Automated Workflow Regression Testing for Multi-tenant SaaS

Integrated Support in Self-Service Configuration Dashboard

Majid Makki Dimitri Van Landuyt Wouter Joosen
iMinds-DistriNet, KU Leuven 3001 Leuven, Belgium
firstname.lastname@cs.kuleuven.be

ABSTRACT

Single-instance multi-tenant SaaS applications allow tenant administrators to (extensively) customize the application according to the requirements of their organizations. In the specific case of workflow-driven applications, the SaaS provider may offer a set of pre-defined workflow activities and leave their composition to the tenant administrators. In such cases, the tenant administrator can instantiate new variants of the application without deploying new software. This effectively makes these tenant administrators part of the DevOps team, and in turn creates the need for the SaaS provider to provide them with Quality Assurance tool support. One such tool is a regression testing framework that allows them to make sure that a new version of a workflow can behave similarly as to a successful execution of a previous version.

This paper highlights the potential and discusses the inherent challenges of running regression tests on workflows in the production environment of a multi-tenant SaaS application and outlines a solution in terms of architecture and automation techniques for mocking and regression detection under control of tenant administrators.

CCS Concepts

•**Applied computing** → *Business process management; Service-oriented architectures;* •**Computer systems organization** → *Cloud computing;* •**Software and its engineering** → *Software testing and debugging;*

Keywords

Automated Regression Testing, Application-level Multitenancy, Software-as-a-Service

1. INTRODUCTION

Single-instance multi-tenant Software-as-a-Service (SaaS) applications serve multiple tenant organizations with different requirements using a single instance of the application. Tenant administrators configure or customize the application such that it behaves according to the needs of the users associated with their own organization. To ensure the economic viability of the cloud offering, such configuration activities are ideally done entirely in a self-service manner [10, 11], requiring no SaaS provider involvement. However, removing the SaaS provider staff from the configuration process requires certain measures for making sure that the tenant administrators configure the system appropriately.

In case of workflow-driven applications, one important point of customization is the workflow definition (cf. [7, 8, 9]). The customization story can go as far as allowing the tenant administrator to design a workflow consisting of activities predefined by the SaaS provider. While designing workflows is a task of the development team and releasing new software variants is a task of the IT operations team, these two tasks are partially delegated to tenant administrators in economically successful multi-tenant settings. Once these tasks are delegated to tenant administrators, they become part of the DevOps[1] team and partially in charge of their own service.

This however, raises the need to provide the tenant administrators with suitable Quality Assurance (QA) tools as part of the self-service configuration dashboard in the production environment itself [10]. They can also update the workflow definition from time to time when their requirements change. Nonetheless, they may still expect an equivalent behavior from the system under a number of similar circumstances and in such cases automated regression testing shows great potential. Provisioning a separate environment for testing which mirrors the production environment is technically possible but considered too costly, potentially hampering the economic viability of the single-instance multi-tenant SaaS offering.

Running a regression test on a workflow definition in the production environment of a SaaS application is different from running a regression test on a piece of code in the developer's workbench or even the entire application in the development team's integration server. There are challenges rooted from the SaaS business model, long lifespan of workflows (compared to computer programs), complex nature of possible regression types in a workflow, and mission-critical status of artifacts in the production environment.

In this work-in-progress paper, we draw a perspective toward automated regression testing of workflows in SaaS production environment where test cases are directly derived

A-TEST'16, November 18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4401-2/16/11...$15.00
http://dx.doi.org/10.1145/2994291.2994302

[1]An abbreviation for Development and Operations.

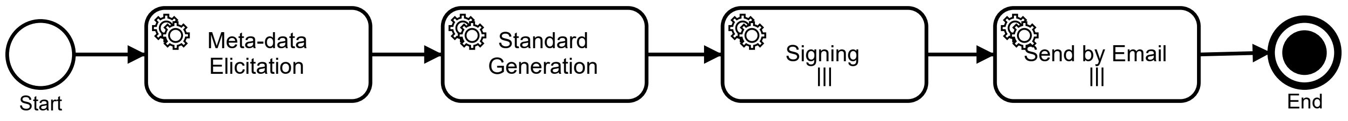

Figure 1: Document Processing Workflow in BPMN 2.0 - Version 1

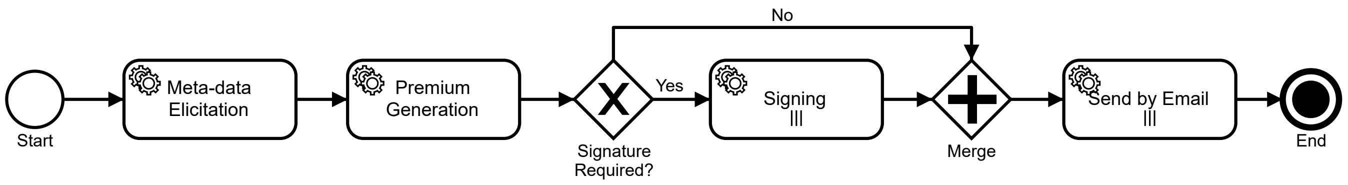

Figure 2: Document Processing Workflow in BPMN 2.0 - Version 2

from previous successful executions. The contributions of this paper are three-fold: (i) we highlight the potential of tightly integrated automation of workflow regression testing in the context of cloud offerings that rely extensively on self-service; (ii) we identify and analyze the inherent challenges of workflows regression testing in a SaaS production environment; and (iii) we outline a solution for overcoming these challenges in terms of an architectural sketch and a set of configurable test automation techniques.

The architectural sketch aims at avoiding the costly mirroring of the production environment and the automation techniques are destined to avoid undesired side-effects of testing in the production environment, to decrease test execution time and to detect regressions vis-à-vis previous versions. The work leading to these contributions are based on frequent and intensive interactions with Belgian workflow-driven SaaS providers in the context of a number of collaborative research projects [1, 3, 6].

The rest of this paper is structured as follows. Section 2 presents a motivating example. Section 3 elaborates on the challenges that the framework should deal with. Section 4 outlines the elements of the testing framework align with test automation techniques. Finally, Section 5 concludes the paper.

2. MOTIVATING EXAMPLE

This section briefly presents a workflow-driven multi-tenant document processing SaaS application where tenant administrators can customize the application behavior by designing a workflow using elements provided by the SaaS provider out-of-the-box. We use Business Process Model and Notation (BPMN) 2.0[2] [2] to illustrate workflows.

The SaaS provider provides tenants with a set of predefined workflow activities and impose some constraints on how these activities can be composed to form a workflow. For instance, using the `Meta-data Elicitation` activity is always mandatory because without the latter it is neither possible to generate documents nor to distribute them. However, for generating documents, two alternative activities are provided namely `Standard Generation` and `Premium Generation`. The tenant administrators can choose any of these two depending on their requirements and budget but they are obliged to put at least one of these activities immediately after `Meta-data Elicictation`. For instance, as shown in Figure 1, *Tenant A* has chosen `Standard Generation` for generating their invoices.

In order to be able to generate and send their leaflets and advertisements using the same system, the administrator of *Tenant A* updates their initially designed workflow into the one depicted in Figure 2. There are two main differences. Firstly, they have chosen `Premium Generation` in order to benefit from the advanced templates offered by the latter. Secondly, the tenant administrator makes the `Signing` activity optional because leaflets and advertisements, as opposed to invoices, are not signed before sending.

Even though the workflow design is changed, it should behave as before for the invoices. In order to make sure that the existing behavior of the system in processing invoices is not disturbed, the tenant administrator wants to run regression tests before activating the new workflow design for their organization. For that purpose, the SaaS provider should offer integrated and automated regression testing in the self-service configuration dashboard.

3. CHALLENGES

In a more realistic scenario, integrating and automating workflow regression tests in the self-service configuration dashboard of a SaaS application involves a number of challenges. This section elaborates on four main challenges that a testing framework should deal with in that regard.

C1: Test Case Generation. Since tenant administrators are not professional QA engineers, the framework should automatically generate a test case by allowing the tenant administrator to select from a previous execution of the workflow. This requires starting the test workflow in exactly the same manner as the selected past execution. This further requires comparing the outcome of two executions based on regression criteria specified by the tenant admin.

C2: Regression Detection Criteria. The effects of workflow executions are reflected in (i) values of workflow variables, (ii) the end nodes where the workflow finishes up, (iii) data implicitly modified or sent out by workflow activities, and (iv) the intermediate events[3] occurred while executing the workflow. However, depending on the change

[2]This is the most recent version of a standardized modeling notation by the Object Management Group (OMG), which, next to the Business Process Exection Language (BPEL) [5] is increasingly being adopted in practice.

[3]Modern workflow technologies (e.g. BPMN2 or BPEL) allow workflows to throw and catch events. For instance, one

scenario urging the regression test, only a subset of these criteria may be relevant. If all items pertaining to these criteria are always considered, the test may end up in a lot of false-positives.

C3: Undesired Side-effects. The SaaS business model, like other areas of cloud computing, is based on the pay-per-use pricing scheme. This implies that executing services involve cost. If workflow activities invoking services not modified in the change scenarios, paying for them only for the purpose of testing is not interesting for the tenants. Moreover, some activities may modify mission-critical data or send them to the outside world (e.g. by sending an email) which is not desirable. Even though such activities should not be executed in the test phase, bypassing them should neither lead to false-positives nor to false-negatives in regression detection. Furthermore, since a workflow is a persistent entity, running a test workflow in the production environment should be performed in such a way that the operational database is not affected by the test execution. This should not partially introduce the costly solution of mirroring the production environment for testing.

C4: Longevity and Interdependence. Modern workflow technologies, such as BPMN2, allows configuring a timer in the workflow definition. However, while executing regression tests, the tenant administrator cannot wait until a timer configured to trigger every Friday at midnight triggers. Furthermore, certain events may not be even thrown at all in the test execution. An example is when the event is thrown by workflow and is supposed to be handled by another. In the normal production execution, the thrown event of the first workflow will be handled by the second workflow. However, if the tenant administrator wants to test the second workflow independently, the event would never be thrown and the second workflow will halt at a certain point. The framework should provide a way to avoid these halts or waiting times.

4. PROPOSED SOLUTION

This section outlines the solution for overcoming the above challenges. This consists of a high-level architectural view of the testing framework, a set of techniques aimed at manipulating the workflow definition before executing test cases and a mechanism for regression detection.

4.1 Overall View

Figure 3 demonstrates the main components and their relationships. The two components highlighted by color are provided by our testing framework and the others are either developed by the SaaS provider or provided by third-parties out-of-the-box. The SaaS developers use the `Test Automation API` to allow the tenant administrators automatically execute regression test cases. The `History Log` repository is updated by the `Snapshot Creator`. The latter is an interceptor which is notified by the workflow engine before and after execution of each step of the workflow. Each time it is notified, it creates a snapshot of the workflow instance and stores it in the `History Log` repository.

The tenant administrator selects one successful execution of the workflow from the `History Log` repository. Then, the

workflow throws an exception if a certain condition is met and a second workflow handles the exception. The workflow engine mediates between the two workflows.

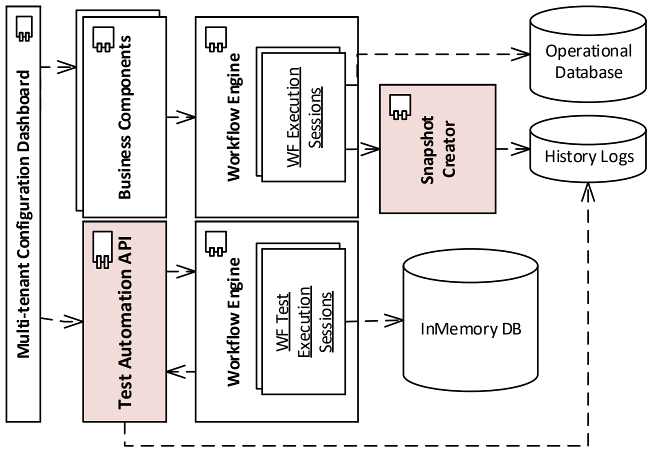

Figure 3: Top-Level Architecture

testing framework should start the test workflow by triggering exactly the same start node[4] and by feeding the workflow instance with the same values assigned to the workflow variables in the selected past execution (cf. ***C1***).

As it is shown in Figure 3, the workflow engine used by the `Test Automation API` is separated from the one used by the `Business Components` in order to keep the data persisted by the workflow engine for the mere purpose of testing out of the `Operational Database`. This is compliant with the third point made in ***C3***, namely the fact that the production environment should not be even partially mirrored, because it allows persisting the test workflow states in an `InMemory Database` which is dropped after the test execution.

4.2 Workflow Manipulation

As implied by ***C3*** and ***C4***, certain workflow steps should be circumvented in test execution. For circumventing such steps, the framework should modify the workflow definition, in the following respects, before starting the test execution.

Mocking Activities. In order to avoid the undesired side-effects discussed in ***C3***, certain activities should be mocked. Mocking means producing the expected effect of an activity without actually executing it. For mocking an activity, the latter should be replaced by a `Mock Activity`.

When the activity has been used in the past executions of the workflow, it is possible to mock the activity by consulting the `History Log` repository. This amounts to querying the snapshots taken before executing the activity when workflow variables holding the same values as mocking time. Subsequently, the test workflow variables should be assigned the same values recorded in the corresponding snapshot of the `History Log` taken after executing the activity.

When the activity has not been used in the past, the only way to mock it is by prompting the tenant administrator for entering the changes that should be made in workflow variables as a result of mocking.

Before applying any of these mocking tactics, the framework should automatically group workflow activities into two groups: *previously-used* and *new*. This can be done

[4]It is assumed that a workflow definition may have multiple start nodes but only one of them can be triggered in every execution. This assumption holds for most modern workflow technologies such as BPMN2 or BPEL.

by comparing the old and the new version of the workflow definition. For instance, by comparing the workflows shown in Figures 1 and 2, the framework should group `Meta-data Elicitation`, `Signing` and `Send by Email` as *previously-used* activities and `Premium Generation` as *new* activity.

Skipping Steps. The time-consuming or hindering workflow steps, discussed in ***C4***, should be skipped over in order to make the test execution feasible and sufficiently rapid. The testing framework should replace such workflow steps by new steps bearing the same name and making no changes in terms of workflow variables. The latter characteristics of the replacement steps guarantees that neither false-positives are caused by them nor do they mask true-positives.

4.3 Regression Detection

As indicated in ***C2***, regression can be detected based on four different criteria. However, the framework should allow the tenant administrator to select any of these as regression detection criteria because any of them might be relevant or irrelevant depending on tenant-specific requirements. For example, a tenant may not be concerned about the end nodes in which the workflow finishes. The tenant admin's control over what to be included in regression detection criteria and what to be excluded from it is even more fine-grained. The framework should allow the tenant administrator to indicate certain elements for each of these four criteria to be ignored in regression detection. This helps the tenant administrator to avoid false-positives where s/he expects a change in the test execution vis-à-vis the successful execution of the past.

The most difficult of the four criteria to evaluate is the data implicitly modified by activities or sent by them to the outside world. Since we do not really execute some activities at the testing phase and mock them instead, it is sufficient to have a trace of all the mocked activities traversed in test execution. The same tracking mechanism is required both for end nodes reached and intermediate events occurred in the course of test workflow execution (cf. ***C2***). Therefore, an observer is injected into the test workflow engine which is notified after each node of the BPMN2 model is traversed. This observer keeps track of all the nodes and the trace will be used in the final phase for detecting regression.

Regression detection consists of comparing any combination (based on tenant-admin's preferences) of the following four elements after the test workflow completes: (i) values assigned to variables; (ii) names of all activities mocked in the course of test workflow execution; (iii) names of all events skipped over (cf. Section 4.2); and (iv) names of end nodes where the test workflow finishes in. Obviously, any variable, activity, event or end node indicated by the tenant administrator to be ignored will be excluded from this final comparison to avoid false-positives.

5. CONCLUSION

In this paper, we discussed the main challenges of embedding a workflow regression testing tool in the production environment of a SaaS application and proposed a solution in terms of a high-level architecture and test automation techniques. The proposed architecture helps avoiding the costly and complex mirroring of the production environment for online test execution by tenant administrators. The testing framework leverages on the workflow executions in the past to both automatically generate a test case and to bypass costly, halting or time-consuming workflow steps. It also enables the tenant administrator to configure the regression detection mechanism.

This work is motivated from our frequent collaborations with industrial partners active in a wide range of application domains, from automated document processing to e-health workflows, and this work fits in our ongoing research on workflow-driven multi-tenant SaaS [1, 3, 7, 6]. We are currently implementing the proposed testing framework on top of jBPM [4] and evaluating it in the context of the industry cases mentioned above.

6. ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven, the ADDIS research program funded by KU Leuven GOA, the DeCoMAdS SBO strategic research project, and the MuDCads O&O project.

7. REFERENCES

[1] D-Base. Decentralized support for Business processes in Application Services (iMinds ICON project), 2014.

[2] Business Process Model and Notation (BPMN). http://www.omg.org/spec/BPMN/2.0/PDF/. Accessed: 2015-08-04.

[3] DeCoMAdS. Deployment and Configuration Middleware for Adaptive Software-as-a-Service (SBO project), 2014-2018.

[4] RedHat JBoss jBPM. http://www.jbpm.org/. Accessed: 2016-02-17.

[5] Web Services Business Process Execution Language. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Accessed: 2016-06-04.

[6] iMinds-OCareCloudS: Organizing Home Care Using a Cloud-based Platform. http://www.iminds.be/en/projects/ocareclouds, June 2016.

[7] M. Makki, D. Van Landuyt, S. Walraven, and W. Joosen. Scalable and manageable customization of workflows in multi-tenant saas offerings. In *Proceedings of the 31st annual acm symposium on applied computing*, pages 432–439. ACM, 2016.

[8] R. Mietzner and F. Leymann. Generation of bpel customization processes for saas applications from variability descriptors. In *Services Computing, 2008. SCC'08. IEEE International Conference on*, volume 2, pages 359–366. IEEE, 2008.

[9] W. M. Van Der Aalst. Business process configuration in the cloud: how to support and analyze multi-tenant processes? In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 3–10. IEEE, 2011.

[10] D. Van Landuyt, S. Walraven, and W. Joosen. Variability middleware for multi-tenant saas applications: a research roadmap for service lines. In *Proceedings of the 19th International Conference on Software Product Line*, pages 211–215. ACM, 2015.

[11] S. Walraven, D. Van Landuyt, E. Truyen, K. Handekyn, and W. Joosen. Efficient customization of multi-tenant software-as-a-service applications with service lines. *Journal of Systems and Software*, 91:48–62, 2014.