Decentralized transaction management in web services environment

Nagwa M. El-Makky
Computer and Systems Engg. Dept., Faculty of Engg., Alexandria University, Alexandria, Egypt
nagwamakky@alex.edu.eg

An increasing number of business transactions are being constructed by combining the execution of multiple Web Services (WS). For efficient processing, these web services-based business transactions are allowed to run with relaxed isolation in web service environments. However, some business transactions do require global transactional guarantees, i.e., isolation and atomicity. This paper proposes an optimistic decentralized protocol for transaction management in web service environments that ensures global transactional guarantees for business transactions. The proposed protocol is inspired by the “local atomicity properties” approach for object-oriented databases. Global correctness is achieved through ensuring local atomicity properties of the schedules of web service providers. The proposed protocol avoids exchanging transaction dependencies information among transactions which saves communication cost and preserves security. A key feature of the proposed protocol is that it is compliant with a representative WS transaction standard, namely, the Web Services Transaction (WS-TX) specifications, for easy integration into existing WS transaction systems. The paper concludes with a comparative analysis showing that the proposed protocol ensures global correctness with a lower message complexity than its counterparts.

ازداد التوسع مؤخرا فى تكوين وحدات معاملات الأعمال عن طريق تجميع العديد من خدمات الشبكة العنكبوتية ، ولزيادة كفاءة الخدمة فى بيئات خدمات الشبكة العنكبوتية يسمح غالبا بارتخاء خاصية عزل وحدات معاملات الأعمال أثناء التنفيذ ، و لكن توجد بعض وحدات معاملات الأعمال التى تحتاج بالفعل الى خاصية العزل وغيرها من ضمانات صحة ودقة البيانات، يقترح هذا البحث بروتوكول متفائل لا مركزي لادارة وحدات المعاملات فى بيئات خدمات الشبكة العنكبوتية بما يضمن صحة ودقة البيانات على مستوى بيئة الخدمات. وقد استلهم البروتوكول طريقة الخواص المحلية لدقة وصحة البيانات- المقترحة من قبل للنظم الشيئية -حيث تتحقق دقة وصحة البيانات على مستوى بيئة خدمات الشبكة العنكبوتية عن طريق تحقيق خواص محلية فى كل مانح للخدمات، ويتجنب البروتوكول المقترح ارسال معلومات اعتمادية بين وحدات المعاملات مما يوفر تكلفة الاتصالات ويحافظ على أمن البيانات. ومن الصفات الهامة للبروتوكول المقترح توافقه مع أحد البروتوكولات القياسية لتنسيق وحدات المعاملات فى بيئات خدمات الشبكة العنكبوتية وهو (WS-TX) مما يسهل تكامل البروتوكول المقترح فى نظم وحدات المعاملات القائمة على خدمات الشبكة العنكبوتية ، وينتهى البحث بتحليل مقارن يوضح أن البروتوكول المقترح يحقق صحة ودقة البيانات على مستوى بيئة خدمات الشبكة العنكبوتية بتكلفة اتصالات أقل من نظائره.

Keywords: Web service environments, Transaction management, Local atomicity properties

1. Introduction

An increasing number of business applications are being constructed by combining the execution of multiple web services. Such WS-based integrated applications should guarantee consistent data manipulation and outcome of business processes. However, these applications often involve long-running computations, loosely-coupled systems, and components that do not share data, location, or administration, and it is difficult to incorporate traditional transaction management techniques within such architectures. For example, traditional protocols like the strict two-Phase Locking (2PL) protocol are impractical in web service environments. A web service provider would not accept to lock its local resources for a long time by web service consumers. Also, approaches relying on a centralized transaction manager, as in multi-database systems, are not appropriate since such an assumption is unrealistic in web service environments.

This paper proposes a decentralized optimistic protocol for transaction management in web service environments that addresses both the recoverability and serializability problems. Under the proposed protocol, global correctness is achieved with no need for locking or testing a serialization graph for a cycle. There is no need for incurring communication cost to send local conflicts and partial serialization graphs between transactions. Avoiding communication between transactions also preserves confidentiality of information. No transaction compensation and subsequently no cascading compensation are required since all effects of a transaction are maintained in an intentions list until the commit of the transaction. This ensures recoverability. A key feature of the proposed protocol is that it is designed to be compliant with a representative WS transaction standard, namely, the Web Services Transaction (WS-TX) specifications [1-3] for easy integration into existing WS transaction systems.

The proposed protocol is inspired by the "local atomicity properties" approach proposed in [4, 5] for transaction management in object-oriented databases. Using the proposed protocol, a scheduler for each web service provider serializes its local transactions depending on its local information. However, by satisfying the same local atomicity property by all schedulers, the global schedule of the system is proved to be correct. The idea is new in the sense that it combines known concepts and techniques for a new purpose.

The rest of the paper is divided into seven sections. In section 2, the related work is reviewed. The idea of the proposed protocol is presented in section 3. Since the proposed protocol is designed to be compliant with the Web Services Transaction (WS-TX) specifications [1-3], these specifications are briefly reviewed in section 4. The proposed protocol is described in detail in section 5. The global correctness of the proposed protocol is proved in section 6. A comparative analysis of the proposed protocol and its counterparts is included in section 7. Finally, section 8 concludes the paper.

2. Related work

There have been industrial proposals for protocols to extend the web services with transaction processing capabilities, e.g., Business Transactions Protocol (BTP) [6],(WS-TX) specifications [1-3], and WS-Composite Application Framework (WS-CAF) [7]. For efficient processing, these protocols relax the isolation property for long-lived business transactions. This means that some activities in a business transaction can commit their results before the whole transaction commits and the results of some activities can be seen before the whole transaction completes. However, atomic execution of the transaction, in case that it fails before getting to the complete phase, can be ensured by cancelling the running operations and compensating the completed ones.

Nevertheless, some business transactions do require global transactional guarantees, i.e., isolation and atomicity. Relaxing isolation can introduce serious inconsistency problems for some business applications. Consider a situation where a participant aborts its transaction after releasing a resource and assume that other participants have already read this resource and completed their own transactions based on this reading. Such a situation implies that different participants hold different states of the same resource, resulting in possibly serious inconsistency problems. In fact, all the known anomalies of "dirty reads", "unrepeatable reads" and "phantoms" can happen in such environments. Examples of such anomalies in web service environments can be found in [8. and 9]. Considering that WS transactions will get more and more prevalent, such situations may occur quite frequently, being a major blocking factor for the WS transaction usage.

There are some academic proposals that address the problems of relaxing isolation for web services-based business transactions, e.g. [8-10 and 11]. The work in [8] addresses only the recoverability problem by managing the completion of dependent transactions. The proposals in [9 and 10] address both the recoverability and serializability problems using serialization graph testing protocols. The proposals in [8 and 9] are made as extensions to a previous version of the Web Services Transaction protocols [1-3], whereas

[10] proposes a new protocol that can be applied in peer-to-peer environments. The work in [11] extends [9] with a non-blocking scheduling scheme to support time constraints of transactions in WS environments.

While these academic proposals are important, they depend on direct (or indirect) communication between transactions to achieve global transactional guarantees. The proposals in [8 and 10] allow sharing transaction dependencies information between the transactions, which does not preserve security. Also, all the above proposals require testing a decentralized graph for a cycle, which adds the cost of propagating conflicts along the edges of local graphs between transactions to the cost of cycle checking and resolution. Moreover, all proposals use transaction compensation. Compensation of some already successfully executed services is sometimes too expensive and often leads to cascading compensation which may result in long and costly replacements of concrete web services, making web service transactions too long.

3. Idea of the proposed protocol

The proposed protocol is inspired by the "local atomicity properties" approach for object-oriented databases in [4, 5]. A local atomicity property is defined in [4, 5] as a property that guarantees, if every object in the system obeys this property, that every schedule in the system behaviour is atomic. This is achieved using only local information with no need for inter-object synchronization. The work in [4, 5] identified three optimal local atomicity properties such that no strictly weaker local constraint on objects suffices to ensure global serializability for transactions. These three properties are dynamic atomicity, static atomicity and hybrid atomicity. Static and hybrid atomicity properties require clock synchronization which can be a problem in web service environments. So, we will use the dynamic atomicity property for the proposed protocol.

The definition of dynamic atomicity states that an object schedule is dynamic atomic if the set of committed transactions in the schedule is serializable in every total order consistent with the precedence relation at the object [4, 5]. The precedence relation is defined as follows: a transaction T_1 is said to precede a transaction T_2 at an object if at least one operation of T_2 is invoked on the object after T_1 commits. If all object schedules satisfy the dynamic atomicity property, then the global schedule is serializable. This aspect is relevant in the context of object-oriented systems, since serializability can now be localized to objects. Another advantage of using dynamic atomicity is that, as long as the objects involved in a transaction guarantee dynamic atomicity, it is irrelevant what specific concurrency control algorithm is used by each object to achieve this behaviour.

The proposed protocol recognizes the similarity between the model of a business transaction that spans multiple web services and the model used in [4, 5] for object-oriented transaction processing systems, where a transaction is a computation involving operations on multiple objects. Each web service provider provides one or more operations as web services that can be invoked within transactions using the service interface of that provider. The sequence of web services invoked by a transaction on a web service provider in a web service environment is analogous to the sequence of operations invoked by a transaction on a particular object in an object-oriented database. In the work of [4, 5], global serializability of the system is achieved by satisfying atomicity properties that are local to individual objects. By analogy, if each of the web service providers can satisfy the same local atomicity property, then it would be possible to have a decentralized transaction management protocol for web service environments. Global serializability will be achieved with no need for a global transaction manager or for sending transaction dependencies information among the transactions.

The proposed protocol introduces a local scheduler component in each web service provider to ensure that local schedules satisfy the same local atomicity property. The dynamic atomicity property is chosen as the local atomicity property to be satisfied by all web service providers. By analogy to the model in [4, 5], it follows that as long as the service providers involved in a transaction guarantee dynamic atomicity, it is irrelevant what specific concurrency control algorithm (pessimistic or optimistic) is used to achieve this behaviour. Satisfying a dynamic atomicity property by a service provider depends only on the characteristics of its supported web services, namely the conflict between these web services. Detecting conflicts at each service invocation, using a pessimistic algorithm, e.g. locking, will be costly for a web service environment. So the proposed protocol uses an optimistic algorithm at each service provider. Web services are checked for conflicts only at transaction commit time. This can be done in a simple validation step by the local scheduler at each web service provider.

Global correctness requires an atomic termination for each transaction on all accessed objects [4, 5]. To achieve a similar behaviour, the proposed protocol applies an atomic commit protocol that is already supported by the WS- Business Activity model [2].

Therefore, by making each web service provider obey a dynamic atomicity property, and by using an atomic commit protocol, the proposed protocol ensures that every global schedule in the system's behaviour is correct. A formal proof of the global correctness of the proposed protocol is given in section 6.

Since the proposed protocol is designed to be compliant with the. WS-TX specifications [1-3], these specifications are briefly reviewed in the following section.

4. Web Services Transaction (WS-TX) specifications

The WS-TX specifications [1-3] define an extensible framework that is aimed at coordinating transactions running across multiple web services. Two key concepts are defined: 1) the Coordinator, which is an entity that resides on the client side and is responsible for reaching a globally agreed upon outcome of the transaction from the client's point of view, 2) the Participant, which is an entity that resides on the web service provider side and is responsible for communicating with the coordinator according to the protocol on behalf of the web service.

The coordinator consists of the services given below (which are all provided as web services).

- Activation service: operations that enable an application to create a coordination instance or context.
- Registration service: operations that enable an application and participants to register for coordination protocols.
- Protocol service: a set of coordination protocols. The coordination protocols that can be defined in this framework can accommodate a wide variety of activities, including protocols for simple short-lived operations, e.g., WS-Atomic Transaction [1] and protocols for complex long-lived business activities, e.g., WS- Business Activity [2].

The specifications in [2] define two specific agreement coordination protocols for the Business Activity transaction model: Business Agreement with Coordinator Completion and Business Agreement with Participant Completion. Developers can use these protocols when building applications that require consistent agreement on the outcome of long-running distributed activities. In the former protocol, the participants rely on the coordinator to inform them when they have received all requests to perform work within the business activity, whereas in the latter one, the participants themselves know when they have completed all requests and should inform the coordinator about that.

Due to the extensibility of WS-Coordination [3], it is possible to define a coordination protocol type that, in addition to specifying the agreement protocol between a coordinator and a participant, also specifies the behaviour of the coordination logic. For example, it may specify that the coordinator will act in an all-or-nothing manner to determine its outcome based on the outcomes communicated by its participants, or that it will use a specific majority rule when determining its final outcome based on the outcomes of its participants. Business activities support the following coordination types: Atomic Outcome and Mixed Outcome. An Atomic Outcome coordination type must direct all participants to close or all participants to compensate. A coordinator for a Mixed Outcome coordination type may direct each individual participant to close or compensate.

This section reviewed the Web Services Transaction specifications. The next section will describe the proposed protocol in detail.

5. Proposed protocol

5.1. System model

Assume a web service environment, consisting of a set of web service providers and a set of service consumers (clients). Each service provider offers one or more operations as web services. A transaction can consist of multiple invocations on a collection of web services that may reside on several web service providers. The proposed protocol is designed for the transactions following the Business Activity model [2] of the WS-TX specifications [1-3]. A client who wants to create a new WS -transaction has to request and receive a coordination context from its coordinator using the Activation service. The client notifies all the participants of the WS transaction by forwarding the coordination context to them. Then, all the participants and the client (WS- transaction initiator) register their own coordination protocols, which are specified in the Business Activity specifications, to the coordinator using the Registration service. The coordinator manages the transaction by exchanging messages with the transaction initiator and the participants via the protocol service. To use the service provided by the proposed protocol, the participants and initiator have to select the Atomic Outcome coordination type and the Business Agreement with Coordinator Completion protocol.

As mentioned in section 3, the proposed protocol introduces a local scheduler component. The local scheduler is an entity that resides on the web service provider side and interacts with transactions' participants at the service provider to satisfy the local atomicity properties. The details of these interactions are given in the next subsection. Fig. 1 adapts the infrastructure of the standard WS- coordination [3] by including the proposed local scheduler component.

Formally, a transaction T will be modelled by the pair $(O_T, <_T)$, where O_T is a set of web services to be invoked by T and $<_T$ is a partial order defined over O_T. A local schedule S_A at a web service provider A is a pair $(O_{SA}, <_{SA})$, where O_{SA} is the set of web service invocations on A and $<_{SA}$ is the order of these invocations. A global schedule S is a pair $(O_S, <_S)$, where O_S is a set of all web service invocations in the web service environment and $<_S$ is the order of these invocations.

For schedules' correctness, the local schedulers rely on a conflict- preserving serializability criterion. There are many conflict serializability-based correctness criteria that basically differ in how they define a conflict. For increasing concurrency, the proposed protocol uses the forward commutativity correctness criterion that can make use of the semantics of operations and their termination conditions [12]. It can be easily adapted to the case of semantically rich web services. In what follows, the notion of conflict is defined based on the commutativity behaviour of web service invocations.

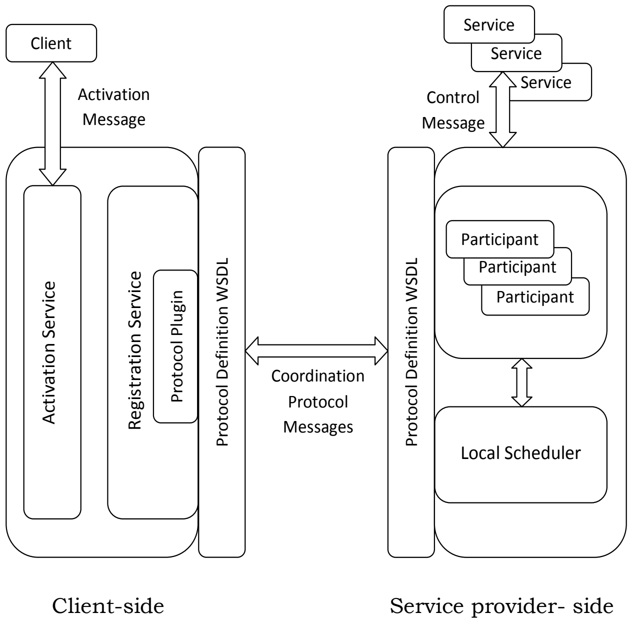

Fig.1. The adapted infrastructure of the standard WS-coordination .

Definition 1: Let ws_1 and ws_2 be two web service invocations on a service provider. Then," ws_1 forward-commutes-with ws_2" if for any pair of sequences α, β of web service invocations on the service provider, the results (return values) are the same in $< \alpha, ws_1, ws_2, \beta >$ and $< \alpha, ws_2, ws_1, \beta>$ Otherwise, ws_1 conflicts with ws_2 (i.e., "does – not-forward – commute – with" it), which makes the pair (ws_1, ws_2) belong to the conflict relation CR of the service provider. The conflict relation of a web service provider is a subset of the cross product: $\{ws_1, ws_2, ..., ws_N\} \times \{ws_1, ws_2, ..., ws_N\}$, where N is the number of web services supported by this service provider.

The local conflict relation of a service provider can be represented by an N*N conflict matrix which is built at design time and contains information about which web services of that provider pair-wise conflict. It is provided as input to the local scheduler to be used for validating transactions.

5.2. Protocol details

The design goals of the proposed protocol were to ensure decentralization, low message complexity, security and compliance with standard specifications for web services transactions. These are common and core requirements in web services environments. Decentralization is achieved by enforcing globally serializable schedules using local atomicity properties of web service providers without relying on a global transaction manager. With local atomicity properties, there is no need for inter-scheduler synchronization or transaction communications which lowers message complexity. Also, transactions dependency information is not required to be exchanged between transactions or transactions' coordinators. Instead, such information is to be stored at distributed web service providers, where each service provider manages only its local transactions dependencies. This preserves security since transactions dependencies can be interpreted as mission-critical information (e.g., confidential contracts between organizations). Regarding the fourth goal, the proposed protocol is designed as an extension to the standard WS- Business Activity Protocol [2]. This is possible because the WS-TX specification is intended as a portfolio of extended transaction models each suited for a specific problem domain. The proposed protocol does not require changes to the standard WS- Business Activity protocol. This allows easy integration into existing WS transaction systems. In what follows, details of the proposed protocol are presented.

The proposed protocol utilizes a dynamic atomicity property, to work in a decentralized optimistic fashion. Each local scheduler at a web service provider can schedule its local transactions by any deferred –update optimistic algorithm that uses a conflict relation based on “does-not-forward-commute-with”. As will be proved in section 6, all such deferred-update optimistic algorithms are dynamic atomic. Using deferred - update, requires the local scheduler to store a set of intentions lists that record the tentative changes of each active transaction invoking web service(s) at the corresponding service provider. When a transaction commits, its intentions list is applied to the permanent state of the service provider. A transaction aborts by discarding its intentions list. The Business Activity model [2] already allows participants in a coordinated business activity to perform "tentative" operations as a normal part of the activity. This feature is utilized by the proposed protocol to support intentions lists.

After each web service invocation, the corresponding participant informs the local scheduler about this action to perform the required bookkeeping. When the participant receives the complete message from its coordinator, it asks the local scheduler for validation. Enabling the service providers' schedulers to locally validate transactions requires equipping each scheduler with its local conflict relation (matrix). Validation ensures that the committing transaction has not been invalidated by the recent commit of another transaction. The used validation algorithm is an adaptation to the algorithm in [13] for the case of web service environments. It works in the following way. The scheduler of each web service provider keeps track of Last (ws_k), the most recent commit timestamp for a transaction that invocated the web service ws_k at this service provider. For each active transaction T and each web service ws_i, each scheduler also keeps track of First (T, ws_i), the logical time when T first invoked the web service ws_i. Validation is governed by the conflict relation CR kept by the local scheduler of the service provider. A scheduler will validate a transaction T if and only if Last (ws_k) < First (T, ws_i) for all ws_i in the intentions list of T and all ws_k such that (ws_i, ws_k) belongs to CR.

A "completed" message or a "cannot complete" message is sent to the coordinator in case of successful validation or invalidation, respectively. A transaction can commit if and only if all its participants are successfully validated (completed) at the involved web service providers. This can be ensured by the Atomic Outcome protocol supported by the WS-Business Activity specification [2]. In case of successful validation, a "close" message is sent by the coordinator to all participants, otherwise a "compensate" message is sent. Algorithm 1 gives a pseudo code for the main part of the protocol that runs on each web service provider. It is concerned with the response to messages received from transactions' coordinators. The protocol that runs on each service provider reacts on received messages as described below.

- In case of a service invocation, the required bookkeeping is done by the local scheduler, the service is executed and the tentative changes are kept in the invoking transaction's intentions list. It is to be noted that the proposed protocol is optimistic, so there is no check of web service conflicts at that time.
- In case of a "complete" message from the coordinator of transaction T, the participant requests validation from the local scheduler of the service provider. According to the response it receives from the scheduler, it responds to the coordinator either by a "completed" or a "cannot complete" message. Algorithm 2 describes the local validation process for a transaction T.
- The other cases represent messages from the coordinator to complete the atomic commitment protocol supported by the WS-Business Activity model [2]. It is to be noted that the received messages are the same standard messages of the Business Agreement with Coordinator Completion protocol specified in [2]. Fig. 2 shows the abstract state diagram of this protocol with possible web service states and messages generated either by a transaction coordinator or a participant.

Algorithm 1 Service Provider Protocol

```
while true do
  wait for next message m ;
   case message m of
     invocation of web service $ws_i$ by transaction T:
       record First (T, $ws_i$);
       execute web service $ws_i$ ;
       keep the tentative changes in intentions list of T;
     cancel :
       discard T's intentions list ;
       send message Cancelled;
     complete :
       validate T// see algorithm 2;
     close :
       update Last($ws_i$) for each $ws_i$ in T's intentions list ;
       apply T's intentions list;
       send message Closed;
     compensate :
       discard T's intentions list ;
       send message Compensated;
     exited:
       discard T's intentions list ;
     failed :
       discard T's intentions list ;
     not completed :
       discard T's intentions list ;
```

Algorithm 2 Validating a Transaction T

```
Mark T as "validated";
for each $ws_i$ in intentions list of T do
   for each $ws_k$ such that ($ws_i$, $ws_k$) is in CR do
       if ( Last($ws_k$) > First(T, $ws_i$)) then
       mark T as "invalidated" and exit;

if T is marked as "invalidated"  then
   send message "CannotComplete" ;

else
   send message "Completed";

wait for the next message from coordinator;
```

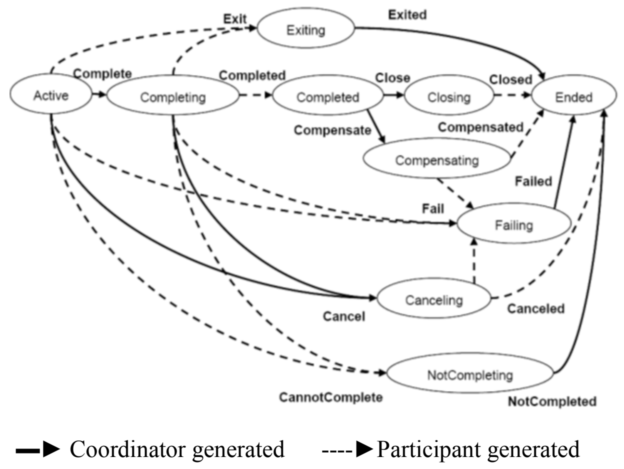

Fig. 2. Abstract state diagram of the business agreement with coordinator completion protocol (adapted from [2]).

6. Correctness proof

This section proves that the proposed protocol guarantees globally correct schedules for a web service environment. First, the global srializability of schedules is proved, and then the schedules are proved to be recoverable.

The following proves that if all schedulers at the web service providers satisfy a dynamic atomicity property and an atomic commit protocol is used, then the global schedule is serializable. Then, it will be proved that if all schedulers use deferred-update optimistic algorithms with a conflict relation based on "does-not-forward- commute- with", then all of them satisfy the dynamic atomicity property. It is to be noted that the definitions and theorems given below are inspired by the work in [4, 5].

By analogy to the work in [4, 5], global serializability can be proved if there is a global serialization order that is consistent with all local serialization orders produced by the web service providers. So, it is required to let all local schedules agree on some ordering of the transactions. The following defines the precedence relation for a web service environment. This relation can be used to define a partial order among transactions, which all local schedules can agree upon, and can also be used as the basis of the dynamic atomicity property.

Definition 2: The precedence relation for a web service environment is defined as follows: a transaction T_1 is said to precede a transaction T_2 at some web service provider site if the participant of T_2 at that site invokes some web service after the service provider has received the commit message from T_1's coordinator. It is clear that if T_1 precedes T_2 at some service provider site, then T_1 must be before T_2 in the global commit order. However the converse is not true. If T_1 and T_2 do not conflict, T_1 might be before T_2 in the global commit order, but not related by the precedence relation to T_2 (all invocations of web services by both transactions might have been done before either commits). Because a transaction cannot invoke any additional web services after it commits, it can never be true that both T_1 precedes T_2 and T_2 precedes T_1 at one service provider; hence the precedence relation is a partial order at each service provider.

It is to be noted that regardless of the scheduler type, transactions that have conflicting web services at some service provider are related by the precedence relation at that provider site (since if T_1 conflicts with T_2 in a pessimistic system, then T_1 must wait for T_2 to commit, while in an optimistic system, T_1 cannot successfully validate unless T_2 has committed prior to the time at which T_1 has invoked a conflicting web service). Furthermore, the serialization order imposed by the conflict is the same as the precedence relation order at that server provider site.

Definition 3: A scheduler is dynamic atomic if it serializes in every total order consistent with the precedence relation [4-5].

Theorem 1: If all the schedulers of the web service providers independently satisfy the dynamic atomicity property and an atomic commit protocol is used, then the global schedule is serializable.

Proof: Assume that each scheduler at a web service provider site satisfies the dynamic atomicity property, i.e., it serializes in every total order consistent with the precedence relation at this site. It is only required to show that the union of the precedence relations at all sites is a partial order. Hence there must be at least one total order, O, that is consistent with the precedence relation at each site. The global schedule is equivalent to the serial schedule based on O. To show that the union of the precedence relations at all sites is a partial order, it will be shown that assuming an atomic commit protocol, it cannot be the case that T_2 precedes T_1 at some site A, and T_1 precedes T_2 at some other site B.

Let the participant of a transaction T on a web service provider A be called T_A. If T_2 precedes T_1 at site A, it follows from the ordering of events in the commit protocol that "T_2's commit event at its coordinator" happened- before" at least one web service invoked by the participant T_{1A}" and that "all web services invoked by T_{1A} "happened-before" T_1's commit event at its coordinator". If it were also true that T_1 precedes T_2 at site B, the opposite conclusion can be drawn as well. Since "happened- before" is a partial order, there is a contradiction, and it can be concluded that the union of the precedence relations at all sites is a partial order. Hence, there is at least one total order consistent with this partial order. All such total orders must be consistent with the ordering of participants imposed by the conflict relation at each site. Hence, any such total order can be used as the basis of a serial schedule, equivalent to the global serializable schedule.

From the above, it follows that if the schedulers at each web service provider all independently satisfy the dynamic atomicity property and an atomic commit protocol is used, then the global schedule is serializable.

Theorem 2: all deferred –update optimistic schedulers that use a conflict relation based on "does-not-forward-commute-with" are dynamic atomic.

Proof: Suppose T_{1A} and T_{2A} are participants (of two transactions T_1 and T_2) at web service provider A. Assume that the scheduler of A uses a deferred- update optimistic algorithm, with a conflict relation based on "does-not-forward-commute-with", to produce the interleaved local schedule S_A. Suppose that T_{1A} and T_{2A} invoked web services ws_1 and ws_2, respectively; that ws_1 follows ws_2 in S_A; and that ws_1 "does-not-forward-commute-with" ws_2. Then, T_{1A} must follow T_{2A} in any serial schedule equivalent to S_A. Either T_{1A} and T_{2A} are concurrently active or they are not. If they are not concurrently active, then T_{1A} will not be validated unless the commit message for T_{2A} has arrived at A before T_{1A} has invoked ws_1. Hence, T_2 precedes T_1 at site A. If T_{1A} and T_{2A} are concurrently active, then T_{1A} will be invalidated and restarted, and it is only the web services of the restarted version that are being related by the precedence relation. Thus, if conflicting web service invocations constrain T_{1A} to follow T_{2A} in any serial schedule equivalent to S_A, T_{1A} is also constrained to follow T_{2A} in any serial schedule consistent with the precedence relation. Thus, the precedence relation imposes on the ordering of participants, in a serial schedule equivalent to S_A, all the constraints that are imposed by conflicts between web service invocations of the participants at A. Hence, any serial schedule consistent with the precedence relation orders conflicting web service invocations in the same way as they are ordered in schedule S_A. Furthermore, the used deferred-update optimistic algorithm employs “does-not-forward-commute-with” as a conflict relation. This relation is based on commutativity of web service invocations, so that if two web service invocations do not conflict, they can be placed in either order in an equivalent serial schedule. Hence, S_A is equivalent to any serial schedule of participants at site A that preserves the order of conflicting web service invocations. It is therefore equivalent to any serial schedule of participants at site A that is consistent with the union of the precedence relations.

It was proved before that (1) the union of the precedence relations at each site is a partial order and hence there is at least one total order consistent with it and (2) all such total orders must be consistent with the ordering of participants imposed by the conflict relation at each site. Hence, any such total order can be used as the basis of a serial schedule equivalent to the global schedule.

It follows that all deferred- update optimistic schedulers that use a conflict relation based on “does-not-forward-commute-with” are dynamic atomic. Therefore, as long as all schedulers of web service providers use deferred- update optimistic algorithms with a conflict relation based on "does-not-forward-commute-with", and the atomic commit protocol supported by the WS-Business Activity model [2] is used, one can be assured that all produced schedules are globally serializable.

Finally, since deferred- update is used, it follows, by definition [14], that the produced schedules are cascadelss (i.e., avoid cascading rollback). Cascadeless schedules are a stricter subset of recoverable schedules, which means that the produced schedules are not only globally serializable but also recoverable.

7. Comparative analysis

As mentioned in section 1, the inconsistency problems resulting from relaxing isolation for business transactions in web service environments were addressed in [8, 9 and 10]. The proposed protocol will be compared to the proposals in [9 and 10], since they address both the recoverability and serializability problems, while the proposal in [8] addresses only the recoverability problem.

The proposed protocol will be compared to the other solutions in terms of message complexity. Message complexity is selected as the performance metric for cost measuring because communication overhead is a dominant factor that affects the overall system performance in a web service environment, as compared with processing speed and storage space. Message complexity will be measured by the number of messages used by the protocol. This is reasonable if individual messages are short. This is the case for the proposed protocol. In particular, this measure is harsh to the proposed protocol since the other solutions have longer messages as will be seen in the following paragraphs.

Using the protocol in [10], dependencies between transactions are managed by the transactions themselves. Therefore, with each web service invocation the corresponding service provider sends the invoking transaction a complete list of conflicts that have occurred with this invocation. Also, when a transaction wants to commit, it informs all service providers on which it has invoked services. Each service provider sends the transaction a list of its post-ordered transactions.

Therefore, it is required to exchange 3P messages (where P is the number of distinct web service providers the transaction invoked services on), between the transaction and the service providers to deliver transaction dependency information.

For validating transactions, this protocol uses a graph cycle checking protocol relying on communicating dependency information between transactions. The used protocol is a variant of the path- pushing approach for distributed deadlock detection [15]. The message complexity of the later approach is known to be $2n^2$, where n is the number of active transactions in the environment. There is also a message complexity of $O(nn_D^2)$ for cycle resolution when a cycle is detected, where n_D is the size of graph cycle [16].

The solution in [9] is similar to that proposed in [10]. The main difference is that it does not allow direct communication between transactions or transaction coordinators (in order to preserve security). Instead, it replaces each single direct communication between two transaction coordinators C_1 and C_2 by two indirect messages: one message from C_1 to the common service provider, and another message from the common service provider to C_2. Therefore, compared to the work in [10], it requires two times the number of exchanged messages between transactions to reach a globally correct solution.

Compared to the previous protocols, the proposed protocol does not rely on delivering dependency or validation information between transactions (or transaction coordinators). Local atomicity properties allow independent validation of each transaction using local information at each service provider. Therefore, the proposed protocol has a message complexity of 3P messages (where P again is the number of distinct web service providers the transaction invoked services on). This is the message cost for applying the atomic commit protocol supported by the WS-Business Activity model [2]. It is clear that the proposed protocol reduces the number of messages required to ensure global correctness.

The protocols in [9 and 10] are optimistic (like the proposed protocol), but they use variants of the distributed serialization graph testing protocol. Therefore, they may achieve a higher degree of concurrency than the proposed protocol. However, this is achieved at a higher cost for transferring dependency information to maintain the serialization graph and for detecting and resolving cycles in it. Knowing that a transaction should not validate until it has been checked that it is not involved in a serialization graph cycle; global cycle detection must take place at least at the same rate as transactions are validated. In typical applications, the cost to do this may be prohibitive. Moreover, the serialization graph will not necessarily be always up-to-date since the cost of its synchronous maintenance is prohibitive as acknowledged in [10]. This implies that correctness is not guaranteed all the time.

8. Conclusions

This paper proposes a decentralized transaction management protocol for web service environments. The protocol is inspired by the "local atomicity properties" approach of transaction management in object-oriented databases. Global correctness can be achieved using local information at each web service provider. This avoids exchanging dependencies information between transactions (or transaction coordinators) which saves communication cost and preserves security. For easy integration into existing WS transaction systems, the proposed protocol is designed as an extension to a representative WS transaction standard, namely, the WS-TX specifications [1-3]. The proposed extension to the web service provider is simple to achieve and there is no need to change the standard messages of the original Web Services Transaction protocols.

The paper gives a formal proof for the global correctness of the proposed protocol. Message complexity of the protocol, in terms of number of messages, is presented together with message complexity of related protocols. A comparative analysis shows that the proposed protocol reduces the number of messages required to ensure global correctness. As a future work, it is planned to perform a detailed performance evaluation study of the proposed protocol compared to its counterparts for web service environments.

References

[1] Web Services Atomic Transaction (WS-Atomic Transaction) Version 1.2, Committee Specification 01, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec.pdf (last access date: 2008-10-17) (2008).

[2] Web Services Business Activity (WS-Business Activity) version 1.2, Committee Specification 01, http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-cs-01.pdf (last access date: 2008-10-17) (2008).

[3] Web Services Coordination (WS-Coordination), Version 1.2, Committee Specification 01, http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01.pdf (last access date: 2008-10-17) (2008).

[4] W.E. Weihl. Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types. ACM Transaction on Programming Language Systems, Vol. 11 (2) (1989).

[5] M. Herlihy and W.E. Weihl. Hybrid Concurrency Control for Abstract Data Types", In Journal of Computer and System Sciences, Vol. 43, pp. 25-61 (1991).

[6] OASIS Business Transaction Protocol. http://xml.coverpages.org/BTPv11-200411.pdf (last access date: 2008-10-17).

[7] OASIS Web Services Composite Application Framework (WS-CAF), OASIS Standard, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf. (last access date: 2008-10-17) (2005).

[8] S. Choi, H. Jang, H. Kim, J. Kim, S. Kim, J. Song and Y. Lee, "Maintaining Consistency Under Isolation Relaxation of Web Services Transactions", In Proc. of WISE 2005 and Springer-Verlag Berline Heidelberg (2005).

[9] M. Alrifai, P. Dolog and W. Nejdl, Transactions Concurrency Control in Web Service Environment", In Proc. of the

fourth European Conference on Web Services (ECOWS'06), IEEE Press (2006).

[10] K. Haller, H. Schuldt, and C. Turker, "Decentralized Coordination of Transactional Processes in Peer to Peer Environments", In Proc. of the 14th ACM International Conference on Information and Knowledge Management (CIKM 2005), pp. 36-43, Bremen, Germany (2005).

[11] M. Alrifai, W-T. Balke, P. Dolog and W. Nejdl, "Non-Blocking Scheduling for Web Service Transactions", In Proc. of the Fifth European Conference on Web Services (ECOWS'07), IEEE Press (2007).

[12] R. Vigralek, H. Hasse-Ye, Y. Breitbart and H-J. Schek, "Unifying Concurrency Control and Recovery of Transactions with Semantically Rich Operations", Theoretical Computer Science, Vol. 190, (2) (1998).

[13] M. Herlihy, "Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types", ACM Transaction on Database Systems, Vol. 15 (1) (1990).

[14] A. Silberschatz, H. Korth and S. Sudarshan, Database System Concepts, Fifth Edition, McGraw-Hill (2006).

[15] R. Obermarck, "Distributed Deadlock Detection Algorithm", ACM Transaction on Database Systems, Vol. 7 (2) (1982).

[16] S. Lee and J.L. Kim, "Performance Analysis of Distributed Deadlock Detection Algorithms", IEEE Transaction on Knowledge and Data Engineering, Vol. 13 (4) (2001).

Received November 12, 2008
Accepted December 24, 2008