Testing the Reliability of Web Services Transactions in Cooperative Applications

Rubén Casado, Javier Tuya
Department of Computing
University of Oviedo
Gijón, Spain
rcasado@lsi.uniovi.es, tuya@uniovi.es

Muhammad Younas
Computing and Communication Technologies
Oxford Brookes University
Oxford, United Kingdom
m.younas@brookes.ac.uk

ABSTRACT
Web services provide a distributed computing environment wherein service providers and consumers can dynamically interact and cooperate on various tasks in different domains such as e-business, education, government and healthcare. Transaction management technology is fundamental to building automated and reliable web services applications. Various models and protocols have been developed for web services transactions. However, they give no attention to the key issue of testing the web services transactions. We propose a novel abstract model for dynamically modeling distinct web services transaction standards and test their reliability in terms of failures. The proposed approach exploits model-based testing techniques in order to automatically generate test scenarios for web service transactions.

Categories and Subject Descriptors
D.2.2 [Design Tools and Techniques]: State diagrams
D.2.5 [Testing and Debugging]: Distributed, testing tools

Keywords
Transactions; web services; testing; cooperative applications

1. INTRODUCTION
Web services provide standard means of communication and interoperation between software applications distributed across the Web. The flexible and cooperative nature of web services allows for composite web services that provide enhanced functionality and cooperation among service providers and consumers. Consider an example of a composite web service such as web travel agency (WTA) application that composes different services (e.g., flight, hotel, and car rental) in order to enable interaction and cooperation among different service providers and consumers. Consumers can use this as a one stop service for booking a flight, hotel and car. While service providers (airline, hotels, and car company) can integrate their services in order to cooperate on business deals as well as offer cheaper and better services.

In order to ensure reliable interaction and cooperation between web services it is crucial that their activities are modeled as transactions such that they achieve a mutually agreed outcome. In order to achieve such outcome various transaction models and standards have been adapted for web services (WS) transactions [1].
Though existing approaches investigate into testing of web service integration, they fall short of testing web services transactions, for instance, in terms of reliability and failures [2]. The process of testing WS transactions is not trivial due to several reasons. First, WS transactions are more complex compared to classical transactions as they involve cooperation among multiple parties, span autonomous and independent organizations, and may have long duration. Thus WS transactions have more intricate sequence of operations and execution environment. Second, WS transactions do not have a homogeneous transaction model such as the ACID (Atomicity, Consistency, Isolation, Durability) model. Instead they are characterized by a diversity of transaction models such as BTP [3], WS-BA [4], WS-TXM [5]. Such diversity of models also complicates the process of testing WS transactions. Third, various kinds of failures may happen during the processing of WS transactions. In the above example, WTA application may suffer from several failures: (i) technical failures such as communication, system and software failures can occur. Such failures result in loss of messages, processing of services, etc. (ii) service acquisition failures may happen - if no flights are available, should the vehicle and hotel reservations be canceled? (iii) what happens if there is a problem with the payment process after the reservations were made?

All the above failures affect the reliability of WS transactions. Thus it is important to have an *abstract (generic) model* in order to analyze different transaction models, generate test case specifications and test their reliability in terms of failures. In this paper we propose an abstract transaction model that serves as a template for modeling and testing different WS transactions standards. The contributions of the work presented in this paper are summarized as follows:

- *To analyse existing transaction models such as ACID models, Advanced Transaction Models (ATM) [6] and WS transactions*. This paper gives a comparative evaluation of the existing standards developed for WS transactions. The rationale behind such analysis and evaluation is to identify the most widely used WS transaction standards for which templates (instances) of the abstract model can be automatically generated and tested in terms of their reliability to failures.
- *To provide a high level and generic template for dynamically modeling existing WS transactions standards*: The abstract model identifies the different roles involved in WS transactions, the relationship between them and models the behavior of each one during the transaction life cycle. The sequence of messages from the abstract model can be automatically translated to a particular syntax of a WS transaction standard. The abstract model provides a high level view of modeling WS transactions.
- *To automatically generate test scenarios in order to test WS transactions standards*: The abstract transaction model automatically generates specific test scenarios for various WS transaction standards and their processing tasks. It adapts the

standard testing techniques of transition coverage for generating test scenarios.

The rest of the paper is organized as follows. Section 2 gives an analysis of WS transaction models and standards. Section 3 presents the proposed abstract transaction model. Section 4 illustrates the process of modeling WS transaction standards. Section 5 presents model-based testing of WS transactions. Section 6 reviews existing work. Conclusions and future work are presented in Section 7.

2. WS TRANSACTIONS STANDARDS

Conventional transaction models follow ACID properties that maintain strict consistency and isolation of data sources [4]. Two Phase Commit (2PC) protocol and its variants [6, 7] have commonly been used for maintaining ACID properties in distributed databases [5]. 2PC protocols implementing ACID properties are vital for transactions (requiring strict data consistency) but they are not suitable for long running applications due to resource locking/blocking problems. Advanced transaction models (ATMs) have been developed in order to address 2PC and ACID related issues. These includes, nested transaction model [7], SAGA model [8], open-nested [9], Split-join [10], Contracts [11], Flex [12], and WebTram [13]. The underlying strategy of these models is to allow compensation of partially completed transactions in order to maintain data consistency and reliability. For instance, if a seat cannot be booked in a flight then the completed hotel reservation should be cancelled through a compensating transaction.

Based on the above transaction models several standard specifications have been developed for WS transactions. For instance, Business Transaction Protocol (BTP) [3] adapts 2PC for short lived transactions and nested transaction model for long-lived transactions. Web Services Composite Application Framework (WS-CAF) [5] is a set of WS specifications for applications composed of multiple Web Services Transaction Management (WS-TXM). WT-TXM defines three models, TXACID, TXLRA and TXBP that address different scenarios. Web Service Atomic Transactions (WS-AT) [14] and Web Service Business Activity (WS-BA) [4] are built on top of Web Service Coordination (WS-COOR) [15] and they follow its coordination mechanism. WS-AT follows 2PC protocol while WS-BA uses the SAGA model.

The above standards are summarized and analysed in Table 1. '*Coordination'* represents whether a particular standard provides coordination facilities. '*Short'* and '*Long'* represent that the underlying models are respectively based on ACID or advanced transaction models. '*Related'* represents the remaining standards which belong to a same family. It is observed that all standards separate the coordination and the management of the subtransactions and also distinguish short-lived transactions from long-lived transactions. It is also observed that these standards have proprietary definitions of their underlying transaction models despite the fact they are based on similar concepts. This makes it difficult to use them in a uniform way. Our analysis shows that WS standards are not homogenous and they need different processing and testing requirements. Thus it is not practical (nor easier) to test a single WS transaction model and evaluate its reliability. Instead different WS transactions standards must be modeled and tested.

Table 1. WS transaction standards

Standards	Coordination	Short	Long	Related
BTP	✓	2PC	Nested	×
WS-CAF	×	×	×	WS-TXM
WS-TXM	✓	×	×	TXACD, TXLRA, TXBP
TXACID	×	2PC	×	WS-TXM
TXLRA	×	×	SAGA	WS-TXM
TXBP	×	×	Open	WS-TXM
WS-COOR	✓	×	×	WS-AT, WS-BA
WS-AT	×	2PC	×	WS-COOR
WS-BA	×	×	SAGA	WS-COOR

3. THE ABSTRACT TRANSACTION MODEL

The abstract model aims to model different WS transaction standards as discussed above. It is designed using the well-known *UML statecharts* notations which reflect the event-driven (message communication) nature of WS transactions. It defines a WS transaction, *wT;* a set of activities (or subtransactions) which are executed in order to achieve an agreed outcome in a WS application (as in the WTA example). Each *wT* has a set of subtransactions, $S=\{s_1,\ldots,s_n\}$. Each *wT* is associated with one Coordinator while each subtransaction, s_i, is executed by an Executor (as defined below). Each s_i could be a single level subtransaction or it may have nested subtransactions, *wT'*. In the proposed model, nested transactions are related in a *parent:child* relationship. Fig. 1 shows such relationship, wherein wT_p, is a parent of s_1, s_2 and s_3. s_1 (or wT_c) is in turn a parent of s_{c1} and s_{c2}. Subtransactions can have different types. A subtransaction is *lockable* if the resources (or data) that it uses can be locked until the completion of the parent transaction. A subtransaction is *compensatable* if its effect can be semantically undone through a compensating transaction. If a subtransaction is neither lockable nor *compensatable* then it is said to be *pivot*. Any *compensatable* subtransaction s_i has a *compensation* denoted by c_i that undoes, from a semantic point of view, the actions performed by s_i. A subtransaction is *retriable* if it can be re-executed without causing data inconsistency. A subtransaction is *replaceable* if there is an alternative that can perform the same task. The outcome of *wT* is called *atomic* if all its subtransactions are either successfully completed or compensated. Alternatively, if subtransactions can differ (some completed and some not), then the outcome is called *mixed*.

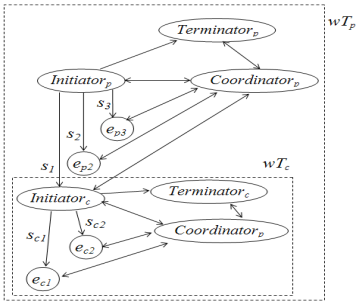

Figure 1. WS transaction relationships

The execution of a *wT* involves different participants, each of which plays a certain role. As shown in Fig. 2, we identify four different roles of the participants involved in processing *wT*:

- *Executor*: a participant responsible for executing and terminating a subtransaction.
- *Coordinator:* coordinates *wT* and manages failures and compensations. It also collects the results from the participants in order to maintain consistency of data after the execution of *wT*.
- *Initiator:* starts *wT*. First it requests the coordinator for a transaction context. Then it asks others participants to participate in *wT*.
- *Terminator:* decides when and how *wT* has to be terminated. It also participates in the coordination tasks. Thus it can be a subcoordinator.

The purpose of defining the above roles is to automatically model the roles of participants in different WS transactions standards.

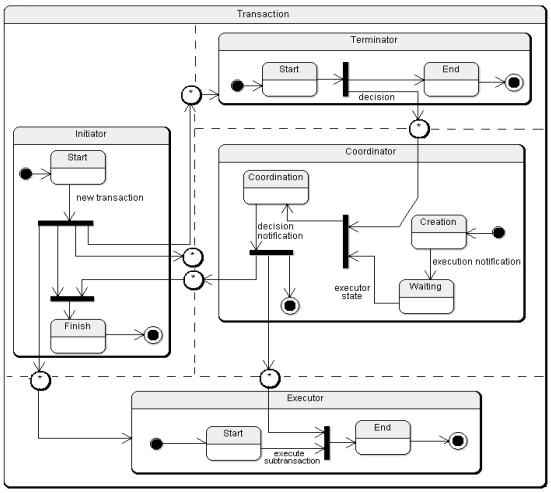

Figure 2. Participant roles

4. MODELING WS TRANSACTIONS STANDARDS

This section explains the process of how different WS standards can be modeled using the abstract model. A prototype tool has been developed that automatically performs the modeling as well as testing of the most widely used WS transactions standards; BTP and WS-BA (see [3, 4] for detailed specifications of BTP and WS-BA). The prototype tool, developed in Java Eclipse, implements the following three algorithms in order to perform the modeling of WS transaction standards:

- *Role identification and modeling*: it identifies the roles of participants in a target WS transaction standard and models it using the roles defined in the abstract transaction model.
- *State transitioning*: it captures the important states of the target WS transaction standard and maps them to the state transitions of the abstract transaction model.
- *Messages syntax*: it maps the messages of abstract transaction model to a specific WS transaction standard.

4.1 Business Transaction Protocol (BTP)

BTP allows coordinating multiple autonomous and cooperating services to ensure that the overall application achieves a consistent result (or agreed outcome). This consistency may be defined a priori (all the work is confirmed or none); or it can be determined according to the type of application (that may agree on partial completion of work).

4.1.1 Roles identificaction and modelling

This algorithm models the roles of the BTP participants involved in executing *wT* and its subtransactions (as defined in section 3). BTP implements nested transaction model [7], wherein a parent transaction, *wT*, is composed of subtransactions. BTP defines *Superior:Inferior* relationship between the parent and subtransactions. Fig. 3 depicts the modeling of BTP using the abstract transaction model. Fig. 3 (a) represents the BTP coordination of *wT* and its subtransactions using the *Superior:Inferior* relationship, and (b) represents the coordination of the same *wT* using the abstract transaction model. In BTP, the superior makes the decision and the inferior abides such decision in order to complete the transaction. In BTP, the *Superior:Inferior* relationship can be recursively extended to define a transaction tree having intermediates nodes as superior and inferior. The superior (of BTP) is modeled as *Initiator* (of the abstract model). Also the superior has to be modeled as *Coordinator* and *Terminator* as it decides on the outcome of the subtransactions. Inferior (in BTP) executes a subtransaction and is therefore modeled as *Executor* (in the abstract model).

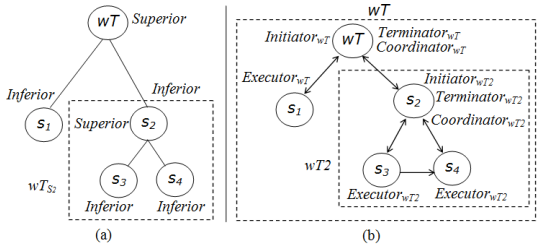

Figure 3. BTP modeling

4.1.2 State transitioning

Figures 4 and 5 show the states and transitions during the processing of *wT*. The abstract model uses these to model the BTP (as well as WS-BA) states and transitions. For instance, an Initiator starts *wT* that causes the creation of a context for a new transaction.

The coordinator replies the context and moves from *INITIAL* state to *ACTIVE* state. Executor receives a context, enrolls with the Coordinator and moves from *READY* to *ACTIVE* state. The Executor moves to *COMPLETED* state after processing its subtransaction. Coordinator moves to *PREPARE* state awaiting decisions from Executors. The Executor sends its outcome to Coordinator and moves to *DECISION* state. The Coordinator collects the outcomes from all Executors and takes the final decision. It moves from *PREPARE* state to *DECISION* state. The final decision is sent to each Executor and then the Coordinator moves to *CONFIRM* state. Executor sends acknowledgement and changes its state to *END* state. Once the coordinator has received all confirmation, it moves to *END* state. Note that an Executor can leave the *wT* before confirming the subtransaction. So it can move from *ACTIVE* state to *CANCEL* state.

Although BTP uses a 2PC protocol, Executors are not required to lock data in the prepared state. This can produce a contradicted decision as some Executors may take their own decisions that could contradict with the Coordinator's decision. When the Coordinator detects a contradiction it notifies the concerned Executor and moves to the *END* state. Further, BTP allows replaceable subtransactions. Thus if an Executor is not able to start or carry on with its subtransaction, it moves to *FAILED* state. A new Executor is selected and the previous one moves to *END* state.

4.1.3 Messages syntax

Table 2 represents the mapping of some of the messages between the abstract transaction model and the BTP. Though this table shows fewer messages the abstract transaction model can capture all the messages required to complete a BTP transaction.

Table 2. BTP message mapping

Abstract model	BTP
Creation	Initiator sends BEGIN to coordinator.
Execution	Initiator sends the context to executor and it sends ENROL to coordinator. It responses with ENROLLED. If the exeuctor is a superior of a new *wT*, it response with CONTEXT_REPLY.
Local commited	Coordinator sends PREPARE to executor. Due a protocol optimization, this transiction could be omitted.
Global commited	Executor sends PREPARED / CANCEL.

Completed sucesffully	Coordinator sends CONFIRM to executor and it responses with CONFIRMED.
Completed rollback	Coordinator sends CANCEL to executor and it responses with CANCELLED.
Preparing	It receives CONFIRM_TRANSACTION from the terminator and sends PREPARE to all executors.
Completed_rollback	Coordinator wants confirm but there is a contradiction. Coordinator sends CONTRADICTION to executor, and/or executor sends HAZARD to coordinator.
Completed_pivot	Coordinator cancels but there is a contradiction. Coordinator sends CONTRADICTION to executor, and/or executor sends HAZARD to coordinator.
Processing failure	The executor is not working. Coordinator knows it receiving a FAIL message or throw a non response message.

4.2 Web Services Business Activity (WS-BA)

WS-BA manages activities (transactions) that apply compensations to handle exceptions which occur during the execution of activities. WS-BA works with WS-COOR coordination protocol. WS-BA supports two coordination types, *MixedOutcome*, and *AtomicOutcome*, and two protocol types. The protocols types differ according to the participant's role in processing subtransactions; Executor (*BusinessAgreementWithParticipantCompletion, BAWPC*) or Coordinator (*BusinessAgreementWithCoordinatorCompletion, BAWCC*).

4.2.1 Roles identificaction

The role of *Initiator* is taken by the first participant who interacts with the Coordinator. In *MixedOutcome*, the Coordinator is the Terminator since each Executor may have its own decision. In *AtomicOutcome* the role of Terminator is taken by all the participants. This is due to the fact that if an Executor cancels its subtransaction, the whole transaction has to be canceled. Also the Coordinator acts as a Terminator since if all subtransactions have successfully confirmed, it has to notify all the Executors about the confirmation.

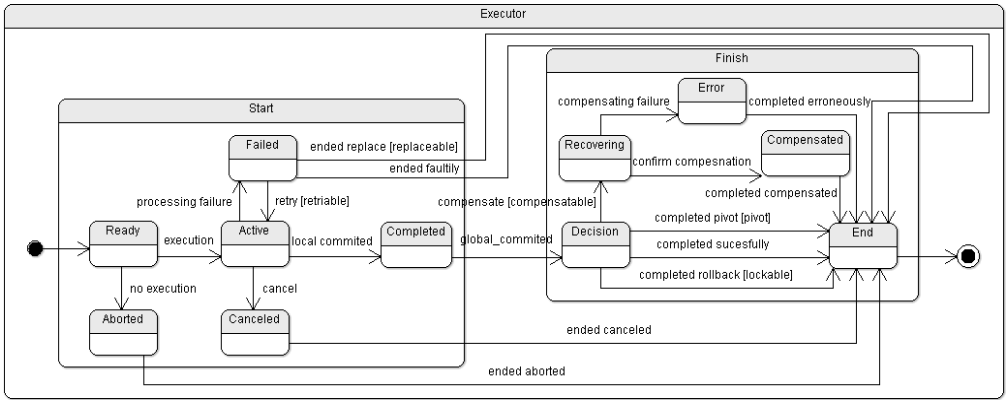

Figure 4. Executor states in the abstract model

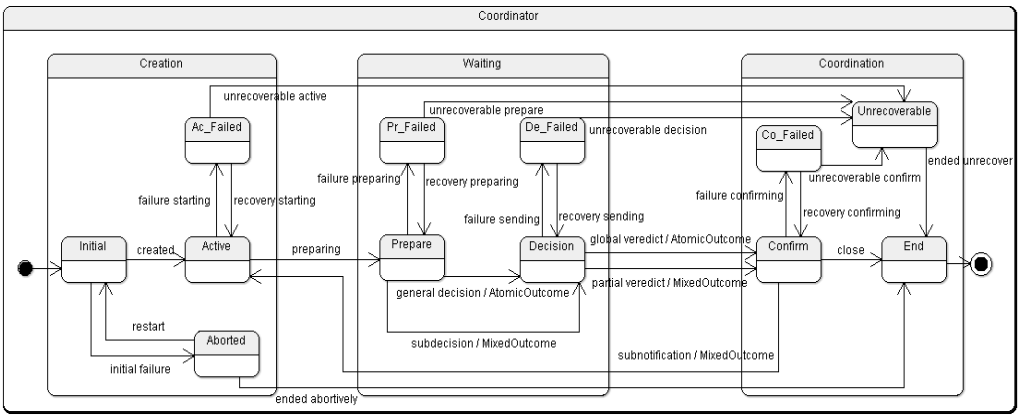

Figure 5. Coordinator states in the abstract model

4.2.2 State transitioning

Similar to BTP the abstract transaction model uses the state/transitions of Figures 4 and 5 to model WS-BA. The Initiator requests a context and moves from *START* to *FINISH*. The Coordinator responds with a context (from *INITIAL* to *ACTIVE* state). The context is sent to Executors by the Initiator. Each Executor joins the current *wT* and moves from *READY* to *ACTIVE* state. After making a decision an Executor moves from *ACTIVE* to *COMPLETED* state and the Coordinator moves from *ACTIVE* to *PREPARE* state. When the transaction is *mixed*, the decision for each subtransaction is taken alone. The Coordinator moves from *PREPARE* to *DECISION* state when it receives an Executor's notification. The Coordinator decides about its outcome and moves from *DECISION* to *CONFIRM*. The Coordinator receives the confirmation and goes back to wait for the rest of Executor's notifications (from *CONFIRM* to *ACTIVE* state). In the *atomic* type, the Coordinator moves from *PREPARE* to *DECISION* state when it has a global outcome about the transaction. The Coordinator then sends the global decision and moves from *DECISION* to *CONFRIM* state.

Finally it waits for the confirmations and moves to *END* state. When an Executor is not able to start executing its subtransaction it moves from *READY* to *ABORTED* state. If the subtransaction was cancelled while it was under execution, the Executor moves from *ACTIVE* to *CANCELLED* state. In case of failure it moves from *ACTIVE* to *FAILED* state.

4.2.3 Messages syntax

Table 3 presents the mapping of some of the messages between the abstract transaction model and the WS-BA. As stated above, the abstract transaction model can capture all the messages required to complete a WS-BA transaction.

Table 3. WS-BA message mapping

Abstract model	WS-BA
Creation	Initiator sends CREATECOORDINATIONCONTEXT to coordinator.
Execution	Each executor sends a REGISTER message to its chosen coordinator. The coordinator responses with a REGISTERRESPONSE message.
Local commited	If the coordination type is BAWCC, coordinator sends COMPLETE to executor. In the other coordination type this transition is omitted.
Global commited	Executor sends COMPLETED to the coordinator.
Completed sucesffully	Coordinator sends CLOSE to executor and it responses with CLOSED.
Global veredict	It is an AtomicOutcome and the coordinator sends CLOSE / COMPENSATE message for all completed executors.
Partial veredict	The coordinator sends CLOSE / COMPENSATE message to a specific executor.
Cancel	Participant sends CANCEL to coordinator.
Processing failure	Participant sends FAIL to coordinator.
Ended faultily	Coordinator sends FAILED to executor.

5. MODEL-BASED TESTING

The main goal of testing is to detect failures and to ensure reliability, i.e., to identify the observable differences between the behaviors of implementation and what is expected on the basis of the specifications of the WS transaction standards. We exploit the model-based testing that encodes the intended behavior of a system and the behavior of its environment. Model-based testing approach is capable of generating suitable test specifications. It has also been used in other WS environments [16].

We describe the process of how the abstract transaction model can be used to generate test scenarios for WS transactions. Since our model is based on states/transitions, we use the well known criterion of transition coverage [17]. The basic concepts used in definition of test scenarios are as follows:

Test criterion: A rule or collection that impose requirements on a set of test scenarios.

Transition coverage criterion: The set of scenarios that must include tests which cause transitions between states.

Abstract test scenario: A sequence of states and transitions of a participant using the abstract model. The notation $S_i \xrightarrow{t} S'_i$ is used to denote that the participant p_i changes its current state S to S' executing the transition labeled, t. If the participant is the Coordinator, it is denoted by K. We use $S^a_i \xrightarrow{t^x} S^b_i - \ldots - S^c_i \xrightarrow{t^y} S^d_i$ to denote a sequence of transitions.

Test scenario: A sequence of messages between participants using a specific WS transaction standard. The notation $i[m_1]j$ denotes that the participant p_i sends a message m_1 to participant p_j. We use $i\ [m_1]j - l_{wT}[m_2]o - \ldots - v\ [m_n]z$ to denote a sequence of messages.

Our prototype tool automatically obtains a set of test scenarios. It applies transition coverage criterion over the abstract model and obtains a set of independent paths. Each path defines an abstract test scenario. Thus the test scenarios reached using this criterion is the minimum set of independent paths that cover all states of a model. Table 4 illustrates an example of an abstract scenario for an Executor. The tool has generated six abstract test scenarios for each Executor and seven for each Coordinator. The tool also generates the mapping from the abstract test scenario to a specific test scenario (sequence of message using the syntax of BTP or WS-BA).

Table 4. Abstract test scenario

Abstract test scenario	Initial $\xrightarrow{creation}$ Ready - Ready $\xrightarrow{execution}$ Active – Active $\xrightarrow{local\_commited}$ Completed – Completed $\xrightarrow{global\_commited}$ Decision – Decision $\xrightarrow{completed\_sucesfully}$ End

As a proof of concept, we have used the tool with the WTA example. In this example there are four Executors (*Flight*, *Vehicle*, *Hotel* and *Payment)*, and one Coordinator (*WTA*), so thirty three test scenarios were automatically generated. Table 5 presents a test scenario for both WS-BA and BTP that are automatically generated using the abstract test scenario.

Table 5. Test scenario

WS-BA test scenario	Agency[CREATECOORDINATIONCONTEXT]K – K[CREATECOORDINATIONCONTEXTRESPONSE]Agency-Agency[CONTEXT]Hotel – Hotel[REGISTER]K – K[REGISTERRESPONSE]Hotel - K[COMPLETE]Hotel - Hotel[COMPLETED]K – K[CLOSE]Hotel – Hotel[CLOSED]K
BTP test scenario	Agency[BEGIN]K – K[BEGUN]Agency – Agency[CONTEXT]Hotel Hotel[ENROL]K - K[ENROLLED]Hotel –Hotel[PREPARED]K – K[CONFIRM]Hotel – Hotel[CONFIRMED]K

Based on the generated test scenarios we can test the failures and reliability of a particular WS transaction standard. We test the BTP and WS-BA transaction standards in terms of their execution of a WS transaction using the WTA case study. As an example, we test a situation where a coordinator does not send a notification to finish the subtransaction, say, executed by *Hotel.* The test scenario in Table 5 will pass using BTP as it does not need this kind of notification. That is the execution of a transaction under BTP will not result in failure. However, it will result in failure using WS-BA (*BAWCC*) standard. This is because WS-BA (BAWCC) needs a notification before sending its result to the Coordinator. In WS-BA (*BAWCC*) implementation, the Coordinator did not receive the confirmation from the Executor (related to *Hotel*) and thus it cancels the reservations despite that the booking can be made. The purchase was not carried out due to a transaction failure and it may result in loss of money. This shows that the abstract

model automatically generates test cases to test different WS transactions standards and identify their reliability to failures.

6. RELATED WORKS

Current work mainly deals with business transaction modeling from a design perspective. A theoretical approach is used in [18] in order to specify, analyze and synthesize advanced transaction models. Transactional patterns that combine workflow process adequacy and the transactional processing reliability are identified in [19]. [20] presents a high level UML-based language to design transaction process with diverse transactional semantics whilst a XML representation is proposed in [21].

Though there exist significant literature on WS transactions but to the best of our knowledge, none of them addresses the testing of WS transactions. In [22] a risk-based approach is used to define general test specifications for compensable transactions. Some others works are focused on verifying long-lived transactions from a theoretical point of view.

In [23], authors have developed a model of communicating hierarchical timed automata suitable to describe long-running transactions. This approach allows the verification of properties by model checking. The work in [24] uses a technique to translate programs with compensations to tree automata in order to verify compensating transactions. Also [25] proposes a formal model to verify the requirement of relaxed atomicity with temporal constraints whilst [26] use event calculus to validate the transactional behavior of WS compositions.

7. CONCLUSIONS AND FUTURE WORK

This paper proposed a novel abstract transaction model which models different WS transaction standards. It exploited the model-based testing technique in order to automatically generate test scenarios for testing the failures and reliability of the WS transaction standards. A prototype tool is developed in order to validate and evaluate the proposed abstract model using a web services application of a travel agency. We showed that the abstract model is capable of dynamically modeling different WS transaction standards such as BTP and WA-BA. We also tested the failure and reliability of these standards using the test scenarios generated through the proposed model. Our future work includes detailed testing of the WS transactions standards and their performance evaluation.

8. REFERENCES

[1] M. Younas, K. Chao, C. Lo, Y. Li, "An Efficient Transaction Commit Protocol for Composite Web Services," *Int. Conf. on Ad. Info. Networking and Applications,* 2006.

[2] G. Canfora and M. Penta, "Service-Oriented Architectures Testing: A Survey," *ISSSE* 2006-2008, Salerno, Italy, 2009

[3] OASIS, "Business Transaction Protocol," http://www.oasis-open.org/committees/tchome.php?wg-abbrev=business-transaction.

[4] OASIS, "Web Services Business Activity," http://docs.oasis-open.org/ws-tx/wsba/2006/06.

[5] OASIS, "Web Services Composite Application Framework" http://www.oasis-open.org/committees/tc-home.php?wg-abbrev=ws-caf.

[6] *Database transaction models for advanced applications*. Morgan Kaufmann Publishers Inc., 1992, p. 610.

[7] E.B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing," MIT, 1981.

[8] H. Garcia-Molina, K. Salem, "Sagas," *SIGMOD* 87, 1987.

[9] G. Weikum, H.-J. Schek, "Concepts and applications of multilevel transactions and open nested transactions," *Database transaction models for advanced applications*: 1992

[10] C. Pu, G. E. Kaiser and N. C. Hutchinson., "Split-Transactions for Open-Ended Activities," *VLDB*, 1988.

[11] Reuter, "ConTracts: A Means for Extending Control Beyond Transaction Boundaries," *3rd Int. Workshop on High Performance Transaction Systems,* 1989.

[12] A. Zhang, M. Nodine, B. Bhargava and O. Bukhres., "Ensuring relaxed atomicity for flexible transactions in multidatabase systems," *ACM SIGMOD Record,* 1994.

[13] M. Younas, B. Eaglestone and R. Holton, "A formal treatment of a SACRED Protocol for Multidatabase Web Transactions," *Database and Expert Systems Applications,* vol. 1873, pp. 899-908, 2000.

[14] OASIS, "Web Services Atomic Transaction," http://docs.oasis-open.org/ws-tx/wsat/2006/06.

[15] OASIS, "Web Services Coordination,," http://docs.oasis-open.org/ws-tx/wscoor/2006/06.

[16] A. Cavalli, T. Cao, W. Mallouli, E. Martins, E. Sadovykh, S. Salva and F. Zaidi, "WebMov: A Dedicated Framework for the Modelling and Testing of Web Services Composition," IEEE *ICWS* 2010.

[17] J. Offutt, S. Liu, A. Abdurazik and P. Ammann., "Generating Test Data From State-based Specifications," *Journal of Software Testing, Verification and Reliability,* vol. 13, pp. 25-53, 2003.

[18] P. K. Chrysanthis and K. Ramamritham, "Synthesis of extended transaction models using ACTA," *ACM Trans. Database Syst.,* vol. 19, pp. 450-491, 1994.

[19] S. Bhiri, C. Godart and O. Perrin, "Transactional patterns for reliable web services compositions," *Int. Conf. on Web Engineering*, California, USA, 2006.

[20] N. Gioldasis and S. Christodoulakis, "UTML: Unified Transaction Modeling Language," *Int. Conf. on Web Information Systems Engineering* 2002.

[21] P. Hrastnik and W. Winiwarter, "Using advanced transaction meta-models for creating transaction-aware web service environments," *International Journal of Web Information Systems,* 2005.

[22] R. Casado, J. Tuya and M. Younas, "Testing Long-Lived Web Services Transactions Using a Risk-Based Approach," *Int. Conf.on Quality Software*, 2010.

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

[24] M. Emmi and R. Majumdar, "Verifying Compensating Transactions," *Int. Conf. on Verification, Model Checking, and Abstract Interpretation,* 2007.

[25] J. Li, H. Zh and J. He, "Specifying and Verifying Web Transactions," *Int. Conf.on Formal Techniques for Networked and Distributed Systems,* 2008.

[26] W. Gaaloul, M. Rouchaded, C. Godart and M. Hauswirth., "Verifying composite service transactional behavior using event calculus," *OTM On the move to meaningful internet systems, Vilamoura*, Portugal, 2007.