Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 121 (2017) 1006–1013

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS / ProjMAN / HCist 2017, 8-10 November 2017, Barcelona, Spain

Extracting and Conserving Production Data as Test Cases in Executable Business Process Architectures

Daniel Lübke[a]*

[a]*Leibniz Universität Hannover, FG Software Engineering, Welfengarten 1, D-30167 Hannover*

Abstract

Because executable business processes are an important and critical software asset of organizations because they control and integrate critical information systems. Thus, testing them thoroughly is a very important task within the software development process. However, failures due to implementation defects still occur in production, which in turn means that the development team needs to analyze, fix and repair the failing processes. In order to support the activities of reproducing the problem outside of the production system and to create better test cases for verifying the fixed implementation, we propose to use process mining techniques on the production process event logs to aid the support & development teams. With our approach it is possible to automatically extract a working unit test case with all partner services being mocked that can run in a development environment. Within in this paper we present the extraction algorithm, our implementation, and possible ways to integrate the tool into the support & development process.

Keywords: Process Mining; Regression Test; Unit Test; Test Case Extraction

* Corresponding author. Tel.: +49-511-762-19667; fax: +49-511-762-19679.
E-mail address: daniel.luebke@inf.uni-hannover.de

1877-0509 © 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies.
10.1016/j.procs.2017.11.130

1. Introduction and Motivation

Executable Business Processes modeled in WS-BPEL or BPMN 2.0 are resembling the implemented business processes of an organization by orchestrating services of other Information Systems. Examples of projects that have successfully used these technologies can be found in various domains, including Telco[1], Mortgage[2] and many other domains[3]. Being a critical software component integrating information systems and passing data over system boundaries, executable processes should be subject to rigor quality assurance. This usually means that executable test cases are developed[4] that cover as much of the processes as possible[5,6]. However, as with any software system, failures in production occur. These failures can be fixed by using the Administrative Console of the run-time environment called a Business Process Management System (e.g. rewinding a process, changing variable values or retrying service calls.) If the failure is due to an implementation error, the support team very often needs to forward the incident to the development team that needs to repair incorrect variables (in case of data transformation errors), fix the root cause and write a regression test that verifies and ensures that the defect has been fixed. This is often complicated because developers have no access to the production environment. For better development performance and improvement of the process repair and the software fix, it would beneficial if the problem could be easily reproduced on a non-production environment and a test case was available: Developers could safely try repairing the process without affecting production data, develop and verify the fix, and eventually use the data generated by the new process version for the repair.

Although all the relevant data is contained in the Business Process Management System, these systems do not offer a way to export the data packaged for such an important task and do not offer support for replicating failing cases outside the production system. Within this paper, we propose to use process mining techniques in order to extract all relevant data of a process instance and store the information in a Replication Test Case, which is a unit test case completely independent of the production environment, i.e. all information systems are replaced with mocks that replay the original service calls.

2. BPEL Process Structure

Within this paper we use process mining[7] techniques on BPEL processes[8]. BPEL (Business Process Execution Language) is a language for modeling executable business processes that orchestrate services in order to realize the business goals. However, our approach works with all process languages that have facilities to call and offer services (or remote interfaces.)

The relevant BPEL concepts are shown in the (simplified) meta-model in Fig. 1: Most importantly, BPEL offers a standardized way to specify, which services are called or are offered by an executable process. BPEL defines *invoke* and *reply* activities for sending messages and *receive* and *pick* activities as well as message handlers for receiving messages. All these activities reference a partner link. A partner link references the concrete service via a partner link type. The service is a port type from a WSDL (Web Service Description Language), which defines operations. Operations can be either one-way operations or two-way operations. This means that an operation has a fire-and-forget semantic with a single message or defines a return message called output message.

Besides defining service calls, BPEL has an assign activity for defining the data-flow. All data is stored in variables: Assign activities read from and write to variables (or parts thereof), and the service-related activities read and write complete messages from/to variables.

The rest of the BPEL activities is concerned with defining the process-flow (loops, conditions etc., e.g. the *sequence*, *forEach*, *repeat* und *while* activities) but are not important to our approach, because we use the process log to reconstruct the ordering of the messages being sent.

BPEL itself is not compiled but it is instead deployed to a Business Process Management System (BPMS). The BPMS has the responsibility of managing the execution of the processes, the versioning of different models and the persistence of all information necessary for executing the processes. Usually, a BPMS can be configured to also write a Process Log (also called Event Log), in which all events regarding the process execution (e.g. when an activity started and was finished) are persisted and can be used later on for feeding Business Reporting or Business Intelligence tools. For example, when process execution is managed by a BPMS, it is easy to do statistics for end-to-end process execution times. However, the format of Process Logs has not been standardized and is dependent on the vendor.

Also the graphical representation of BPEL Processes is not defined by the standard. Because of this, we use BPMN[9] to visualize the executable business processes in this paper.

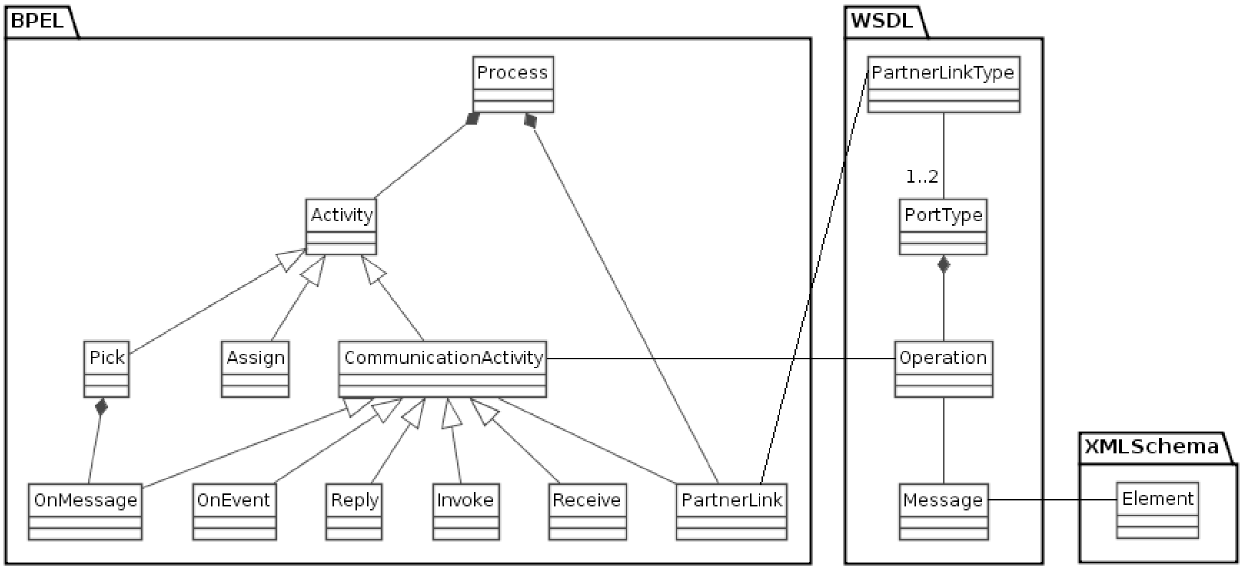


Fig. 1. Simplified Meta-Model of BPEL with Relationships to WSDL and XML Schema.

3. Process Log Analysis

Because the Process Log is specific to the BPMS used, we first needed to analyze a commercially available BPMS. The analysis was done with the aim of identifying the relevant information for mining a Replication Test Case: How can the execution order of activities be reconstructed and how can the data sent from and to the executable process be extracted? The first finding was that the necessary information is only contained in the Process Log, if the BPMS is configured to use the highest process persistence level. When this prerequisite is fulfilled, all relevant events - including the data - is recorded. This especially includes the following Event Types:

- Start Executing Activity
- Activity Completed
- Variable Changed
- Message Received

All events contain the Event Type but also a reference to the affected process element. Usually this is an activity except in the case of the *Variable Changed* event, which references a variable. A *Variable Changed* event contains also the new data of the updated variable.

In the next step we created different Process Logs by deploying and executing small processes that combined contained all message-related activities. This way, we identified the event patterns for the different message exchanges containing SOAP one-way and two-way operations as shown in Table 1. Messages received by the process (and thus later sent by the Replication Test Case) were indicated by their own event type. The *Message Received* events are followed by a respective *Variable Changed* events. Both are created between the start and the completion of an activity.

There is no dedicated event that indicates that a message is sent. This data must be resolved by finding the variable that carries the message sent via the structure of the process model. This means that the mining algorithm must combine information from the Process Model with information contained in the Process Log. The variable value must then be constructed from the event log by locating the last *Variable Changed* event executed before the message is sent.

One challenges was identified during the implementation of our approach: Because BPEL Activities do not have a unique ID but only a non-unique name, BPEL vendors need to find a way to uniquely reference a process model element in the events. The BPMS, which we used, did this by constructing path expressions similar to XPath. However, the path expressions could not be mapped trivially to the process model but we needed to reverse engineer how these path expressions are constructed.

Table 1. Process Log Excerpts for different Activities

Receive	Reply	Invoke (One-Way)	Invoke (Two-Way)	Pick w/ OnMessage	OnEvent Handler
1 Executing Receive	1 Executing Reply	1 Executing Invoke	1 Executing Invoke	1 Executing Pick	1 Executing EventHandlers
2 Message Received	2 Message Reply	2 Completed Invoke	2 Variable Changed	2 Ready to Execute onMessage	2 Executing OnEvent
3 Variable Change	3 Completed Reply		3 Completed Invoke	…	…
4 Receive Completed				3 Message Received	3 Message Received
				4 Variable Changed	4 Ready to Execute Scope
				…	5 Executing Scope
				5 Completed OnMessage	6 Variable Changed
				6 Completed Pick	…
					7 Completed Scope

4. Test Case Extraction Algorithm

With the information about the Process Log structure available, we designed an algorithm that recreates the process instance and stores the information in a Replication Test Case. In general, we save all variable values if we encounter a variable change, thereby recreating the variable states as we replay the events (lines 18-20). For *receive*, *reply*, and *invoke* activities the handling is similar: We wait for the completion event (line 6), resolve the activity in the BPEL model (line 7), follow the model to the operation (lines 8+9) and from there to the variable name (lines 10+11) and finally extract the variable data in the variables being sent or received (line 11). With this information, a new activity in the test case is created (lines 12-16). Because the *onMessage* and *onEvent* elements are stored differently in the Event Log, we need to remember if the last encountered event was a *Message Received* event. Whether it is directly followed by a *Variable Change* event (*onMessage*) or followed by a *scope* and a *Variable Change* event (*onEvent*), we can differentiate and properly handle the two cases. This is realized by remembering the previous activities in a variable called *lastEventEventWasMessageReceive*, which is set to true when encountering a message receive on an *onEvent* and *onMessage* (line 22) and is kept at true if a scope follows directly (lines 2-4). The whole algorithm is as follows:

```
1   for e in E
2     if NOT(lastEventEventWasMessageReceive AND e.activity.type = scope) then
3       lastEventEventWasMessageReceive <- false
4     end if

5     onEvent <- (lastEventEventWasMessageReceive AND e.type == VARIABLE_CHANGED)

6     if (e.type = COMPLETED AND e.activity.type IN (invoke, receive, reply)) OR onEvent then
7       communicationActivity <- onEvent ? lastMessageReceiveOn : e.activity
8       partner <- resolvePartner(communicationActivity)
9       operation <- resolveOperation(communicationActivity, partner)
```

```
      inputMessage <- variables[communicationActivity.inputVariable]
      outputMessage <- variables[communicationActivity.outputVariable]

      if e.activity.type = receive then
         a.testCase.partners[partner].add(new TestActivity(operation, inputMessage))
      else if e.activity.type = reply  then
         a.testCase.partners[partner].last.outputMessage <- outputMessage
      else if e.activity.type = invoke and operation.isOneWay then
         a.testCase.partners[partner].add(new TestActivity(operation, outputMessage))
      else if e.activity.type = invoke and operation.isTwoWay then
         a.testCase.partners[partner].add(
                     new TestActivity(operation, outputMessage, inputMessage))
      end if
    end if

    if e.type == VARIABLE_CHANGED then
      variables[e.variable] <- e.variableValue
    end if

    if e.type = MESSAGE_RECEIVED AND e.activity.type IN (onEvent, onMessage) then
      lastEventWasMessageReceive <- true
      lastMessageReceiveOn <- e.activity
    end if
  next
```

5. Example

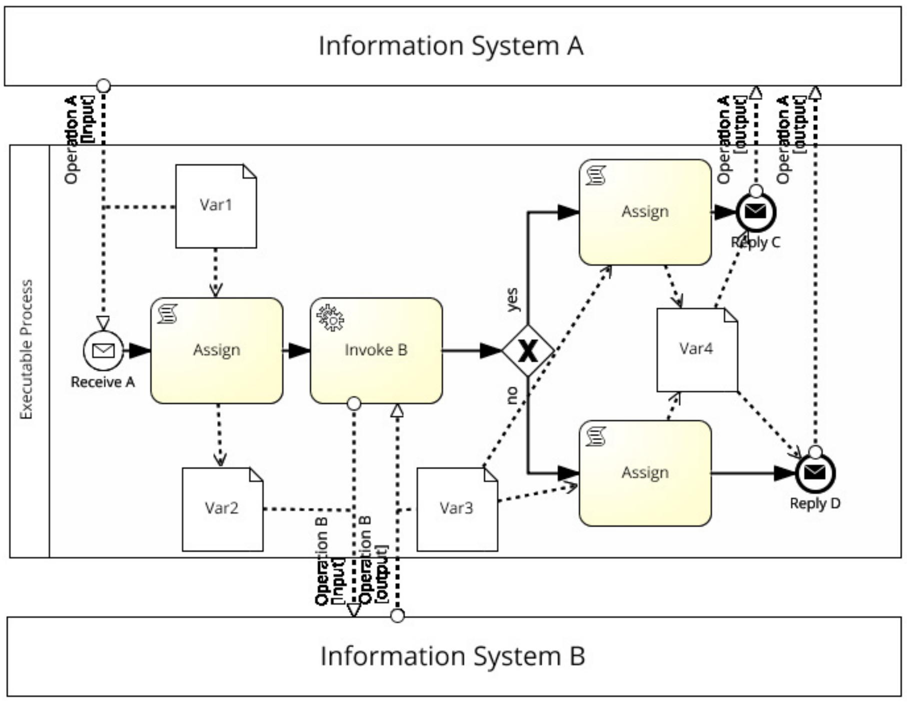


Fig. 2. Simple Example Executable Process integrating two Information Systems.

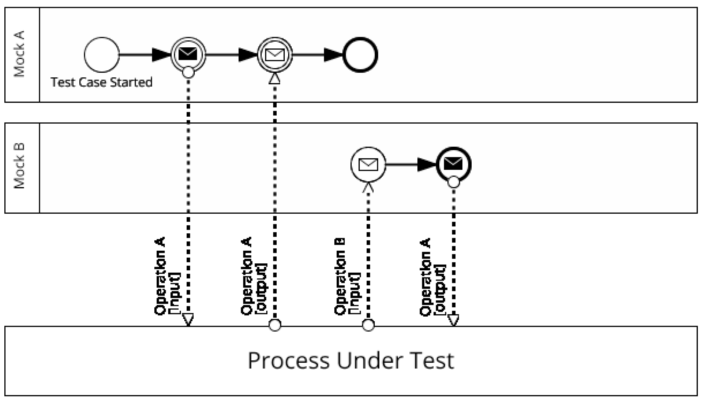
Within this section we illustrate our approach by providing an example. The process shown in Figure 2 is very simple but sufficient to demonstrate how the algorithm works. We replay the Process Log of one process instance on top of the process model. An excerpt with the important events is shown in Table 2: First, the variable change is processed that stores the message used for starting the process into *Var1*. Then the completion of the *Receive* A is processed. For this, the algorithm resolves which variable, partner and operation is referenced by this activity in the process model. Thus, a new test partner with a two-way test activity is created. The test activity sends the value stored in *Var1*. At this point, the message for the reply to the partner is not known. Next, the variable changes for *Var2* and *Var3* are processed. The success completion of the *Invoke B* activity is processed similar to the processing of *Receive A*: The variables, partner and operation is resolved. Consequently, a new test activity for answering a two-way operation with receiving the value of *Var2* and sending the value of *Var3* back to the process is created in a new partner that mocks *Information System B*. In the next step, the variable change event for *Var4* is processed. Finally, the completed successfully event for *Reply C* is processed. Because it is a reply activity, the reply message in *Var4* is added to the test activity originally created by processing *Receive A*. As can be seen from this example, all start activity events and the successful completion of all other activity types, including the completion of the data transformation activity type assign, are ignored. This allows to fetch only a small part of the Process Log from the production system in order to reduce the load imposed by the data transfer.

The resulting (simple) test case is shown in Fig. 3: Two mocks have been generated and every mock serves the messages for the two operations.

Table 2. Example Process Log.

Event Type	Referenced Element	Data
Variable Changed	**Var1**	**<Message1 />**
Completed Successfully	**/process/sequence/receive[@name='A']**	
Variable Changed	**Var2**	**<Message2/>**
Completed Successfully	/process/sequence/assign	
Variable Changed	**Var3**	**<Message3/>**
Completed Successfully	**/process/sequence/invoke[@name='B']**	
Variable Changed	**Var4**	**<Message4/>**
Completed Successfully	/process/sequence/assign[2]	
Completed Successfully	**/process/sequence/reply[@name='C']**	

Fig. 3. Extracted Test Case.



6. Integration into the Support & Development Process

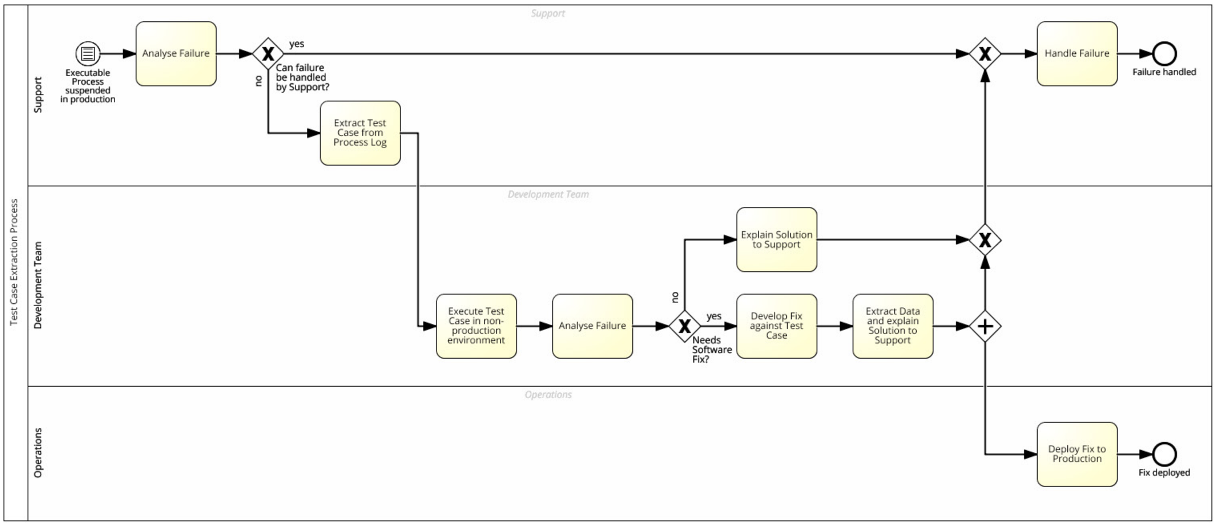

Test Case Extraction Process
Support
Development Team
Operations
Executable Process suspended in production
Analyse Failure
Can failure be handled by Support?
yes
no
Extract Test Case from Process Log
Handle Failure
Failure handled
Execute Test Case in non-production environment
Analyse Failure
Needs Software Fix?
no
yes
Explain Solution to Support
Develop Fix against Test Case
Extract Data and explain Solution to Support
Deploy Fix to Production
Fix deployed

Fig. 4. Failure Handling Process involving the Support, Development and Operations Organizations.

With a tool that implements the described algorithm it is possible to export a failing process instance from the production system and have developers analyze and fix the problem without giving them access to the whole production system, e.g. by following the incident process as shown in Fig. 4: Whenever an executable process fails and the *Support Team* is not able to successfully resolve the problem by themselves, they can extract a Replication Test Case. For this operation, only read-only access to the BPMS is required. The Replication Test Case is then given to the *Development Team*, which can run it on a non-production environment and get to the same process state as the failing one in production repeatedly. Analysis can be done from there and – in case a software fix is required – the Replication Test Case can serve as a validation whether the original problem has been fixed. The information on how to repair the failing process is then passed from the *Development Team* to the *Support Team*. Also the fix could then be passed to the *Operations Team* in order to deploy it.

7. Conclusions & Outlook

Within this paper we demonstrated that process mining techniques can be used to build a Replication Test Case, which can be used to isolate a failing process instance and transfer it off the production environment in order to better analyze and fix the underlying root cause. The Replication Test Case is completely independent of any surrounding services, because the exchanged data is also extracted and consequently can be mocked during test-time. The ability to transfer and analyze executable processes has been used to suggest a new incident management process, which allows to strictly separate roles if required and allows the parties with minimal rights to perform their tasks. Especially the developers can get the required information, which they need without having access to the production system.

However, the approach is heavily dependent on the Business Process Management System being used because there is no existing standard how Process Logs should be created and saved. The algorithm and thus the implementation of this approach is dependent on the events, event types, and event order. As such, the algorithm could look very different if used with a differently structured Process Log.

Possible future work can be to help development teams build automated regression test suites. If the data in the extracted test suite is anonymized and assertions are generated from the messages sent by the executable process automatically, regression tests can be developed by extracting and anonymizing productive or manually tested process instances. In this way, it would be easily possible to add automated tests for already implemented and frequently used features.

References

1. Zimmermann, O.; Doubrovski, V.; Grundle, J. & Hogg, K. Service-oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, ACM, 2005, 301-312.
2. Berli, W.; Lübke, D. & Möckli, W. Plödereder, E.; Grunske, L.; Schneider, E. & Ull, D. (Eds.) Terravis -- Large Scale Business Process Integration between Public and Private Partners Lecture Notes in Informatics (LNI), Proceedings INFORMATIK 2014, Gesellschaft für Informatik e.V., 2014, P-232, 1075-1090.
3. Hertis, M. & Juric, M. B. An Empirical Analysis of Business Process Execution Language Usage IEEE Trans. Software Eng., 2014, 40, 738-757.
4. Lübke, D. Unit Testing BPEL Compositions; in Baresi, L. & Nitto, E. D. (Eds.) Test and Analysis of Service-Oriented Systems Springer, 2007.
5. Yang, Q.; Li, J. J. & Weiss, D. M. A survey of coverage-based testing tools The Computer Journal, Br Computer Soc, 2009, 52, 589-597.
6. Lübke, D.; Singer, L. & Salnikow, A. Calculating BPEL Test Coverage through Instrumentation Workshop on Automated Software Testing (AST 2009), ICSE 2009, 2009.
7. van der Aalst, W. Process Mining – Data Science in Action, Springer, 2016.
8. OASIS. Web Services Business Process Execution Language Version 2.0 2007.
9. Object Management Group (OMG). Business Process Model and Notation (BPMN), Version 2.0, 2011.