

# The Safety Issue in a Web Travel Booking Services Scenario Based on Business Process Execution Language

Apostolos Axenopoulos

Charalampos Doulaverakis

Informatics and Telematics Institute

Centre for Research and Technology Hellas, Thessalonica, Greece

E-mail: {axenop, doulaver}@iti.gr

Nicholas Protogeros

Panayiotis Tahinakis

Department of Accounting and Finance

156 Egnatia str., 546-00, Thessalonica, Greece

E-mail: {nproto, tahi67}@uom.gr

John Mylonakis

10 Nikiforou str., Glyfada, 16675, Athens, Greece

E-mail: imylonakis@vodafone.net.gr

## Abstract

In the Business Process modeling area, the Business Process Execution Language (BPEL) seems to be gaining ground in favor of its other competitors. The simplicity of the protocol coupled with its ability to hide the complex mechanisms that lie underneath along with advanced control features and integration with other concrete standards as XPath and XSLT have contributed to its acceptance. BPEL has been thoroughly investigated in literature as a solution to workflow management in an enterprise environment. In this paper, an intrinsic property of BPEL, related to compensation handling, is investigated as an alternative approach to two-phase commit used in distributed transactions. The applicability of the proposed solution is demonstrated on a travel booking scenario and interesting conclusions are drawn regarding the system's response in cases of service failures.

**Keywords:** Web services, Business process execution language, Business process management, Database transactions, Compensation handling mechanism

## 1. Introduction

Service Oriented Architecture (SOA) is the de facto standard for designing complex software systems, where each component is offered as a service described in a uniform and reusable way. Today, SOA is mainly implemented using web services. The combination of web services and SOA has had a substantial impact on business process workflow and management by enabling automation of procedures that involved either intra or inter corporate structures while at the same time being able to sustain a high level of interoperability. However, it was soon realized that business process logic and business process implementation had to be separated which in turn led to the development of protocols and standards that would allow an adequate distinction of logic from implementation, an active area of research known as Business Process Management (BPM).

The Business Process Execution Language (BPEL) emerged as an industry backed standard for describing long-running processes. Its adoption rate as a BPM protocol is partly due to endorsement from companies like IBM, Microsoft and Oracle who have all contributed to the standard and also provide commercial products as BPEL engines and designers for easy implementation of BPEL workflows although reliable open source engines have been released, e.g. ActiveBPEL from Active Endpoints (<http://www.activevos.com>).

Each party in the business process is referred to as a partner. The Business Process Execution Language communicates between partners using partner links with the exchanged messages defined in WSDL. All BPEL processes perform three basic operations. These are:

- a) **Invoke**, involves the calling and execution of the services that comprise the process. These are web services implemented on each partner side.
- b) **Assign**, allocates values
- c) **Reply**, is the termination point of each called service and contains the result of the execution. Reply is also the endpoint activity of the process as a whole.
- d) between variables. These values either come from the reply function of a service or they can be user defined. The Business Process Execution Language is often used as an orchestration service for managing complex processes that require access to different and/or distributed databases of the different organizations (partners) involved. Due to the inherent heterogeneity between software systems of the involved partners, a matter which often arises is that of consistency of data. One could imagine the situation in a BPEL travel booking process for example, where a customer issues a reservation request for both a hotel room and a flight and something goes wrong in the room reservation service while the reservation for the flight is completed. In this case a mechanism should exist where a rollback of the flight reservation would be initiated automatically within the process. BPEL can handle these situations with appropriately defined compensation handlers. A compensation handler can only be called through a fault handler and its main use is to undo previous actions inside a scope when an exception or error occurs. While it provides an efficient way to cope with abnormal behavior, its main drawback is that the only way to be invoked is inside of a fault handler and not during the normal process execution.

In the travel booking scenario, if an error occurred during the final phase of the reservation for both the hotel and the flight ticket in the hotel process for instance, this would have the effect of invoking the compensation actions and both reservations of flight and hotel and informing the customer that an error occurred. Then the whole process would start from the beginning and would again reserve the flight ticket that was originally reserved and subsequently cancelled. It is obvious that this behavior adds overhead to the process and could have been avoided if compensation could be invoked outside of a fault handler.

The purpose of this paper is to investigate the Business Process Execution Language using an implementation of a travel booking business process.

## 2. Background and Literature

### 2.1 The 2-Phase Commit Protocol

In relational database management systems, transaction management aims to keep a database in a consistent state even when concurrent accesses and failures occur. That is, if a database was in a consistent state before a transaction is initiated, transaction management ensures that the database will return to a consistent state after the completion of the transaction.

In order to maintain a consistent state in RDBMS systems, the 2-phase commit protocol has been extensively used (Alkhatib, 2002). 2-phase commit is a distributed algorithm used to ensure the integrity of a committing transaction. The protocol results in either all nodes committing the transaction or aborting. In 2-phase commit, one node is designated the coordinator and the rest of the nodes are called cohorts. In the first phase, namely the “commit-request” phase, the coordinator sends a “*query-to-commit*” message to all cohorts waiting for them to reply with an agreement message. In the second phase, the “commit” phase, if the coordinator receives an agree message from all cohorts, it sends them a “*commit*” message. If all agreement messages do not come back, the coordinator sends an “*abort*” message. If the cohort receives a “*commit*” message, it releases the resources held during the transaction and sends an acknowledgement. If it receives an “*abort*” message, then the cohort undoes the transaction and releases the resources (Singhal, 1994).

### 2.2 Transactions in Business Activities

2-phase commit protocol is appropriate for supporting atomic distributed transactions with the traditional ACID (atomic, consistent, durable, isolated) properties. However, in real world business transactions, the 2-phase commit is difficult to implement. The major issue is the isolation of a database transaction. The protocol requires that the processes waiting for resources locked by other processes will have to wait until the resources are released. This resource locking is impractical for the business world (Charfi, 2007).

Real world business transactions are usually long-running transactions, also known as business activities. A defining characteristic of the business activities is that instead of locking resources (using e.g. a 2-phase commit protocol) they can undo previous actions by using appropriate compensation logic. These transactions are

characterized by relaxed isolation, i.e. state information can be shared across distributed resources before the completion of the business activity (Kolawa, 2006).

Transactional semantics support for long-running transactions is available in some business process composition languages, like the extended transaction model of BPML (Arkin, 2001). The extended transaction allows two possible modes of recovery: backward recovery, where the transaction initiates some compensating activities to cancel the effect of the failed transaction, and forward recovery, where the business process instance is allowed to continue its execution taking into account that the transaction failed.

An alternative approach to the two-phase commit protocol can be provided indirectly by BPEL. In case of a service failure, a compensation handler can be used to undo those actions that completed successfully. This approach is appropriate for business process instances that extend over a long period or cross organizational boundaries, where it is impractical to have transactions waiting to commit.

Several efforts that exploit the exception handling features of BPEL in business oriented applications have been proposed in the literature. The Fault, Compensation and Termination (FCT) handlers of BPEL have been examined by He & Watters (2007) in order to show the features that might be modified and be suitable for business processes in 3<sup>rd</sup> party logistics applications. AT framework for automatic handling of business process failures is presented by Kareliotis (2007). The framework employs a pre-processor that enhances BPEL scenarios with code that detects failures, discovers alternate WS implementations and invokes them resolving the exception.

### 3. The Travel Booking Scenario

Following the above considerations and in order to test the abilities of the compensation mechanism of BPEL, a virtual travel booking scenario has been implemented. The scenario involves the collaboration of various organizations as are hotels and airline companies along with user input. The travel booking scenario has been used extensively (Decker, 2007, Rosenberg, 2005), providing a relatively easy to understand example of BPEL's orchestration and communication behavior. This Section shall provide the details of the developed platform on which all tests were carried out.

In the travel scenario, the user seeks to book a holiday package through an appropriate web portal which orchestrates all services from providers using a BPEL process. The providers are a number of airline companies and hotels where each has a searching and booking service. The user selects destination and duration of holidays and the system requests availability on airplane tickets and hotel rooms. The response to the user is presented in a sorted list of available package compilations according to total cost. The user selects a package and a confirmation is sent to the user if the whole process was successful. If an error occurred, a compensation strategy is followed and user is informed. The overall process is visualized in Figure 1.

#### 3.1 Requirements and System Description

A travel booking portal should fulfill several requirements in order to be dealt with as a real world scenario. These requirements and how BPEL copes with them according its specifications noted below:

- **Scalability.** As the number of hotels and airline companies that are linked to the system (portal) is not constant but will be changing, there should exist the possibility to easily scale up or down according to the needs. The scaling should be done in a manner that will not compromise functionality of the system and should also be done without much complexity. BPEL can handle a large number of *partner links* and communication with partners is done individually through messages. As a result, a change in the number of partners and services will not compromise any functionality.
- **Reliability.** All communications should be done in a reliable environment. If a response to a request is not issued or an error occurs the system should be able to recover and proceed with normal operations as planned accordingly. According to BPEL specifications message exchange is reliable. Session timeouts can be implemented so as not to end up in a deadlock and special fault handlers, which will be discussed later, can be defined in order to deal with errors.
- **Asynchronous communication.** Responses from partners registered in the system could either be received instantly, after a period of time or even not received at all if an error occurs. The process should be able to continue execution without having to wait for a response. BPEL has native support of asynchronous messaging as it is especially designed to orchestrate long running business processes.
- **Exception handling.** As the number of partners (hotels and airline companies) increases, errors or faults will become more common. Fault handling should be employed in order to manage errors that could

lead to unpredictable situations or even stop process execution. Additionally as asynchronous communication is adopted, compensation of previous actions in case of errors should be supported. BPEL incorporates fault and compensation handlers and their construct will be investigated in particular in the next Section.

Following the above requirements, an appropriate travel booking portal where requests can be issued has been implemented. Simulated hotel and airline services were also developed along with a BPEL process specifically for orchestrating them according to the diagram of Figure 1. The travel booking portal was developed with JSP while the BPEL engine used was the ActiveBPEL Engine (<http://www.activebpel.org>) along with Apache Tomcat (<http://tomcat.apache.org>) and Apache Axis (<http://ws.apache.org/axis>). The BPEL process was developed using the ActiveBPEL Designer.

### 3.2 Managing Exceptions in BPEL

A travel booking process may involve processes that can span multiple enterprises. It requires successful completion of several activities, such as hotel reservation, car rental and flight booking. These services, no matter how they are distributed across platforms and companies, all succeed or fail as a unit. In such a business environment, where even more complex workflows are built and communication with several partners may take place, more faults and exceptions are likely to appear, which may cause unpredictable results during the process execution. Therefore, the success of a business process is strongly related to the ability of the system to deal with exceptions.

In order to handle exceptions during workflow execution, BPEL provides backward-recovery procedures based on compensation handling. More specifically, in the case of a fault, several operations that were completed successfully need to be undone. The role of compensation handling is to undo these successfully executed units of work.

In the travel booking scenario described in this paper, a fault is likely to occur in several phases of the process. For example, when the user performs a search for available solutions, the web service of a hotel partner may be unavailable, which causes a fault during the invocation of its partner link. This fault, however, should not affect the overall process since the remaining solutions can be returned to the user. This problem can be easily overcome by using a fault handler, which catches the fault and ensures the normal completion of the process. The situation is different considering the booking phase, where the user submits a request for hotel, flight and car reservation. In this case, a failure in the flight reservation service should force cancellation of the entire travel reservation process. This requires that the rest of the remaining successfully completed services (hotel / car reservation) should be undone, by applying an appropriate compensation mechanism.

In Figure 2, the BPEL code for the implementation of the compensation handling mechanism is presented. This part of the process is enclosed in a *scope* container (ScopeBookAll), which is responsible for the invocation of the room reservation and flight reservation services. These two services are executed in parallel, as they are enclosed in a *flow* container (lines 10-35). The scope "ScopeBookRoom" (lines 11-22) includes an *invoke* activity for the execution of the hotel booking service along with an accompanying compensation handler. If the hotel booking operation needs to be undone, the compensation handler undertakes the task to cancel the booking by invoking the cancellation service. The flight booking service execution is treated in a similar way (lines 23-34).

The fault handler of the scope "ScopeBookAll" (lines 2-9) includes two *catch* activities. Each one is associated with a fault name, "cannotBookRoom" and "cannotBookFlight", which are described in the WSDL files of the respective web services. When a fault occurs in the room reservation service, the *catch* activity "cannotBookRoom" is activated and calls the *compensateScope* with "ScopeBookFlight" as its target. This means that if the room reservation fails and the flight reservation completes successfully, a compensation mechanism is invoked to undo the flight reservation (cancel Flight Booking).

## 4. Conclusions

From the implementation of the travel booking scenario described above, the compensation mechanism of BPEL was analyzed. This analysis has shown that the compensation handling mechanism provided by BPEL has several limitations. The most significant one stems from the fact that in BPEL the process is both the initiator and the coordinator of the transaction. The process itself is responsible for managing the transaction, i.e. listen to faults, start compensation, etc. In such a local coordination model, the BPEL process decides alone on whether compensation is needed. In case a fault occurs, the process initiates compensation without interacting with the

partner to check if it is desirable. Consequently, if a fault occurs at the partner, it will not be noticed by the process, which may result in failure of the compensation operation and produce an inconsistent state.

For large-scale business processes that span across enterprises, the implementation of compensation mechanisms with BPEL is a challenging task. The developer has to write code to take care the transaction handling and the scope definition and to nest them correctly. Additionally, the developer has to take care not only of the process transactions but also of the mechanisms for reversing the operations of the partner web services

However compensation handling in BPEL is a quite sophisticated mechanism that gives developers great flexibility to model operations for undoing previous actions. It can be used as an alternative approach to two-phase commit, but it is more appropriate for long-running transactions. Two-phase commit is advisable for database transactions that can be handled as ACID transactions and allow resources to be locked. On the other hand, for long-running business processes, it is impractical to lock resources for several hours or days. By using BPEL Compensation Handlers, locking of resources is avoided. Regarding the travel booking scenario, the use of compensation handling allows transactions related to room or flight reservation to be completed without the need to lock resources. This is very important taking into account that these transactions may last for long periods of time.

Overall, BPEL despite its addressable limitations in error and compensation, presents a unique ability to integrate and assemble individual Web services into standards-based business processes. It is an important element of the service-oriented enterprise and the overall Web service technology stack. It provides developers with an easy-to-use framework to design sophisticated distributed business processes, enabling creation of complex workflows and orchestration of services from multiple platforms and enterprises.

#### References

- Alkhatib G., Labban R.S. (2002). Transaction Management in Distributed Database Systems: the Case of Oracle's Two-Phase Commit. *Journal of Information Systems Education*, Vol.13, pp. 95-104.
- Arkin A., Agrawal, A. (2001). *Business Process Modelling Language (BPML)*. Working Draft 0.4, pp.1-100.
- Charfi A., Schmeling B. and Mezini M. (2007). *Transactional BPEL Processes with AO4BPEL Aspects*. In Proc. European Conf. Web Services, IEEE CS Press, pp. 149-158.
- Decker G., Kopp O., Leymann F., and Weske M. (2007). *BPEL4Chor: Extending BPEL for Modelling Choreographies*. ICWS, pp. 296-303.
- He Yi, Watters P.A. (2007). *On the complexity of compensation handling in WS-BPEL 2.0 for 3<sup>rd</sup> party logistics*. Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST), pp.118-121.
- Karelitis C., Vassilakis C. and Georgiadis P. (2007). *Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling*. ICWS, pp. 51-758
- Kolawa A. (2006). BPEL in a Service-Oriented Architecture. *InfoManagement Direct*, June 2, pp.1-5.
- Rosenberg F., Dustdar S. (2005). Business rules integration in BPEL - a service-oriented approach. *E-Commerce Technology*, pp. 476-479.
- Singhal M., Shivaratri N. (1994). *Advanced Concepts in Operating Systems*. McGraw-Hill, 1-522.
- <http://www.activevos.com>
- <http://www.activebpel.org>
- <http://tomcat.apache.org>
- <http://ws.apache.org/axis>

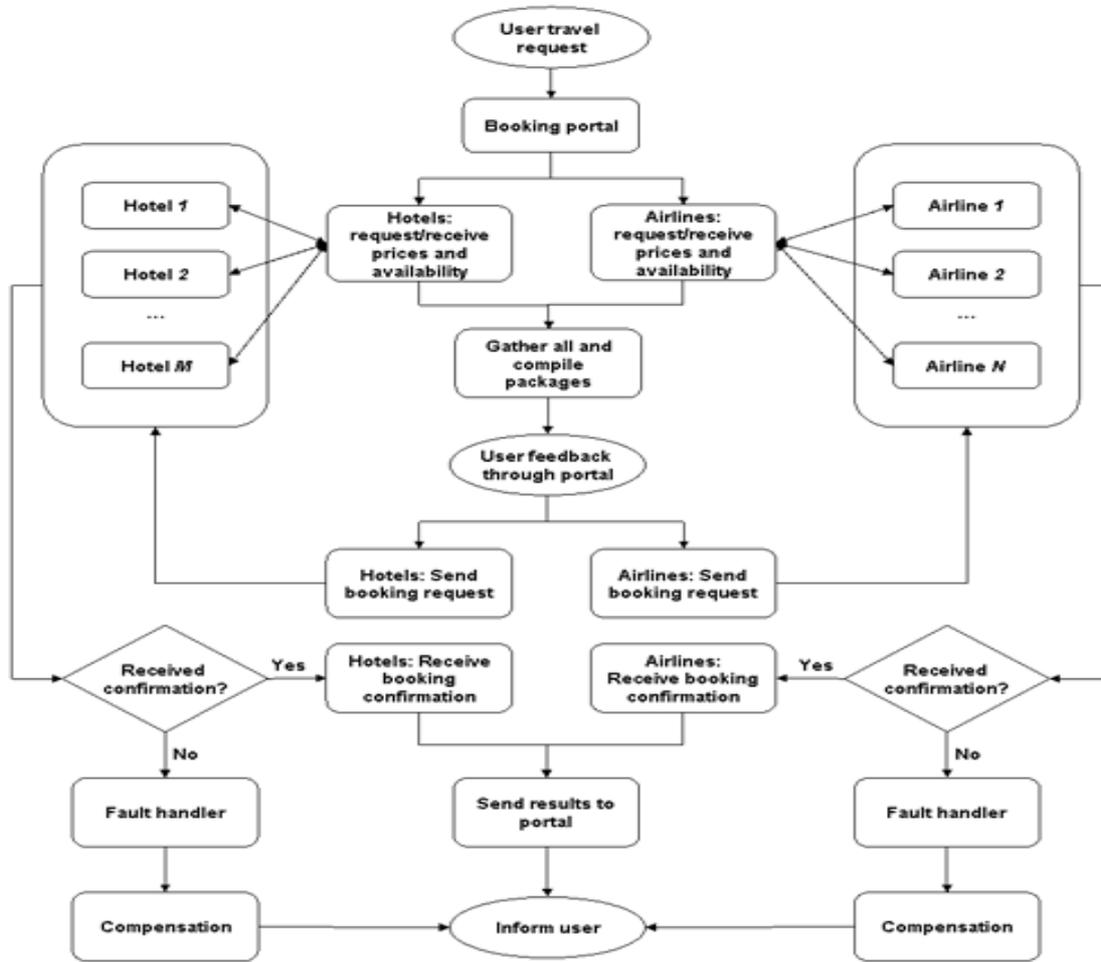


Figure 1. Travel booking portal scenario

```
1 <scope name="ScopeBookAll">
2   <faultHandlers>
3     <catch faultName="ns6:cannotBookRoom">
4       <compensateScope target="ScopeBookFlight" />
5     </catch>
6     <catch faultName="ns6:cannotBookFlight">
7       <compensateScope target="ScopeBookRoom" />
8     </catch>
9   </faultHandlers>
10  <flow>
11    <scope name="ScopeBookRoom">
12      <compensationHandler>
13        <invoke partnerLink="HotelServicePortType_PL"
14          operation="cancelHotelBooking">
15        </invoke>
16      </compensationHandler>
17      <sequence>
18        <invoke partnerLink="HotelServicePortType_PL"
19          operation="makeHotelBooking">
20        </invoke>
21      </sequence>
22    </scope>
23    <scope name="ScopeBookFlight">
24      <compensationHandler>
25        <invoke partnerLink="FlightServicePortType_PL"
26          operation="cancelFlightBooking">
27        </invoke>
28      </compensationHandler>
29      <sequence>
30        <invoke partnerLink="HotelServicePortType_PL"
31          operation="makeFlightBooking">
32        </invoke>
33      </sequence>
34    </scope>
35  </flow>
36 </scope>
```

Figure 2. Managing exceptions in travel booking using BPEL compensation handling