Testing Long-lived Web Services Transactions Using a Risk-based Approach

Ruben Casado, Javier Tuya
University of Oviedo
Spain
rcasado@lsi.unovi.es, tuya@uniovi.es

Muhammad Younas
Oxford Brookes University
Oxford, UK
m.younas@brookes.ac.uk

***Abstract*—Transactions are crucial to ensuring the quality (such as recovery and reliability) of web services applications by constraining them to a mutually agreed outcome. This paper addresses the issue of testing the long-lived web services transactions which has been given little attention by the current research. It proposes a risk-based approach and also defines a set of properties for web services transactions. The proposed approach identifies for each property a set of potential situations that must be tested. We present an analysis for the Recovery property using a Fault Tree diagram where the leaf nodes represent potentially dangerous scenarios that must be checked. Finally we show with a case study how this Fault Tree can be used to derive test cases for web services transactions.**

Long-lived transactions; web service testing; risk-based testing

I. INTRODUCTION

Web Services (WS) provide a paradigm for developing Internet-based applications using standard technologies and protocols and enable standard means of communication and collaboration between web applications running on a variety of platforms. Web services transactions are used to build reliable web-based applications. The fundamental principle of WS transactions is to ensure that all the component WS achieve a mutually agreed outcome. Transactions have been traditionally based on the ACID model which enforces strict isolation and atomicity properties and the locking of resources. WS transactions are generally complex and of long duration as they involve multiple parties and span many organizations. Thus ACID model does not fit well to the nature of WS transactions. In order to deal with such transactions, various extended transaction models have been adapted for WS. These models mainly relax the atomicity and isolation policy of ACID properties [1].

In order to manage long-lived WS transactions several standard specifications have been proposed [2,3,4], including Web Service Coordination (WS-COOR) [5], Web Service Atomic Transactions (WS-AT) [6] and Web Service Business Activity (WS-BA) [7]. However, there are no practical works on testing WS transactions [8]. Further, [9], [10], and [11] develop approaches for testing long-lived transactions but these works are focused on theoretically verifying the transactions properties.

This paper addresses the issue of testing the WS transactions. The objective is to define criteria for testing the transactions that comply with WS-COOR and WS-BA standards as these are the most recent and widely accepted standards. Our approach applies a risk analysis method to a set of properties such as Composition, Sorting, Visibility, Consistency, Durability, Controllability, and Recovery in order to find possible failures during the WS transaction life-cycle. Contributions of this paper include (i) specification of notation for testing the behavior of long-lived WS transactions (ii) definition of a set of system properties (iii) a method for deriving test case specifications using a risk analysis of the recovery property, and (iv) practical case study which illustrates the proposed approach.

II. BACKGROUND

A. WS Transactions standards

WS-COOR standard defines protocols for distributing the coordination context of a transaction among participants. It specifies an interface of a transaction manager (or *coordinator*) for creating a new or joining an existing transaction [5]. Both WS-AT and WS-BA [6, 7] are built on top of WS-COOR. WS-BA coordinates long-running compensation-based activities that may consist of several atomic transactions.

Fig. 1 depicts the main states during the life cycle of a participant using WS-BA and the messages exchanged between the participant and the coordinator. WS-AT is used to coordinate ACID-based transactions.

B. Transaction characteristics

Based on the existing works [12] that address the new features of this kind of transactions, we have identified the important characteristics that should be tested.

A long-lived transaction is *composed* of subtransactions [13] that have autonomy to commit or abort unilaterally. So the atomicity is relaxed and isolation is violated because the results of the committed subtransactions are *visible* to other transactions. This new concept of atomicity (called semantic atomicity [14]) means that if any of the subtransactions is aborted then the effects of the committed transactions must be compensated [15].

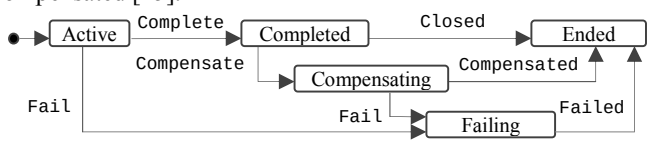


Figure 1. Ws-BusinessActivity abstract state diagram

Hence the system must be able to *recover* [16] its previous state maintaining the *consistency* [17]. Both the subtransactions

and compensations must be executed in the correct *order* [18] and to manage these operations a *coordinator* is required [19]. As in the ACID model, the *durability* of the results of a completed long-lived transaction is a desired property [20].

III. Conceptual Framework

In [21], we presented the hierarchical conceptual framework in order to test long-lived WS transactions. The input or level zero is the analysis of the transactions characteristics. First level defines the system properties as described in Section II.B. At the next level a risk analysis is applied for each property in order to identify potential failures and their possible causes. This analysis is carried out taking into account the WS transaction standards behavior. The third level states a set of test case specifications which are identified through previous risk analysis. The last level specifies the execution of the proposed tests. Our work in this paper concerns the *system properties, risk analysis* and *the specification of test cases*.

A. Definitions and Notations

1) A long-lived Web Service Transaction *(wT)* comprises a group of subtransactions that execute different web services *(participants)*. A *wT* is defined by $wT=<S, C, \Psi_I, \Psi_E, \Psi_C>$ where *S* is a set of subtransactions, *C* is a set of *compensatory actions,* Ψ_I is a set of states that define the requirements so that each subtransaction can be executed *(initial states),* Ψ_E is a set of states that define the requirements after each subtransaction is completed *(executed states)* and Ψ_C is a set of states that define the requirements after each subtransaction is compensated *(compensated states)*.

The set $S=\{s_1, ..., s_n\}$ defines the subtransactions where each s_i is an atomic transaction or another *wT'*.

Any subtransaction s_i has a *compensatory action* denoted by c_i. A c_i undoes, from a semantic point of view, the actions performed by s_i, but does not necessarily return the transaction to the state that existed when the execution of s_i began. The set of all compensatory actions is denoted by $C=\{c_1, ..., c_n\}$. Any c_i could be executed only if s_i has been completed. Any c_i may be an empty action, denoted by λ.

2) Participants and coordinator

A *participant* p_i is the agent responsible for executing the subtransaction s_i and its compensatory action c_i. A *transaction notification* is the communication between two participants of *wT*. The notation $i_{wT}[m_1]j_{wT}$ is used to denote that the participant p_i intervenes in the transaction *wT* and sends the message m_1 to another participant p_j which is also part of the transaction *wT*. If the participant is the coordinator, it is denoted by *K*. We use $i_{wT}[m_1]j_{wT} - l_{wT}[m_2]o_{wT} - ... - v_{wT}[m_n]z_{wT}$ to denote a sequence of transaction notifications.

A *Coordinator K* is the participant that manages the subtransactions of a *wT*. It executes different subtransactions, manages failures and compensations, and collects the results from the participants in order to provide system with a consistent state after the execution of a transaction.

3) Phases of a subtransaction

The *Initial state* σ_i is the necessary requirement so that the participant p_i can execute s_i. The set of all initial states is denoted by $\Psi_I=\{\sigma_1, ..., \sigma_n\}$.

The *Executed state* σ^*_i defines the requirement that the participant p_i has to satisfy once the s_i execution has correctly finished. The set of all executed states is denoted by $\Psi_E=\{\sigma^*_1, ..., \sigma^*_n\}$. The requirements specified in an executed state σ^*_i may be included in the necessary requirements to execute the next subtransaction s_{i+1}, specified in the initial state σ_{i+1}.

The *Compensated state* σ'_i defines the requirements that the participant p_i has to satisfy once the c_i execution has correctly finished. The set of all compensated states is denoted by $\Psi_C=\{\sigma'_1, ..., \sigma'_n\}$.

The *Compensatory action* c_i undoes, from a semantic point of view, the actions carried out by s_i. The system changes to σ'_i. This state is not necessarily equal to σ_i due to the impossibility of undoing some operations (e.g. a sent email).

We use $\sigma_i \xrightarrow{s_i} \sigma^*_i$ to denote that, starting from the initial state σ_i, the executed state σ^*_i is reached after the subtransaction s_i is completed and the next subtransaction(s) can be executed. In the same way, we use $\sigma^*_i \xrightarrow{c_i} \sigma'_i$ to denote that, stating from the executed state σ^*_i, the compensated state σ'_i is reached after the compensatory action c_i is completed and the next compensatory action(s) can be executed. Fig. 2 depicts these notations.

B. First Level: System Properties

In Table I, we define properties that are shared and applicable to all long-lived transaction-based WS applications. If any of these properties are infringed a failure could appear.

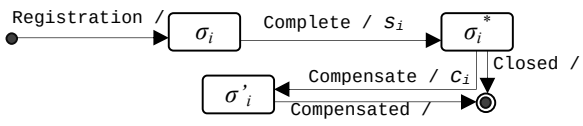
IV. Risk Analysis For The Recovery Property

A risk is defined as an undesirable event that, if it occurs, represents a threat to the correct behavior of a process [22]. Risk analysis is a set of techniques used to investigate problems created by uncertainty and to assess their effects. Originally it was used in areas such as the nuclear, chemical and space industries. Nowadays it is used in software development where safety is also very important [22].

In this section the Fault Tree Analysis [23] notation will be used to hierarchically organize the identified risks. In this technique there is a top event (the root node) that represents the general problem. This top event will be discomposed in the next levels generating a tree of potential causes. These causes are connected through logic gates. The leaf nodes represent specific causes that must be controlled. A circle is used to denote leaf nodes and a rectangle is used to denote intermediate nodes. A triangle represents a subtree in order to generate images which are easier to understand.

In our approach each system property will be a top event, therefore each one will generate a fault tree. The leaf nodes specify the potential problems that are necessary to test in order to avoid a wrong behavior of a *wT*. In this paper we present some details of the recovery property risk analysis. When we reach a leaf node in a branch it is used to generate a test case. It is illustrated with an example in Section V.B.

Fig. 3 depicts a small part of the fault tree for the Recovery (1) property in order to model the risks of the WS transaction compensatory mechanism, which includes: the compensatory action is not executed at all (2) or incorrectly executed (3) or it



fails due to the loss of messages between the participants and the coordinator (4). This could be due to problems with

participant messages (5) or coordinator messages (6). The problems related to participant are that it does not receive the compensation message (7), it receives the message when it should not receive this message (8) or the compensate message has finished with timeout (9). When a participant has problems receiving compensate messages from the coordinator, it may receive the message in an unsuitable state and wrongly execute the compensatory action. It means that the participant receives the message without executing its subtransaction (10), the participant receives a compensate message when it has finished its participation in the transaction (11), it receives the compensate message when it has already executed its compensatory action (12) or it receives the compensate message when it does not need to execute any compensatory action (13).

One of the possible situations identified in (10) is that a participant was in a failing state (14). This situation means that a participant *I* was registered in a transaction *wT* and then, while it was executing its subtransaction, the participant reaches the *failing* state because a failure happened. One possible situation in (11) is that a compensate message is received when the participant has executed its subtransaction and it was not necessary to execute the compensatory action, therefore the participant is in the *ended* state (15).

Based on the situations identified in the leaf nodes, we can define a test scenario for each one specifying the transaction notifications to reach this situation. For example, the sequence *I [register]K - K[complete]I – I[fail]K – K[compensate]I* represents the test scenario for the situation specified in (14). Test case for this scenario is shown in Section V.B.

Please note that only one branch of the large fault tree has been presented. In the current version the whole fault tree consists of 26 leaf nodes that represent potentially dangerous situations that should be checked.

V. THE TRAVEL AGENCY EXAMPLE

This section presents the advantages of proposed approach using a travel Agency example in which customers are offered

TABLE I. SYSTEM PROPERTIES

Composition	A *wT* is composed of subtransactions. That is, $S=\{s_1, \ldots, s_n\}\ S \in wT$
Sorting	This shows that subtransactions can be executed in parallel or serially by satisfying the *wT* specification. The notation used is $O(S)=\{s_i[;\|]\ \ldots[;\|]_{sn}\}$
Visibility	A *wT* allows other (sub)transactions to see the partial results of its subtransactions.
Durability	When the transaction is finished successfully the results will remain permanent in the system
Consistency	Subtransactions or their compensating transactions must maintain the required consistency of web services
Recovery	A subtransaction can be undone executing its compensatory action. c_i. It moves the system from the state to the previous state σ_i. So any *wT* can be undone reaching an initial equivalent state if all executed subtransactions are compensated.

Controllability	This requires that the coordinator has to ensure the Composition, Consistency, Durability and Recovery properties of the transactions.

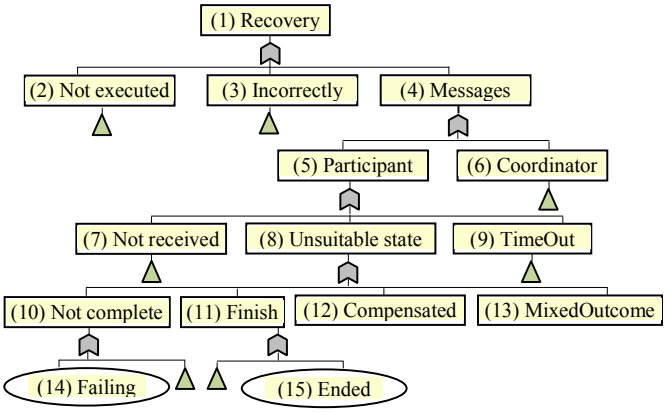


Figure 3. Recovery Fault Tree

with the facilities for making travel arrangements as follows. The *Agency* service receives an itinerary from a customer. After checking the itinerary for errors, the process determines which reservations to make, sending simultaneous requests to the appropriate airline, hotel and car rental agencies. If any of the reservation tasks fails, the itinerary is canceled by performing the compensatory action and the customer is notified of the problem. *Agency* service waits for confirmation of the three reservation requests. Upon receipt of confirmation, the *Agency* service notifies the customer and sends the reservation confirmation and final itinerary details and makes the payment.

A. Transaction Specification

wT_{agency} is defined below using the notation specified in Section III.A. Due to space limitation, we just show the necessary requirements to illustrate the test case derived from leaf node (14) (Fig. 3).

1) Definition

$wT_{agency}=S_{agency},C_{agency},\psi_{Iagency},\psi_{Eagency},\psi_{Cagency}$

$Participants=\{Itinerary,Airline,Hotel,Car,Bank,Email,Coordinator\}$

$S_{agency}=\{s_{itinerary},s_{airline},s_{hotel},s_{car},s_{bank},s_{email}\}$

$C_{agency}=\{c_{itinerary},c_{airline},c_{hotel},c_{car},c_{bank},c_{email}\}$

$\psi_{Iagency}=\{\sigma_{itinerary},\sigma_{airline},\sigma_{hotel},\sigma_{car},\sigma_{bank},\sigma_{email}\}$

$\psi_{Eagency}=\{\sigma^*_{itinerary},\sigma^*_{airline},\sigma^*_{hotel},\sigma^*_{car},\sigma^*_{bank},\sigma^*_{email}\}$

$\psi_{Cagency}=\{\sigma'_{itinerary},\sigma'_{airline},\sigma'_{hotel},\sigma'_{car},\sigma'_{bank},\sigma'_{email}\}$

2) Subtransactions

$s_{itinerary}$= To check the dates, places and customer information. It decides the appropriate airline, hotel and car rental agencies to make the reservations.

$s_{airline}$= To make the flight reservation using the itinerary information.

s_{hotel}= To make the hotel reservation using the itinerary information.

3) Compensatory actions

$c_{itinerary}=\lambda$

c_{hotel}= To cancel the hotel reservation.

4) Initial states

σ_{hotel}= The hotel service has received the dates and destination of the travel.

5) Executed states

σ^*_{hotel}= The room is booked in the hotel system according to the itinerary.

6) Compensated states

$\sigma'_{itinerary}=\sigma^*_{itinerary}$

σ'_{hotel}= The room cancellation is registered in the hotel system. If the cancellation was executed by the agency two days before the first reservation day, or the cancellation was executed by the hotel company, the agency has received the amount paid. If the cancellation was carried out by the agency one day before the first reservation day, the agency has received 50% of the amount paid.

B. Test Cases Using the Recovery Fault Tree

In this section a specific test is defined for the scenario shown in the fault tree (leaf node (14), Fig. 3). The failure scenario refers to a participant that received a compensate message when it is in a *failing* state. This means that a problem has occurred while the participant was executing its subtransaction so that all its functionality could not have been completed. Therefore if it receives a compensate message and executes its compensatory action a failure will occur because some features of the subtransaction will be undone when they had not been carried out. In the *Agency* example there are four participants involved in the transaction so this scenario must be checked for each participant. An example of test case definition for the *Hotel* participant following the previous scenario is defined is shown in Table II.

If the implementation accepts the compensation message, a failure appears. In this example the consequence would be the loss of money for the *Hotel* company. Assume that the *Hotel* had a problem while it was making the reservation so it did not execute the charge. But if it receives the compensate message and executes its compensatory action the *Hotel* will return to the customer the money that he/she had supposedly paid.

VI. Conclusions and Future Work

We presented a new approach to test long-lived transactions in WS environments. Risk-based techniques are applied in order to identify failures. In this paper, we have shown the preliminary results including specific notations and a set of system properties. We have also developed a risk analysis of one of these properties, Recovery, using a Fault Tree diagram. We have shown how to use our approach in order to generate test cases specifications with an example. Our future work is to validate the test cases in a real implementation.

Acknowledgment

This work was partially funded by the Department of Science and Technology (Spain) and ERDF funds under the National Program for Research, Development and Innovation, project Test4SOA (TIN2007-67843-C06-01, grant BES-2008-004355

TABLE II. Test Case Using The Fault Tree Analysis

Preconditions	wT_{agency} is correctly initialized.
Input sequence	$Hotel_{agency}$ *[register]K* *K[complete]*$Hotel_{agency}$ $Hotel_{agency}$*[fail]K* *K[compensate]*$Hotel_{agency}$
Expected output	*Hotel* ignores the compensate message so does not execute c_{hotel}. Thus the system remains in σ_{hotel} so σ^{*}_{hotel} is not met. *Hotel* still waits for the *failed* message.

References

[1] M. Younas, K. Chao, C. Lo and Y. Li, "An efficient transaction commit protocol for composite web wervices," 20th Int. conf. on Advanced Information Networking and Applications, 2006, vol. 1

[2] Business Transaction Protocol. [Online]. Available: http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf

[3] Web Services Composite Application Framework. [Online]. Available: http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html

[4] WS-BPEL. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[5] Web Service Coordination. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html

[6] Web Service Atomic Transactions. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html

[7] Web Service Business Activity. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os/wstx-wsba-1.2-spec-os.html

[8] G. Canfora and M. Di Penta, "Service-Oriented architectures testing: a survey," Lecture Notes in Computer Science, 2009, vol. 5413

[9] R. Lanotte, A. Maggiolo-Schettini, P. Milazzo, A. Troina, "Design and verification of long-running transactions in a time framework," Science of Computer Programming, 2008, vol. 73, pp. 76-94.

[10] M. Emmi, R. Majumdar, "Verifying compensating transactions," Lecture Notes in Compute Science, 2007, vol. 4349, pp. 29-43.

[11] J. Li, H. Zhu, J. He, "Specifying and verifyng web transactions," 28th IFI WG 6.1 Int. conf. on Formal Techniques for Networked and Distributed Systems, 2008, pp. 149-168

[12] M. Younas, B. Eaglestone and R. Holton, "A review of multidatabase transactions on the web: from the ACID to the SACReD," Lecture Notes In Computer Science, 2000, vol. 1832, pp. 140-152.

[13] E. B. Moss, "Nested transactions: an approach to reliable distributed computing," Massachusetts Institute of Technology, Technical report TR-260, 1981

[14] M. Younas, B. Eaglestone, R. Holton "A formal treatment of a SACReD protocol for multidatabase web transactions," LNCS, 2000, vol. 1873, pp.899-908.

[15] H. Garcia–Molina and K. Salem, "Sagas," ACM SIGMOD Record, 1987, vol. 16, pp. 249-259.

[16] M. Schäfer, P. Dolog, W. Nejdl, "An environment for flexible advanced compensations of web service transactions," ACM Transactions on the Web, 2008, vol. 2, issue 2.

[17] S. Choi, H. Kim, H. Jang, J. Kim, S. M. Kim, , J. Song, and Y. Lee, "A framework for ensuring consistency of Web Services Transactions," Information and Software Technology, 2008, vol. 50, pp.684-696.

[18] N. Lakhal, T. Kobayashi, H. Toyota, "FENECIA: failure endurable nested-transaction based execution of composite Web services with incorporated state analysis," V LDB Journal, 2009, vol. 1, pp. 1-56.

[19] C. Guidi, R. Lucchi, and M. Mazzara, "A formal framework for web services coordination," Electronic Notes in Theoretical Computer Science, 2007, vol. 180, pp. 55-70.

[20] S. Ehikioya, K. Barker, "A formal specification strategy for electronic commerce," IDEAS Symposium, 1997, pp.201.

[21] R. Casado, J. Tuya, "Testing transactions in service oriented architectures," 9th Int. conf. on Web Engineering, DC, 2009.

[22] J. Bennett, G. Bohoris, E. Aspinwall and R. Hall, "Risk analysis techniques and their application to software development", European Journal of Operational Research, 1996, vol. 95, pp. 467-475.

[23] W. Vesley, F. Goldberg, N. Roberts and D. Haasl, "Fault tree handbook," NUREG-0492, U.S. Nuclear Regulatory Commission, 1981.