Testing workflow-enabled applications

Uroš Goljat, Marjan Heričko
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17, SI - 2000 Maribor, Slovenia
{uros.goljat, marjan.hericko}@uni-mb.si

Abstract. *Business processes are a part of every company and are becoming increasingly more complex. Due to this, the need for workflow-based applications has never been more apparent. The Windows Workflow Foundation (WF) provides a simple and consistent way to model and implement complex problems that arise in workflow-enabled applications. Because of the increased use of such applications, we also need supporting tools and frameworks to enable the quality assurance of workflows that present the core of workflow-enabled applications.*

In this paper, we will present approaches on how to enable the testing of workflows built using WF inside .NET applications.

Keywords. Windows Workflow Foundation, Testing, Unit-testing

1 Introduction

In last few years, companies and governmental institutions have been paying increasing attention to automation and the management of business processes with the use of IT technologies. To accomplish these tasks, we use workflow management systems, like the Windows Workflow Foundation (WF), which provide new techniques for modeling business processes as workflows. Typical usage scenarios are document-centric applications, ordering systems, hiring/payroll applications, etc.

Workflow-based applications are designed (and existing applications are reengineered) so that the main business logic resides inside workflows. Software testing is, at the moment, the most important and most often-used quality assurance technique. Consequently, we need to ensure quality assurance for these workflows by testing them with appropriate techniques (for example: unit-testing) and testing tools.

This paper is organized as follows: Section 2 provides a brief overview of WF. Section 3 discusses unit-testing workflow–based applications. Section 4 presents tools for testing WF-based applications. In Section 5, we compare unit-testing workflow-based applications and the use of WorkflowInspector tool for testing workflow-based applications. Section 6 concludes the paper.

2 Applications based on WF

It is important to note that WF is a framework for developing workflow-based applications, and not a full-featured product that can be immediately used by end users. WF provides a foundation on which to build workflow-based systems. All the pieces required for building workflows and manipulating the workflow infrastructure are provided. The rest is up to developers. For example, WF does not include a full-featured tool for monitoring workflow execution but it exposes the information needed for developing such useful and often required tools.

WF provides a programming model, an engine and tools that then allow developers to build and deploy workflow-based applications on the .NET Framework. It was first released in November 2006 as part of .NET 3.0 [1].

2.1 Basic/Key concepts

To build workflow-enabled applications with WF one needs to be familiar with some key concepts that are fundamental for the development of WF applications. These key concepts are :
- **Workflow Designer**: a graphical tool to visually design and model workflows. Workflows are designed inside Visual Studio but the tool can be integrated into any Windows application.
- **Activities**: these are the basic building blocks of every workflow built with WF. They are standalone

pieces of functionality that can be reused across multiple workflows. The work an activity implements can be very simple (e.g. a send e-mail activity) or quite complex (e.g. a composite activity that executes nested activities in a transaction).
- **Workflow**: a group of activities that represent the implemented business process or its parts. The type of activities contained in a workflow defines its type.
- **Base activity library (BAL)**: a set of activities that range from the most basic workflow control to more complex activities, such as invoking WCF services. Activities are building blocks for defining workflows. BAL includes nearly 30 different activities .
- **Runtime engine**: a WF component that is responsible for executing workflows. It also manages the addition, removal and execution of runtime services that are vital for workflows to properly execute.
- **Host process and Run-time services**: The host process is needed to host and manage the run-time engine that executes workflows. The host process is also responsible for providing run-time services that are responsible for providing services such as transactions and persistence to the run-time engine. A host process can be any type of .NET application such as console applications, web applications, web services or Windows SharePoint Services.

The relationship between the described concepts is shown in Fig. 1.

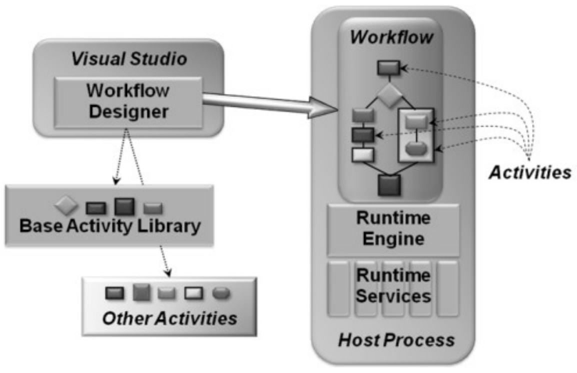

Figure 1. WF fundamental components

2.2 Types of workflows

The Windows Workflow Foundation provides three different ways to model workflows: the finite state machine (state machine workflows), sequences of single activities (sequential workflows), and data-driven workflows (special types of sequential workflows).

The three types of workflows are discussed briefly in the next sections.

2.2.1 State machine workflows

State machine workflows (Fig. 2) are defined as a set of states and application events. The transition between these states is triggered by application events, which occur when a workflow is in a specific state. Each state machine workflow has exactly one initial state and one or more terminal states in which the workflow completes. In most cases, workflows modeled as state machines are non-deterministic. This type of workflow is ideal for business processes where the workflow itself includes a lot of user interactions.

2.2.2 Sequential workflows

Sequential workflows (Fig. 3) are defined as a series of steps that are predefined and executed in a prescribed order. Hence, the path through the workflow is deterministic. The flow of control within workflow is defined through well-known constructs such as *if-else* branching and *while* loops. They are ideal for modeling business processes. This type of workflow is used mainly when little or no user interaction is needed.

2.2.3 Data-Driven workflows

Data-Driven workflows are usually presented as a special type of sequential workflow that contains constrained activity groups and policies. In data-driven workflows, activities are executed in an order that is determined by conditional expressions. Conditional expressions are presented by rules that check external data to determine the path of a workflow instance.

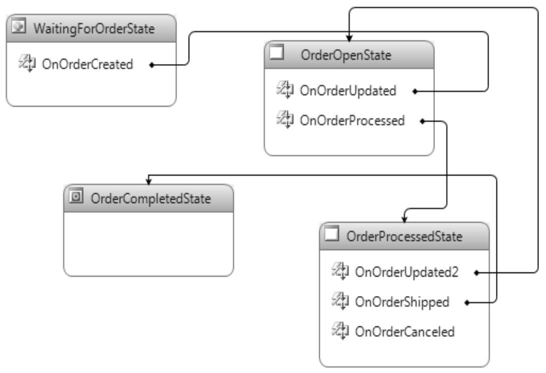

Figure 2. State machine workflow

3 Unit-testing workflow based applications

In this section, we will describe the approaches and issues that arise during unit-testing workflow-based applications. As a unit-testing framework, NUnit can be used. NUnit is a general unit-testing framework for the .NET framework that was ported from Java . Alternatively, Visual Studio could also be used for unit-testing workflow built using WF.

Because workflows implemented using WF are basically .NET classes, we can use the unit-testing approach to test them. When unit-testing such applications built using WF, there are several components that need to be tested. These components are: custom activities, workflow rules and workflows as a whole.

Common issues arise when developers test these components, such as managing workflow runtime, providing appropriate run-time services, running workflow instances and waiting for them to complete, etc. For managing runtime, for example, there is a good practice for using the test setup and teardown methods to initialize the runtime and gracefully shut it down when the test method executes .

Because unit testing of WF workflows adds additional complexities to unit-testing code, custom-unit testing libraries have been developed. These libraries hide the complexities mentioned above and can be used in conjunction with any unit-testing framework.

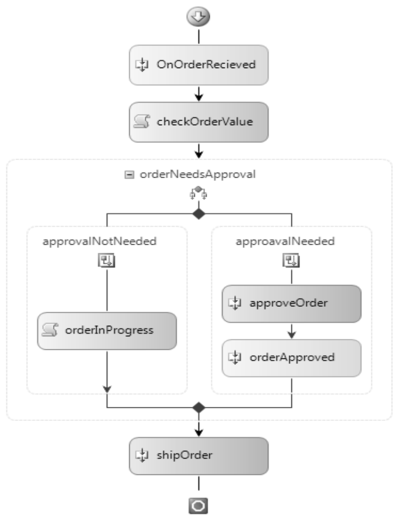

Figure 3. Sequential workflow

3.1 Unit testing custom workflow activities

Although BAL includes a wide range of activities, there are always situations when developers need to develop their own custom activities. An example of a custom activity would be an activity for sending e-mails based on its properties.

Because custom activities are classes in the code, they can be unit tested. When developing unit tests for custom activities, developers must do this in isolation by providing inputs and testing expected outputs . This means that they must be tested as individual components rather than testing inside a whole workflow. What is advantageous is the architecture of classes involved in the development of workflows. Every workflow in WF is also an activity. This means that the WF runtime can execute any activity as a workflow, even if this activity is as simple as an activity that writes a line of text to a console window. So when unit testing custom activities, developers need to provide a workflow instance with a single tested activity, which the workflow runtime executes.

When writing unit tests, developers also need to test the behavior of a tested component with exceptional cases. The NUnit testing framework, in our case, has the ability to declare expected exceptions to simplify test development ("failure as success"). But when unit-testing custom activities, developers must use another approach. The reason behind this is that the WF runtime catches raised exceptions and passes them to the host through the use of events (when an exception is thrown by any of the activities inside the workflow, a *WorkflowTerminated* exception is raised). So the approach for taking advantage of the expected exceptions mechanism is to handle the appropriate event trigged by WF runtime and then re-throw the exception. When re-throwing the exception there can be three problems: either the exception type is lost, the call stack is lost, or both. So when re-throwing the exception developers need to be careful that this important data is not lost.

In many cases when testing activities, developers need to check the values of different properties on tested activities. This presents another problem with unit testing custom workflow activities. As such, direct access to an instance of a tested activity is not directly available. Again, the reason for this is the WF runtime that is responsible for creating an instance of an activity class and hides a created instance inside a workflow instance. Hence, a workaround is needed to get access to an instance of a tested activity.

3.2 Unit testing workflows

After all custom activities involved in a workflow have been tested separately, workflows as a whole can be tested.

Developers face even greater challenges when unit-testing entire workflows. This is due to the fact that workflows often model long-running processes that interact with many different services and applications and also often people. A given workflow can, for example, call several web services using the Windows Communication Foundation (WCF) and display the results. Then, it can require the user to confirm the acquired data, in which user confirmation is required to continue the process. This simple scenario has two potential unit-testing challenges: calling external services using WCF and user interaction.

Those challenges are discussed in the next two sections.

3.2.1 External services calls using WCF

When unit-testing workflows involving web service calls, interactions need to be handled by using mocking frameworks (such as), which inject mock objects to replace real service calls.

BAL includes an activity for calling web services using WCF (*Send* activity), which uses the

ChannelManagerService class to resolve web service endpoints. A test class can easily add named endpoints (through the *ChannelManagerService* class) that match those expected in the workflow, so that they match with a local service implementation .

3.2.2 User interaction

When workflows involve user interactions, we need a way to mimic user's choices to continue the execution of a workflow. Two separate problems need to be addressed: how do we know that the workflow is waiting for user input and how do we collect data from the user and then input it into workflow from a running test?

To determine when the workflow is waiting for user input, we need to keep track of the workflow's execution path so that we know at every moment which activity is currently executing inside the workflow. The WF tracking infrastructure and services provide the facility to monitor and query the execution of a workflow instance. This allows us to query a running workflow if it is currently executing a *HandleExternalEvent* activity, which is used to model user interactions.

When we are sure that the workflow is waiting for user input, we need to display a user input form through which the user can enter appropriate information. This form has to be built dynamically based on the data that is needed by the user interaction. Which data is needed to continue a workflow execution can be obtained from the *HandleExternalEvent* activity's properties.

To enable automation, we have to consider saving the user input to a persistence medium (relational database, XML file, etc.) and automatically enter the saved data into an appropriate user input form when running automated tests.

4 Tools for testing workflow based applications

Without appropriate tools, testing a workflow-based application is a tedious and time-consuming task.

Although workflow-based applications are more and more popular, there is still a lack of tools that enable the testing of workflow-based applications built using WF. In the next section, we will describe the WorkflowInspector as a tool that enables the testing of workflow-enabled applications in the .NET Framework.

4.1 WorkflowInspector

WorkflowInspector (WI) is a custom tool developed by the University of Innsbruck, the research group Quality Engineering and world-direct eBusiness .

WI helps developers conduct early testing of workflows to enable the verification of modeled workflows by end users. The main goals of the WI tool are :

- Enable automated workflow testing, without the use of development environments (such as Visual Studio), debuggers, etc.
- Enhance WF in the field of testing workflow-enabled applications by adding development specific features (for example, a path coverage test).

The tool is developed as a standalone Win32 application that allows the testing of workflows without a development environment. Workflow definition is loaded from an assembly. This means that workflows are not altered for testing purposes, which was a fundamental requirement when the tool was being developed.

4.1.1 Implemented test types

WI implements three coverage test types that provide metrics about workflow execution. The three implemented coverage test types are :

- **Activity Coverage** test type: This reveals which activities have been executed during a test run. Full activity coverage is achieved when every activity in the workflow is executed at least once.
- **Branch Coverage** test type: This includes the activity coverage test and is used to reveal which edges of the workflow's control flow graph were executed at least once. Full branch coverage means that all edges have been executed at least once.
- **Path Coverage** test type: This is used to enable the coverage of all possible paths through the workflow. This type of test is introduced because of workflows that contain loops. In such cases, Activity and Branch coverage tests are not sufficient.

4.1.2 Main features

To give an overview of WI capabilities, some of the major components should be described. The described features are summarized from .

- Testing of State Machine Workflows: WI is designed to support testing state machine workflows. But sequential workflows can also be tested because state machine almost always include nested sequential workflows.

- Workflow and Data visualization: WI provides a compelling graphical user interface that inherits the graphical notion of the workflow designer in Visual Studio. On the other hand, WI also includes the possibility of displaying tracked data (this data is acquired through tracking the workflow execution during a test run) in a user-friendly way, and as detailed as possible.

- **Generic way to input test data:** through this feature, WI removes the need of building basic user interfaces and input forms for each workflow separately. When WI is running a test and external data is required to continue workflow execution, WI dynamically creates an input form. This form contains

all the input fields that are needed to resume workflow execution.
- **Graph Visualization:** WI provides an additional view of the workflow. This view displays the workflow as a directed graph of activities. It provides a better view of the entire workflow and can also be used for presentation purposes.
- **Pre- and Post- condition Validation:** with the use of the Pre- and Post- condition validation, WI enables a simple verification of test runs. Basic boolean constraints for workflow parameters can be defined.

4.1.3 Testing workflows with WI

The testing procedure used by WI is quite simple and easy to use. 1) The workflow that is defined in Visual Studio is compiled into an assembly. The compiled assembly contains the definition of the workflow. 2) Inside the workflow inspector, a new test suite is added where an assembly with a workflow definition must be provided and pre- and post- conditions can be defined. 3) Perform coverage tests (Activity, Branch or Path coverage test). Once the workflow is in a test run, its behavior cannot be modified. Executing different paths inside the workflow are achieved by providing different input values when the workflow requires external input. 4) After the test run, test results are displayed through the WI's graphical user interface.

5 Comparing NUnit and WorkflowInspector

In this section, we will present the differences, advantages, and disadvantages between unit-testing workflows and testing workflows using the WI tool.

Table 1 shows the main differences between the presented approaches for testing workflow-based applications. It shows that both types of workflows can be tested with either approach. But real-world workflows always include either human interaction or external partner interaction through web service calls. From Table 1 we can see that only WI supports the testing of such workflows.

When modeling complex, real-world processes, there is always the need for custom activities. When we create custom activities, they need to be tested in isolation with providing inputs and testing outputs with positive and exceptional cases. Only NUnit supports this scenario.

Automated testing is also an important feature, especially when continuous integration is used. Automated testing can be run on the integration server whenever changes in a workflow definition are submitted to the code repository. Both the NUnit and WI tool supports automated testing.

Testing workflows as a whole is supported by both approaches. But when testing workflows, its graphical representation and the path through which the workflow has been executed (whose activities were executed and were not) are also important. The graphical workflow representation and workflow execution path are only supported by WI tool.

Last but not least is test coverage. With unit testing, we can only do test coverage based at the code. With the use of the WI tool, test coverage can be measured on the level of activities or workflow as a whole.

When it comes to real-world scenarios where business processes become more and more complex to model. Hence, both approaches need to be combined to deliver quality workflows.

Table 1. Comparison of test capabilities

	NUnit	**Workflow Inspector**
Testing sequential workflows	☑	☑
Testing state machine workflows	☑	☑
Testing-human based workflows	☒	☑
Support for web service call interception	☒	☑
Automated testing	☑	☑
Testing of a single activity in isolation	☑	☒
Testing of a workflow as whole	☑	☑
Visual representation of tested workflow	☒	☑
Visual representation of test execution path	☒	☑
Test coverage at the code level	☑	☒
Test coverage at the activity level	☒	☑
Test coverage at the workflow level	☒	☑

6 Conclusion

In this paper, WF has been presented as a technology that enables the modeling of business processes as workflows.

Because workflows present the core of workflow-enabled applications, they have to be properly tested to ensure their quality assurance. For this reason, techniques and challenges for unit-testing workflows have been presented. The WorkflowInspector tool has also been described, which enables the testing of workflow-based applications and extends the WF framework in the field of testing workflow-enabled applications in .NET.

To achieve the best quality for workflows, both techniques need to be used: Unit-testing for custom activities and other business logic that resides in other classes outside of the workflow; and the WI tool to perform visual workflow testing and coverage tests.

References

[1] Microsoft: **Windows Workflow Foundation**, available at http://msdn.microsoft.com/en-us/netframework/aa663328.aspx, Accessed: 31st March 2009

[2] Chappel D: **Introducing Windows Workflow Foundation**, available at http://download.microsoft.com/download/f/3/2/f32ff4c6-174f-4a2f-a58f-ed28437d7b1e/Introducing_WF_in_NET_Framework_35_v1.doc, Accessed: 3rd April 2009

[3] Kitta T: **Proffesional Windows Workflow Foundation**, Wiley Publishing Inc., Indianapolis, USA, 2007

[4] Lechner A, Breu R: **Workflow Inspector - A Test Tool for Microsoft Workflow Foundation**, Proceedings of the 2008 international Conference on Software Testing, Verification, and Validation, 9th–11th April, Lillehammer, Norway, 2008, pp. 498-501.

[5] Breu R, Lechner A, Willburger M, Katt B: **Workflow Testing**, Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation Third International Symposium, ISoLA 2008, 13th–15th October, Porto Sani, Greece, pp. 709-723

[6] NUnit.org: **NUnit**, http://www.nunit.org/, Accessed: 28th April 2009

[7] Milner M: **Unit Testing Workflows And Activities**, available at: http://msdn.microsoft.com/en-us/magazine/dd179724.aspx, Accessed: 28th April 2009

[8] Kennedy M C: **Significant Advances in Unit Testing Windows Workflow**, available at: http://www.michaelckennedy.net/blog/2009/01/18/SignificantAdvancesInUnitTestingWindowsWorkflow.aspx, Accessed: 30th April 2009

[9] Ayende.com, **Rhino Mocks**, available at: http://ayende.com/projects/rhino-mocks.aspx, Accessed: 3rd May 2009

[10] Gristwood D: **Windows Workflow Foundation: Tracking Services Introduction**, available at: http://msdn.microsoft.com/en-us/library/bb264459(vs.80).aspx, Accessed: 5th May 2009