A Software Test Approach to Evaluate the Enforcement of a Workflow Engine

Laurent Bobelin, Christian Toinard, Tuan Hiep Tran
LIFO Computer Science Laboratory of Orléans
INSA Centre Val de Loire
88 bld Lahitolle CS 60013 18022 Bourges, France
{laurent.bobelin, christian.toinard, tuan.tran}@insa-cvl.fr

Stéphane Moinard
Qualnet
24 Route du Creton, 18110 Vasselay, France
stephane.moinard@qualnet.fr

Abstract—Conformance testing is a critical problem of workflow systems. Indeed, the execution engine must satisfy the specification of the workflows coming from the end users. The problem of testing is un-decidable in the general case. The difficulty is thus to provide a heuristic that minimizes the suite of test sequences. Despite many theoretical works exist in the literature, concrete solutions are really missing. This paper describes a software solution to this problem. It is implemented as a standalone component that can take a workflow specification as input and gives the test sequences as outputs. That approach enforces workflows coming from the Intraqual environment of Qualnet but it can serve as a general solution for testing any other workflow engine.

Keywords—workflow, tests, validation, execution engine

I. INTRODUCTION

Nowadays workflows are central elements for improving the quality of human processes but also to enforce quality procedures for finance, medical or governance applications. Many tools exist now and form a whole ecosystem dedicated to workflow management, ranging from business process modeling tools to workflow engine. Companies now sell online solutions hosting workflows for their clients and integrating those workflows in such an ecosystem. Such hosting allows their clients to externalize the cost of deploying and maintaining such a complex environment. But it comes with a risk: as business processes managed by workflows are most of the time really critical for the client this client must have a strong confidence in his workflow solution provider.

To strengthen client’s confidence, many providers allow their users to run some tests before the workflow is deployed for production. However, the testing of a workflow engine execution remains a difficult problem. Indeed, testing is similar to the problem of proving software that is known as un-decidable [1]. Moreover, workflow testing is much more complex than usual testing due to the concurrency that is allowed in the workflow model. There are limitations also in doing real tests on workflow, as some time-based conditions may be hard to simulate and time consuming. Finally, testing can be done at many levels, ranging from high level integrated tests to conformance testing of the workflow engine itself [6].

Another problem is that it is usually up to the Workflow designer to model and implement tests: it is the case in popular workflow environments such as IBM Business Process Manager [8] or K2 [7]. This means that the designer has to be an expert in workflow testing to run properly its test campaign.

A workflow defines the specification of the business process and corresponds an automaton. Thus, theoretical works dealing with testing of automata have to be considered. [2] presents an overview of different methods to address the difficulty to test an automaton. They consider different classes of execution faults such as the timing or logical faults. Those types of fault must be tested and prevented first since the major objective of the workflow automata is to guaranty the correct sequencing of the different steps and events. In order to evaluate the sequencing, the test must generate a set of sequences that must be consistent with the sequencing of the automaton. It highlights the difficulty to be a testing expert in workflows.

A final problem is associated with black-box workflow testing. Such a testing requires consistent input data (i.e. possibly generated through a mockup environment) allowing the workflow transitions. Generating these input data is complex and costly.

Qualnet is a company providing and hosting a software ecosystem named Intraqual dedicated to workflows. The Intraqual environment provides a user interface easing the modeling of a workflow. A dedicated execution engine uses a web approach to control the workflow. Currently, end users need conformance indicators for the Intraqual engine since...
the workflows usually address procedures of safety, quality insurance or audit trails. Qualnet developed testing methods using standard tools such as a Selenium driver [3]. This testing approach is nice but limited since it does not benefit from the knowledge of the workflow model. Moreover, tests with Selenium evaluate the whole integrated environment including the web frontend. A more in-depth testing approach is to validate the conformance of the workflow engine regarding the user model. The key point of this conformance is the satisfaction of the sequencing with the user model. Evidences of the sequencing provide an interesting feedback. Moreover, these evidences enable to check the non-regression of the execution engine.

This paper proposes the following contributions:

- Definition of a workflow environment that supports the automatic testing reusing the workflow specifications so that the designer does not have to be an expert in workflow testing.
- Introduction of a workflow testing based on a mixed black box/white box approach to avoid the cost of real black box workflow testing.
- In-depth testing to verify that the workflow engine satisfies the requested sequencing. Thus, designers and end-users have evidences that the engine respects the sequencing.
- Preventing the design of complex mockup for the input data. Thus, the approach of testing is more efficient since the input data can be factorized as abstract transitions.
- An approach that generates and checks the test suite for the workflows while assuring a good quality. Indeed, the generated test suite completely covers the workflow.
- A real implementation and experiments on various types of workflows validate our approach.

This paper addresses these different topics. It presents the general approach and architecture. It presents the current implementation and the experiments. Finally, it concludes with future works.

II. GENERAL APPROACH

A. Workflow environment

Our work is dedicated to workflow environment such as the Qualnet Intraqual ecosystem. One can define 3 main roles implied in such an environment: 1) provider (e.g. a local Web service) that is responsible for hosting the workflows, securing data and execution, 2) Workflow designers setting up the workflows, and 3) end-users using the workflow instances. An overview of the roles in the environment and of the process of setting up and executing a workflow is given in figure 1. Elements in yellow are under the designer responsibility, elements in blue are under the providers’ responsibility and elements in red are under the end-users’ responsibility.

Designers produce specifications that are stored in a workflow database. The workflow engine extracts the model from this database and executes it given the input data provided by end-users. The workflow engine contains two parts: 1) an environment that provides and products all the necessary data for the application 2) an automata engine which executes the business process regarding the considered model.
It may even be a silent fault such as passing from a state to another one without any translation between them. To secure the automata engine, a more in-depth approach of workflow testing is mandatory.

We then decided to run tests at the level of the automata engine as depicted in Figure 2. The yellow part corresponds to a classical workflow environment such as the one from Intraqual. The rest of the figure describes our in-depth agnostic approach. First, a parser produces an abstract target automata. A generator computes the test sequences for the abstract automata. Thus, an agnostic solution is proposed supporting various formats for the workflow models. Second, the testing process continues with a list of transitions entering into the real engine. Thus, in-depth testing is supported since only the core parts of the engine are tested. The application environment (e.g. the user interface) can be tested solely by reusing standard tools such as Selenium. A clear separation is proposed between testing the core and testing the environment.

By doing so, we skipped possible but unlikely upper levels failures and the cost associated to integrated tests. Moreover, as we rely only on workflow specification and the abstract automata engine, this workflow testing is agnostic to the whole environment. This allows us to port our test suite to another environment seamlessly.

III. IMPLEMENTATION

A. Generation of test sequences

Our test campaign of the workflow engine relies in using a generated set of test sequences as a base for a test suite. A test sequence is an ordered sequence of successive transition. Test sequences are input of the workflow implementation. For each test sequence, the workflow implementation outputs an ordered sequence of states. We evaluate how the test as passed by comparing this sequence of states and expected results generated from workflow specification using an oracle described later.

Eventually the quality of the test suite depend on the significance and coverage properties of the set of tests sequences. A minimal property is that any state and transition is covered by a test at least once. This is however insufficient to ensure the correctness of the workflow execution. To ensure a good testing of the workflow, we decided to generate tests sequences so that tests will cover most common workflow executions.

The set of test sequences we use is composed of two subset. First subset is composed of any acyclic test sequence that starts from the initial state and terminates at any of the ending states. Second subset is composed of any test sequence starting from the initial state and stopping at the second occurrence of a state already contained in the test sequence. Figure 3 shows the set of test sequence corresponding to a given workflow specification.

![Figure 3 Paths considered for tests](image)

Path 1 in the figure is one of the execution path we want to test: Begin → S1 → S2 → S3 → S4 → End. Its corresponding test sequence is the ordered transitions sequence (S1, e0) → (S2, e1) → (S3, e2) → (S4, e3b) → (End, e4).

The algorithm to generate the test sequences is described by the following pseudo-code. It starts from the initial state and traverses any possible acyclic path in the workflow using a breadth-first search approach. For each possible successor of current state, if this successor state is not a final state or a state that is already traversed in this path, we add the next state to the current path. In our example, we have 6 paths that cover all the states and transitions of workflow.

Function 1:

```
// This function generates a set of test sequences
Function GenerateTestSuiteAutomate
    // A path is a list of couple (state, transition)
    Variable (List) path
    // Add the first state of automata
    Variable array(2) firstElement
    firstElement(0)= firstState
    firstElement(1)= firstTransition
    path.Add(firstElement)
    // This variable represent the actual path
    Variable (List of path) actualPath
    // At the beginning the actual path is the first transition
    actualPath.Add(path)
    // This variable is true when the process of building path is finished
    Variable (Boolean) finished
    Variable (List of path) cumulatedPath
    // This variable stocks the path at each state. The value of result(1) is the path at this moment and result(0) is a flag to tell us if the path can be evolved
    Variable array(2) result
    While Not finished
        result = buildPath(actualPath)
        cumulatedPath = result(1)
        // If this path can not be evolved
        If result(0)= False Then
```
Function 2:
// This function builds the path at each state cumulatively
Function BuildPath(actualPath)
Variable (List of path) cumulatedActualPaths
Varibale (Boolean) evoluted = False
For Each path In actualPath
    // The actual state is in the last element of the list which represents the actual path
    Variable actualState = (path.Last)(0)
    Variable nextStates
    // If the actual state is not the End so we get the next states of this actual state
    If Not actualState = End Then
        nextStates = GetNextStates(actualState)
    End If
    // If the actual State is not present in the path two times, it means that the path doesn’t return to one traversed state
    If Not CheckStateInPath(actualState, path) = 2 And nextStates IsNot Nothing Then
        For Each nextState in nextStates
            // If the state is not traversed, we will add it to the path. If the state is traversed so it’s already stocked in the path, we will add it this time as the terminal
            Variable (List of path) cumulatedPath
cumulatedPath.AddRange(path)
cumulatedPath.Add(nextState)
cumulatedActualPaths.Add(cumulatedPath)
        Next
        evoluted= True
    Else
        // The last state of path is the terminal
        cumulatedActualPaths.Add(path)
    End If
Next
// This variable stock the result of process of building path at each state
Variable array(2) result
result(0)= evoluted
result(1)= cumulatedActualPaths
Return result
End Function

Function 3:
Function TestWorkflow
// Get test suite
Variable testSuite = GenerateTestSuiteAutomate()
// Get the implementation of automata
Variable automate= GetImplAutomate
// Flag of test case
Variable satisfied= True
Variable List failedTestCases
Variable List satisfiedTestCases
For Each testCase in testSuite
    If Not CheckTestCase (testCase, automate)
    Then
        Satisfied = False
        failedTestCases.Add(testCase)
    Else
        // The last state of path is the terminal
        cumulatedActualPaths.Add(path)
    End If
Next
// This variable stock the result of process of building path at each state
Variable array(2) result
result(0)= evoluted
result(1)= cumulatedActualPaths
Return result
End Function

In figure 4 we illustrate the execution of our algorithm with workflow depicted in figure 3 as input data. As it is breadth-first search algorithm, we represented the different paths states by level (ranging from 1 to 5). For sake of clarity we cut the tree in three parts, from left to right. Paths in red are considered as complete because they reached a final state; paths in green are considered complete because their next state is already included in the path.

B. Evaluation of test sequences

We build an oracle to check tests results using workflow specification provided by users as pictured in the following figure. We parse the workflow database to obtain a target automata. We then use our generator to produce our set of test sequences and for each of these test sequences its corresponding expected list of states. Finally we run our test suite and compare each test results with the expected list of states. This enables us to test conformance of the workflow engine. The whole workflow testing and its different component is depicted in figure 2. Components not dedicated to workflow testing (workflow database, automaton implementation and workflow engine) are yellow, components dedicated to workflow testing are in red.

We give below in pseudo-code algorithms that check any test case as well as the whole test suite.
satisfiedTestCases.Add(testCase)
End If
Next
Variable array result(3)
result(0) = satisfied
result(1) = failedTestCases
result(2) = satisfiedTestCases
Return result
End Function

Function 4:
// The tableTransitions here is an automata
Function CheckTestCase(testCase, tableTransition)
Variable satisfied = True
Variable preState
Variable preTransition
// The first element of array testCase(i)(0) is the actual state // and the second testCase(i)(1) is the traversed transition
For i = 0 to testCase.count - 1
If i = 0 Then
    preState= Begin
Else
    preState= testCase(i-1)(0)
End If
// The tranversed transition
preTransition = testCase(i)(1)
// If there is a failed transition so the test case is not satisfied
If
preState doesn’t exist in table transition
or
preTransition doesn’t exist in table transition
or
we can not go from preState to actual State by the preTransition
Then
satisfied= False
Return satisfied
End If
Next
End Function

IV. EXPERIMENTATION

A. Test sequences
We realized experiments with 5 automata pictured in figure 5. Automata 1 to 4 depicts the most usual workflows: simple, with a cycle, with a short circuit, with a switch while the 5th combines the features of all previous workflow. For each of those automata, we give at the bottom of the figure the corresponding specifications. In the specification, each line contains the information concerning one transition.

For example:
Automata 1- first line: 0, S1, e0 means that the state Begin (0) is changed to state S1 by the transition e0.

Automata 4 – fifth line: S3, S5, e3c means that the state S3 is changed to state S5 by the transition e3c.

Figure 5 Tested workflow
For any of the five previous workflow, we generated a test suite and check their results using our oracle. Figure 6 depicts tests suites. It shows the number of test cases for each tested workflow as well as each verified path. Each test case is the execution of one of the generated test sequences.

Figure 6 Test suite

B. Workflow engine testing
We then perform conformance testing using these tests suites. We run tests campaign to evaluate conformance of 5 automata implementations given in figure 7 that may or may not behave correctly. Each automata implementation is stored in a text file. Each line describes a transition: first term is the initial state,
second one the final state and the third is the event associated to the transition. For the five automata given upper, we added some faults in their implementation. The oracle is based on automata initial specifications. This enables us to test if faulty workflow engines are detected. Correct automata implementations are 1, 2 and incorrect ones are 3, 4, 5.

![Figure 7 Automata implementation](image)

We used the automata implementation given in figure 7 and run a test campaign with a test suite generated for any of them. Eventually we used a function to export the result to a file .txt to visualize it; the results are given in figure 8. For example we find that 2 test cases doesn’t work correctly for workflow 5 since they are supposed to include the transition (S3, S4). These failures correspond to the transition (S3, S4, xxx) that is incorrect in the implementation of automata 5. Our tests have detected this error. Considering workflow 4 the transitions (S2, S3, xxx) and (S4,-2, yyy) are incorrect; our test suite has detected 2 incorrect paths as well.

V. RELATED WORK

Recently, a big attention has been paid to workflow testing in the industry, because of the massive adoption of business process management tools. In practice this has become a major issue; however, just a few research paper recently focused on this point.

From the theoretical point of view, workflow testing is well studied since the middle of the 90’s as workflow is considered as a finite state automata (see [2] for an overview). The closest theoretic problem from our is known in the literature as conformance testing, machine verification or fault detection depending on the field of study of authors. In this problem, one wants to check the behavior of a finite state automata engine considered as a black-box. Different methods then exists to generate input data and its associated family of sequences such that testing this family will cover the different possible faults of the finite state automata. However, as the method considers the automata as a black box, generating such a sequence may lead to sequences too long to be tractable in practice [9]. Moreover, these kind of tests is unrealistic in our case, because of actions and possible delay induced by the real workflow.

![Figure 8 Tests results](image)

As far as we know, no practical workflow solutions relies on such algorithms. Authors of [12] reviewed in a survey attempts to apply various methods for solving this problem, such as Petri nets [14] or randomized selection of test cases [13]. In our case, as we avoid the permutation state problem by not actually running the state automata but only generated automata corresponding to a finite sequence of states we address a slightly different problem. This enabled us to develop a simpler and more practical solution than those described in theoretic papers.

On the practical side, solutions relies mainly on unit test framework such as BPEL Unit Test Framework [10]. Authors of [11] for example describe their approach using BPEL Unit Test and BDD as an automation framework for their tests. While this
represent a practical implementation of continuous integration scheme for BP testing, it is still to the end user to design his tests. Authors of [15] relies on UML modeling for testing BP when faced to dynamic composition of web services. Those solution all share the same drawback, as it faces the user with a complex software stack he has to understand in order to design his tests. Our method eases the end user experience as test campaign is generated from his model.

CONCLUSION

In this paper we presented a new practical method to test workflow engine that mixes black box and white box approach. In contrast with the previous theoretic works, our test method is integrated into a real workflow environment while easing the test for the workflow designer. Our approach gives evidences of the enforcement of the workflow for both the designer and the end-users. It minimizes the number of tests while achieving a good coverage of the workflows.

The workload of the designer is very low thanks to the automatic generation of the tests from the workflow specifications. The good coverage of the workflow strengthens the confidence in the workflow engine.

As future work we aim at considering testing of actions attached to transition to further guarantee the conformance of the engine, as well as how to define tests suite that ensure good coverage even in case of concurrency.

ACKNOWLEDGMENT

This work is partially supported by Centre-Loire Valley region (France).

REFERENCES