Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (2) 295-302 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Agreement and Compensation Verification in Long-running Transactions

S. VIGHIO, F. H. CHANDIO*, G. M. SHAH, Z. H. ABRO, I. A. KOREJO*

Department of Information Technology, Quaid-e-Awam University, Nawabshah, Pakistan
Email: saleem.vighio@quest.edu.pk , gm.sayed@quest.edu.pk, zahidabro@gmail.com, imtiaz@quest.edu.pk

*corresponding author: Email: F. H. CHANDIO chandiofida@usindh.edu.pk Ph. No. +92-229213177

Received 07th August 2011 and Revised 22nd March 2012)

Abstract: This paper presents formal analysis of an application example that makes use of long-running, nested, and compensation-based services which require coordination and agreement in order to perform a joint business task. The communication is supported with the use of Web Services Business Activity protocol suitable for implementation of complex business processes. The main contributions of the paper include the analysis of the suitability of the business activity protocol in order to be used in complex services that the designers claim this protocol is developed for. Furthermore, the paper proposes a separation of compensation logic from the protocol about which the protocol specification is not clear. Analysis is performed using the model-checker UPPAAL and the property of main interest is to verify the consistent outcome of the business process once the services complete their work.

Keywords: Web services; Agreement protocols; Model-checking; Verification.

1. INTRODUCTION

Modern e-business applications require coordination with other applications in order to perform a joint task (typically a business task) (Erl, 2005, Gray, 1992). For example, consider an online reservation application that facilitates its users with the booking of a plane ticket and a hotel room, solely not managed by the reservation application by itself. In this case, after the request has been received from the customer, the reservation application communicates with the airline and the hotel services in order to process customer's request. Finally, depending on the replies received from the corresponding services about the availability of resources (plane ticket, hotel room), the reservation application decides on the final outcome of the (reservation) task, and communicates the result with the customer. The tasks performed by the services which span organizational boundaries are usually complex in nature and require mechanisms in order to recover from faults that occur during the processing of these tasks (Liu, *et al*., 2008). Furthermore, services intending to perform a joint task require coordination and agreement among themselves in order to reach a common business goal. The Web Services Coordination and Agreement (WS-CA) protocols (Newcomer, *et al*., 2009) are specially designed in order to deal with these challenges. These protocols allow coordination among multiple services and direct them to reach a common goal. Furthermore, these protocols allow services to rollback or to perform compensations in cases wherefaults occurs during the processing of tasks. The two types of agreement protocols defined in the Web Services Coordination and Agreement Framework (WS-CAF) (Newcomer, *et al*., 2009) include the Web Services Atomic Transaction (WS-AT) (Newcomer, *et al*., 2009) and the Web Services Business Activity (WS-BA) (Newcomer, *et al*., 2009) protocols used respectively for transactions that are short in duration and the transactions that run for longer time.

This paper presents formal analysis of an extended version of the well-known travel reservation process example where a customer sends a reservation request for an air-ticket and a hotel room to the travel agent. The extension of the example introduces a new role: *the pilot* which sends reservation request for a hotel room and a rental car to the airline service. The coordination and agreement among services is achieved using the Web Services Business Activity (WS-BA) protocol that is suitable for communication among complex services (long-running, nested, and compensation-based). With its usage in this case study,

*Institute of Mathematics and Computer Science, University of Sindh, Jamshoro, Pakistan

the paper presents an evaluation of the suitability of the protocol in order to be used in complex business applications, for e.g., considers a case where the airline service processes requests both for the customer for an air-ticket, and the pilot for a room and a car. In this case, the airline has to proceed with the requests from the pilot before it can process a request from the customer (via travel agent). This shows a hierarchy of activities: the concept called nested transactions (Moss, 1981). Because of the processing dependencies nested transactions are more complex than flat transactions (see Section II and III for more details on nested transactions).

Furthermore, the paper also proposes a separation of compensation logic (including the business logic) from the WS-BA protocol about which the protocol specification is not explicit. However, studies show that the business and compensation logics are application dependent and should be performed at the service (or application) level and not at the protocol level (Korth, *et al*. 1990). One of the key advantages of this separation is the ease and flexibility with which the application designers can modify these logics according to the changing needs of the organizations. Efforts have already been taken in this regard, for example, in (Vogt, *et al*., 2005) the authors have introduced a transactional middleware that separates the business logic from the WS-BA protocol for an easier client implementation. That approach is further extended in (Erven, *et al*., 2007) where the authors uncouple the initiator and the coordinator and thus suggest possible extension of the protocol standard. On the other hand, this paper proposes a middle way without modifying the protocol standard. The technique proposed here allows applications to perform business logic independently by themselves, however, the compensation logic is also performed by the services by themselves, but the indication comes from the protocol (details are followed in Section III).

The case study example presented here is implemented and verified using the model checker UPPAAL (Behrmann, *et al*., 2004). The key properties verified are: (i) the *termination* property, which guarantees that all the parties involved in the process must complete their work, and (ii) the *common agreement* property, which ensures that once the parties complete their work the outcome seen by these parties is the same.

The rest of the paper is organized as follows: Section II gives details of the case study example along with a brief introduction of the WS-BA protocol. Formal verification of the application example and results are discussed in Section III. Finally, Section IV concludes the work with a discussion and gives suggestions for the future work.

2. MATERIAL AND METHODS

A. Application example

The simple travel reservation process example involves four roles: *Client*, *Travel agent*, *Hotel*, and *Airline*. In that process, initially, the client sends a reservation request for an air ticket, and a hotel room to the travel agent. The travel agent after receiving requests, contacts the airline and the hotel services for an air-ticket and a room respectively. Once the travel agent receives replies from all the services, it decides on the final outcome of the reservation task and sends the decision to the client. The decision reached depends on the coordination type and the agreement protocols (see, Newcomer, *et al*., 2009) used for communication. This paper presents extension of the above example by introducing a new role: the *Pilot* which requires a hotel room and a car. The requests of the pilot go through the airline which further communicates with the hotel, and the car-rental services.

As it can be seen from **(Fig. 1)**, every service produces an atomic outcome of "ok" or "refused", and the final outcome of the

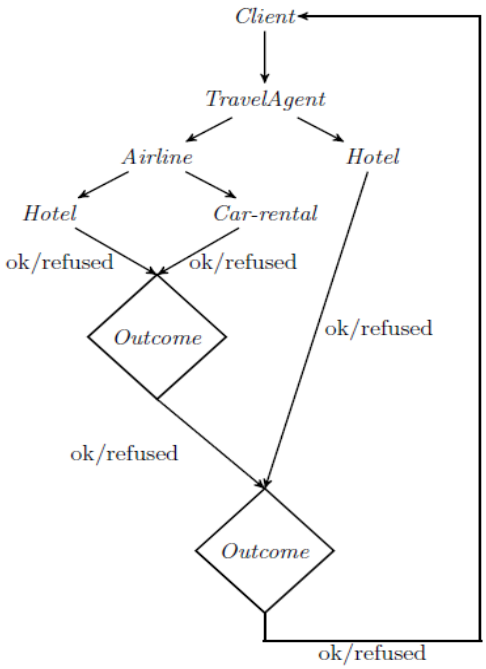


Fig. 1: Flow of activities

reservation task is based on the outcomes produced by the individual services. This means that, if all the services reply with an "ok" then the overall outcome of the task is successful, otherwise it is refused. In WS-BA terms (Newcomer, *et al*., 2009), the client is an application that initially requests the travel agent for the creation of a new activity, i.e., the reservation task. The

travel agent and the airline services play coordinator role and register participants to agreement protocols. The participants in this case are the airline, hotel, and car services. Precisely, the travel agent is a coordinator for the airline and the hotel services, whereas the airline is a coordinator for the car-rental and the hotel services. Note that, the airline plays a dual role, on the one hand, it is a participant of the travel agent, and on the other hand, it is a coordinator for the car-rental and hotel services. The coordinator and the participant communicate with each other respectively using the coordinator and participant roles of the same protocol that they register with, see (Newcomer, *et al*., 2009) for more details on protocol roles.

The WS-BA standard (Newcomer, *et al*., 2009) defines two types of agreement protocols namely, Business Agreement with Participant Completion (BAwPC) and Business Agreement with Coordinator Completion (BAwCC), and two coordination types: AtomicOutcome and MixedOutcome. The agreement protocol considered for communication here is the Business Agreement with Participant Completion (BAwPC for short) and the coordination type used by this protocol is the AtomicOutcome. As the main idea of the analysis presented here is to evaluate the suitability of the WS-BA standard in order to be used in compensation-based and nested transactions, so what type of BA protocol is considered for communication does not matter.

3. RESULTS AND DISCUSSION

B. Formal analysis

Depending on the complexity of operations performed by each service, this paper divides services into complex (nested, as discussed earlier) and simple services. A complex service is one which requires other services to perform its work, whereas a simple service is self-contained and does not require other services to perform its work. In the example case presented in this paper, the travel agent and the airline are complex services, whereas the hotel and the car-rental are simple services. As discussed before, in the WS-BA context, complex services communicate using the coordinator role of the protocol, whereas the simple services communicate using the participant role of the same protocol. Based on this information, the processes implemented here include the models for the complex services (travel agent and airline services), simple services (airline, hotel, and the car-rental services), coordinator and participant roles of the BAwPC protocol used respectively by the complex and simple services for communication, and the coordination logic for complex services in order to communicate with simple services. Note that, in the implementation presented here business logic is embedded in complex and simple services, whereas the simple services also include the compensation logic.

The idea is this: the client initiates the reservation task by sending a request to the travel agent (complex service). The travel agent after executing its business logic, i.e., checking the availability of resources (contractual airlines and hotels) invokes its coordination logic which further invokes (sequentially in the case presented here) the airline and the hotel services. On the other hand, in order to process pilot's requests, the airline service after checking the availability of resources (contractual hotels and car-rental agencies) initiates its coordination logic in order to send (sequential) requests for a room and a car to the hotel and the car-rental services respectively. The hotel and the car-rental services operate concurrently. Initially, these services check the availability of resources (rooms and cars respectively) and communicate the information with the coordination logic of the airline; the services then invoke their corresponding participant protocols for coordination (coordination is required in order to perform a joint work and to reach a mutual agreed outcome) with the calling service (airline in this case). On the other hand, the airline service invokes its coordinator protocol for communication with the hotel and the car-rental services. The coordinator role of the protocol checks if all the replies remained successful. In the positive case it announces the outcome of the task to the participating services (hotel, car-rental). However, if any of the services replies negatively (failure of the participating service or unavailability of resources), the coordinator protocol sends a compensate message to the participating services who already have completed their work successfully; this is required in order to reach a common outcome of the business task. The services which receive a compensate message invoke their corresponding compensation logic in order to revert the changes. Furthermore, once the airline service finishes (successfully or unsuccessfully) requests for the pilot, it works as a simple service for the travel agent similar to the car-rental and hotel services. The travel agent then goes through the steps similar to the airline service worked for the pilot. In the text to follow, we give UPPAAL (Behrmann, *et al*., 2004) models of the roles involved in the reservation process **(Fig.2)**. The UPPAAL (Behrmann, *et al*., 2004) is a tool for the modeling, simulation, and verification of distributed and embedded systems. The processes in UPPAAL are modeled as timed automata machines that execute concurrently and communicate using channels and shared variables. The tool offers powerful C-like syntax for the description of systems in terms of guards, updates, and real-valued clocks. The paper refers the reader to (Behrmann, *et al*., 2004) for details about the tool and its usage.

- ***UPPAAL modeling of the protocol***

***Client*:** The UPPAAL model for the client is shown in **(Fig.2).**

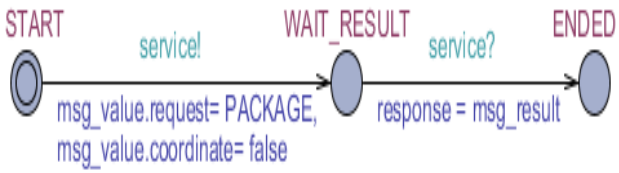


Fig. 2: Client

`Initially, the client sends a service request to the travel agent using the channel **service**. The variable **msgs_value** is defined as a record type specifying the type of request (**msgs_value.request**) which in this case is a combined request (**PACKAGE**) for a room and an air-ticket. The record type specifies also the coordination link **msgs_value.coordinate** for communication. The coordination link is initially set to false indicating that the client is not a coordinator and that the receiver of the request (the travel agent in this case) should not run the participating protocol (this means that the receiver which is a travel agent in this case, should run the coordinator protocol).

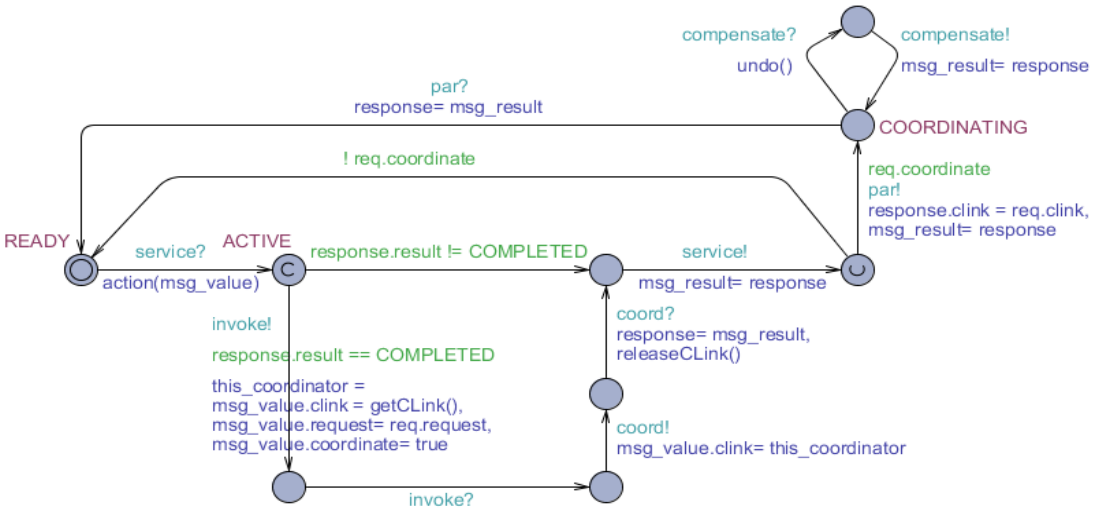
***Complex Service*:** As discussed earlier at the start of this section, travel agent and airline work as complex services. Fig. 3 shows UPPAAL model for the travel agent. Once the travel agent receives service request from the client (i.e., synchronizes on channel service?), it executes its business logic action(msg_value**)** in order to check the availability of resources (contractual airlines, hotels); see code below:

void action(MsgValue r)
{req = r;
if (available > 0) { response.result = COMPLETED;
available --; return;}
response.result = FAULT; return; }

In the function above, available is declared as a local integer variable and is initialized to the maximum number of resources (int[0,MAXRESOURCE available). If the resources are available, i.e., if (available > 0), then the resulting response is set to COMPLETED and the resources are decreased by one. However, if the resources are not available, then the response is set to FAULT. In the positive case when response.result == COMPLETED, the complex service invokes (invoke!) its coordination logic (for communication with simple services), gets and sets the coordination link, and sends a service request to the coordination logic:

this coordinator =
msg_value.clink = getCLink(),
msg_value.coordinate = true,
msg_value.request = req.request

As shown in **(Fig. 3)**, after receiving a reply (invoke?) from the coordination logic, the complex service synchronizes (coord!) with the coordinator protocol and communicates with it a coordination link msg_value.clink=this coordinator for communication. Once the travel agent is acknowledged (coord?) by the coordinator protocol about the completion of communication (with the participating services via their corresponding participant protocols), it saves the outcome of the task respose = msg_result and releases the coordination link releaseCLink(). At this point, all the service requests are completed and the complex service for the travel agent communicates (service!) the outcome of the transaction with the client. The paper refers the reader to (Vighio, *et al*., 2011) for complete UPPAAL models of this application example. Note that, in our implementation every service (complex, simple) embeds business logic, whereas, simple



services additionally embed compensation logic as well. This means that compensation logics are performed at simple services. Note also that, as the model for the complex service is parameterized (the UPPAAL feature for using the same model for different processes), the location COORDINATING in Fig. 3 is only reached when the model is used for the airline service which behaves as a simple service for the travel agent and complex service for the pilot. In that case, the airline service can be told (compensate?) by its participant protocol in order to perform the compensation. The compensations considered here are very simple and only perform an undo operation that reverts the changes performed and brings the service to its previous state. Precisely this means that if a room is booked then its corresponding compensation would be to unbook the same room. Details about simple services are given in the text to follow.

***Coordination logic for complex services*:** In the implementation presented here, complex services communicate with simple services using the coordination logic. The UPPAAL model of the coordination logic used by the travel agent is shown in Fig. 4. After the coordination logic has been invoked by the travel agent (invoke?), it sends a request to the airline service using the channel airline_s. Once it is acknowledged by the airline service, the coordination logic proceeds to send a request to the hotel service using the channel hotel_s, see (Fig. 4). When all the services requests are sent and acknowledgements are received, the coordination logic invokes (invoke!) the travel agent service for further processing.

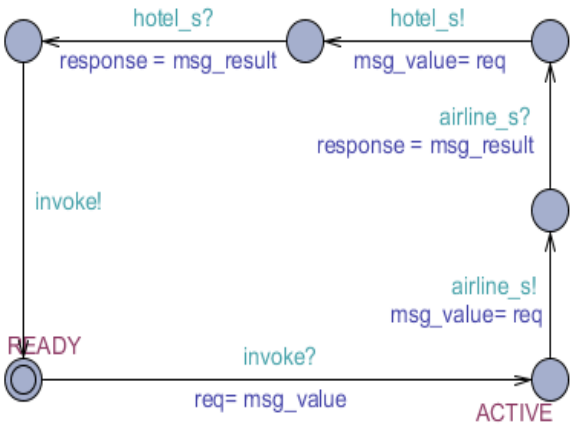


Fig. 4: Coordination logic for travel agent

***Simple service*:** In the application example presented here, hotel and car-rental services are modeled as simpleervices. **(Fig. 5)** shows the UPPAAL model for the hotel service.

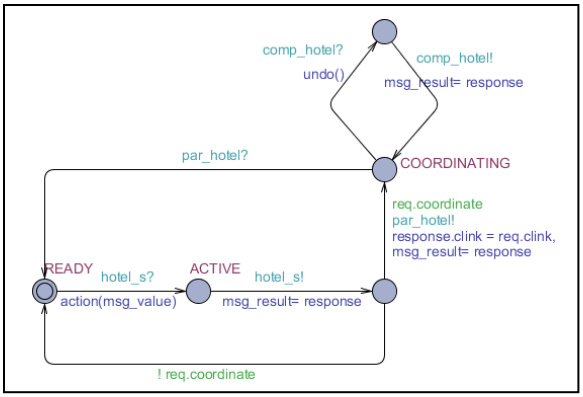


Fig. 5: Simple service for hotel

Recall from the description of the complex services that the model for the airline service works both as simple and complex services.

The hotel service is called (hotel_s?) by the coordination logic of both the travel agent and the airline services. After sending back the acknowledgement (hotel_s!) to the coordination logic, the hotel service synchronizes with the participant protocol on channel par_hotel in order to communicate with the travel agent and the airlines services via their coordinator protocol. During synchronization with the participant protocol, the hotel service also sets the link information (response.clink = req.clink) in order to receive the protocol messages. At location COORDINATING, the simple service can perform compensation if informed by the coordinator protocol (via participant protocol), see Fig. 5.

***Participant and coordinator protocols*:** As discussed in (Newcomer, *et al*., 2009), services register for agreement protocols and once registered they communicate using the coordinator and the participant roles of the same protocols. In the case study presented here, hotel, airline, and car-rental services communicate using the participant role of the BAwPC protocol, and the travel agent and the airline services communicate using the coordinator role of the same protocol. The coordinator manages the overall business process and directs the services to a common outcome.

As discussed in Section I, the agreement protocol considered for communication here is BAwPC; a protocol type of the WS-BA standard. The protocol is described in its specification using control

flow diagrams supported by the English text and the state-transition-tables, all of which further describe how the protocol behaves and what properties it should satisfy. The implementation presented in this paper considers only the abstract behavior of the protocol and omits message retransmissions and the states that represent the invalid behavior, see (Newcomer, *et al*., 2009) for more details on the BAwPC protocol. The protocol allows asynchronous communication and defines a finite number of messages transmitted by each role (coordinator, participant). The UPPAAL model of the participant protocol for the hotel service is shown in Fig. 6. At location READY the participant protocol receives a request (par_hotel?) from its registered service (see Fig. 5 for simple hotel service). The protocol then sets up using setup (msg result), a communication link form communication.

In **(Fig. 6),** MsgsPAR is defined as an integer type (typedef int[0,PAR_MESSAGES-1] consisting of six messages (const int PAR_MESSAGES = 6) as defined in the protocol specification. These messages are transmitted from the participant to the coordinator role. As shown in Fig. 6, each message contains the link information the message is transmitted on. For example, msgsPar[id][COMPLETED]++ represents that the message COMPLETED is sent from the participant protocol to the coordinator on a particular communication link identified as id. Furthermore, with

presented here does not allow loss or retransmission of messages. This count is necessary also for the coordinator protocol in order to determine the number of participants which sent the
message COMPLETED, and in case of a failure of that message from any participant, the coordinator sends a compensate message to the participants who have successfully completed their work. This is required in order to reach a consistent business outcome. It is easy to observe from Fig. 6 that some transmissions (of messages) are dependent on the reception of a message (implemented as guards on transitions) from the coordinator protocol, whereas others are not. For example, the transmission of messages COMPLETED and CANCELED is dependent, whereas the transmission of EXIT and FAULT is not. The operation of the coordinator protocol is similar to that of the participant protocol and is self-explanatory. The paper refers the reader to (Vighio, *et al*., 2009) for complete UPPAAL models of this case study example.

Properties verification

Verification here refers to the guarantees about the intended behavior of the protocol and the business process. The first key property verified here is *termination* which ensures that all the services (travel agent, airline, hotel, and car-rental) and their corresponding coordination logics and protocols involved in the process must reach their final state (independent of the final outcome). The UPPAAL formulation of this pro each transmission of a message perty is shown below:

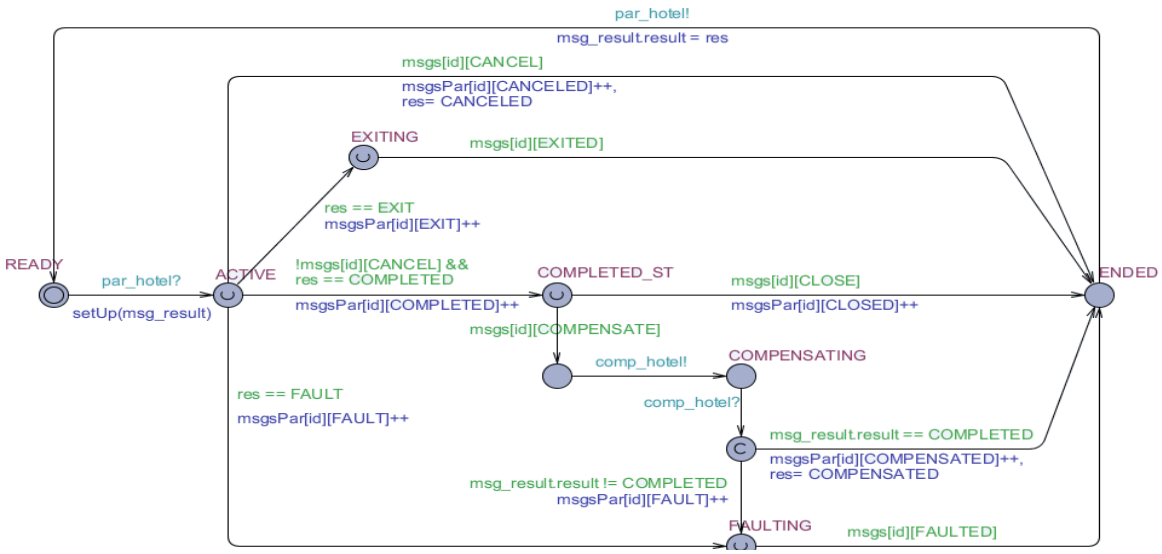


Fig. 6: Participant role of BAwPC protocol

the implementation presented in this paper puts a count on the number of times the same message is transmitted. This is due to fact that the implementation

A<>(car.READY and hotel.READY and hotelPC.READY and carPC.READY and

airCoord.READY and VIPair.READY and airLogic.READY and airPC.READY and taCoord.READY and VIPta.READY and taLogic.READY and client. ENDED)

This is a liveness property and it ensures that all the parties involved in the process reach their READY state and at the same time the client process enters its ENDED state. Note that with the client in its ENDED state and the services and their corresponding coordination logics and protocols in their READY state represent in this implementation that the services have completed their tasks. The property is satisfied. The satisfaction of termination property is highly desirable in order to prove in practice that services always complete their work and then become ready to perform the next task.

The second key property verified here is the *consistency* on the outcome of the business process. This property checks that if all the parties finish their work without failures, the the outcome of the process is successful (*COMPLETED*). The UPPAAL formulation of this property is the extension of the termination property shown above but with an addition of a check on the final outcome when the client process receives the response from the travel agent. It is in fact a logical implication that suggests that termination implies successful completion.

A[](not(car.READY and hotel.READY and hotelPC.READY and carPC.READY and airCoord.READY and VIPair.READY and airLogic.READY and airPC.READY and taCoord.READY and VIPta.READY and taLogic.READY and client.ENDED) or client.response.result == COMPLETED)

This property is also satisfied for available resources and no failures of services. Note that in the opposite case, if any of the participants does not want to participate (exit) in the process, or runs out of resources, then the final outcome is aborted, and as a result the above property will not be satisfied.

To show one of these scenarios, we introduce an erratic service for the hotel **(see Fig. 7),** which non-deterministically selects one of the outcomes (**COMPLETED** or **EXIT**) in the business logic. Furthermore, it is also verified by simulation that in cases when one of the participating services fails, a compensation is performed on services which already have successfully completed their work. The compensations considered here bring the service to its previous state before the changes took place.

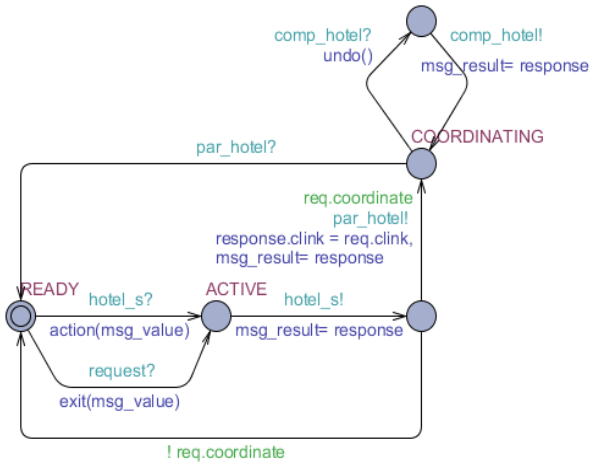


Fig. 7: Erratic service for hotel

4. CONCLUSION AND FUTURE WORK

In this paper, we presented formal verification of an extended version of an application example that is based on the well-known travel reservation process. The main contributions of this paper include the separation of business and compensation logics from the BAwPC protocol about which the protocol specification is not clear, and the implementation of nested and compensation-based transactions. The analysis is performed using the model checker UPPAAL. Verifications are the guarantees that all the parties (services, protocols, coordination logics) involved in the process always finish their work by reaching their final state. Furthermore, it is also verified that in cases when one of the transactions fail, corresponding compensation are performed. Finally, the consistency on the outcome of the transaction is also verified by ensuring that if all the parties finish their work without failures then the overall outcome of the transaction is successful otherwise it is aborted.

For the future work, further extensions of the application example are planned, for example, the consideration of penalty policies if the client cancels the successful booking. Furthermore, it is also planned to implement this case study using specially designed languages e.g., BPM and WS-BPEL for easier implementation and bridging the gap between the implementation and the execution.

REFERENCES:

Behrmann G., A. David and K.G. Larsen (2004) A tutorial on UPPAAL. In *Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFMRT'04)*, number 3185 in LNCS, 200–236. Springer-Verlag, 2004.

Erl T. (2005) *Services Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

Erven H., G. Hicker C, Huemer and M. Zaptletal (2007) The web services-business-activity-initiator (ws-ba-i) protocol: an extension to the web services-businessactivity specification. IEEE International Conference on Web Services, 216-224.

Gray J. and A. Reuter (1992) *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Korth H. F., E. Levy and A. Silberschatz (1990) A formal approach to recovery by compensating transactions. Technical report.

Liu C. and X. Zhao (2008) Towards flexible compensation for business transactions in Web service environment. *Service Oriented Computing and Applications.* Vol. (2): No., 2-3, 79-91.

Moss E. B. (1981) Nested transactions: An approach to reliable distributed computing. Technical report.

Newcomer E. and I. (chairs). Robinson (2009) Web services coordination (WScoordination) version 1.2, 2009.

Newcomer E. and I. (chairs). Robinson (2009) Web services atomic transaction (WS-atomic transaction) version 1.2.

Newcomer E. and Robinson I. (chairs). (2009) Web services business activity (WS-businessactivity) version 1.2.

Vogt F. H., S. Sambrovski B. Gruschko and P. Furniss (2005) Green A. Implementing web service protocols in SOA: WS-Coordination and WS-Businessactivity, .

Vighio S., A.P. Ravn and F.A. Jokhio (2011) UPPAAL model of the Business Agreement with Paprticipant Completion (BAwPC) protocol, 2011. Available at http://people.cs.aau.dk/~vighio/bawpc.zip.

.