistent, the two transaction models used are namely *Composite* and *Distributed* allow smooth recovery to a previous "safe" state.

The set of emerging features mentioned earlier, which are a combination of requirements mostly coming from the areas of databases and workflows, provide the basis for identifying the most relevant requirements for transactions in service-oriented systems. We list these next, starting from the classic ACID properties, then considering behaviors, models and further issues of service transactions.

3.1 ACID properties

3.1.1 Atomicity

Atomicity is the property of a transaction to either complete successfully or not at all, even in the event of partial failures. In the Drop Dead Order Example, it should not happen that the supplier's resources are committed while the Carrier is not.

3.1.2 Consistency

Consistency is the property of a transaction to begin and end in a state which is consistent with the intended semantics of the system, i.e., not breaking any integrity constraints. A state in which the Carrier is committed but has never prepared to commit is inconsistent.

3.1.3 Isolation

Isolation is the property of a transaction to perform operations isolated from all other operations. One transaction can therefore not see the other transaction's data in an intermediate state. The Customer should not be aware of the state of the transaction between the Distributor and the Supplier/Carrier regarding a different order.

3.1.4 Durability

Durability is the property of a transaction to record the effects in a persistent way. Whenever a transaction notifies one participant of successful completion, the effects must persist, even when subsequent failures occur. When the Supplier is notified of a successful completion, but somehow the connection with the Carrier fails, the changes with the Carrier should still be made.

3.2 Transaction behaviors

3.2.1 Rollback

Rollback is the operation of returning to a previous state in case of a failure during a transaction. This may be necessary to enforce consistency. In the DDO, when the Distributor assigns a Supplier, but cannot assign a Carrier, the changes made with the Supplier (and Customer) should be rolled back.

3.2.2 Compensating actions

Compensating actions are executed in the event of a failure during a transaction, all changes performed before the failure should be undone. If the Distributor assigned a Supplier and committed it but cannot assign a Carrier, the changes made with the Supplier (and Customer) should be compensated.

3.2.3 Abort

Abort is the returning to the initial state in case of failure or if the user wishes so. When the Distributor assigns a Supplier but cannot assign a Carrier, the entire transaction is to abort.

3.2.4 Adding deadlines

Adding deadlines to transactions involves giving timeouts to operations. Suppose that the Customer needs the goods before a certain time, then the Distributor and the Carrier need to comply with certain time constraints, too.

3.2.5 Logical expressions

Logical expressions for specifying constraints are used for giving unambiguous and semantically defined rules for guaranteeing consistency. For instance, the fact that the account of the Distributor cannot be debited while the account of the Customer is not credited in the event of a money exchange can be expressed by debited(distributor) + credited(customer) = 0.

3.3 Transaction models

3.3.1 Composite transactions

Composite transactions are nested transactions. In the Drop Dead Order example, the distribution transaction consists of two sub-transactions, namely, the supply and the deliver transactions. These transactions depend on the global outcome, that is, all three succeed or the whole composite transaction fails.

3.3.2 Distributed transactions

Distributed transactions are transactions between two or more parties executing on different hosts. The transaction should support transactions through a network between two different hosts. A customer can place a drop-dead order at the Distributor through a network connection.

3.4 Transaction Behavior - Alternatives

3.4.1 Transaction recovery

Transaction recovery by dynamic rebinding and dynamic re-composition at run-time is the possibility of replacing a faulty Web service when the current service is not able to fulfill its promises. Dynamic re-composition is the forming of a new composition by replacing one or several services by another composition that fulfills the same function. Imagine that the first Carrier somehow fails and is unreachable. If this happens during a transaction, then automatic re-bind with a service that offers the same service should take place. Re-composition through re-binding with a third Carrier through the Supplier is also a possibility.

3.4.2 Optimistic or pessimistic concurrency control

Optimistic or pessimistic concurrency control refers to the support of different types of concurrency control to enforce consistency. This control could either be optimistic or pessimistic. The pessimistic approach prevents an entity in application memory by locking it in the transaction for the entire time. While the optimistic simply chooses to detect collisions and then resolve the collision when it does occur. This scheme has better performance. When two transactions are concurrent, they should not both claim the same supply of goods from one Supplier.

For the Drop Dead Order example, we see that all these requirements are necessary with the exception of the last two points. Existing transaction protocols are based on pessimistic concurrency control (locking). But let us look at this in more detail by considering, first existing standards and composition languages, and then tools referring to the just listed requirements.

4 Transaction Standards and Service Composition Languages

WS-Transactions [3], [4] and Business Transaction Protocol (BTP) [18] are the two most representative standards that directly address the transaction management of Web service-based systems, while for representing compositions of services the Business Process Execution Language (BPEL) [19] and the Choreography Description Language (WS-CDL) are most widely known and adopted. WS-Transactions consists of two coordination protocols: WS-AtomicTransaction (WS-AT) [3] and WS-BusinessActivity (WS-BA) [4] which live in the WS-coordination framework [2]. WS-AT provides the coordination protocols for short-lived simple operations, while WS-BA provides the coordination protocols for long-lived complex business activities. The WS-coordination framework is extensible and incremental. That is, WS-coordination can enhance existing service oriented systems with transaction properties by wrapping them with a specific coordination. On the other hand, BTP is a model for long-lived business transaction structured into small atomic transactions, and using cohesion to connect these atomic operations. Its motivation is to optimize the use of resource involved in a long-lived transaction under loosely coupled Web service environments and avoiding the use of a central coordinator. BPEL provides the facilities to specify executable business processes with references to services' interfaces and implementations. It does handle some basic issues of transactions, such as compensation, fault and exception handling, but other transaction requirements are not managed. WS-CDL provides the infrastructure to describe cross-enterprise collaborations of Web services in a choreographic way. The transactions are not explicitly addressed, but some facility can be used to satisfy some basic transaction properties, as we see next.

Consider the proposed protocols that take the transaction and the business perspective of Service-oriented systems with respect to the requirements identified in the previous section. In Table 1, we summarize the results of the evaluation for all requirements-each row-and for all protocols-each column-by denoting the satisfaction with the '$\oplus$' symbol, the partial satisfaction with '$\odot$', and no support with '$\ominus$'. We refer to [20] for details on the evaluation.

First, we notice that WS-Transaction actually consists of two different protocols with different properties, which we analyze separately. WS-AT is a traditional protocol which satisfies the basic ACID properties. WS-BA, on the other hand, renounces atomicity to accommodate long-lived transactions. BTP has included confirm-sets. These confirm-sets let the application element choose which operations with parties in the transaction are to be canceled and which are to be confirmed. In this way, the application element is able to contact more services which perform the same task and to choose the best option. Unfortunately, BTP is not part of the WS-Stack, which limits its compatibility with other Web service technologies. In addition, BTP does not support long-lived transactions. There is also a difference in granularity between the above transaction standards. WS-AT contains simple two phase commit protocols, WS-BA contains non-blocking protocols and BTP consists of a sequence of small atomic transactions. As for security, WS-Security [21] can be combined with WS-Transaction as well as with BTP.

TABLE 1
Evaluation Results

Requirements	BTP	WS-AT	WS-BA	BPEL	WS-CDL
3.1.1 Atomicity	$\oplus$	$\oplus$	$\ominus$	$\odot$	$\odot$
3.1.2 Consistency	$\oplus$	$\oplus$	$\odot$	$\odot$	$\odot$
3.1.3 Isolation	$\ominus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
3.1.4 Durability	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\odot$
3.2.1 Rollback	$\oplus$	$\oplus$	$\odot$	$\odot$	$\oplus$
3.2.2 Compensating actions	$\ominus$	$\ominus$	$\oplus$	$\oplus$	$\odot$
3.2.3 Abort	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\ominus$
3.2.4 Adding deadlines	$\ominus$	$\oplus$	$\oplus$	$\odot$	$\oplus$
3.2.5 Logical expressions	$\ominus$	$\ominus$	$\ominus$	$\oplus$	$\oplus$
3.3.1 Composite trans.	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
3.3.2 Distributed trans.	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
3.4.1 Trans. recovery	$\ominus$	$\ominus$	$\ominus$	$\oplus$	$\ominus$
3.4.2 Concurrency control	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\oplus$

Dynamic rebinding is supported only by BPEL, though only at the implementation level. WS-CDL supports most requirements, while its major disadvantage is that the large players in the field do not support it and that no implementation is available.

We can further draw the following conclusions in terms of extensions to the traditional transaction model. WS-AT is a very conservative business transaction model especially with respect to blocking. WS-BA is more appropriate for services, by renouncing to the concept of the two-phase commit. BTP places itself in the middle (two phase commit is followed in a relaxed way). As for BPEL and WS-CDL they address the business process perspective with limited transaction support.

We refer to [13] for survey and comparison of tools available for transaction management. The survey includes tools such as Apache Kandula, Active Engine 2.0 or JBoss Transactions.

5 Proposal for Integrating Transactions into BPEL

The above survey shows that there are standardized protocols for describing transactions and languages for describing processes in terms of flows of activities. The connection among these is, to say the least, very loose. The problem is that processes are described in terms of activities and roles capable of executing the activities, but semantic dependencies among these activities are not represented beyond message and flow control. It may happen that several operations from a single Web service are invoked within a BPEL process, and dependencies among these operations may exist.

For example, before a supplier provides the product requested by a distributor, he needs first to process the request and then reply to the requester. The two operations correspond to two activities in the BPEL process, namely *providing products* and *processing request*, which need to be managed in some transactional way, but BPEL is unable to capture the right granularity and the dependencies among operations.

Our proposal consist of making the dependencies among the activities explicit via an automatic procedure and performing a restructuring step of the process, where necessary. The identified dependencies among activities can be then identified by the designer of the process as being transactions or not. In case they are, the designer will decide which kind of transactions they are and simply annotate them. The execution framework then takes care that transaction annotations are correctly managed at run time. The need for the human design decision in the process is necessary due to the lack of semantic annotations of the BPEL processes. Only the designer can decide whether a set of activities that seem to have a dependency in the process are to be executed transactionally or not.

Let us be more precise on what the phases of the proposed approach are. Consider Figure 3, where data transformation goes from left to right and we distinguish three layers: the data layer at the bottom, the middle execution layer defining the data transformation, and the knowledge level indicating from where the knowledge to transform the data comes. We start with a generic business process designed to solve some business goal. An automatic processing step, which we define next, identifies dependencies among activities. These are then reviewed by an expert that decides which are actually transactions and which not. For those who qualify, s/he further decides what kind of transactions they are and annotates them. For instance some may be long running while others may be atomic ones. We remark how this is a design step performed by an expert who understands the domain, the specific process and the consequences of choosing a transaction policy in favor of another. This step cannot be automated unless further semantic annotations are made on the BPEL. The restructured and annotated process is then ready to be sent for execution. We notice that the restructured process may be sent to execution several times. That is, the manual effort will occur only once and allow for many execution instances. In fact, at this stage no concrete binding has occurred. This will be postponed till the execution phase and will be handled by the execution framework. Next we consider the three phases of the approach individually.

5.1 Preprocessing

Preprocessing the BPEL specification is performed in two steps, namely (i) identification and (ii) resolution of transaction dependencies. In order to illustrate the two steps, we introduce an abstract model of BPEL.

5.1.1 Abstract model of BPEL specifications

A BPEL process specification describes the interaction between services in a specific composite Web service. Its abstract model, known as *behavioral interface*, defines the behavior of a group of services by specifying constraints on the order of messages to be sent and received from a service [22]. In this sense, a BPEL specification S is a set of activities A and its associated links L, represented by $S = (A, L)$. The links, which are directed, define a partial ordering over the set of activities and are thus well represented as a directed graph (e.g., Figure 4).

- An activity a in A having a type represented by T_a, has the following properties:
  - name N_a.
  - operation OP_a, which is usually implemented by the Web service at a specific port.
  - input variable IV_a and output variable OV_a, which specifies the parameters required and produced by the OP_a, respectively.
  - set of source links SL_a and set of target links TL_a, which specify the outgoing and incoming links (transitions), respectively.
- A link l in S has a unique name N_l and is indirectly defined through two activities a_1 and a_2 which indicates not only the direction l^d of the transition, but also the conditions l^c for the transition to take place.

Furthermore, the *Customer-to-distributor* link l_{c-d} is one of the source links of the *ReceiveOrder* activity a_1, namely $l_{c-d} \in SL_{a_1}$. Furthermore, $l_{c-d} \in TL_{a_6}$, where TL_{a_6} is the target link of the *CompleteDistribution* activity a_6. Therefore, the link l_{c-d} connects the transition between a_1 and a_6, denoted as $a_1 \xrightarrow{l_{c-d}} a_6$. Figure 4 provides an illustration of $a_1 \xrightarrow{l_{c-d}} a_6$.

5.1.2 Dependencies identification algorithm

If one specifies a set of activities within a given BPEL specification S, there may exist dependencies among activities that can hinder the application of transaction management as described above. Assume that

$$S_t = \{a_i \mid a_i\, is\, a\, transactional\, activity\, of\, a\, transaction\, t\}$$

is a transaction t specified within the BPEL specification S.

For any two activities a_m and a_n where $a_m, a_n \in S_t$ and $a_m \neq a_n$, if there exists a path $a_m \xrightarrow{l_{j_1}} ... \xrightarrow{l_{j_k}} a_n$ where l_{j_1} and l_{j_k} are some links connecting activities, we

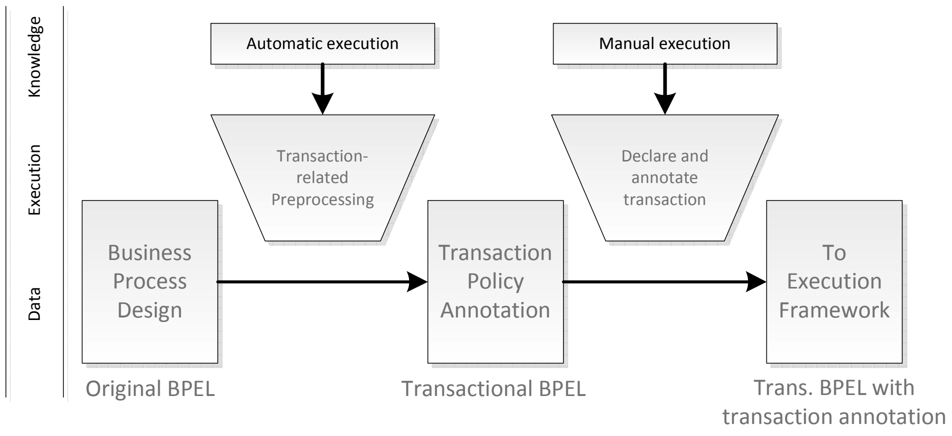
Knowledge
Execution
Data
Automatic execution
Manual execution
Transaction-related Preprocessing
Declare and annotate transaction
Business Process Design
Transaction Policy Annotation
To Execution Framework
Original BPEL
Transactional BPEL
Trans. BPEL with transaction annotation

Fig. 3. Approach to integrating transactions into BPEL processes.

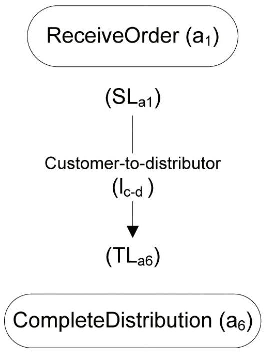

Fig. 4. Representation of activities and the link that connects them.

say that a_n is *reachable* from a_m, denoted as $a_m \xrightarrow{*} a_n$, and $\{l_{j_1}...l_{j_k}\}$ is a link chain of $a_m \xrightarrow{*} a_n$ denoted as $LC(a_m \xrightarrow{*} a_n)$. For any two activities a_m and a_n in a transaction S_t that are implemented by the same Web service, if $a_m \xrightarrow{*} a_n$ and $OV_{a_m} \in l^c$ where $l \in LC(a_m \xrightarrow{*} a_n)$, then a transaction dependency exists between a_m and a_n.

To identify the existence of transaction dependencies within a given BPEL specification S, we propose Algorithm 5.1. The algorithm is a standard graph algorithm similar to those for reachable set construction, e.g., [23]. The function *IdentifyDependency* takes S as input and outputs a boolean value that represents the existence of transaction dependencies *td*. The function first creates a path p for any two activities a_m and a_n. Then traverses the links in the link chain l_s obtained from p. When a link l is detected and its transition condition l^c contains the output variable OV_{a_m} of the first activity a_m, or if it contains an output variable OV_{a_i} which is identical to OV_{a_m} semantically, the algorithm stops and returns TRUE. Otherwise, it continues until all pairs of activities in S_t have been visited. Finally, if no transaction dependencies are detected, the algorithm returns FALSE.

Algorithm 5.1: IDENTIFYDEPENDENCY(S)

```
$td =$ **false**
**for each** $a_m \in S$
  **do** {
    **for each** $a_n \in S$ **and** $a_m \neq a_n$
      **do** {
        **if** $\exists p, and\, a_m \xrightarrow{p} a_n \in S$
          **then** {
            $ls = \phi$ **comment:** ls is a set of links.
            $p \xrightarrow{store\ transitions} ls$
            **for each** $link\, l \in ls$
              **do** {
                **if** $l^c \neq \phi$ **and** $OV_{a_m} \in l^c$
                  **then** {$td =$ **true**
                  **else if** $l^c \neq \phi$ **and** $OV_{a_i} \in l^c$
                **and** $OV_{a_i} \in OV_{a_m}$
                  **then** {$td =$ **true**
**return** $(td)$
```

5.1.3 Resolution of dependencies

Once transaction dependencies are identified, it is necessary to handle them. To solve this problem, we merge the dependent activities into one transaction. Algorithm 5.2 resolves the transaction dependencies within a BPEL specification *S*. It employs Algorithm 5.1 to detect transaction dependencies and it asks the user for confirmation that it is indeed a transactional dependency. The output is a new BPEL specification referred to as preprocessed BPEL where conflicts are resolved.

Algorithm 5.2: DEPENDENCYRESOLVER(S_t)

```
$PS = S_t$
**for each** $a_m \in S_t$
  **do** {
    **for each** $a_n \in S_t$ **and** $a_m \neq a_n$
      **do** {
        **if** $IdentifyDependency(a_m, a_n, PS) =$ **true**
        **and** $user\_agrees\_that\_it\_is\_transaction$
          **then** {$PS \longleftarrow PS(a_m/a_n)$
**return** $(PS)$
```

Figure 5 and 6 illustrate the schematic flowcharts of the BPEL specification for the drop-dead example

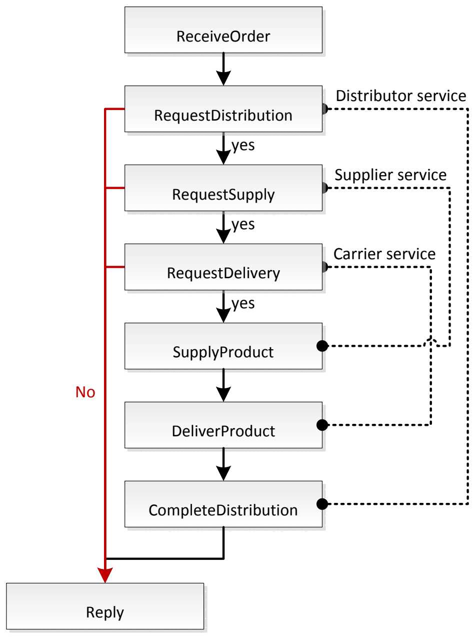

Fig. 5. The schematic flowchart for the drop-dead order example.

before and after the processing using Algorithm 5.2, respectively.

Figure 5 shows the existence of transaction dependencies in the original BPEL specification for the drop-dead order example. That is, for some activities from the same Web service, there is always another companion activity. For example, before the *supplier* service supplies products (the *SupplyProduct* activity in Figure 2), a request activity (the *RequestSupply* activity in Figure 2) is performed. These two companion activities are similar to *Prepare* and *Commit* in the two-phase-commit protocol. When Algorithm 5.1 is applied to the drop-dead order example, it can identify the transaction dependency between the *RequestSupply* activity (denoted as a_2) and the *SupplyProduct* activity (denoted as a_4), because both a_2 and a_4 are implemented by the *supplier* service, and the precondition for executing a_4 contains the output variable of a_2. Similarly, the algorithm identifies the transaction dependency between *RequestDistribution* and *CompleteDistribution*, and the transaction dependency between *RequestDelivery* and *DeliverProduct*. In this case, all identified dependencies lead to an actual transaction.

Figure 6 illustrates transactional activities within the preprocessed BPEL specification where the transactional dependencies have been resolved using the algorithm 5.2. For example, our algorithm merges the *RequestSupply* activity (a_2) and the *SupplyProduct* activity (a_4), resulting in one single transactional activity namely the *SupplyProduct* activity (a_4). Similarly, *CompleteDistribution* and *DeliverProduct* are treated as transactional activities. Note that three transaction scopes are specified in Figure 6. The *supply* transaction (T2) and *deliver* transaction (T3) are nested in the distribution transaction (T1). For each

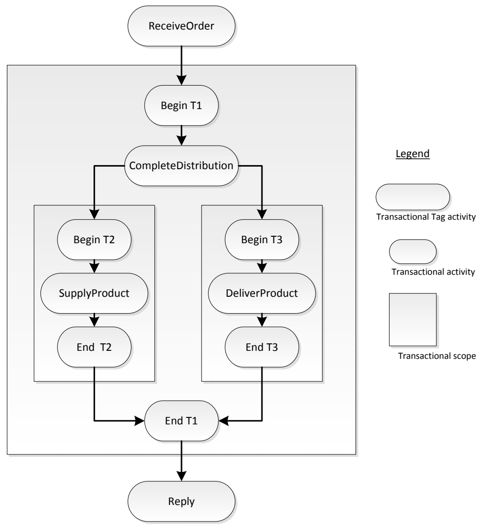

Fig. 6. Transactional activities in the preprocessed BPEL specification (DDO).

of these three transactions, one transactional activity (an *invoke* activity) is annotated with *transaction policy*. The meaning of transaction policy will be explained in the subsequent sections.

Note that although the detection and elimination are done automatically, a human interaction is necessary when a design choice must be made to represent the relationship between different transactions. Consider for instance, the Drop-dead Order example, there are three transactions (T_1, T_2 and T_3). One could decide that T_2 is going to run in parallel with T_3, and both of these are nested in T_1 as we do in Figure 6. However, another valid choice is that of including T_3 in T_2 this would represent the case where the supplier accepts the order, the carrier is going to deliver it, and this way the parallelism between T_2 and T_3 is removed. Moreover, Algorithm 5.1 detect all possible transactions within a process. Consequently, due to the fact that multiple solutions might exist for even a simple BPEL process, the human interaction is mandatory at this step.

5.2 Declaration of transaction policies

Once transactions are identified and BPEL has been accordingly restructured, one needs to define the desired transactional behavior. To this end, we introduce a reference transaction policy declaration schema, shown in Figure 7. With this schema, one can declare the transaction policy using the following elements:

1) *Trans_ID* is a non-zero integer, representing transactions within a business process.
2) *Trans_Protocol* specifies a protocol for the transaction, such as WS- AtomicTransaction (WS-AT) or WS-BusinessActivity (WS-BA).
3) *Trans_Root* indicates the parent transaction identified by Trans_ID. The value 0 is used to indicate the root transaction within the business process.

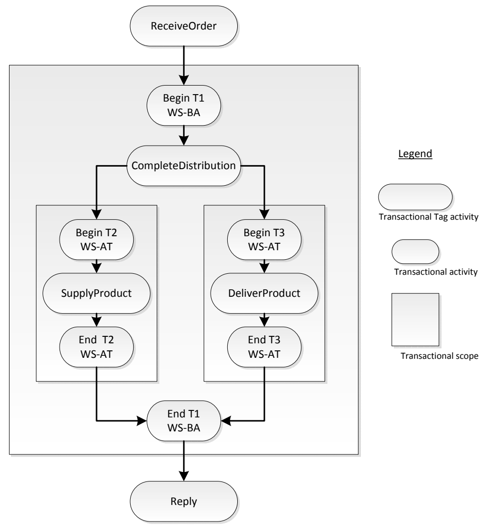

Fig. 8. An illustration of transaction policies for the drop-dead order example.

One can specify the hierarchy of transactions by assigning appropriate Trans_IDs and Trans_Roots.

With such a schema, one can annotate constraints or preferences to a specific activity in the BPEL specification. The annotated activity must be an *invoke* activity. One can separately specify the desired constraints or preferences in the *design-time-info* or *run-time-info* sections. For transaction management, we declare the transaction policies in the section of the *trans-info* which is embedded within the section of run-time-info, since a transaction policy is a run-time constraints. Together with the other types of process information, transaction policies are stored in an XML file for use at run-time.

Let us consider again the drop-dead order example. After the preprocessing, we found three transactional activities belonging to different transaction scopes (Figure 6). Then, one declares the desired transaction policies. For the purpose of illustration, suppose that one would like to implement the *SupplyProduct* activity and the *DeliverProduct* activity using the WS-AtomicTransaction protocol (WS-AT), and implement the *CompleteDistribution* activity using the WS-BusinessActivity protocol (WS-BA), respectively. Figure 8 illustrates such a declaration of the preferred transaction policies for the drop-dead order example[1].

In order to support the declaration of transaction policies, we need to annotate the transactional activities using the proposed transaction policy declaration schema. By analyzing these annotation profiles, one may deduce that the CompleteDistribution transaction (T1) is the top transaction because its Trans_Root is 0, while both the SupplyProduct transaction (T2) and the DeliverProduct transaction (T3) are sub transactions of the CompleteDistribution transaction, since the latter Trans_ID is 1 and the SupplyProduct and DeliverProduct Trans_Root are 1.

1. Notice that the "Transaction Scope" used here is not related to the BPEL "scope" tag.

5.3 The Execution framework

The proposed approach transforms a generic business process into a restructured one in which transactions are identified and annotated. Now one needs an execution framework that is richer than a simple BPEL engine. In fact, one needs to interpret the annotations, make sure that activities are executed according to the transaction conditions and also that the binding among dependent activities is consistent with the transaction semantics. To achieve this we rely on the SeCSE platform in the context of which the current approach has been developed.

Service Centric System Engineering (SeCSE) is an European sixth framework integrated project, whose primary goal is to create methods, tools and techniques for system integrators and service providers and to support the cost-effective development of service-centric applications [24], [25]. The SeCSE service composition methodology supports the modeling of both the service interaction view and the service process view [26]. A service integrator needs to design both the abstract flow logic and the decision logic of the process-based composition. Therefore, the SeCSE composition language allows the definition of a service composition in terms of a process and some rules that determine its dynamic behavior [27]. Correspondingly, the flow logic can be represented by a BPEL specification, while the decision logic is defined by rules.

Based on the architecture of the SeCSE platform, we built a transaction management tool called *DecTM4B*. It consists of three modules, namely

- The *Preprocessor for T.M.* is used to identify and eliminate transaction dependencies occurring in the original BPEL specification. The output is the preprocessed BPEL specification. The SeCSE platform will deal with the binding of abstract services before the BPEL engine executes the BPEL specification. The preprocessing executed by Preprocessor for T.M. happens just before the binding. Currently, ODE and ActiveBPEL [28] are two BPEL engines supported by the SCENE platform.
- The *Event Adapter* maps the low-level events from the BPEL engine onto the binding-related events. The first version of SeCSE event adapter is extended to support the mapping of transaction-related events.
- The *Transaction Manager* is a separate component in the executor and deployed in the Mule container (Mule is a messaging platform based on ideas from Enterprise Service Bus (ESB) architectures). The Transaction Manager consists of the following two transaction-specific components.
  1) *TransLog* is responsible for managing the lifecycle of transactions, such as creating transaction instances, maintaining the status of transaction instances, and destroying transaction instances.

```
 <activity-info activityName=@ncname>
  <design-time-info>
        <!-- To do: define design-time information -->
  </design-time-info>
  <run-time-info>
      <!-- To do: define other run-time  information -->
      <trans-info>
             <Trans_ID> i </Trans_ID> <!-- where i is a non zero integer -->
             <Trans_Protocol> [WS_AT] | [WS_BA] </Trans_Protocol>
             <Trans_Root> Trans_ID </ Trans_Root >
      </trans-info >
  </run-time-info>
</activity-info>
```

Fig. 7. A transaction policy declaration schema.

TransLog is also responsible for transferring the information among the components in the executor. For example, it listens the transaction related events from the Event Adapter, and it is responsible for the communication between Transaction Manager and JBoss Transaction Server.

2) *PolicyOperator* retrieves the transaction policies from the XML file, and parses the transaction policies, and then maps transaction policies onto the coordination context. It provides a set of APIs which are to be called by the TransLog.

As for the implementation of transaction protocols, we rely on JBoss Transaction Server [29]. JBoss Transaction Server is an open source implementation of WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity. It provides a set of APIs to support the coordination services and transaction protocols. JBoss Transaction Server is selected for this purpose because it (1) is a complete, standalone, open source software tool, (2) has sufficient documentation, (3) and supports WS-Coordination and WS-Transaction.

6 A Run of the DDO Example

Let us consider again the drop-dead order example introduced in Section 2 and apply the proposed methodology for transaction management. In particular, let us consider the drop-dead order example in the context of the automotive industry with a BPEL process specification (denoted as S_1) and three Web services instances. The process starts with an order request through the *Receive* activity, then it transfers the request variable to the *carrier* and *supplier* services for their confirmation, and finally replies to the request by the *Reply* activity. The process returns a positive result if both the supplier and carrier services can provide the declared service; otherwise it returns a rejection. The Web services developed and deployed are transactional Web services that execute the business operations and at the same time are aware of the coordination responses. In our case, they are participants of a transaction and must implement the coordination interfaces for specific transactional protocols. Table 2 lists signatures in the coordination interfaces for different transactional protocols applied [29] (DecTM4B reuses the coordination infrastructure in the JBoss Transaction Server). We refer to the specifications of WS-AtomicTransaction and WS-BusinessActivity for details, to [13], [30] for the reference implementation for these signatures and we omit them here for brevity.

TABLE 2
Lists of signatures for different transactional protocols.

Protocols	Signatures supported by participants
WS-AT Durable2PC	prepare, commit, rollback, commitOnPhase, unknown, error
WS-AT Volatile2PC	prepare, commit, rollback, commitOnePhase, unknown, error
WS-BA with Participant Completion	close, cancel, compensate, status, unknown, error
WS-BA with Coordinator Completion	close, cancel, compensate, complete, status, unknown, error

Then, the BPEL specification s_1 is processed using the *Preprocessor for T.M.* to identify and eliminate the transaction dependencies. The preprocessing is implemented using the Algorithms 5.1 and 5.2. The result of the preprocessing is a BPEL instance denoted as s_2. Then, we declare transaction policies and annotate transactional interpretation rules with respect to s_2. The next step is to obtain the transformed BPEL specification (denoted as $s3$) by applying the Preprocessor for Binding to $s2$ in order to support dynamic binding.

Let us consider an execution within the SeCSE platform after all relevant Web services and tools have been properly deployed and initiated for execution, such as *ActiveBPEL* (a BPEL engine) [28], *Drools* (a rule engine) [31], and the SeCSE related modules: *Binder* (implementing the binding process), the *Transaction Manager*, *JBoss Transaction Server*, the *Event Bus* and *Event Adapter*. Suppose the transactional invoke activity SupplyProduct is executed, the following sequence of events represents a possible successful execution:

1) The Event Adapter captures the event ActivityEnableEvent produced by the BPEL engine, and maps it to the activitiyBindingEvent and the transactionCreatedEvent events.
2) When the Drools rule engine receives the event ActivityEnableEvent from the Event Bus, the predefined binding rule is invoked. During the binding process, the Binder finds the service http://localhost:8084/services/supplierWS and returns it for execution via the operation *setBinding(AbstractService as, Service s)* where AbstractService is the proxy and Service is the concrete service discovered.[2]
3) Then the Drools rule engine receives the event transactionCreatedEvent from the Event Bus, and the rule defined in Figure 8 is invoked. The transaction-related process proceeds as follows:
    a) Drools is invoked for interpreting the predefined transaction rule.
    b) A PolicyOperator module retrieves transaction policies from the XML file, it parses it and it translates it into the appropriate transaction context.
    c) The TransLog module, which is responsible of managing the lifecycle of the transaction, prepares the coordination context with transaction policies, creates an instance transaction entry and transfers the coordination context to the JBoss Transaction Server via the Event-Bus.
    d) JBoss Transaction Server controls the proceeding of the transaction between the participant (http://localhost:8084/services/supplierWS) and the coordinator according to the WS-AT transaction protocol.
    e) When the transaction is successfully committed, the TransLog receives a message from JBoss Transaction Server via the Event-Bus, destroys the transaction instance, and publishes an event of TransactionResultEvent via the Event-Bus.
4) The next activity (DeliverProduct) within the BPEL specification is ready for being invoked.

7 Related Work

The study of transactions is quite old in the history of computer science. Scholars investigated various topics, such as nested transactions [1], [32], transaction processing monitor [33] or using declarative techniques for integrity control [12]. The interest has been mainly motivated by databases applications, but as workflows became a popular way to manage information systems, the problem of handling exceptions and transactions in workflows arose [34].

2. In the SeCSE platform, there is a separate component responsible for service discovery based on quality of service properties.

Bonner et al. [35] propose to solve transaction problems using *Transaction Datalog (TD)*. He identified different types of transactions and exceptions that might occur, and wrote *TD* rules for them. However, the detection and elimination of dependencies among workflow activities is missing, as well as automatic modification of workflows to account for transactions. Other work such as [36], [37] emphasizes on exception handling within a transactional workflow. This is done by identifying the error cases and creating specific defined rollbacks. If on the one hand, this approach is similar to what we proposed, one remarks the limited applicability of the approach and the need to define rollbacks on a per case base rather that generally. In [38], [39] an exception management framework to define policies in a declarative manner is proposed. The methodology is similar to ours, though it lacks the possibility of identification and resolution of dependencies, thus transforming the original workflow. Additionally, the application to the context of Web services is not straightforward.

7.1 Web service transactions

With the shift in interest toward Internet-based applications, Web service transactions have received growing attention from both industry and academia. Unlike traditional transactions, Web service transactions are usually complex, involve multiple parties, spanning over many organizations, and may last over a relatively long time [9]. Various advanced transaction models and architectures for Web services have been proposed [40], and their middleware support [41]. An overview of service transaction behaviours is offered in [42], which is a superset of those used in this paper as requirements. [43] offers a description of possible failures during workflow executions, including transaction failures. Our work does not offer another Web service transaction model or architecture, we rather consider the integration of existing transaction models with composite Web services, having the objective of increasing the reliability of the composition. Similarly to [44], we consider the transactionality of processes and services, not simply data.

Curbera et al.[6] point out that Web services are moving from their initial “describe, publish and interact” capability to a new phase in which robust business interactions are supported. They indicate that transactional (transaction-aware) Web services will be available in the near future and will need to be managed. This need is what we addressed with the present proposal.

As a response to transactional Web services, Vasquez et al. [45] developed an open source middleware that enables Web services to participate in a distributed transaction as prescribed by the WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity specifications. Their work focuses on the implementation of the transactional Web services only from the Web service’s point of view, in particular, on the recovery of Web services in case of failure. Although the architecture of the middleware employs a BPEL engine, it does not deal with the issue of how to process BPEL specifications.

7.2 Declarative approaches and transaction models

Tai et al. [8], [10] present a WS-Policy based method to implement transaction management within BPEL processes. In their model, the coordination services described by WS-Coordination are implemented as a coordination middleware. Coordination participants are a set of Web services, which support not only application specific port types (interfaces), but also coordination middleware interfaces. A declarative coordination policy specifying WS-Coordination types and protocols can be attached to a Web service through its WSDL specification. In the BPEL definitions, the coordination policy is attached to BPEL partner links and scopes. These are correspondingly called coordinated partner links and coordinated scopes, respectively. In order to support the implementation of transactions, the method extends existing Web services to support coordination interfaces, and it changes the original BPEL definitions. This method is similar to the one we propose since both of them are declarative. The difference is that the method we propose is based on the preprocessing and the rule-based annotation mechanism, transaction policies are declared using a specific XML schema, while Tai et al.'s method is based on WS-Policy. In addition, it affects partner links in the BPEL specifications and relies on the Java implementation of BPEL constructs for executing the BPEL composition. Furthermore, our method does not need any modification of existing BPEL engines.

Green and Furniss [46] present a method of extending BPEL for transaction management. The method introduces transaction contexts and coordination contexts as variables of BPEL processes. The set of BPEL actions is extended with transaction activities including 'new', 'confirm' and 'cancel'. At the same time, the method extends compensation handlers with *confirmHandler* and *cancelHandler* operations. In order to support the transaction management in BPEL processes, it is necessary to extend the BPEL language itself, and thus modify the available BPEL engines. It is not clear how the method incorporates WS-Coordination and WS-Transactions into BPEL processes. Till this date, no implementation of this method is reported.

Papazoglou et al. [9] propose a business-aware Web services transaction model and support mechanism, which is driven by common business functions. They focus on cross-enterprise business process automation, and they include quality of service (QoS) information to guarantee integrity of information. The authors distinguish between business transaction and Web service transactions, the latter rely on technical requirements such as coordination, data consistency and recovery; while business transactions depend on economic needs. In fact, their objective is met once a final agreement between parties is made.

7.3 Performance

Recently, Schäfer et al. [47] discuss an approach for dealing with compensation when a service failure occurs during a transaction and propose a heuristic to improve performance. The authors describe an environment for advanced compensation operation adopting *forward recovery* within Web service transactions. The idea is to prevent a rollback when an error occurs at some point in the transaction. Forward recovery proactively changes the state and structure of a transaction after a service failure has occurred, allowing the process to complete successfully. This is done by using a component called *abstract service* that acts as a mediator for compensations. The proposal by Limthanmaphon et al. [48] is on a similar line. It provides a different solution to compensate when a service failure occurs. The authors use the *Tentative Hold and Compensation* approach to allow tentative, non-blocking holds or reservations to be requested for a business resource. By granting non-blocking reservations on their services, the resource owners still keep control of their resources while allowing many potential clients to place their requests, thus minimizing the need for cancellation. The issue of recovery is very important in transaction management, though different in scope from our proposal that has to do with the design and annotation of transactions.

7.4 Technologies

A commonly used method in practice for supporting Web service transactions resorts to generic middleware. Some representative transaction management tools, such as IBM Web services AtomicTransaction for WebSphere Application Server [49], JBoss Web service Transactions [29], and Apache Kandula [50], have employed this method to support distributed Web services atomic transactions. These tools focus on the implementation of transactions within some types of application servers using the Java Transaction API (JTA). JTA provides three main interfaces, namely UserTransaction interface, TransactionManager interface and Transaction interface [51]. These interfaces share transaction operators, such as *commit()*, *rollback()*, *suspend()*, *resume()* and *enlist()*. The application servers act as Transaction Manager, and implement the coordination services described by the WS-Coordination specification. This method is practical and reuses the available generic middleware. Though, it does not take into account transaction management within BPEL processes. In [11], Loechner presents a survey of issues in implementing transactions with service technology and proposes a model-based approach to managing transactions. The proposal is not backed up by any implementation.

8 DISCUSSION AND FUTURE WORK

Web services are being increasingly adopted by organizations in order to run their businesses more effectively and efficiently. However, current technologies lack

TABLE 3
Requirement satisfaction for the proposed approach.

Requirements	Proposed approach
3.1.1 Atomicity	⊕
3.1.2 Consistency	⊕
3.1.3 Isolation	⊕
3.1.4 Durability	⊕
3.2.1 Rollback	⊕
3.2.2 Compensating actions	⊕
3.2.3 Abort	⊕
3.2.4 Adding deadlines	⊙
3.2.5 Logical expressions	⊙
3.3.1 Composite trans.	⊕
3.3.2 Distributed trans.	⊕
3.4.1 Trans. recovery	⊕
3.4.2 Concurrency control	⊕

the support often required by such organizations. The success of Web services lies, among other factors, in their reliability, especially when economic interests are involved. One key feature is that of being able to deal transactionally with a set of operations, but this is far from being easy, especially when the operations in the transaction come from different remote service instances.

In this paper, we highlight the key requirements of transaction management in Service-oriented systems and propose a novel declarative transaction management approach for Web service compositions. The key to implementing transaction management into BPEL processes is to consider the combination of business logic with transactions, taking into account the challenges that make it impossible to directly apply transaction models to all BPEL processes.

The proposal consists of first a preprocessing of the BPEL to identify and manage transaction dependencies among a group of activities. Then it proceeds with the annotation with transaction policies. Finally, the interpretations of the declared transaction policy are specified as event-action-condition rules to be processed at run-time.

Consider again Table 1 where requirements for transactions in Service-oriented systems are listed, one may wonder how our proposed approach performs with respect to these. Basic atomicity and isolation properties (3.1.1,3.1.3) are supported in our approach by the adoption of the underlying transaction protocol such as WS-AT. Similarly for Consistency, we note that Adding deadlines (3.2.4) and Logical expression (3.2.5) currently have no direct implementation in Web service transaction protocols.

Secure transactions of different types (Confidentiality, Integrity, Authentication and Non-repudiation) referring to the fact that participants in a transaction may be authorized and authenticated are beyond the scope of this work and the current implementation does not support it. However, we do not see this as a limitation of the approach, but rather as something to be addressed by resorting to an underlying secure layer, moreover security will be addressed in future work. Table 3 summarizes the analysis.

The proposed methodology has been fully implemented and tested on a number of cases from the automotive industry (Fiat and Daimler-Chrysler who are partners of the SeCSE project [24], [25]). For illustration purpose, here we use a simple example, the drop-dead order nested transaction. This example is simple as it contains only few activities and three partners, but it is explanatory enough as it captures nested and atomic transactions among independent partners. The execution of the platform on the example is also described.

The proposal fills an existing gap of composition languages with respect to transactions. The main advantages are that it integrates with existing BPEL processes (independently of how these were engineered) and it is declarative. If an organization decides to change the transaction policy within one of its processes, then, it simply needs to change a few lines of XML. One can then have a general process with attached many transaction policies. One could even have a different set of policies for each customer interacting with the organization.

The described research opens a number of items for future investigation. First, one would like to automate as much as possible the process of identifying transactions in BPEL instances. This could be achieved, for instance, by mining typical transaction patterns in BPEL processes and their most recurrent translation into a transaction [52]. Another possibility is to use semantic annotations that can allow for fully automatic identification of transactions. Second, we will consider other transaction protocols beyond WS-AtomicTransaction and WS-BusinessActivity when these become available. From the more technological point of view, it is currently hard to achieve coordination among services living in distinct containers, e.g., rolling back operations of a delivery service living in a JBoss container and of a purchase service living in an Apache Tomcat container. Thus, nifty implementations have to be devised to overcome other heterogeneity issues in Web service coordination.

REFERENCES

[1] J. Gray, “The transaction concept: Virtues and limitations,” *Very large Data Base (VLDB)*, vol. 6, pp. 144–154, 1981.
[2] WS-C, “Web Services Coordination (WS-Coordination),” Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies and Microsoft, Tech. Rep., 2007.
[3] WS-AT, “Web Services Atomic Transaction (WS-AtomicTransaction), Version 1.1,” Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies and Microsoft, Tech. Rep., 2007.
[4] WS-BA, “Web Services Business Activity Framework (WS-BusinessActivity), Version 1.1,” Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies and Microsoft, Tech. Rep., 2007.
[5] BPEL, “Business Process Execution Language for Web Services Version 1.1,” IBM, Microsoft, BEAT, SAP and Siebel Systems, Tech. Rep., 2003.
[6] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, “The next step in Web services,” *Communication of the ACM*, vol. 46, pp. 29–34, 2003.

[7] F. Leymann and D. Roller, "Business processes in a Web services world: A quick overview of BPEL4WS," 2002, http://www-128.ibm.com/developerworks/library/ws-bpelwp/.

[8] S. Tai, R. Khalaf, and T. Mikalsen, "Composition of coordinated Web services," *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, vol. 78, no. 5, pp. 294–310, 2004.

[9] M. Papazoglou, "Web services and business transactions," *World Wide Web: Internet and Web Information Systems*, vol. 6, no. 1, pp. 49–91, 2003.

[10] S. Tai, T. Mikalsen, I. Rouvellou, J. Grundler, and O. Zimmermann, "Transactional Web services," in *Service-Oriented Computing*, G. Georgakopoulos and M. Papazoglou, Eds. MIT Press, November 2008.

[11] S. Loecher, "Model-based transaction service configuration for component-based development," in *Component-Based Software Engineering*, vol. LNCS 3054. Springer, 2004, pp. 302–309.

[12] P. W. P. J. Grefen, "Combining theory and practice in integrity control: A declarative approach to the specification of a transaction modification subsystem," in *19th International Conference on Very Large Data Bases*, R. Agrawal, S. Baker, and D. A. Bell, Eds. Morgan Kaufmann, 1993, pp. 581–591.

[13] C. Sun and M. Aiello, "Requirements and evaluation of protocols and tools for transaction management in service centric systems," in *IEEE Int. Ws. on Requirements Engineering For Services at IEEE COMPSAC*, 2007, pp. 461–466.

[14] A. Lazovik, M. Aiello, and M. Papazoglou, "Planning and monitoring the execution of Web service requests," *International Journal on Digital Libraries*, vol. 6, no. 3, pp. 235–246, 2006.

[15] M. Aiello and A. Lazovik, "Monitoring assertion-based business process," *International Journal of Cooperative Information Systems*, vol. 15, no. 3, pp. 359–390, 2006.

[16] B. Haugen and T. Fletcher, "Multi-party electronic business transactions. Version 1.1," UN, Tech. Rep., 2002.

[17] M. Little, "Transactions and web services," *Communication of the ACM*, vol. 46, no. 10, pp. 49–54, 2003.

[18] OASIS, "Business transaction protocol," OASIS, Tech. Rep., 2004.

[19] BPEL, "Business Process Execution Language for Web Services Version 1.1," IBM, Microsoft, BEAT, SAP and Siebel Systems, Tech. Rep., 2003.

[20] C. Sun, D. Hammer, G. Biemolt, and H. Groefsema, "An evaluation of desctiption- and management- standards and languages for Web service transactions," Univ. of Groningen/SeCSE Project, Tech. Rep., 2006.

[21] OASIS, "WS-Security Specification," OASIS, Tech. Rep., 2006.

[22] C. Ouyang, E. Verbeek, W. M. van der Aalst, S. Breutel, M. Dumas, and A. ter Hofstede, "Formal semantics and analysis of control flow in BPEL," *Sci. Comput. Program.*, vol. 67, pp. 162–198, 2007.

[23] G. Chiola, "A reachability graph construction algorithm based on canonicaltransition firing count vectors," in *Petri Nets and Performance Models*, 2001, pp. 113–122.

[24] S. Consortium, "http://www.secse-project.eu/," European Union, Tech. Rep., 2005–2007.

[25] The SECSE Team, "Designing and deploying service-centric systems: The secse way." in *Service Oriented Computing: a look at the Inside (SOC@Inside'07)*, 2007.

[26] Various Authors, "Report on methodological approach to designing service compositions (final), version 4.0 SeCSE A3.D3," ESI, CA and CEFRIEL, Tech. Rep., 2005, http://www.secse-project.eu/.

[27] ——, "Report on methodological approach to design service compositions (v2.0) SeCSE A3.D3.2.b," CEFRIEL Unisannio, Tech. Rep., 2006, http://www.secse-project.eu/.

[28] ActiveBPEL, "Activebpel engine 2.0," 2009, http://www.activebpel.org.

[29] JBoss, "JBoss transaction service 4.2.2 -Web service transactions programmers guide," 2008.

[30] A3.DY, "Preliminary design of transaction management. SeCSE A3.DY," Univ. of Groningen, Tech. Rep., 2006, http://www.secse-project.eu/.

[31] Drools, "Java rule engine," 2006, http://www.drools.org. [Online]. Available: http://www.drools.org

[32] J. Moss, *Transactions: an approach to reliable distributed computing*. MIT Press, 1985.

[33] P. A. Bernstein, "Transaction processing monitors," *Commun. ACM*, vol. 33, no. 11, pp. 75–86, 1990.

[34] G. Alonso, D. Agrawal, A. E. Abbadi, M. Kamath, R. Gunthor, and C. Mohan, "Advanced transaction models in workflow contexts," in *Proc. 12th International Conference on Data Engineering, New Orleans, February 1996.*, 1996, pp. 574–581.

[35] A. J. Bonner, "Workflow, transactions and datalog," in *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 1999, pp. 294–305.

[36] J. Eder and W. Liebhart, "Workflow recovery," in *Conference on Cooperative Information Systems (CoopIS 96)*, 1996, pp. 124–134.

[37] S. R. Van, T. D. Meijler, A. Aerts, D. Hammer, and R. L. Comte, "TREX, workflow transactions by means of exceptions," in *EDBT Ws on Workflow Management Systems*, 1998, pp. 21–26.

[38] L. Zeng, H. Lei, J. jang Jeng, J.-Y. Chung, and B. Benatallah, "Policy-driven exception-management for composite Web services," in *CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 355–363.

[39] A. Erradi, P. Maheshwari, and V. Tosic, "Recovery policies for enhancing Web services reliability," in *ICWS '06: Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 189–196.

[40] J. McGovern, S. Gyagi, S. Stevens, and S. Mathew, *Java Web Services Architecture*. Morgan Kaufmann, 2003.

[41] W. Emmerich, M. Aoyama, and J. Sventek, "The impact of research on middleware technology," *ACM SIGSOFT Software Engineering Notes*, vol. 32, pp. 21–46, 2007.

[42] P. Greenfield, A. Fekete, J. Jang, and D. Kuo, "Compensation is not enough," in *Proceedings of the 7th International Enterprise Distributed Object Computing Conference*, EDOC 2003, pp. 232–239.

[43] D. Kuo, A. Fekete, P. Greenfield, J. Jang, and D. Palmer, "Just what could possibly go wrong in B2B integration?" in *Proceedings of the Workshop on Architectures for Complex Application Integration (WACAI2003) at COMPSAC*. IEEE Computer Society, 2003, p. 544.

[44] A. Portilla, "Providing transactional behavior to services coordination," *VLDB Ph.D. Workshop*, vol. 170, 2006.

[45] I. Vasquez, J. Miller, K. Verma, and A. Sheth, "Openws-transaction: Enabling reliable web service transaction," in *Int. Conf. on Service Oriented Computing (ICSOC05)*, vol. LNCS 3826. Springer, 2005, pp. 490–494.

[46] A. Green and P. Furniss, "BPEL and Business Transaction Management," 2003, submission to OASIS WS-BPEL Tech. Committee.

[47] M. Schäfer, P. Dolog, and W. Nejdl, "An environment for flexible advanced compensations of Web service transactions," *ACM Trans. Web*, vol. 2, no. 2, pp. 1–36, 2008.

[48] B. Limthanmaphon and Y. Zhang, "Web service composition transaction management," in *Australian Computer Society*. Australian Computer Society, Inc, 2004, pp. 171–179.

[49] WS-AT, "Web Services Atomic Transaction for WebSphere Application Server (WS-AT for WAS)," 2003, http://www.alphaworks.ibm.com/tech/wsat. [Online]. Available: http://www.alphaworks.ibm.com/tech/wsat

[50] Apache, "Kandula," 2005, http://ws.apache.org/kandula. [Online]. Available: http://ws.apache.org/kandula

[51] S. Maple, "Distributed transaction with WS-AtomicTransaction an JTA," IBM, Tech. Rep., 2004, http://www-128.ibm.com/developerworks/library/ws-transjta/?ca=dnt-54. [Online]. Available: http://www-128.ibm.com/developerworks/library/ws-transjta/?ca=dnt-54

[52] O. Zimmermann, J. Grundler, S. Tai, and F. Leymann, "Architectural decisions and patterns for transactional workflows in SOA," in *Int. Conf. on Service Oriented Computing (ICSOC07)*, vol. LNCS 4749. Springer, 2007, pp. 81–93.