JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 261-287 (2018)
DOI: 10.6688/JISE.2018.34.1.16

Concurrency Control of Real-Time Web Service Transactions

DE-PENG DANG[1], XUE JIANG[1], NAN WANG[1],
YING-TING YAO[1] AND WEN-BIN YAO[2]
[1]*College of Information Science and Technology*
Beijing Normal University
Beijing, 100875 P.R. China
[2]*Beijing Key Laboratory of Intelligent Communication Software and Multimedia*
Beijing, 100875 P.R. China
E-mail: ddepeng@bnu.edu.cn; jiangxue@ustb.edu.cn; xiao_wangnan@163.com;
yyingting@126.com; yaowenbin@bupt.edu.cn

Transactional property is crucial to the success of web service composite applications. In the dynamic and open web service environment, maintaining consistency is a new challenge. In this study, we propose a flexible web service transaction model FleWeSeT, which supports replaceability and compensation. We expand the WS-Frame framework and add various components to support the transaction-dependence detection for transactions in FleWeSeT. Furthermore, we propose the optimistic verified concurrency control protocol ACoDe based on the transaction-dependence detection. Finally, we establish a randomly colored Petri net model and perform analysis and evaluation of the ACoDe protocol. This study's experimental results indicate that the protocol can help more transactions meet their deadlines and significantly improve the average response time of transactions and the success rate of the web service.

Keywords: web service transaction, transaction framework, concurrency control, performance evaluation, service computing

1. INTRODUCTION

The number of web services posted on the Internet is growing rapidly. Especially with the explosive development in the fields of e-commerce and e-government, inter-enterprise and Internet-based service composition has become a prevailing trend [1-3]. The beneficiaries of web service composition are not only value-added service providers who exploit new customized services through the composition of some existing services, but also service consumers who receive on-demand services or benefit from web service composition systems by combined services. Conversely, Service is regarded as the "link" and "binder" between infrastructure capabilities and ubiquitous user experiences [6, 7]. The operation mode "Anything as a Service," which supports consumption and utilization of resources for users, is a way of "using rather than owning" and pay-on-demand. Therefore, in the future of the Internet, there will be a phenomenon of "sea serving," that is, simple services that read-write on data objects and complex services such as journey reservation [8-10]; vast numbers of equivalent services provide rich selections for users by using different service quality indicators. Moreover, a flexible service invocation model can call the most appropriate service as needed on the Internet [10, 11].

However, web services are faced with a widely distributed and heterogeneous environment, and relying only on these basic functions, it is still difficult to accurately char-

Received April 7, 2016; revised October 16, 2016 & February 4 & March 1, 2017; accepted March 9, 2017.
Communicated by Haiying Shen.

acterize complicated business processes; most real-world business processes require cooperative work to achieve a specific target [5, 6]. We urgently need to composite existing web services to build a value-added web service software structure [7, 8]. Because of stateless calls among web services (*i.e.*, there is no conversation among web services), the process of web service transactions has become the key to the composition of web service applications [9, 10, 12].

Actually, many modern web service systems are dynamic and real-time. Meeting time constraints is very important in some business processes [9, 13], such as burst exigency-based service systems (*i.e.*, ice disasters, earthquakes, and security) [14, 15]; battlefield information management and command service systems [16, 17]; traffic information management and scheduling service systems [8, 18]; location-related query service systems [8, 18]; (mobile) stock trading service systems [14, 18]; (mobile) auction-based service systems [18, 19]; (mobile) e-commerce [9, 21]; (mobile) e-health systems [22-24] and so on. If stock exchange requests from users cannot be completed before the deadline, severe financial losses may result. Moreover, data involved in these systems, such as the stock price, auction bidding and test data, are valid only within a certain range of time, and decisions or inferences are meaningless over time. Services requested in these systems require access to the latest and consistent data at all times and places; otherwise, wrong judgments and decisions will ensue [21, 25].

For software application systems based on web service composition, concurrent execution of web services with multiple granularities and timely updates of the data are required. This contributes to reliable and consistent service [26, 27]. Existing web services cannot take effective measures to alternate and concurrently execute web service transactions, and this has become a serious obstacle to the development of web service technology [27, 28].

Concurrency control of real-time web service transactions is a new problem. In this paper, we propose a flexible web service transaction model called FleWeSeT, which supports the replaceability and compensation of web service transactions, and extend the WS-Frame framework by adding various components to support transaction-dependence detection for transactions in FleWeSeT. Furthermore, to achieve alternative and compensatory operations for web service transactions, we propose the optimistic verified concurrency control protocol ACoDe. Finally, we establish a randomly colored Petri Net model and undertake analysis and evaluation for the ACoDe protocol. Experimental results indicate that ACoDe performs better than WS-BA in the aspects of average response time and success rate of web service transactions.

Section 2 reviews some related work. In Section 3, we describe our web service transaction model called FleWeSeT and our extended framework called WS-Ultimate. In section 4, we discuss our web service transaction concurrency control protocol called ACoDe, which can help concurrent transactions globally obtain access to serializability. In section 5, we describe our simulation model based on colored Petri Nets and compare our performance with WS-BA. Section 6 presents the study's conclusions.

2. RELATED WORK

Existing research studies about concurrency control of real-time web service trans-

actions originally came along with the development of real-time transaction processing. However, this processing was first used in databases and later developed for web services. In 1993, K. Ramamritham first clearly introduced the application background of research on real-time transaction management, the features of real-time transactions and the key problem of real-time transaction processing [30, 31]. In the new century, a boom of research on distributed real-time databases and transactions emerged [32, 33].

However, these studies focused on short-lived database transactions in tightly-coupled environments, which are composed of data operations. To ensure reliability and consistency, these methods for database transactions adopt centralized decision-making, restrain each other strictly and require all participants to be active and obedient; however, they are unacceptable for autonomous web services in loosely coupled environments. If a web service system adopts these protocols, it will lead to long periods of continuous waiting for a large number of transactions and cause fatal damage to the system. The web service environment needs new forms of transactions and management mechanisms that do not need to strictly follow the ACID property of transactions.

With the development of web service transactions, the composition of real-time web service transactions for complex business transactions has become a study hotspot. Mark Little said: "Only are the basic services combined to form new, composite web services integrated with more functions, web service technology can play its role in the practical application actually. The key problem is to ensure the reliability and consistency of the result conducted by multiple web services. It's impossible to meet the complex business demand of composite web service applications without the transaction capacity." [35]

Research institutions have started research projects on web service composition. They are focusing on automatic service composition based on business flow, AI planning and program combination, and they explore these regarding various aspects, such as service description, service matching, service selection, service quality, service security and composition validation and test [36-42]. However, all of the research studies are concentrated on how to rapidly find appropriate existing services and design a composite service to meet the demand of a single user. They do not take the time constraint and the consistency issues into account. When plenty of users are requesting various composite web services on the Internet, a vast number of composite services will call services on the lower layer to execute in parallel within the deadline, which poses a large challenge for consistency.

The composition of real-time web service transactions is a new problem that has appealed to many researchers. References [43-47] have studied the modeling, formation, arrangement and infrastructure of real-time web service composition based on time constraints, but they have not involved transactions. References [48-51] have studied modeling, formation, arrangement, combination formal modeling and accuracy verification based on transactional constraints, which were not real-time. Research studies in these two areas have just focused on the specification of real-time and transactional constraints in the design of web service composition, and the concurrent execution of real-time web service transactions is not given attention.

Academia and industry have put forward some coordination specifications of transaction operations that are suitable for the web service environment: BTP [51], WS-C/T [52], WS-CAF [53] and the standard specification WS-TX approved by OASIS (including WS-Coordination, WS-AT, and WS-BA) [29]. Reference [29] analyzed transactional

requirements in SOC and provided an integrated framework for service and WS-TX. References [54-57] raised extended proposals for the WS-TX framework, and references [58-61] studied the cascade compensation in the distributed and P2P environment, which did not consider the need of real-time. It is still necessary to build particular applications or services to deal with concurrency control and recovery [54-57].

Therefore, our concern is more related to the concurrency control of real-time web service transactions within deadlines.

In the Internet, message transmission and the network are unreliable. It is a tough challenge to effectively manage and control complex behaviors of real-time web service composition. However, the replaceability and compensation of web service transactions make it possible to improve the performance of real-time web service transactions. By replaceability between equivalent services, we can eliminate conflicts between web service transactions and implement compensatory measures, which can undoubtedly help more web service transactions meet their real-time requirements.

3. WEB SERVICE TRANSACTION MODEL AND FRAMEWORK

We propose a flexible web service transaction model called FleWeSeT in the form of a transaction tree based on replaceability. The model contains two types of nodes, as shown in Fig. 1. Non-leaf nodes represent composite web service transactions, leaf nodes represent basic transactions, and the edge between two nodes denotes a parent-child relationship. In this model, the root node (such as WST in Fig. 1) represents the entire business transaction. Additionally, each node has a set of alternative transactions for its own sub-transactions. As shown in the dashed-line circle in Fig. 1, WS6 and WS7 are the alternative transactions for WS1, which guarantees that WST11 can commit the request for the execution of other alternative sub-transactions through the index when WS1 breaks down within the deadline. For each non-leaf node, the degree and depth are both dependent on functional logic, the service composition process and alternative service flexibility. According to the functional logic and the service composition, the parent-child relationship between two nodes can be confirmed, and an edge can be added between the two nodes. As the process goes on, a transaction tree can be established. Each sub-transaction has corresponding alternative and compensatory transactions, which may also be a transaction tree. Therefore, the depth of a non-leaf node can be computed along the path from the node to the root node, and the degree implies the sum of child nodes it owns. Moreover, because of alternative transactions offered by different providers with the same function and similar performance, the implementation of the entire business transaction varies in different paths, which influences the degree and depth of a non-leaf node as well.

A flexible web service transaction model satisfies the request for the deadline and allows flexible web service composition, recovery and concurrency in granularity. Each sub-transaction possesses an independent submission right, compensates for failure transactions, and significantly adapts the characteristics of the autonomy and isomerism of the web service. To improve concurrency, the results of the submitted sub-transactions are available for other concurrent executions of sub-transactions.

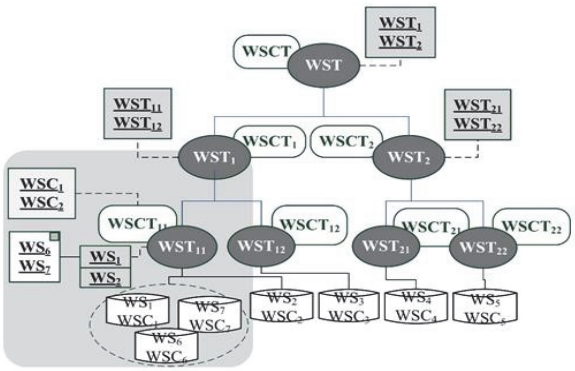


Fig. 1. Flexible web service transaction model based on alternates.

3.1 Transaction Dependency

In the process of a web service transaction, according to the transaction tree based on function logic, composition and alternatives, each web service has a set of alternative transactions and local relevant compensatory services. That is, once a transaction breaks down, the relevant transaction node searches the list of compensatory services by the current service id and requests compensatory operation. Considering replacement and compensation, there may be four dependency relationships: logic dependency, resource dependency, alternate dependency and compensation dependency.

Definition 0 Transaction dependency Suppose there are two web service transactions T1 and T2; if the outcome of transaction T2 can be affected by transaction T1, we call the relationship between them transaction dependency. In this case, transaction dependency can be described as T2→T1, while T2 is a parent node and T1 is a child node.

Definition 1 Logic dependency In a web service transaction flow, the outcome of the parent transaction is affected by the sub-transaction, namely, the parent transaction needs to wait until all of the sub-transactions have been submitted. We call the dependency relationship between the parent and sub-transactions logic dependency, in which the child transaction dominates the transaction and the parent transaction is dependent on the transaction. For example, there is a logic dependency between an online payment service and online bank services in an online ticketing system. The online payment service cannot confirm an order until the online bank service is submitted. That is to say, an online payment service logically depends on online bank services.

Definition 2 Resource dependency When two web service transaction flows request the same sub-transaction, because of the semantic relationship between the various operations of the sub-transaction, there must be a certain order for the two transactions. We refer to this dependency relationship as resource dependency. For example, in an online ticketing system, when two users are operating on the same ticket resource, transaction A is refunding a ticket and transaction B is booking the ticket. Transaction B should be performed after transaction A is submitted. That is to say, there is an order between the two transactions, so transaction B depends on transaction A for resources.

Definition 3 Alternate dependency Alternate dependency is a relationship based on replaceability. A new alternative transaction keeps some original dependency relationships of the replaced one, such as the logic dependency relationship between the parent and the sub-transaction, so there is an alternate dependency between the alternate transaction and the original transaction. For example, for an online ticketing system, the alternative service of the online service in bank A is the online service in bank B, so there is an alternate dependency between online services in bank A and bank B.

Definition 4 Compensation dependency Considering two sub-transactions that have a logic dependency or resource dependency relationship (no matter whether they belong to the same or different business transaction flow); when one sub-transaction needs to be compensated, another one also needs to trigger the compensation transaction, and we refer to the relationship between these two transactions as compensation dependency. Transactions with compensation dependency can belong to the same web service transaction flow or different transaction flows. It is reasonable that compensation dependency can be caused by either logic dependency or resource dependency. For example, there are two business flows that separately handle deposits and withdrawals to the same account, and deposit compensation is performed after withdrawal compensation to avoid system mistakes caused by deposit compensation.

To facilitate the expression of dependency, we adopt a dependency graph in the transaction process. A dependency graph is a directed graph in which nodes represent transactions and edges represent the transaction dependency between nodes. Each edge is depicted from dependent transaction to dominated transaction. The dependency graph is updated correspondingly when a new transaction enters or leaves the system. Fig. 2 indicates that transaction T1 depends on T2 and T3.

In a transaction flow, the parent and child transactions are predetermined, so logic dependency belongs to global dependency and static dependency, which can be reflected in the global dependency graph. Resource dependency depends on the transactions that dynamically go through the service site, so it belongs to local and dynamic dependency. Compensation dependency is caused by either logic dependency or resource dependency, so the compensation sequence is determined by the two dependency relationships when triggering a compensation transaction. Nevertheless, an essential alternate dependency is the newly added relative logic dependency or resource dependency. Therefore, we mainly consider the detection of the logic dependency and resource dependency among sub-transactions when processing the concurrency control and the submission. Both dependency relationships can possibly cause a dependency cycle among sub-transactions in a practical process. Fig. 3 shows two web service transactions with dependency, wherein

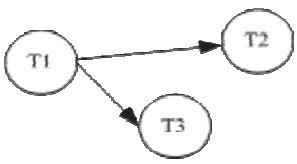


Fig. 2. Dependency graph.

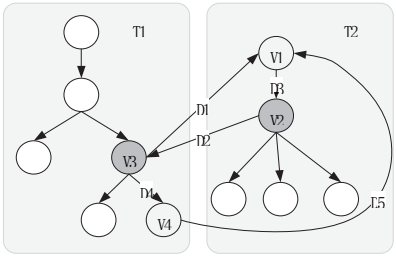


Fig. 3. Two web service transaction with dependency.

transactions T1 and T2 are nested transactions and a solid line represents a parent-child relationship with logic dependency. In other words, the parent transaction depends on the sub-transactions. A dashed line denotes resource dependency between two sub-transactions. For example, there are resource dependencies between V1 and V3, V4 and V1, and V2 and V3. There are two cases where a dependency cycle may occur. First, for V1, V2 and V3, we assume that logic dependency D3 and resource dependency D2 exist, and then it is impossible for resource dependency D1 to exist. That is because resource dependency is a dependency relationship formed by requests to the same service or sub-transactions from two transactions. The establishment of D2 means that V2 and V3 submit a request to the same service or sub-transaction simultaneously; likewise, the establishment of D1 means that V1 and V3 submit a request to the same service or sub-transaction. However, there is a parent-child transaction relationship between V1 and V2, in which resource dependency does not occur considering the division and combination of services in the practical applications. Therefore, under the circumstance of D1 and D2, D3 does not exist. One case that should be considered is the dependency cycle D2, D4, D3, and D5. In the context of the establishment of D2, D3 and D4, if D5 exists, resource dependency and logic dependency will constitute a dependency cycle. When there is a dependency cycle of sub-transactions, a circular wait for submission will occur. Because the dependent transaction has to wait for the submission of its dominated transactions, permanent waiting for multiple transactions will be caused. Therefore, dependency detection of sub-transactions is a crucial problem for the concurrency control of nested web service transactions.

3.2 Extended Framework WS-Ultimate

OASIS has proposed the WS-Coordination, WS-Atomic Transaction and WS-Business Activity protocol as a service transaction framework. To maintain web service consistency, we extend the web service transaction standard framework and rename the extended framework to WS-Ultimate. WS-Ultimate can implement the concurrency control of web service transactions, support transaction dependency detection and alternate operation. The extended WS-Ultimate framework is shown in Fig. 4.

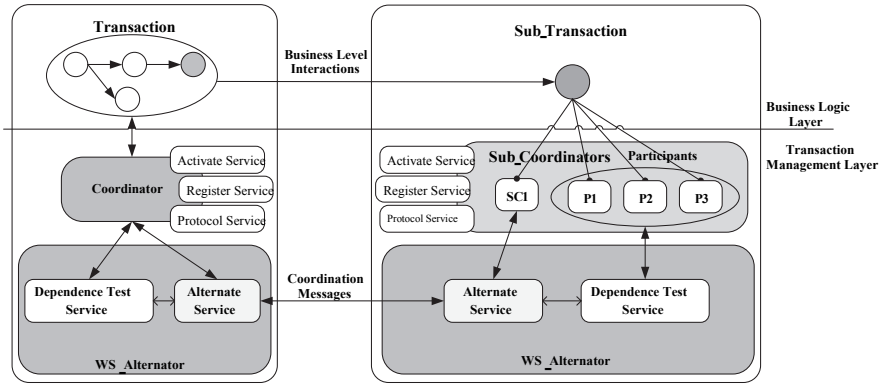


Fig. 4. WS-Ultimate extended framework.

The WS-Ultimate framework adds WS-Alternator components in the coordinator end and participant end. In addition, because of the characteristics of the web service transaction, in this model, participants in the original standard framework also play a coordinator role for the subordinate transaction. Therefore, the web service end is extended into the sub-transaction end. Besides participants, we also add a sub-coordinator to allow web service participants to act as the subordinate coordinator and call sub-transactions. Therefore, in the WS-Ultimate framework, a coordinator plays two roles, the parent coordinator and the sub-coordinator. Although the two roles correspond to the same component, the different roles' tasks differ. The extended framework also retains the three component services of the standard framework: activate service, registration service and protocol service.

Newly added WS-Alternator components are responsible for the two ends transferring messages and two services, which are the Dependence Detection Service and Alternative Service.

• Dependence Detection Service

Sub-coordinator end (also known as subordinate coordinator): The Dependence Detection Service located at the sub-coordinator end maintains a local resource dependency graph. When the WS-Alternator in the sub-coordinator end receives a submission request from the coordinator end, the Dependence Detection Service can detect whether the submission request can be submitted immediately, namely, whether there exists a dominated transaction, and then report the detection result to the Alternative Service.

Coordinator end (also known as superior coordinator): The Dependence Detection Service located at the coordinator end has the same function as described above when playing the role of sub-coordinator, namely, to receive a submission request from the superior coordinator, conduct dependence detection and determine whether to enable the submission. In the structure of the transaction, besides the root node, the coordinator of the transaction node always works as both sub-coordinator and parent coordinator at the same time, which means it needs to send and receive requests. The Dependence Detection Service will not be enabled when acting as the parent coordinator.

• Alternative Service

Sub-coordinator end: The Alternative Service located at the sub-coordinator end is responsible for receiving messages from the local Dependence Detection Service and sending them to the coordinator end.

Coordinator end: The Alternative Service located at the coordinator end receives messages from the sub-coordinator end, requests an alternative transaction for the current sub-transaction in the waiting state, registers and performs an alternative transaction, and notifies the sub-transaction participant to stop waiting.

All coordination messages from the coordinator or reply messages from the sub-coordinator should be received or processed by WS-Alternator before being transferred. For example, when WS-Alternator receives a submission message from the coordinator, it should check whether the transaction can be submitted before transferring the message to the corresponding sub-coordinator. Therefore, all different web service statuses (such as submission, compensation, suspension, *etc.*) defined by WS-Business Activity are stored in WS-Alternator.

The benefit of importing WS-Alternator is that the sub-coordinator end can obtain the coordination context of different transactions that request the same sub-transaction, so it can detect potential transaction dependencies and handle them appropriately to prevent circular dependency.

3.3 Representation and Detection of the Transaction Dependency

Because all the web service states in different contexts defined by WS-Business Activity are saved in WS-Alternator, the maintenance of the dependency detection and dependency relationship can be implemented by the WS-Alternator components of the extended WS-Ultimate framework.

3.3.1 Resource dependency

For the resource dependency among transactions mentioned in section 3.1, we use the method of a conflict matrix to detect it. In this paper, a conflict matrix is established and updated by the sub-transaction or service provider and accessed by WS-Alternator.

The conflict matrix is a matrix of $n \times n$, wherein n is the number of operations in the web service at the service provider end. Whether there are conflicts between operations is defined by the semantics of these operations. These operation conflicts can reflect the interaction relationship when they perform. Here is the definition of operation conflicts: if the order of two operations changes, it may lead to a different final state, and these two operations have a semantic conflict. The conflict matrix is established by a sub-transaction provider before providing a web service while designing sub-transactions, because only the service provider understands the conflict relationship among operations. It also can help WS-Alternator detect a potential transaction dependency while transactions are running.

The conflict matrix is determined by the attributes of the operation, and it is static and determined. Logic dependency and compensation dependency are always dynamic and depend on the actual business flow.

3.3.2 Logic dependency

For the logic dependency mentioned in section 3.1, we use a logic dependency tree to represent and maintain the logic dependency relationship. A logic dependency tree is created in the process of establishing the transaction. Each sub-transaction node maintains its own parent and sub-transactions that have a call and called relationship and establish a local logic dependency tree, which is maintained by each corresponding coordinator. Specifically, we usually adopt a multi-tree. Therefore, a logic dependency tree can be a good description of the overall transaction structure and preserve logic dependency.

The coordinator determines the logic of transaction submission according to the nodes' parent-child relationships on the logic dependency tree. When the current node needs to be submitted, we check whether the sub-transaction of the current node is allowed to be submitted based on the local logic dependency tree and report the information to the superior coordinator.

3.3.3 Compensation dependency

For the compensation dependency mentioned in section 3.1, we use the list of compensation to describe and store the compensation dependency relationship. Because the compensation dependency relationships of transactions are determined by both logic dependency and resource dependency, the process of establishing a compensation list is dynamic. The compensation list is maintained by the coordinator when building and executing transactions, and it is represented by a linked list. According to the definition of compensation dependency in section 3.1, the child node is a dominated transaction, and the corresponding parent node is a dependent transaction. That is to say, parent nodes can trigger child nodes' compensation. Therefore, in the compensation linked list, parent nodes are the predecessor nodes of compensation.

In addition, for resource dependency, we take the dependency relationship we get from the resource dependency graph into consideration to determine which one is the dominated transaction. Similarly, dependent transactions are predecessor nodes in the compensation list.

The representation and detection of these three dependency relationships can contribute to the transaction modeling, transaction submission and concurrency control, which we mention in section 4.

4. CONCURRENCY CONTROL PROTOCOL OF WEB SERVICE TRANSACTIONS: ACODE

An extended web service transaction framework with WS-Alternator can detect transaction dependence in a concurrent process. The concurrency control protocol proposed in this paper is based on the nested web service transactions and the new extended framework. After detecting resource dependence, we alternate the current waiting subtransaction to submit within the deadline, thus breaking the wait state of the entire transaction requested for submission and creating a new transaction execution path that can effectively avoid circular wait.

The ACoDe protocol maintains the dependence graph of the transaction. While ensuring that the graph does not contain any cycles, the protocol guarantees that concurrent transactions obtain access to serializability globally. We also integrate ACoDe into our WS-UItimate framework, in which each Dependence Detection Service component in WS-Alternator maintains partial views of the global dependence graph. Partial views preserve the dependence relationships among transactions when calling local sub-transactions. Each WS-Alternator ensures that its partial dependence subgraph is without cycles by requesting replaceability through the Alternative Service and applies the Commit-Order strategy to control the order of committing concurrent transactions. Commit-Order is an optimistic concurrency control strategy that allows the immediate acceptance of concurrent access to local services and a consistency check at the time of submission. Therefore, deadlock and circular wait never appear in experiments. WS-Alternator applies the following three rules to ensure the consistency of transactions:

- A transaction can be submitted only when all its dominated transactions are submitted within the deadline.

- When a transaction is aborted or compensates local activities, the local activities of all its dependent transactions should be compensated automatically.
- When a transaction is in a waiting state for dependence problems or in an unfinished state for faults, transaction replaceability should be considered.

The result of the first rule is that the submission request of any dependence transactions should be delayed until dominated transactions are submitted. For the dependence transactions that are delayed to submit, the coordinator can decide whether to start transaction replaceability or not according to the waiting time.

Fig. 5 shows the transaction state transition of the leaf nodes in the web service transaction tree, and Fig. 6 shows the transition graph of the non-leaf nodes. In the model

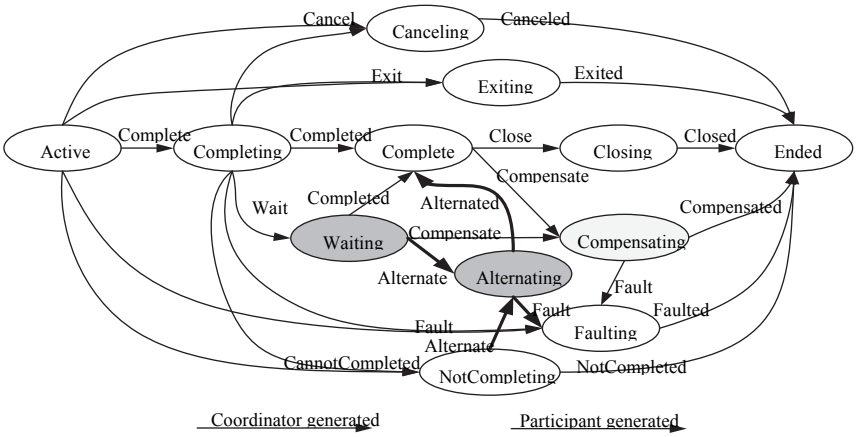


Fig. 5. Transaction state transition of leaf nodes.

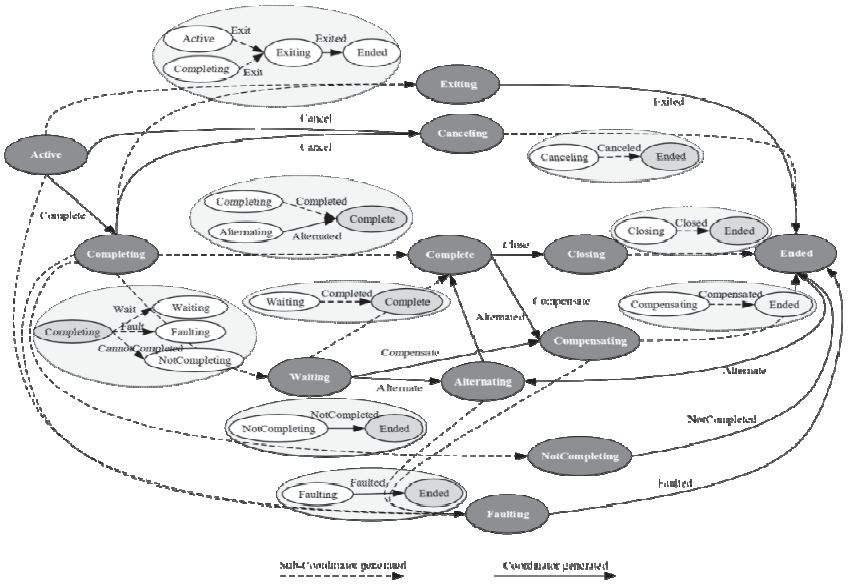


Fig. 6. Transaction state transition of the non-leaf nodes.

of the web service transaction tree, the transaction state transition of the non-leaf nodes is determined by the process of the sub-transaction state transition, whereas the state transition of the leaf nodes is triggered by the result of the web service.

We first demonstrate the transaction state transition of leaf nodes, as shown in Fig. 5. Dashed arrows represent trigger messages from participants, while solid arrows represent trigger messages from coordinators. The text on the arrows indicates the messages.

We add two new states into WS-Business Activity, *i.e.*, the waiting state and the alternating state, which are represented as two dark ovals in Fig. 5. We define three new types of messages, *i.e.*, Wait, Alternate and Alternated, which are imported into the protocol.

- Wait Message

WS-Alternator manages communications between participants and coordinators, including sending a Wait message to notify them that their requests have to be delayed due to consistency problems and then changing the state transition from Completing to Waiting. WS-Alternator traces the current state of the current concurrent participants and the context of the transactions. Once realizing the submission of all transactions dominated by the waiting transactions, the delayed submission request will be sent to the web service, and then coordinators will receive a Completed message, which changes the state from Waiting to Complete.

- Alternated and Alternated Message

When a transaction is in the Waiting state, the coordinator sends an Alternate message to notify the participant that the alternative service is successfully registered, but it can also choose to continue waiting. At this time, the transaction state changes from Waiting to Alternating. If the new alternative service is submitted successfully, the coordinator sends an Alternated message to notify the original participant to cancel the current waiting submission, which changes the state from Alternating to Complete. It indicates that the alternative service for the delayed submission has been performed successfully. However, in a service with no replaceability, transactions have to wait until all dominated transactions are submitted within the deadline. Then, the delayed submission requests are sent to the web service, and coordinators receive a Completed message in the end.

The role of participants is played by sub-coordinators for the web service transactions of non-leaf nodes. Therefore, the transaction state transition of non-leaf nodes, as shown in Fig. 6, is triggered by the sub-transaction state transition. The corresponding relations between every message sent by participants in Fig. 5 and the state transition processes of the sub-transactions are shown in Table 1.

A submission request (*e.g.*, Completed message) from the superior coordinator is received by WS-Alternator as a representation of the subordinate coordinator. Then, the Dependence Detection Service inspects its partial dependence graph to decide whether there is an outgoing edge of the transaction. In Fig. 7 (a), the submission request is received when WS-Alternator finds that the transaction node has no outgoing edges and is sent directly to the appropriate subordinate coordinator. Upon receiving all of the Completed messages from all subordinate coordinators within the deadline, WS-Alternator deletes the transaction node in the dependence graph and sends a Complete message to the superior coordinator.

Table 1. The corresponding relations between the message.

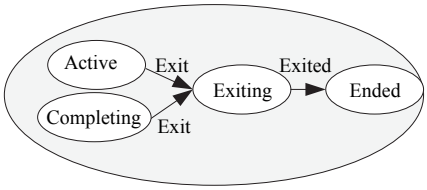
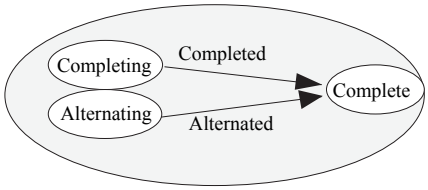
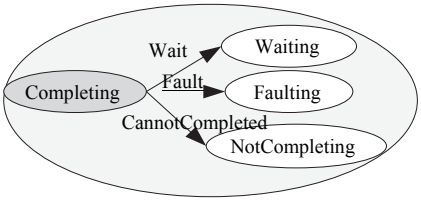
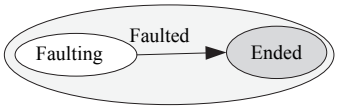
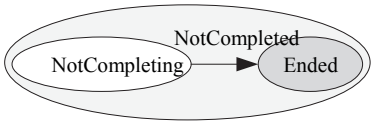
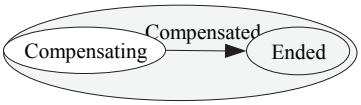
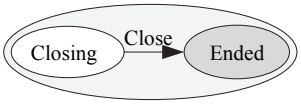
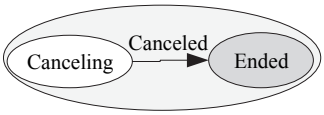
Messages sent by sub-coordinators	Sub-transaction state transition
Exit	
Completed	
Wait	
Fault	
Cannot Completed	
Compensated	
Closed	
Cancelled	

Table 2. Simulation parameters.

Parameter	Value
The number of coordinators	2
The network packet loss rate	20%
Registration success rate of alternating sub-transactions	80%
Submission success rate of alternating sub-transactions	90%
The alternating probability for failing sub-transactions	49%
Compensation success rate of sub-transactions	49%
Error rate of sub-transactions	10%
Unfinished rate of sub-transactions	10%
Growth coefficient of transaction dependence with concurrent degree	0.1
Dependence detection delay	49
Network transmission delay	49
Submission delay of sub-transactions	49
Waiting delay of sub-transactions	100
Registration delay of alternate sub-transactions	200
Submission delay of alternate sub-transactions	49
MinSlack	2

In Fig. 7 (b), WS-Alternator finds an outgoing edge of the transaction node and delays the submission until all dominated transactions are submitted. That is, WS-Alternator temporarily does not send a Complete message to the subordinate coordinator but returns a Wait message to the superior coordinator.

At the same time, the superior coordinator is retrieving an alternative transaction for the waiting sub-transaction. Once an alternative transaction is available and registered successfully (Procedure 4), the superior coordinator sends an Alternate message to the participant (Procedure 5) to notify the subordinate coordinator of the completion of the transaction replacement. At this point, the superior coordinator can continue to be in the waiting state. After the alternative transaction is submitted successfully within the deadline (Procedure 6), the superior coordinator sends an Alternated message (Procedure 7) to the subordinate coordinator. Once WS-Alternator receives the Alternated message, it will send a Cancel message (Procedure 8) to the original subordinate coordinator, which stops waiting and aborts the submission, while the transaction is removed from the dependence graph by WS-Alternator. Thus, the sub-transaction in the waiting state is released by executing alternative transactions.

Fig. 7 (c) describes the compensatory process of web service transactions. For submitted sub-transactions (Procedure 6), the superior coordinator sends a Compensate message (Procedure (7)) and then the subordinate coordinator sends a Compensate message as well (Procedure 7). After the compensation of the subordinate coordinators is over, the Compensated message carrying compensation information is returned to the superior coordinator in turn (Procedure 9, 10, (10)), which receives all Compensated messages from all its subordinate coordinators, and the compensation succeeds within the deadline.

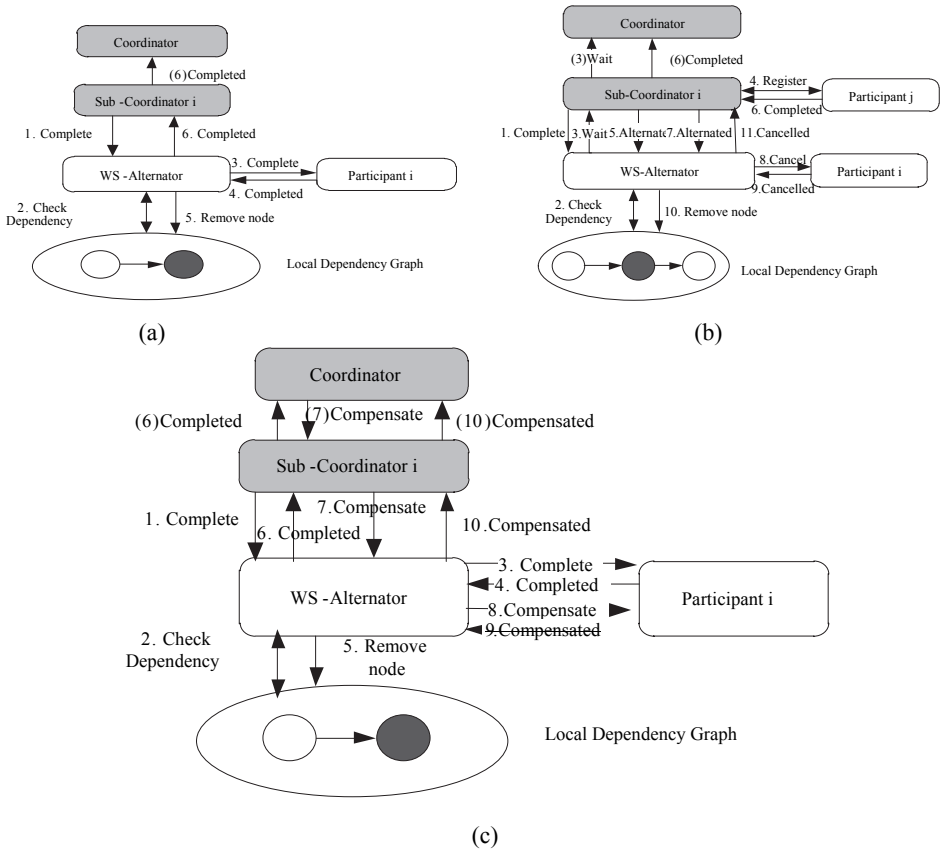


Fig. 7. The process of how WS-Alternator carries out a replacement transaction to solve resource dependence and compensation.

5. PROTOCOL SIMULATION AND PERFORMANCE EVALUATION

5.1 ACoDe Simulation Based on a Colored Petri Net

In this study, we establish a concurrency control model of web service transactions based on a randomly colored Petri Net (CPN), which can describe concurrent behaviors accurately and concisely, to simulate the ACoDe protocol in the new WS-Ultimate framework. We then compare the model with the standard framework protocol Business Agreement With Coordinator Completion (the WS-BA protocol for short) in two aspects, the transaction success rate (*i.e.*, the ratio of the number of transactions successfully submitted within the deadline and the total number of transaction requests) and the average response time (*i.e.*, the response time for transactions that are completed within the deadline is the actual time they execute, even with many restarts; however, for transactions that are not completed before the deadline, the response time is the time from the start to the deadline; the average response time equals the total response time in both situations of success and failure divided by the total number of transaction requests).

5.1.1 Experimental model and parameters

The simulation model contains a superior coordinator and a subordinate coordinator. The superior coordinator simultaneously sends transaction submission requests to the subordinate coordinator through the network, which receives these requests and returns processing results to the superior coordinator according to the situations of the transaction submission requests. Drawing on the experience of parameter setting in reference [46, 47], the system simulation parameters are shown in Table 2 (time in ms). We then adopt the parameter values in Table 2 to model the ACoDe protocol in the WS-Ultimate framework and the Business Agreement with Coordinator Completion protocol (WS-BA) in the standard framework. The constituent elements of an arrival transaction include the transaction ID, message contents and deadline. The deadline is (current time + slack factor* time taken), where the slack factor is uniformly distributed in [MinSlack, MaxSlack]. In our model, we define two enumerated types of color sets to represent different message types between the superior coordinator and the subordinate coordinator. To facilitate the expansion and reuse of the model, we adopt a hierarchical Petri Net, including one parent page (WS-Alternator) and two subpages (Alternator_A and Alternator_B), which contain the dependence detection module, the alternate processing module, and other transaction processing modules, respectively.

In the Petri Net model, the first step is to set color sets. The color set Coormess defines message types from the superior coordinator. In the protocol proposed above, the superior coordinator can send a Complete message, a Compensate message, an Alternate message and an Alternated message to the subordinate coordinator when in search of alternate sub-transactions. Additionally, the superior coordinator can also actively send Cancel, Close and End messages and respond to Exited, Faulted and NotCompleted messages from the subordinate coordinator.

Similarly, the color set PartiMess defines message types from the subordinate coordinator. In particular, the subordinate coordinator can send a Wait message to make subtransactions enter the delayed submission state. There are also Completed, Compensated, Cancelled, and Closed messages.

Furthermore, the color sets TIDxCoorMessxWSTime and TIDxPartiMessxWSTime define the token type received by libraries A, B, C, and D in the parent page, respectively. This type is a triple, consisting of the TID (transaction ID), CoorMess/PartiMess (corresponding message type) and WSTime (time taken by the sub-transaction).

To improve the reliability of the experimental data, the system remains running for 5000s, continuously, to ensure that our system runs in a stable state, and we start to collect the results after 1000s.

5.1.2 Simulation model construction based on the petri net

In this section, we will describe the modeling process based on the Petri Net in detail.

ACoDe simulation in the WS-Ultimate framework

The simulation model of the parent page is shown in Fig. 8, in which the two alternative transmissions Alternator_A and Alternator_B correspond to the Alternator_A and

Alternator_B subpages, respectively. As a component in the superior coordinator, Alternator_A is in charge of receiving and handling coordination messages from the subordinate coordinator Alternator_B in library D and sending processed messages to library A. Similarly, Alternator_B, as a component in the subordinate coordinator, is in charge of receiving and handling coordination messages from the superior coordinator Alternator_A in library B and sending processed messages to library C. The transmissions TransmitMessage1 and TransmitMessage2 simulate the transmission of the coordination messages and obtain the network packet loss rate by calling the trans_success function, which is set at 20%. Fig. 9 shows the statistics of the transaction success rate.

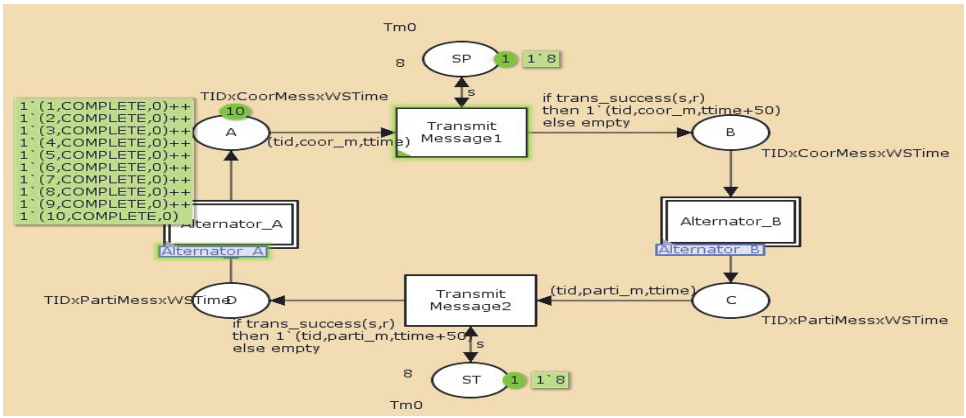


Fig. 8. CPN model of the parent page in the ACoDe protocol.

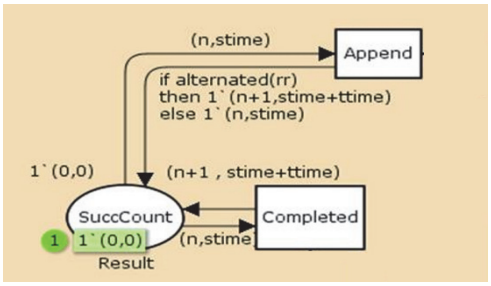


Fig. 9. The statistics of the transaction success rate.

As shown in Fig. 9, referring to Fig. 5, when the transaction changes its state from "Completing" or "Waiting" to "Complete" after the sub-coordinator sent the message "completed," the SuccCount is incremented by 1. Likewise, when the state changes from "Alternating" to "Completing" by the message "Alternated," the SuccCount also needs to be incremented by 1. That is, when the state of a transaction becomes "Complete" within deadline, the SuccCount needs to be increased by 1.

We take the Alternator_A subpage as an example to demonstrate the simulation process, as shown in Fig. 10. Alternator_A plays the role of the superior coordinator and sends messages that are used in the superior coordinator. Alternator_ A also contains 7 transmissions, Judge, Completed, wait_Alternate, Alternated, Faulted Process, Cannot-

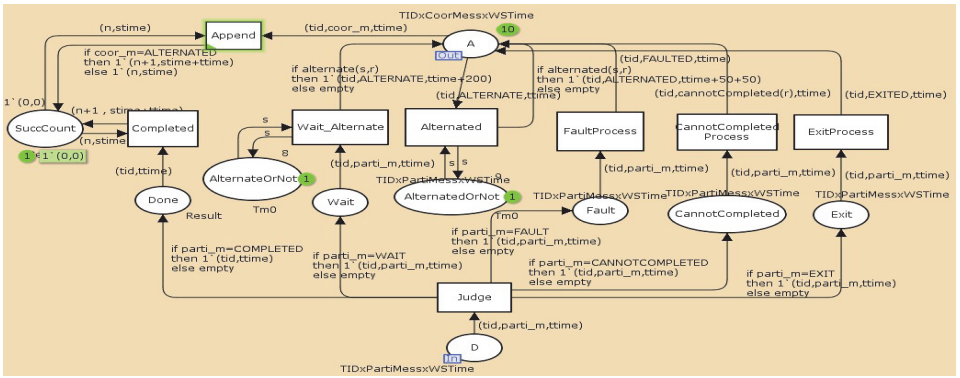


Fig. 10. CPN model of Alternator_A subpage in the ACoDe protocol.

Completed Process and Exit Process. The Judge transmission processes and responds to messages from the subordinate coordinator Alternator_B. For example, if Judge receives a Completed message, it will transmit it to library Done, which temporarily stores the TIDs and time spent by completed transactions. After the Completed transmission, library A returns an empty record to the parent page (because the Completed message indicates that the transaction has been successfully submitted) and submits the number of successful transactions and response time through the library SuccCount. Another example is, if Judge receives a Wait message, it will decide whether to notify the superior coordinator to wait or to choose an alternative transaction in accordance with the Wait_Alternator transmission. This process is simulated by random numbers generated by the library AlternateOrNot. For the first case, the Wait_Alternator transmission will send a token containing an Alternate message to library A. Otherwise, Wait_Alternator sends a token containing an Alternated message (simulated by library AlternateOrNot). Additionally, the success of alternating sub-transactions also belongs to a trigger condition, so the SuccCount statistics accumulate accordingly.

Fig. 11 shows the CPN model of the Alternator_B subpage. Alternator_B plays the role of the subordinate coordinator and sends messages that are used in the subordinate coordinator. Alternator_B also contains 11 transmissions, Dependence Detection, Judge,

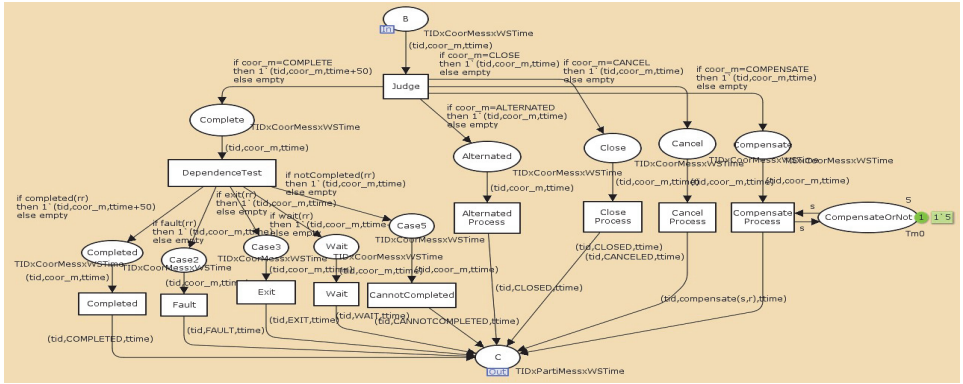


Fig. 11. CPN model of the Alternator_B subpage in the ACoDe protocol.

Alternated Process, Close Process, Cancel Process, Compensate Process, Completed, Fault, Exit, Wait, and Not Completed. The Judge transmission processes and responds to messages from the superior coordinator Alternator_A. If library B receives a Completed message from Alternator_A, Judge Transmission will transmit it to the left library complete, and the Dependence detection transmission will detect the dependence. If the result is the permission to submit, the process will be executed by the Completed transmission, and otherwise by the Wait transmission. The functions of other libraries and transmissions are similar to the functions described in Alternator_A.

In addition, we have also modeled the standard framework Business Agreement with Coordinator Completion based on the Petri Net to contrast it with our protocol ACoDe. The modeling results are shown in Figs. 12, 13, and 14.

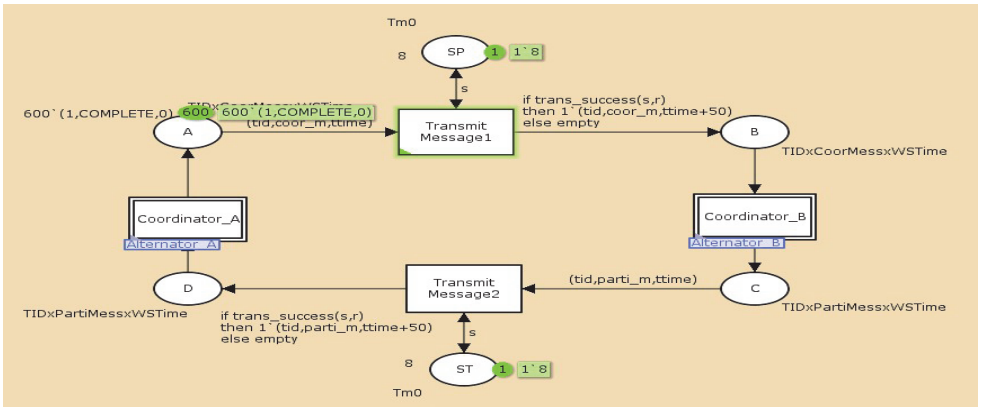


Fig. 12. CPN model of the parent page in the WS-BA protocol.

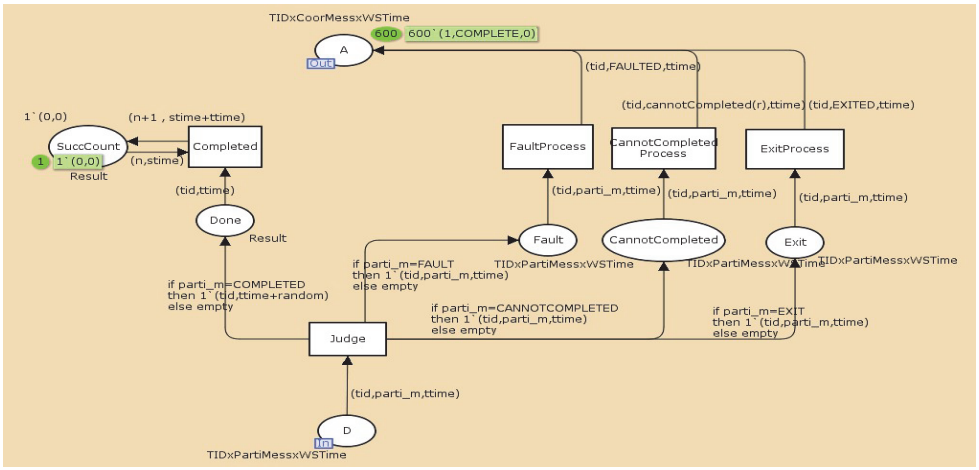


Fig. 13. CPN model of the Coordinator_A subpage in the WS-BA protocol.

WS-BA Simulation

Fig. 12 depicts the simulation model of the parent page for WS-BA. The context of

the web service transaction that the parent page expresses is consistent between the two protocols. Therefore, the model of the parent page should maintain consistency in the two protocols.

Fig. 13 depicts the simulation model of the Coordinator_A subpage for WS-BA. This model lacks the Wait_Alternate and Alternated transmissions.

Fig. 14 depicts the simulation model of the Coordinator_B subpage for WS-BA. This model lacks the Alternate Process and Wait transmissions. The original WS-BA does not have the ability to handle the access to conflict.

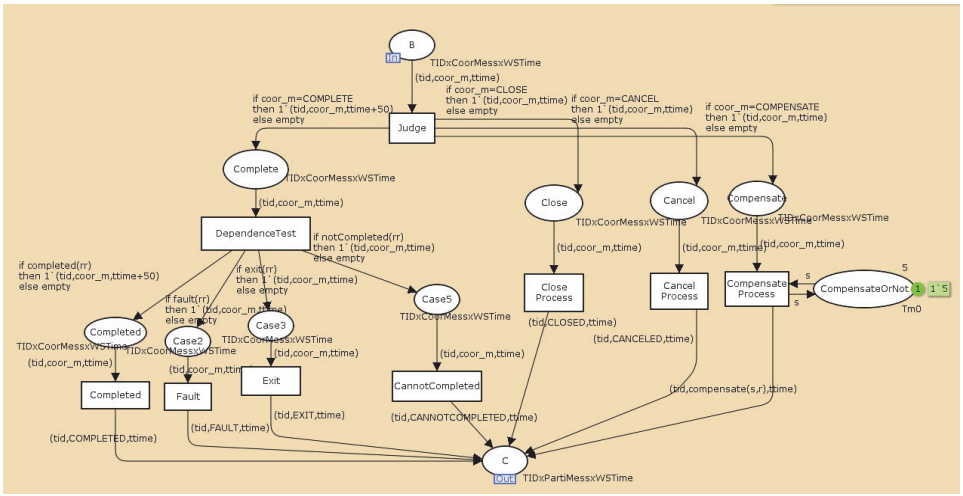


Fig. 14. CPN model of the Coordinator_B subpage in the WS-BA protocol.

The models of the two protocols are set to the same initial transaction state and simulation parameters. In the next section, we will analyze and compare the simulation results and evaluate the ACoDe protocol performance.

5.2 Analysis and Evaluation of the Simulation Results

Based on the simulation models above, we regard the transaction success rate as a performance evaluation metric to compare the ACoDe protocol in the WS-Alternator framework and Business Agreement with the Coordinator Completion protocol (WS-BA) in the standard framework. Considering every concurrent transaction situation, we conduct the experiment five times to record the number of successful transactions and the transaction response time, and then we calculate the average values to compare the results. The experimental data and comparative results are shown in Figs. 15 and 16.

Experimental results show that the transaction success rate in the new framework protocol ACoDe is significantly higher than in the standard framework of the WS-BA protocol. This is because, when a sub-transaction that the subordinate coordinator handles cannot be submitted normally, a Wait message will be sent to the superior coordinator, which starts an alternative transaction if the waiting period is too long but still within the deadline. However, for WS-BA, if a transaction cannot be submitted normal-

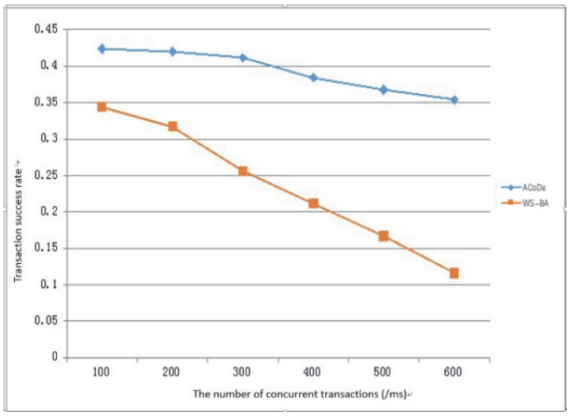


Fig. 15. Comparison of the transaction success rate.

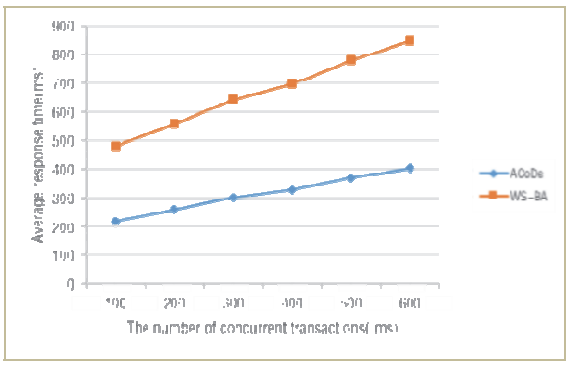


Fig. 16. Comparison of the average time.

ly, it will restart the same transaction again and again until it meets the deadline or it is completed successfully. Thus, we empirically verify that alternating a new transaction is more likely to be completed successfully than restarting the transaction that has failed before. That is why the transaction success rate improves significantly. In addition, with the increase of the transaction concurrency, the transaction success rate declines slightly, and because of the limit of the subordinate coordinator and the increase of concurrent transactions, transaction dependency will be increased when submitting requests, which results in response to more submission waiting cases.

Therefore, the new framework of the ACoDe protocol performs better than the WS-BA protocol in the aspect of the transaction success rate. Furthermore, by the reason of the higher transaction success rate, which means more transactions in ACoDe are completed within the deadline, the average response time of transactions in the new framework protocol ACoDe is lower than that in the standard framework protocol WS-BA.

6. CONCLUSIONS

In a dynamic open web service environment, maintaining consistency is a new cha-

llenge. In this paper, we have proposed a flexible web service transaction model FleWeSeT, which regards the root transaction and all sub-transactions as a transaction tree. For each non-leaf node of the transaction tree, the degree and depth are both dependent on functional logic, the service composition process and alternative transaction flexibility. We have expanded the WS-Frame framework as WS-Ultimate and joined some components to support the replaceability, compensation and transaction-dependence detection of real-time web services. We have also proposed an optimistic verified concurrency control protocol, ACoDe, for replaceable and compensatory web service transactions based on the transaction dependence detection. Furthermore, we have integrated the ACoDe protocol into our proposed WS-UItimate framework, in which each dependence detection service component in WS-Alternator maintains partial views of the global dependence graph. Finally, we have established a concurrency control model of the web service transactions to conduct an ACoDe protocol simulation based on a randomly colored Petri Net and compared the model with the standard framework protocol Business Agreement with Coordinator Completion to evaluate the performance of ACoDe. Experimental results show that the new protocol can significantly improve the transaction success rate and reduce the average response time through alternate transactions.

ACKNOWLEDGMENTS

This research was supported by the National Natural Science Foundation of China under Grant No. 61672102, No. 61073034 and No. 61370064; the Program for New Century Excellent Talents in the University of Ministry of Education of China under Grant No. NCET-10-0239; the Science Foundation of Ministry of Education of China and China Mobile Communications Corporation under Grant No. MCM20130371; and the Open Project Sponsor of Beijing Key Laboratory of Intelligent Communication Software and Multimedia under Grant ITSM201493. Corresponding author. Tel: +86 13121915269. E-mail address: ddepeng@bnu.edu.cn (D. Dang)

REFERENCES

1. D. Guinard, S. Karnouskos, D. Savio, *et al*., "Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Transactions on Services Computing*, Vol. 3, 2010, pp. 223-235.
2. A. Segev and Q. Sheng, "Bootstrapping ontologies for web services," *IEEE Transactions on Services Computing*, Vol. 5, 2012, pp. 33-44.
3. A. Barker, D. Robertson, and C. Walton, "Choreographing web services," *IEEE Transactions on Services Computing*, Vol. 2, 2009, pp. 151-166.
4. M. Abdelkader, S. Benslimane, and M. Mimoun, "Locating candidate web service in legacy software: A search based approach," in *Proceedings of International Conference on Information Technology and e-Services*, 2012, pp. 1-6.
5. Y. Badr, N. Faci, Z. Maamar, *et al.*, "Using social networks for web services discovery," *IEEE Internet Computing*, Vol. 15, 2011, pp. 47-53.
6. I. Giannoukos, V. Loumos, I. Lykourentzou, *et al.*, "Web-based decision-support system methodology for smart provision of adaptive digital energy services over

cloud technologies," *IET Software*, Vol. 5, 2011, pp. 444-454.
7. V. Altmann, F. Golatowski, J. Skodzik, and D. Timmermann, "Investigation of the use of embedded web services in smart metering applications," in *Proceedings of the 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 6072-6077.
8. J. Y. Gong, L. P. Di, N. C. Chen, *et al.*, "A flexible data and sensor planning service for virtual sensors based on web service," *IEEE Sensors Journal*, Vol. 11, 2011, pp. 1429-1439.
9. K. Don and S. H. Son, "QoS management in Web-based real-time data services," in *Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2002, pp. 129-136.
10. Z. Shelby, "Embedded web services," *IEEE Wireless Communications*, Vol. 17, 2010, pp. 51-56.
11. D. K. Chen, "A context-aware recommender system for web service composition," in *Proceedings of the 8th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2012, pp. 227-229.
12. A. C. Pathan and M. A. Potey, "Detection of malicious transaction in database using log mining approach," in *Proceedings of International Conference on Electronic Systems*, *Signal Processing and Computing Technologies*, 2014, pp. 262-265.
13. C. S. Ran and G. L. Guo, "The research of real-time performance in EAM system based on web services," in *Proceedings of International Conference on Computer Science and Electronics Engineering*, 2012, pp. 44-46.
14. F. Andrade, A. Jose, R. Bertagna, *et al*., "Exigency-based real-time scheduling policy to provide absolute QoS for web services," in *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*, 2007, pp. 254-262.
15. C. J. Hu, J. Liu, and H. J. Yuan, "Application of web services on the real-time data warehouse technology," in *Proceedings of International Conference on Advances in Energy Engineering*, 2010, pp. 335-338.
16. A. Eggen, T. Hafs, F. T. Johnsen, *et al*., "Robust web services in heterogeneous military networks," *IEEE Communications Magazine*, Vol. 47, 2010, pp. 78-83.
17. A. Eggen, C. Griwodz, T. Hafs, *et al.*, "Web services discovery across heterogeneous military networks," *IEEE Communications Magazine*, Vol. 47, 2010, pp. 84-90.
18. L. Kulik, "Privacy for real-time location-based services," *ACM SIG Spatial*, Vol. 1, 2009, pp. 9-14.
19. F. Golatowski, G. Moritz, S. Pruter, and D. Timmermann, "Web services on deeply embedded devices with real-time processing," in *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 432-435.
20. X. X. Wu and Y. X. Chen, "Trustworthiness expectation of real-time web services," in *Proceedings of the 7th IEEE International Conference on Ubiquitous Intelligence and Computing*, 2010, pp. 292-298.
21. L. Lin and P. Lin, "Orchestration in web services and real-time communications," *IEEE Communications Magazine*, Vol. 44, 2007, pp. 44-49.
22. R. Deters, S. Jamal, and R. Lomotey, "SOPHRA: A mobile web services hosting infrastructure in health," in *Proceedings of the 1st IEEE International Conference on Mobile Services*, 2012, pp. 88-95.

23. C. Capua, A. Meduri, and R. Morello, "A smart ECG measurement system based on web-service-oriented architecture for telemedicine applications," *IEEE Transactions on Instrumentation and Measurement*, Vol. 58, 2010, pp. 2520-2528.
24. T. Bubhaus, S. Fischer, and D. Gregorczyk, "A proof of concept for medical device integration using web services," in *Proceedings of the 9th International Multi-Conference on Systems*, *Signals and Devices*, 2012, pp. 1-6.
25. A. Ahmadi, R. Bakhshi, A. Elci, *et al.*, "Microcontroller-based AWGNG for security enhancement of embedded real-time web services," in *Proceedings of the 33rd IEEE Annual International Conference on Computer Software and Applications*, 2009, pp. 110-115.
26. M. E. Cambronero, G. Diaz, Pardo, and V. Valero, "Using UML diagrams to model real-time web services," in *Proceedings of the 2nd International Conference on Internet and Web Applications and Services*, 2007, pp. 24-29.
27. B. Kawtar, N. Manuel, R. Carlos, *et al.*, "Real-time web services orchestration and choreography," in *Proceedings of the 6th International Workshop on Enterprise and Organizational Modeling and Simulation*, 2010, pp. 142-152.
28. A. J. Beaumont, M. J. Eccles, and D. J. Evans, "True real-time change data capture with web service database encapsulation," in *Proceedings of the 6th IEEE World Congress on Services*, 2010, pp. 128-131.
29. C. A. Sun, E. Khoury, and M. Aiello, "Transaction management in service-oriented systems: Requirements and a proposal," *IEEE Transactions on Services Computing*, Vol. 4, 2011, pp. 167-180.
30. K. Ramamritham, "Real-time databases," *Distributed and Parallel Databases*, Vol. 1, 1993, pp. 199-226.
31. J. Haritsa and K. Ramamritham, "Real-time databases in the new millennium," *Real-Time Systems*, Vol. 19, 2000, pp. 205-208.
32. E. Kayan and O. Ulusoy, "Real-time transaction management in mobile computing systems," in *Proceedings of the 6th International Conference on Database Systems for Advanced Applications*, 2002, pp. 127-134.
33. D. P. Dang and Y. S. Liu, "Concurrency control in real-time broadcast environments," *Journal of Systems and Software*, Vol. 68, 2003, pp. 137-144.
34. D. Georgakopoulos and M. P. Papazoglou, *Service-Oriented Computing*, The MIT Press, UK, 2009.
35. Mark Little, "Transactions and web services," *Communications of the ACM*, Vol. 45, 2003, pp. 48-53.
36. S. Agarwal and C. Petrie, "An alternative to the top-down semantic web of services," *IEEE Internet Computing*, Vol. 16, 2012, pp. 94-97.
37. A. Bouguettaya, J. Yang, W. Zhao, and H. Zheng, "QoS analysis for web service compositions with complex structures," *IEEE Transactions on Services Computing*, 2012, pp. 99-101.
38. A. Andrekanic and R. Gamble, "Architecting web service attack detection handlers," in *Proceedings of the 19th IEEE International Conference on Web Services*, 2012, pp. 130-137.
39. A. Mourad, H. Otrok, C. Talhi, *et al.*, "Towards a BPEL model-driven approach for web services security," in *Proceedings of the 10th Annual International Conference on Privacy*, *Security and Trust*, 2012, pp. 120-127.

40. P. Giani, M. Lovera, and M. Tanelli, "Linear parameter-varying model identification with structure selection for autonomic web service systems," *Control Theory and Applications*, Vol. 6, 2012, pp. 1889-1898.
41. C. J. Hu, J. Liu, and H. J. Yuan, "Application of web services on the real-time data warehouse technology," in *Proceedings of International Conference on Advances in Energy Engineering*, 2010, pp. 335-338.
42. A. Eggen, T. Hafs, F. T. Johnsen, *et al*., "Robust web services in heterogeneous military networks," *IEEE Communications Magazine*, Vol. 47, 2010, pp. 78-83.
43. A. Eggen, C. Griwodz, T. Hafs, *et al*., "Web services discovery across heterogeneous military networks," *IEEE Communications Magazine*, Vol. 47, 2010, pp. 84-90.
44. T. Bubhaus, S. Fischer, and D. Gregorczyk, "A proof of concept for medical device integration using web services," in *Proceedings of the 9th International Multi-Conference on Systems*, *Signals and Devices*, 2012, pp. 1-6.
45. A. Ahmadi, R. Bakhshi, A. Elci, *et al*., "Microcontroller-based AWGNG for security enhancement of embedded real-time web services," in *Proceedings of IEEE 33rd Annual International Conference on Computer Software and Applications*, 2009, pp. 110-115.
46. C. S. Wang and M. J. Tsai, "Adaptive real-time computation coordination for the web services based computing architecture," in *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design*, 2007, pp. 552-556.
47. M. E. Cambronero, G. Diaz, Pardo, and V. Valero, "Using UML diagrams to model real-time web services," in *Proceedings of the 2nd International Conference on Internet and Web Applications and Services*, 2007, pp. 24-29.
48. I. Saleh, G. Kulczycki, and M. B. Blake, "Formal specification and verification of transactional service composition," in *Proceedings of IEEE World Congress on Services*, 2011, pp. 464-471.
49. Y. Cardinale, J. E. Haddad, M. Manouvrier, and M. Rukoz, "CPN-TWS: a coloured petri-net approach for transactional-QoS driven Web Service composition," *International Journal of Web and Grid Services*, Vol. 7, 2011, pp. 91-115.
50. J. Hadad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition," *IEEE Transactions on Services Computing*, Vol. 3, 2010, pp. 73-85.
51. F. Montagut and R. Molva, "Augmenting web services composition with transactional requirements," *IEEE International Conference on Web Services IEEE Computer Society*, 2009, pp. 91-98.
52. OASIS, "Business transaction protocol," http://www.oasis-open.org/committees/documents.php?wg_abbrev=business-transaction.
53. IBM, Microsoft and BEA, "Web services transactions specifications," http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/.
54. Arjuna Technologies Ltd., Fujitsu Software, "IONA Technologies PLC, Oracle Corp., and Sun Microsystems," Web Services Composite Application Framework, http://developers.sun.com/techtopics/webservices/wscaf.
55. M. Schafer, P. Dolg, and W. Nejdl, "An environment for flexible advanced compensations of web service transactions," *ACM Transactions on the Web*, Vol. 2, 2008, pp. 1-36.

56. A. Liu, L. S. Huang, and M. J. Xiao, and Q. Li, "FACTS: A framework for fault-tolerant composition of transactional web services," *IEEE Transactions on Services Computing*, Vol. 3, 2010, pp. 45-58.
57. K. Haller, H. Schuldt, and C. Türker, "Decentralized coordination of transactional processes in peer-to-peer environments," in *Proceedings of ACM CIKM International Conference on Information and Knowledge Management*, 2005, pp. 28-35.
58. S. Fink, M. Husemann, and N. Ritter, "Rule-based coordination of distributed web service transactions," *IEEE Transactions on Services Computing*, Vol. 3, 2010, pp. 59-72.
59. M. Alrifai, W. Balke, and P. Dolog, "Distributed management of concurrent web service transactions," *IEEE Transactions on Services Computing*, Vol. 2, 2009, pp. 289-302.
60. M. Alrifai, P. Dolog, and W. Nejdl, "Transactions concurrency control in web service environment," in *Proceedings of IEEE European Conference on Web Services*, 2006, pp. 109-118.
61. S. Choi, H. Jang, H. Kim, J. Kim, S. Kim, J. Song, and Y. Lee, "Maintaining consistency under isolations relaxation of web services transactions," in *Proceedings of International Conference on Web Information Systems Engineering*, 2005, pp. 245-257.

De-Peng Dang (党德鹏) received his Ph.D. degree in Computer Science and Technology from Huazhong University of Science and Technology, China, in 2003. From June 2003 to June 2005, he did his postdoctoral research in the Department of Computer Science and Technology, Tsinghua University, China. Now, he is a Full Professor and supervisor of Ph.D. students in Computer Science and Technology at Beijing Normal University, China. Up to now, he has chaired Four NSFC projects. His research interests include service computing, service recommendation, big data and MapReduce.

Xue Jiang (姜雪) is currently studying at the College of Information Science and Technology, Beijing Normal University, China. She has received her Bachelor's degree in Computer Science and Technology from Beijing Normal University. Her research interests include service computing and advanced transactions.

Nan Wang (王楠) has received her Bachelor's degree in Computer Science and Technology from Beijing Normal University. She is currently studying at the College of Information Science and Technology, Beijing Normal University, China. Her research interests include service computing, big data and advanced transactions.

Ying-Ting Yao (姚颖婷) has received her Bachelor's degree in Computer Science and Technology from Beijing Normal University. She is currently studying at the college of Information Science and Technology, Beijing Normal University, China. Her research interests include service computing, service recommendation and big data.

Wen-Bin Yao (姚文斌) received his Ph.D. degree in Computer Science and Technology from Harbin Institute of Technology University, China, in 2003. From 2003 to 2005, he did his postdoctoral research in the Department of Computer Science and Technology, Tsinghua University, China. Now, he is a Full Professor and supervisor of Ph.D. students in Computer Science and Technology at Beijing Key Laboratory of Intelligent Communication Software and Multimedia, China. His research interests include service computing, advanced transactions, big data and MapReduce.