Fault-Tolerant Business Processes *

Sanka Samaranayake, Ricardo Jiménez-Peris, Marta Patiño-Martínez

Facultad de Informática
Universidad Politécnica de Madrid, Spain
{ssamaranayake,rjimenez,mpatino}@fi.upm.es

Abstract. Service-oriented computing (SOC) paradigm promotes the idea of assembling application components into a network of loosely coupled services. Web services are the most promising SOC-based technology. A BPEL process definition represents a composite service that encapsulates some complex business logic including the invocation to other (external) web services. The complexity of a BPEL process together with the invocation of external services subject to network and computer failures requires countermeasures to tolerate this kind of failures. In this paper we present an overview of FT-BPEL, a fault-tolerant implementation of BPEL that copes both with failures of the machine running the BPEL process and network failures in a transparent way, that is, after a failure the system is able to resume the BPEL process consistently.

1 Introduction

Service-oriented computing paradigm promotes the idea of assembling application components into a network of loosely coupled services. Services are autonomous, independent entities that can be described, published, discovered and loosely coupled. Application developers can compose rapid, low cost and evolvable applications by discovering and invoking the network-available services to accomplish some task ranging from a simple query to execution of sophisticated business logic [11].

Composite services, services that invoke other services, often are long running activities, loosely coupled and span across multiple layers of service consumer and providers [12]. Web services are the most promising SOC-based technology. They use Internet as the communication medium and open Internet-based standards, including Simple Object Access Protocol (SOAP) [6] for packaging/routing data, Web Services Definition Language (WSDL) [7] for defining services, Business Process Execution Language for Web Services (BPEL-WS) [3] for orchestrating services. A BPEL process definition represents a composite service that encapsulates some complex business logic. Web services consists of complementary standards set, atop the core to provide various Quality-of-services including WS-Addressing [5] to encapsulate message routing information, WS-Reliable Messaging [4] for ensuring reliable message delivery etc. Due to the complexity and

* This work has been partially funded by the Spanish Research Council (MiCCIN) under TIN2010-19077, by the Madrid Research Foundation, S2009/TIC-1692 (co-funded by FEDER & FSE), and CCG10-UPM/TIC-5855.

long running nature of composite services they are susceptible to a wide variety of failures, for instance, lost messages due to unreliable communication links, crash of the services, crash of the composite service.
Inconsistencies may arise if the solution to tolerate computer failures (crash failures) is simply restarting a composite service. The process may invoke third party stateful services, for instance booking a hotel room or buying an airplane ticket. Repeating the process execution from the very beginning leads in such scenarios to undesirable situations where invocations are repeated. That is, there is no guarantee that a service invocation is performed *exactly once*. This paper presents the architecture and design of FT-BPEL, a framework for reliable web services orchestration built in BPEL. The framework implements passive replication, that is, the process state is persisted and in case of a crash, the process is restarted on a different machine from the point where the failure happened avoiding repeating the execution of the actions the process successfully executed before the failure. The framework tolerates communication failures.

The rest of the paper is organized as follows. Section 2 presents some concepts of fault-tolerance and design goals. Section 3 presents the proposed architecture. Section 4 describes how we deal with failures. Finally, Sections 5 and 6 present related work and conclusions.

2 Fault Tolerance

FT-BPEL deals with two kinds of failures: crash of the machine running BPEL and communication failures. The goal is to provide failure transparency, that is, after a failure the system recovers and resumes execution from the point where the failure happened avoiding duplicate requests. This is important for any stateful service. In order to tolerate crash failures, FT-BPEL implements the so called *passive replication model*. That is, the state of running processes is persisted during normal execution (*checkpoint*). If there is a failure, the process is started on a different computer and its execution resumes from the last checkpoint. The main challenge here is to avoid repeating the execution of the process from this point till the point where the failure happened, especially those actions that may change the state of the process or invoke another service. For instance, let us assume that a checkpoint of a BPEL process is made and then, it invokes an external service to buy a theatre ticket. The BPEL process fails before the response is received. When the process is resumed from the last checkpoint, the service invocation is repeated and the result is that two tickets are bought. These duplicate invocations are avoided by FT-BELP.

A similar situation arises in the presence of network failures. If the external service is executing on a different organization, network disconnections may happen and invocations may not reach its destination. So, the client of this service upon a failure notification, it may retry the invocation. It may happen that the invocation reached the service but the response did not reach the client because a disconnection happened in between. In this case, the client invocation will be executed twice.

The goal of FT-BPEL is to deal with these situations where a failure may lead to duplicate executions. This is an important issue in the context of web services where service autonomy must be preserved. We have designed our framework to meet the following goals:
Failure Transparency. FT-BPEL is able to recover from crash failures of the machine running BPEL processes in such a way that is completely transparent to the users and external services of the system.
Interoperability. FT-BPEL framework relies only on open Internet-based standards for web services including WS- Addressing for message correlation, WS-Reliable Messaging for reliable message delivery and duplicate filtering.
Ease of Use. FT-BPEL uses Apache ODE [1] as the orchestration engine and WSO2 Mercury [2] as the WS-Reliable Messaging provider. Any standard BPEL process definition can be deployed in FT-BPEL without any modification.

3 FT-BPEL Architecture

Figure 1 provides an architectural overview of FT-BPEL framework. Its execution environment consists of an ODE runtime as the BPEL orchestration engine, Mercury runtime as the WS-Reliable Messaging provider and FT-BPEL integration layer that mediates the communication between the orchestration engine and reliable messaging service provider.

WS-ReliableMessaging allows reliably sending SOAP messages between two web services over an unreliable infrastructure. It provides the following guarantees:*AtLeastOnce*, *AtMostOnce*, *ExactlyOnce* and *InOrder*. *AtLeastOnce* means that messages may be delivered more than once (duplicates). If a message cannot be delivered, an error is raised. *AtMostOnce* means that there will be no duplicates but a message may not be delivered. *ExactlyOnce* guarantees that a message is delivered exactly once (no duplicates). If a message cannot be delivered, an error is raised. *InOrder* guarantees FIFO delivery of messages. This property can be combined with the previous ones.

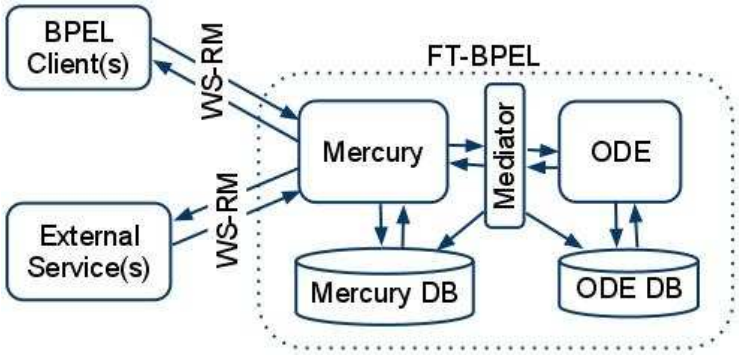

Fig. 1. FT-BPEL Architectural Overview

Our architecture uses *WS-ReliableMessaging* with *ExactlyOnce* delivery for the communication among web services. Both clients invoking a BPEL process and invocations from a BPEL process use *WS-ReliableMessaging* with Exactly-Once delivery guarantees.

WS-Reliable Messaging is transparent to the existing applications. In WS-Reliable messaging systems, there are handlers or agents which reside in the client's or server's SOAP processing engines that transfer messages, handle retries and deliver to the application. These agents are not visible at application level and they ensure that messages get retransmitted if lost or undelivered.

FT-BPEL mediates the communication between the orchestration engine and the reliable messaging service. FT-BPEL assumes that all clients and external services use a WS-Reliable Messaging protocol for reliable communication. Each message exchanged is passed through a logical reliable messaging channel called sequence. Each sequence carries a unique sequence identifier. We are using Mercury as the WS-Reliable Messaging provider. Mercury uses a database to store the state of the communication and guarantee the delivery of messages in case of failures. Mercury is able of retransmitting any messages that are outstanding in the outgoing buffer and the ones that are sent but not acknowledged after predefined (configurable) amount of time. It discards any duplicates. Therefore, any message losses due to network failures are dealt accordingly.

ODE also uses its own database to persist the state of each process instance after an activity is executed. So, if there is a failure, the instances running at the time the failure happened are restored and execution resumes from the last activity executed.

In order to provide high availability, the databases may be replicated in different nodes. We are currently using MySQL replication as replicated database. However, our implementation is independent of the database management system. High available LAN configuration can be used to detect a failure of FT-BPEL runtime and redirect any incoming traffic to FT-BPEL slave runtime.

4 System Failures and Recovery

In this section we present how we deal with failures during the execution of a request from a BPEL client to the BPEL server. Request from the BPEL server to other services are implemented in a similar way.

Upon the successful reception of an invocation request from a BPEL client via Mercury, FT-BPEL mediator will inject it to ODE runtime. If a failure occurs during communication between BPEL Client and FT-BPEL Server, Mercury will resume any incomplete reliable messaging sequences upon recovery. Mercury delivers any messages it receives to FT-BPEL mediator by executing a single callback function. After the completion of this callback, Mercury marks the sequence as a successful delivery by writing the appropriate changes to its database. FT-BPEL injects any invocation request to ODE runtime in the context of a transaction. If a failure occurs during this transaction, ODE runtime would be restored to state before the injection. Since this failure happens during

the callback from Mercury to FT-BPEL and while awaiting its completion, Mercury will re-attempt the delivery. Hence FT-BPEL recovers from these failures.

However, if the failure occurs just after the completion of the ODE transaction, but before Mercury marks the sequence as a successful delivery, duplicate invocations can be produced. Upon recovery, Mercury will re-attempt the delivery and the invocation request will get injected again to ODE. This leads to duplicate invocations of the BPEL process. FT-BPEL utilizes ODE's capability to associate an identifier (i.e. client key) with the request to be injected. ODE guarantees that the response will contain the same identifier. FT-BPEL mediator uses the reliable messaging sequence identifier through which it receives the request as the client key. Before FT-BPEL mediator injects the request to ODE runtime, it first checks for the existing of the client key in the ODE database. If it exists, the request is discarded as a duplicate.

When the response is ready, ODE runtime notifies FT-BPEL mediator by executing a callback function. FT-BPEL mediator extracts the client-key from the ODE response which is the sequence identifier from which it received the request. It writes the response message to the correct reliable messaging sequence in the Mercury database. This triggers Mercury to deliver the response to the BPEL client. This happens within a Mercury transaction. Upon completion, ODE runtime is notified explicitly. If a failure occurs during the Mercury transaction, Mercury runtime will be restored to the state before the response was written to its database. ODE runtime will re-execute the callback since it was not explicitly notified about the completion. Hence, the response will be delivered at the client.

However, if the failure occurs just after the completion of the transaction but before ODE is notified, the failure causes the ODE runtime to re-execute the callback which in turn will write the same response message to the reliable messaging sequence in Mercury database causing it to deliver the same response message to the client again. To prevent this, FT-BPEL mediator always checks whether the response exists in Mercury database before writing it. If exits, FT-BPEL mediator discards the response and notifies ODE runtime explicitly.

5 Related Work

Several techniques have been proposed for reliable service orchestrations in unreliable environments. [9] suggested the use of atomic execution of services and exception handling. [13] proposed the use of declarative, reusable fault handling logic (exception handling policies) and use of system infrastructure to generate an exception-aware process schema. [10] presented an approach where a BPEL process is annotated with recovery actions and then converting the annotated WS-BPEL into a standard BPEL. [8] presented a set of extensible recovery policies to specify how to handle and recover from typical faults in web services composition. Their approach delegated the enforcement of recovery policies to the underlying messaging middleware. In general, they adopt two fault-tolerant mechanisms, transactions for backward recovery and exception handling for for-

ward recovery. Fault recovery logic is embedded into process description and converted into standard BPEL process with various degrees of automation.
Our work is different as we focus on providing a fault-tolerant mechanism against system failures (crash failures) and not against faults that are thrown by services.

6 Conclusions

In this paper we have presented a fault tolerant framework for web services orchestration. We argue that state persistence combined with web services reliable messaging can be used for consistent crash recovery and guarantee 'exactly once' external service invocations. The framework is constructed using state of the art technologies. Currently, we are running a thorough evaluation of the system.

References

1. Apache ODE. http://ode.apache.org/.
2. WSO2 Mercury. http://wso2.org/projects/commons/mercury.
3. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, version 1.1. *Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation*, 2003.
4. R. Bilorusets, D. Box, L.F. Cabrera, D. Davis, D. Ferguson, C. Ferris, T. Freund, M.A. Hondo, J. Ibbotson, L. Jin, et al. Web services reliable messaging protocol (WS-ReliableMessaging). *Specification, BEA, IBM, Microsoft and TIBCO, http://www-128. ibm. com/developerworks/library/specification/ws-rm*, 2005.
5. D. Box, F. Curbera, et al. Web services addressing (WS-Addressing), 2004.
6. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (SOAP) 1.1, 2000.
7. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1, 2001.
8. A. Erradi, P. Maheshwari, and V. Tosic. Recovery policies for enhancing web services reliability. 2006.
9. A. Liu, L. Huang, Q. Li, and M. Xiao. Fault-tolerant orchestration of transactional Web services. *Web Information Systems–WISE 2006*, pages 90–101, 2006.
10. S. Modafferi and E. Conforti. Methods for enabling recovery actions in WS-BPEL. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 219–236, 2006.
11. M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.
12. C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.
13. Liangzhao Zeng, Hui Lei, Jun jang Jeng, Jen-Yao Chung, and B. Benatallah. Policy-driven exception-management for composite web services. *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on*, pages 355–363, July 2005.