Testing Event-Driven Business Processes

Josef Schiefer, Gerd Saurer
Senactive IT Dienstleistungs GmbH, Vienna, Austria
Email: {josef.schiefer, gerd.saurer}@senactive.com

Alexander Schatten
Institute for Software Technology and Interactive Systems, Vienna, Austria
Email: aschatt@ifs.tuwien.ac.at

***Abstract*—Today's business climate requires to constantly evolve IT strategies for responding to new opportunities or threats. While the fundamentals of IT — reliability, availability, security and manageability — are still crucial, rapid results are mandatory for business success. These business challenges can be solved by acting with agility – striking the proper balance between the introduction of leading-edge technology and the pragmatic application of IT. In this paper, we introduce a testing framework for business solutions dealing with complex and dynamic IT environments. Our framework enables a test-driven development and adaptation of business processes in order to implement flexible and reliable business solutions. We compare our testing framework with model-driven development approaches and show how we applied it to an event-driven process management platform called SARI (Sense And Respond Infrastructure).**

***Index Terms*—Testing, Test-Driven Development, Business Process Management, Business Process Engineering, Event Management.**

I. INTRODUCTION

Business Process Management (BPM) systems are software solutions that support the management of the lifecycle of a business process. This includes the *definition*, *execution* and *monitoring* of business processes. For the execution of business processes, many organizations are increasingly using process engines supporting standard-based process models (such as WS-BPEL [18]) to improve the efficiency of their processes and keep the testing independent from specific middleware.

A major challenge of current BPM solutions is to continuously adapt a business process over time according to responses from the business environment, and keep them robust and operational for critical business processes. Making mistakes when designing and changing e-business processes can cause delays and costs when deploying process improvements. Any defect or outage in a business process solution is likely to affect large portions of the enterprise, potentially causing loss of revenue and data.

Many business process solutions are deployed with little or no testing. Testing, if any, is usually done manually and sporadically. One of the reasons why process solutions are often not tested thoroughly is the fact that testing of heterogeneous message-driven systems is a challenging undertaking. These solutions are complex, distributed, heterogeneous and asynchronous in nature and there are very few tools available to aid in these testing efforts.

In this paper, we discuss the challenges of functionally testing business processes and propose a testing framework to tackle the inherent complexities. With our testing framework, we address the following issues:

- Validation of the quality and correctness of a process model
- Automated testing of business process execution within a distributed environment
- Testing side effects from modifications of a process model
- Detecting and diagnosing performance problems of process deployments
- Facilities for investigating discovered malfunctioned processing steps

Traditional business process management solutions use simulation for analyzing the quality of process models. Process simulation is a top-down approach for process reengineering programs and provides valuable insight in variations of process models. However, process simulation is only as good as the underlying assumptions for the simulation parameters. Typical process simulation tools lack in providing tools for simulating the events of a real-world business environment in order to verify the correct *runtime behavior* of a deployed business process. The verification of correct process runtime behavior is often done by debugging and tracing facilities which usually is a manual and time consuming process. Through the standardization of process description (e.g., WS-BPEL), but lacking a standard for testing, companies can choose from various IT platforms to deploy and execute a business process. Nevertheless, a switch to a new platform can introduce new process runtime behavior caused by integration issues.

In this paper, we propose a testing framework which follows a bottom-up approach for gaining insight into existing business processes or business process solutions in development. Starting from an existing process model, we use our framework to validate it with tests bottom up.

For the testing, we not only consider the description of the process model, but also runtime aspects such as:

- Verifying the outcomes of processing steps by actually invoking services and components.
- Measuring the throughput of processing engines.
- Testing the robustness of the process management system during failover scenarios.
- Testing changes for configuration settings such as parameters for processing engine, business rules, or resource assignments.
- Creating simulation tools and mock-up strategies to replace complex business environments like SAP during test scenarios.

The remainder of this paper is organized as follows. In Section II, we discuss test-driven development of business solutions and compare it with model-driven development. In Section III, we give an overview of our testing framework. In Sections IV and V, we show how we applied the testing framework to an event-driven process management platform and demonstrate a test scenario for monitoring stock levels. In Section VI, we outline how the proposed testing concepts can be also used for WS-BPEL processes. Finally, with Section VII, we conclude our paper and give an outlook for future work.

II. Test-Driven Development For Business Solutions

Looking back the way software has been developed – particularly considering the last years – dramatic changes can be observed. Starting in 2001 with the "Agile Manifesto" [6], agile software developing processes became commonly accepted development strategies, hence having significant impact on the way software projects are planned, managed and implemented nowadays. Agile methodologies like XP [2], Scrum [3] or Crystal [4], which formed the foundation for the Manifesto, strongly emphasize the role of software tests [17] by integrating them early into the software development instead of just "accepting" it as part of software production at the end of iteration or the whole process. Parallel with the new software development processes, new software architectures also appeared on the scene, driven by the widespread usage of Internet related technologies. Particularly noteworthy are so-called Service Oriented Architectures (SOA) [9]. SOA has gained popularity as a new software engineering paradigm and arose from the necessity of creating components providing clearly defined components that later on can be assembled into complex (usually distributed) applications. Splitting up complex applications into smaller components and services aims not only at increased reusability, but should also leverage composing and refactoring (business) processes. Evidently, this approach fits well for fast changing environments, where business systems have to continuously adapt to new business needs.

Service-oriented systems usually don't try to substitute systems in the first place that are already installed in companies; more often they try to provide an infrastructure that enables to utilize older applications as services and allow to update legacy systems later on. Taking all this into account, it appears to be obvious, that such systems often result in an operational environment with complex dynamical behavior which is hard to predict and evaluate [13]. Hence, (automatic) testing of an infrastructure is a difficult issue. An additional challenge for such environments is to bring the requirements defined by a user or several stake holders into a model that can be understood from technical architects, developers and of course testers (which is the main focus of this work).

Using models to design systems help us understand complex problems and their potential solutions through abstraction. Selic proposes in [12] the following five characteristics of engineering models for building complex systems (see Table I).

With our testing framework, we want to address these characteristics in different ways as traditional model-driven development approaches. Figure 1 shows a

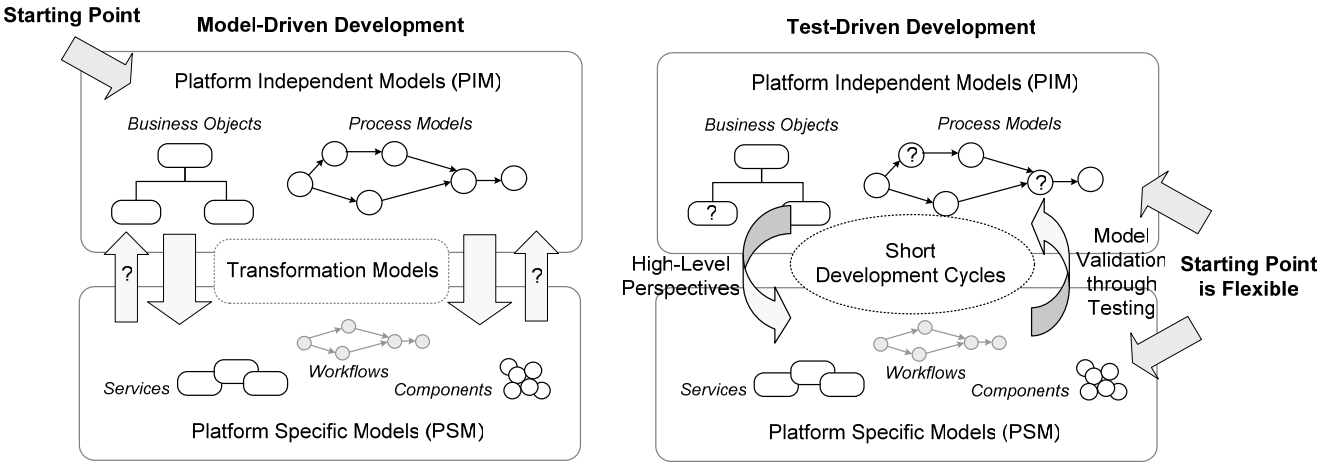

Figure 1. Model-Driven Development vs. Test-Driven Development

comparison of a model-driven and test-driven development of business solutions.

TABLE I.
CHARACTERISTICS OF ENGINEERING MODELS

Abstraction: A model is always a reduced rendering of the system that it represents. By removing or hiding details that are irrelevant for a given viewpoint, it lets us understand the essence more easily.
Understandability: Is a direct function of the expressiveness of the modeling form used (expressiveness is the capacity to convey a complex idea with little direct information). A good model provides a shortcut by reducing the amount of intellectual effort required for understanding.
Accuracy: A model must provide a true-to-life representation of the modeled system's features of interest.
Predictive: The model should correctly predict a system's interesting but non-obvious properties, either through experimentation (such as by executing a model on a computer) or through some type of formal analysis, which depends greatly on the model's accuracy and modeling form.
Inexpensive: It must be significantly cheaper to construct and analyze than the modeled system.

The focal point of model-driven development is to describe business solutions ideally with models without any consideration of platform specific details (= Platform Independent Models) and automating the transformation of these models into executable software components. Model-driven development holds the promise of being the first true generational leap in software development since the introduction of the compiler.

One of the biggest challenges for model-driven development is the transformation between the platform-independent and platform-specific models. Roundtrips between these models are very difficult and sometimes it is even not feasible to maintain the traceability between model artifacts. In particular, the traceability links from platform specific models back to platform independent models are often missing which makes it difficult to create a true-to-life representation of the modeled system. Model-driven development approaches often only predict how platform independent models will map into an execution environment.

With our testing framework, we also make this process reversible and further consider how an existing execution environment conforms to a platform independent model. Test-driven development does not assume that all platform specific models (which include for instance also code) are generated in a top-down fashion. A software engineer can decide whether to start with a model which provides a high-level perspective of the system, or with platform specific artifacts such as software components.

A software engineer can use software tests to validate the behavior of an existing system with platform dependent and independent models. Although it would be desirable to validate all details of a platform independent model, this is often too time consuming and not cost-efficient. Software tests facilitate users to test out various aspects of the system behavior that are relevant to them. As we will show in the next sections, they can do this on various abstraction levels.

III. TESTING FRAMEWORK OVERVIEW

In this section, we introduce a testing framework for business processes which is based on so-called *process test specifications*. Process test specifications are a skeleton for defining functional tests which are executed within a managed test environment. Any artifacts needed by a test besides the testing framework itself are considered a part of the process test specification. A process test specification is used by testers to describe and implement tests for typical scenarios for a business process. They help to mount any required system functionality or data into the test environment, such as data from a database, configuration files or external services. Sometimes, it is time-consuming to include existing services into a test scenario (e.g., placing an order with a SAP system for testing purposes) and dummy placeholders are sufficient to perform the tests. Such placeholders are also called "mocks". The process test specification contains a list of tests which initialize the test environment, invoke the tested unit (such as a service) and assert the results. The last step is the most crucial one since the assertions check the current behavior of a business process.

Figure 2 shows the elements of process test specifications and how they are linked with a business process model. After defining a process test specification, our testing framework is able to extend a process description in order to make it testable. In the following, the elements of a process test specification are discussed in detail.

Simulation Models. A simulation model is used to generate real-world input data for the business process. It allows defining the data of a sequence of events for typical and exceptional business scenarios. A key requirement of the simulation model is that it generates consistent data sets for entire scenarios. For instance, if the processing of an order for a customer should be simulated, certain information of the events which occur during the order processing — such as order information (e.g., order ID, order items) or information about the customer (e.g., customer ID, customer preferences) — have to be maintained.

Mocks for Systems and Services. Instead of invoking the real source systems, the tested process calls a mock object that merely makes sure that the correct functions were called, with the expected parameters, in the correct order without actually contacting a real source system. For example, to test an access to an SAP system, it may be burdensome to install, configure, and seed a local copy of SAP, run test tests, then tear the local SAP installation down again. Mock objects provide a way out of this dilemma. A mock object conforms to the interface of the real object, but has just enough code to fool the tested process and track its behavior. For example, a SAP mock object might record the system request while always returning the same hardwired result. As long as the process being tested behaves as expected, it won't notice the difference, and the process test can check that the proper SAP request was emitted.

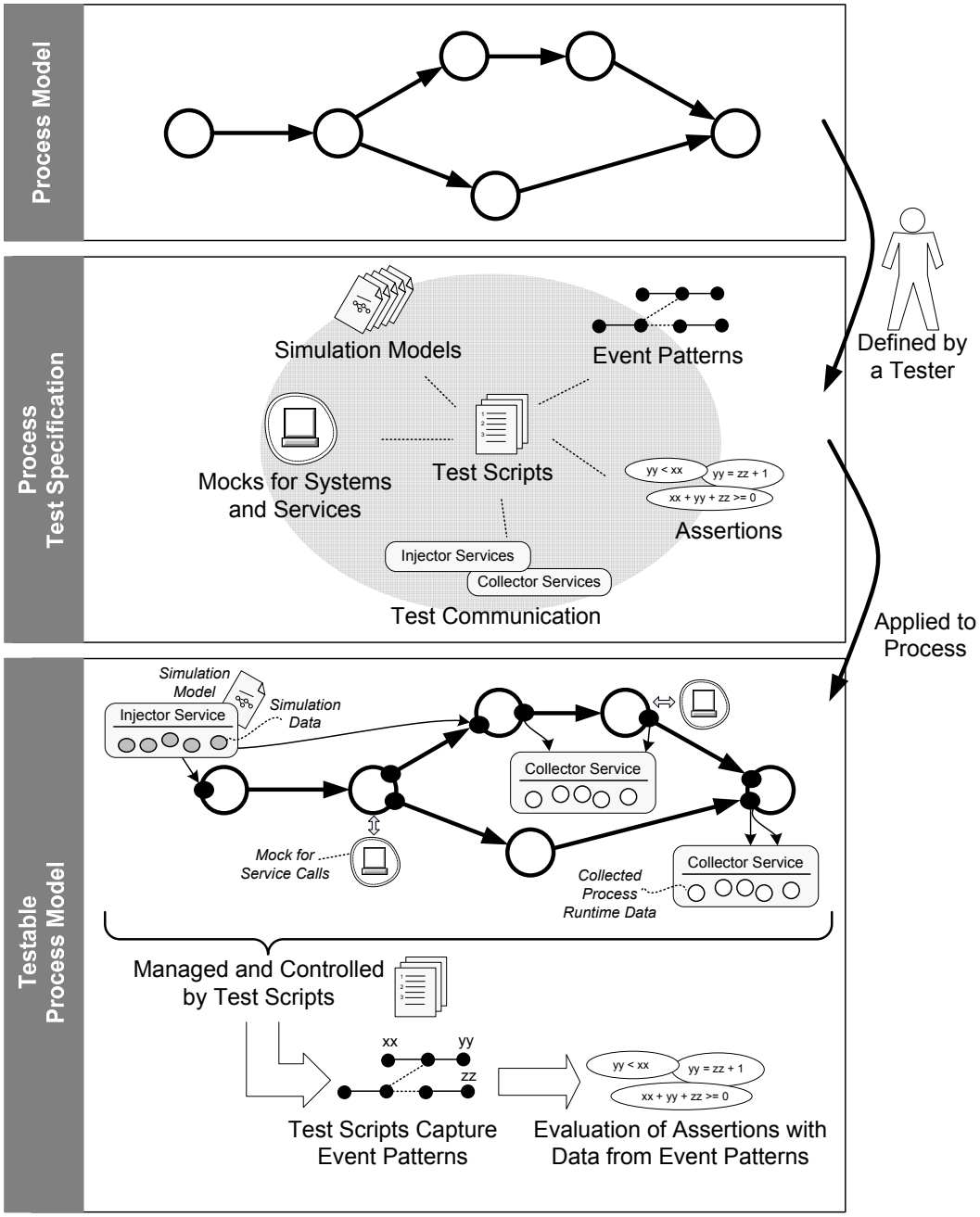

Figure 2. Testing Business Processes with a Process Test Specification

Test Communication. Supports test scripts in order to communicate with the process runtime environment. It is used to inject and collect process runtime data as well as to control the process engine. The test communication infrastructure has to be linked and integrated with parts of the business processes and the process engine in order to enable a data exchange. In our framework, we provide two components: 1) *injector services*, which take input data from a simulator and send it to the appropriate system interfaces which initiate some action within a business process, and 2) *collector services*, which gather runtime data from a business process which occur during the course of executing.

Event Patterns. Associate the captured process runtime data in order to identify event sequences which are relevant for the testing. Event patterns describe selected events for an execution path of a business process. Test scripts can use the events of the discovered event patterns in order to evaluate assertions.

Assertions. An assertion is an expression that enables testing the assumptions made about a business process. For example, if a business process is modeled that automatically closes a pending approval request after 3 days, one might assert that every approval request is not older than 3 days. Each assertion contains a boolean expression that is believed will be true when the assertion executes. If it is not true, an error will be thrown. By verifying that the boolean expression is indeed true, the assertion confirms the assumptions about the behavior of the business processing, increasing the confidence that

the business process solution is working correctly and free of errors. Experience has shown that writing assertions while programming is one of the quickest and most effective ways to detect and correct bugs [19]. We take advantage of this technique and adapted it for using it for business process testing. As an added benefit, assertions serve to document the inner behavior of business processes, enhancing maintainability. For our testing framework, assertions are based on captured event patterns. An assertion can evaluate to true 1) if an event pattern occurred, or 2) a boolean expression which uses data of captured events or event patterns evaluates to true.

Test Scripts. Test scripts are the glue for the previously discussed artifacts. They use simulation models for generating a set of input data which is sent to the injector services. Test scripts invoke parts or the entire business process with simulated data, and test whether the process is executing correctly. Test scripts are snippets of code defined by a tester which usually perform one of the following tasks:

- Setup of test environment: This includes setting up the process engine, deploying the target process to be tested. The target process is extended with mocks and proxies for the test communication which enables the test script to control the process execution.
- Initialization of the test environment: During this step, the test script generates the input data for the test run. A simulation model is loaded by a simulator which sends a consistent set of input data to the injection services. Furthermore, the test script registers event patterns which should be discovered during the execution of the test scenario.
- Execution of the test: After correctly initializing the test environment, the test script performs the actual test with the process. In this step, the test script remotely controls the process engine and performs some steps, which are relevant for the test (e.g., triggering a set of activities of the process).
- Capturing test results: During the execution of a test, collector services gather runtime information from the process engine. The testing environment uses the data from the collector services in order find out whether excepted event sequences occurred.
- Checking assertions: The test script uses data collected from collector services or discovered event sequences for evaluating assertions. The test script records the results of the assertions.

IV. A Testing Framework For Sense And Respond Processes

The testing framework was tailored to an agile business process management platform called SARI (Sense And Respond Infrastructure). In the following, we want to give an overview of the SARI system in order to better understand how the testing framework was applied.

A. Brief Introduction of SARI

SARI's main objective is to help organizations to control and monitor their business processes and IT systems in order to respond to business situations or exceptions with minimal latency. SARI is able to continuously receive, process and augment events from various source systems, and transforms these events in near real-time into performance indicators and intelligent business actions. SARI is a distributed, scalable platform, which allows modeling and executing various forms of sense and respond processes (see Figure 3).

The data processing is controlled by "Sense & Response loops" which can be divided into five stages.

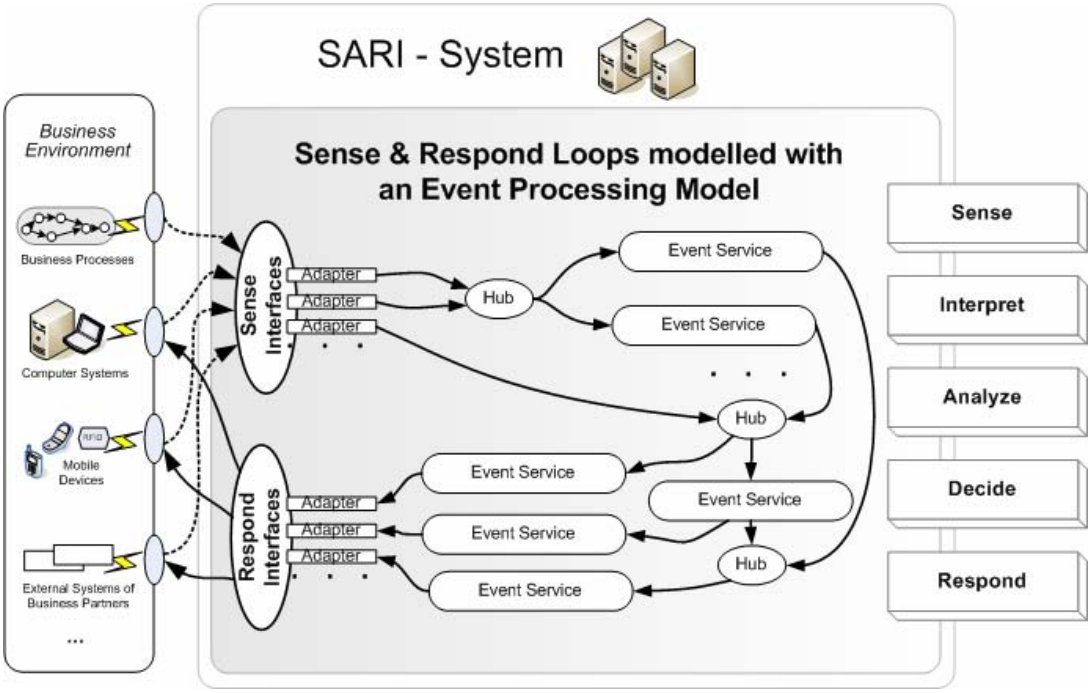

Figure 3. SARI Sense & Respond Loops [11]

During the "Sense" stage events are continuously captured and transmitted to the SARI system where they are initially unified before the actual data processing starts. In the "Interpret" stage, the captured events are transformed into meaningful business information, such as key performance indicators, business situations and exceptions. The next stage is the "Analysis" of the generated business information in order to determine root causes for business situations and exceptions. The analysis has the goal to find the best possibilities to improve the current situation of the enterprise. The analysis also allows predicting the business performance and risks for making changes to the business environment. In the following "Decide" stage, the best option for improving the current business situations is selected and determines the most appropriate action for a response to the business environment. Finally, in the "Respond" stage, the decision made has to be implemented in the business environment by communicating the decision as a command or suggestion (e.g., by e-mail), or by directly adapting and reconfiguring business processes and IT systems. During the processing in the five stages, business information is continuously generated and decisions are made to which a response follows. The response has an effect on the source systems (from which SARI originally might have received events) and consequently also on the performance and the success of the organization.

Using the SOA approach, we model in SARI Sense & Respond loops with a pool of services and establish the infrastructure that enables a robust communication and interaction between them. SARI uses event processing maps (EPMs) for modeling Sense & Respond loops. Similar to a construction kit, the EPM offers various building blocks which can be used to construct a Sense & Respond loop. Dependent on the requirements and the business problem, these building blocks can be flexibly conjoined or disconnected. Links between the building blocks represent a flow of events from one service to the next. Typical building blocks are event services for various data processing tasks in each stage. For more details please refer to [11].

B. Overview of the Testing Framework

In this section, we introduce the testing environment and framework we built for the SARI system. The idea of the sense and response paradigm is to capture business events, and build processing steps that sense situations relevant to the business. If such a "situation" takes place, the system responds intelligently.

As shown in the previous sections, SARI uses a modular approach for constructing Sense & Respond loops. This is achieved by using an EPM, which allows flexibly conjoining and disconnecting elements (e.g., event services) of the model, dependent on the requirements and the business problem. Links between the elements represent a flow of events from one service to the next.

TABLE II.
ELEMENTS OF AN EVENT PROCESSING MAP (EPM)

Event Service: A component which performs a certain type of event processing. Event service can consume events and can generate new events as a result of the event processing. The EPM manages the exchange of events between event services (see Figure 4).
Adapter: Component for receiving events from or responding to an external system.
Hub: A component for merging and splitting event streams.
Worker Node: A computation node which hosts event services and adapters.
Coordinator Node: A computation node which coordinates the worker nodes in order to ensure a consistent and reliable event processing.

Figure 4 shows an EPM and the deployment environment of SARI in the context of our testing framework. Table II summarizes some elements of the SARI system that we want to use later in our discussion. We used the testing framework NUnit [8] as a foundation for the code & unit tests, component tests, integration

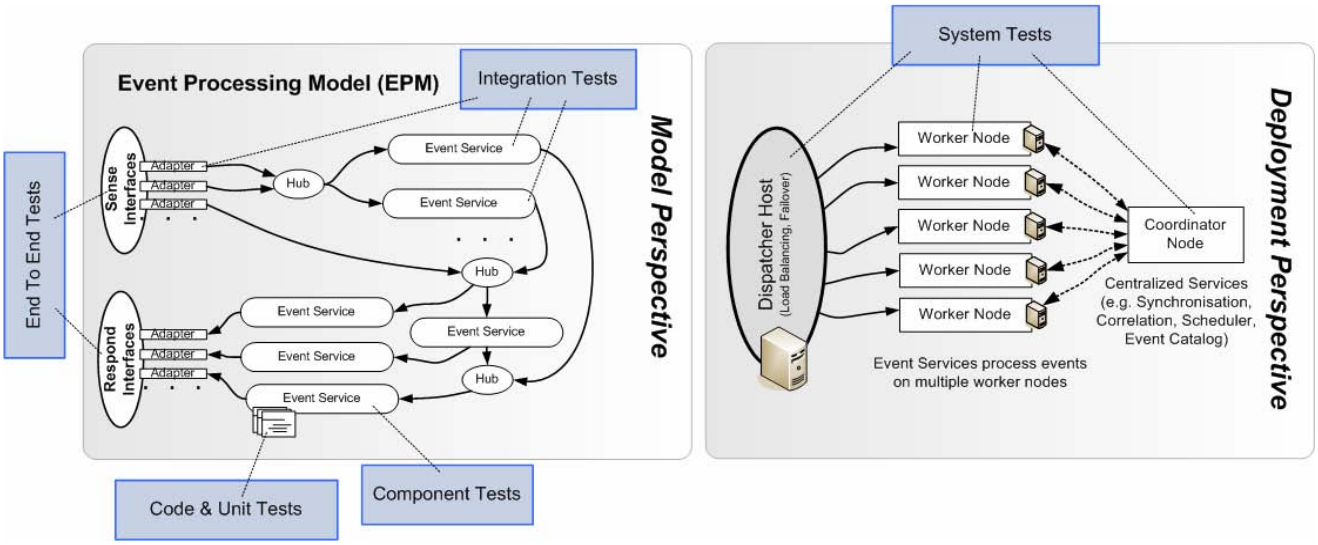

Figure 4. SARI Testing Environment

tests and system test. We extended the NUnit framework with new test fixtures and mocks (as will be explained later) in order to support complex tests. For the end-to-end tests, a simulator was developed. Figure 5 shows a summary of the different abstraction levels for tests.

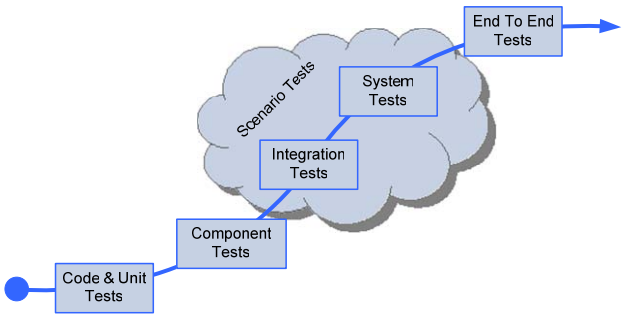

Figure 5. SARI Framework Tests

Code & Unit Tests. In SARI, data is collected and received through adapters and continuously processed by event services. Each adapter and event service is a component that can consist of multiple classes. For instance, when implementing an event service for mapping event data to a database, a set of classes for describing the data mappings, a class for parsing data mappings from an XML file, and an event service performing the data mappings needs to be implemented. Code and unit tests are fully supported by NUnit; hence an extension of this testing framework was not required at this level of testing.

Component Tests. Interfaces of a component and services are validated using component tests. In SARI, interfaces are defined by event types describing the structure of specific event data. The event types are used for input and output ports of event services and adapters with EPMs. E.g., when testing an event service, a tester sends a set of events to the input ports of the event services, and tests the result of the event processing by investigating the results at the output ports of the event services. Figure 6 illustrates this process. Within a test fixture, a tester uses an event injector to send events to the input ports of the event service. During the event processing, the event service might use various kinds of system services (such as event correlation [10], transaction management, and queuing) which are also provided by mock objects. These mock objects are light-weight implementations of system services which results in shorter execution times for the component tests (the initialization of component tests typically occurs in less than a second). After the event processing, an event collector mock object is used to capture the events published on the output ports of the event service. The tester can use the collected events for assertions within the test fixture.

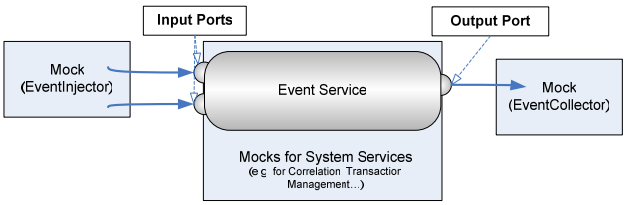

Figure 6. Testing Event Services

Integration Tests. An event service can depend on the results of other event services or adapters. The aim of integration tests is to consider a larger set of components and services within EPMs, and test whether they work correctly in concert. Figure 7 shows a set of dependent event services. Event injector mocks are used to send events to the event services from a test fixture. Similar as in component tests, event collector mocks collect events from output ports of an event service. The collected events can be used for assertions within the test fixture. The difference to component tests is as follows:

- Testing of the outcome of an event processing problem including multiple event services.
- Multiple event injectors and event collectors are allowed.
- Additional map elements such as hubs (shown in Figure 7).
- Event services can use “external” parameters such as map parameters.
- A comprehensive set of heavy-weight mocks for system services is available.

With integration tests, users can test parts of an EPM or an EPM in its entirety. Integration tests include a larger set of mocks for system services. Some of the mocks are heavy-weight because they are built on top of SARI system services. Examples for such heavy-weight services are the correlation service, the event emitter for

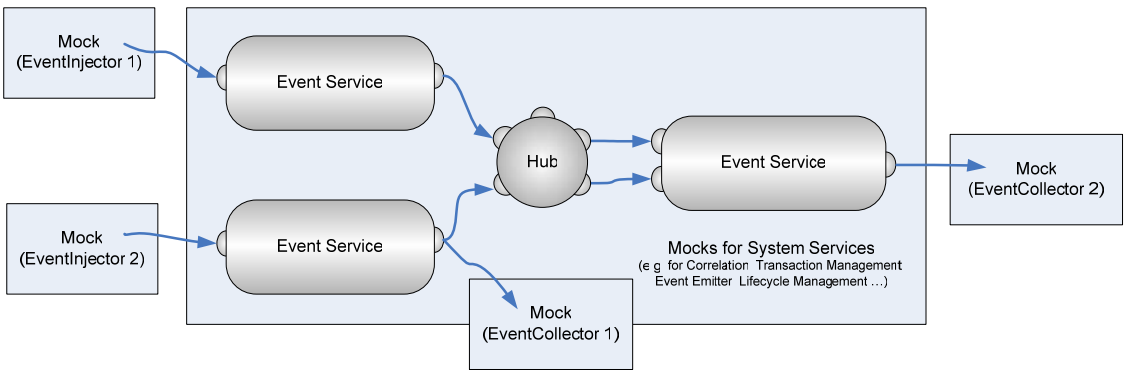

Figure 7. Integration Test for Event Services

forwarding events within an EPM, and the lifecycle management for event services and adapters. The execution of an integration test takes some seconds in our system due to the longer instantiation times of complex mocks for the system services.

System Tests. The SARI system is a distributed platform for event processing (see Figure 4 – deployment perspective). It includes various types of nodes that perform dedicated processing tasks. E.g., the dispatcher host performs load balancing, the worker nodes host the event services and process events, and the coordinator node provides centralized services such as a timer service and synchronization. System tests allow to test the processing tasks within this complex distributed environment. Test fixtures for system tests can assert processing results for EPMs as well as system states (e.g., lifecycle state of an event service) in cases of exceptional situations such as failures of a working unit. The repetition of such failure situations is extremely difficult in real-world scenarios, as the system state is difficult to preserve. Using a test fixture that prepares the system state of worker nodes and other working units elegantly solves this problem, since the tester has full control and access to the working units (e.g., worker nodes) and can initialize and verify the system state.

End-To-End Tests. End-to-end tests are used to test complete sense and respond business scenarios in order to verify their correct behavior from an end user perspective. In order to facilitate the testing of end-to-end business scenarios, we developed a simulator for simulating event sequences of real-world scenarios. End users can either generate events by initiating state changes in the real business environment (which results in events that are sent to the SARI system), or by using a simulator to generate the events for a test. A detailed discussion of end-to-end tests lies outside the scope of this paper and is part of our future work.

V. Example – Monitoring Stock Levels

Figure 8 shows an example of an EPM which monitors stock levels and triggers reorders when supplies are running out of stock. The EPM includes two adapters for receiving external events: 1) when an order has been received, and 2) when supplies have been replenished. Both events have an effect on the stock levels. Additionally, it contains three event services which process the incoming external events and determine the best time for reorders. In this example, it is assumed that users can use business rules for making reorder decisions. Table III explains the components that are used within the EPM. Please note that the replenishment process was simplified for demonstration purposes.

TABLE III.
Components of Event Processing Map (EPM)

Order Management System: A web service adapter which receives an event when a new order has been placed by customers.
Supply Chain Management (SCM) System: A SAP adapter which receives an event when supply materials have been replenished.
Stock Analysis: A service which analyzes the current stock levels and calculates indicators from historical stock developments (e.g., deviations from seasonal order levels).
Reorder Rule Processing: This service decides based on rules, whether a reorder should be triggered or not.
Reorder Service: This service processes reorder requests by sending them to an SAP system.

In the following, it will be shown how the testing framework can be applied to the illustrated sample process. For a tester, it should be easily possible to run tests in every development stage of a designed or modified process. With the information provided by the process test specification, the testing framework is able to extend the original EPM (designed for the production environment) in a way that the process becomes testable. In the example, adapters and services are used, which communicate with source systems such as SAP. For running a testing scenario, production systems like SAP are usually not available for a testing environment. Therefore, these systems have to be "mocked". Figure 9 shows the replenishment process with injection services and collector services that replace any communication interface to a source system. Using this EPM, the testing framework is able to send arbitrary events from a testing script to the injection services, which trigger the processing of the event services for the stock analysis and reorder rule processing. The results of the event processing are captured by a collector service.

In the following, it is explained how all elements of the process test specification are defined and how they are used by the testing framework.

Simulation Models. The SARI simulation model allows to model sequences of events which can be sent by the test script to the injector services. The simulation model supports event generators, which use templates that are dynamically filled during runtime with data by

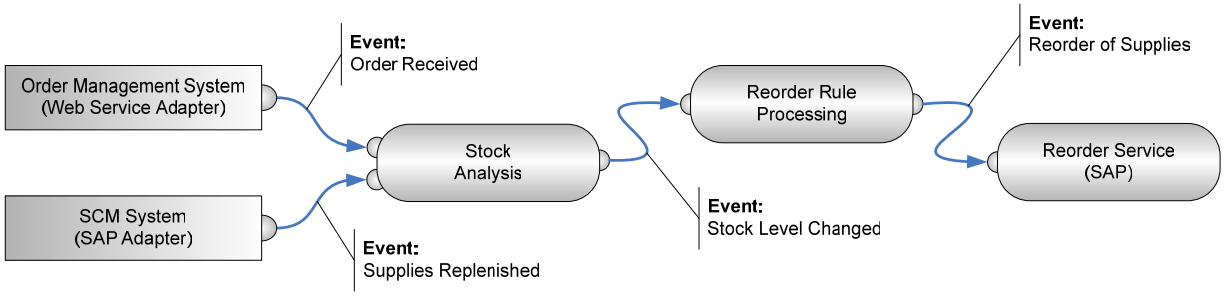

Figure 8. EPM for Monitoring and Managing Stock Levels

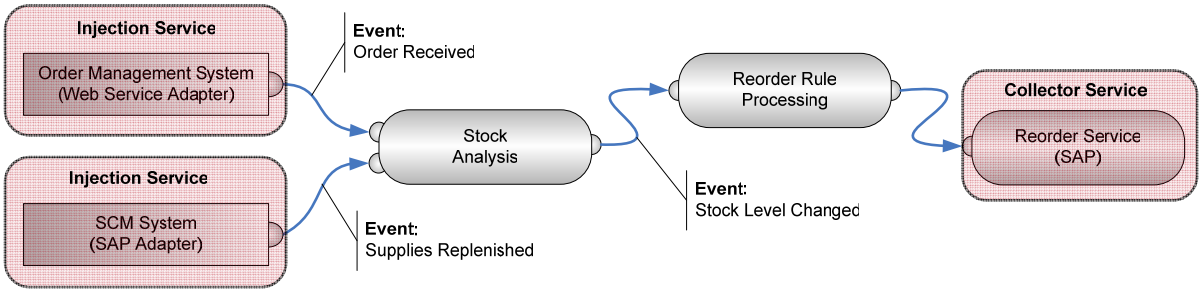

Figure 9. Injection and Collector Services for Mocking External Systems

the simulator. Figure 10 shows on the left side a typical template for an event generator for order events. The template is an XML fragment with value generators that are replaced during the simulation with real values. On the right side, a typical definition for an event simulation is shown, which describes a simulation scenario that generates 80 % orders of product A and 20 % orders of product B. A detailed description of the simulation model lies outside the scope of this paper. Nevertheless, this example should show the principle for declaratively modeling a simulation run for a test scenario.

Mocks for Systems and Services. As shown in Figure 9, our testing framework uses injection and collector services to mock services that interact with external systems. These mocks are crucial to decouple a process from the underlying target systems. The SARI testing framework allows to extend or replace any services of a process with mock objects which can be controlled from a test script. As mentioned before, injection services and collector services are part of the test communication. Nevertheless, they can be also mock objects replacing external systems (such as SAP). An injection service gives a tester the possibility to send data directly to the process engine in order to trigger the execution of process tasks. This input data can be either created manually within the test script or automatically generated by the simulator. On the other hand, the output data of the process execution is captured by collector services. Collector services are able to sense intermediate results during process execution, and signal the test script when certain event patterns occurred.

Test Communication. SARI uses a communication bus for interacting with the process runtime environment. Figure 11 shows the interaction between the process engine and the test environment in which test scripts are executed. The communication bus is a key component for effectively testing process scenarios, since it allows to remotely control the process engine as well as the injection services and collector services. It is deeply embedded into the testing framework, and it ensures that testers have convenient and easy access to any runtime data during process execution.

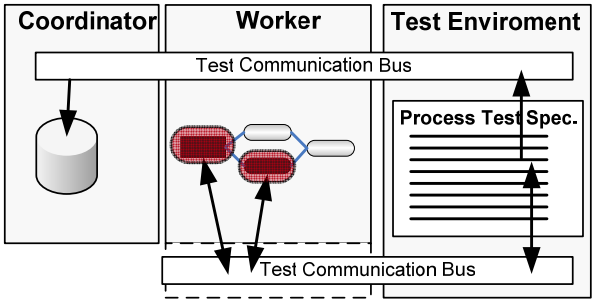

Figure 11. Test Communication Bus of SARI

Event Patterns. Part of the SARI system is an ECA service, which is able to evaluate conditions for events ,and sends a notification depending on the evaluation result. The acronym ECA stands for Event-Condition-Action, and is the fundamental principle of discovering event patterns: After receiving an *event*, the data of this event is evaluated by checking on whether some specific *conditions* are fulfilled. If this is the case, an *action* will be taken. This action step is implemented to send notifications about discovered event patterns to the testing environment. A test script has access to the results of ECA sets, and uses them for further assertions. Figure

Event Generator:

```
<EventGenerator identifier="ProductA">
  <Template>
   <OrderEvent>
    <OrderID>
     <ValueGenerator identifier="OrderID"/>
    </OrderID>
    <CustomerID>
     <ValueGenerator identifier="CustomerID"/>
    </CustomerID>
    <OrderItems>
     <ValueGenerator identifier="OrderItems"/>
    </OrderItems>
   </OrderEvent>
  </Template>
</EventGenerator>
```

Event Simulation:

```
<EventSimulation identifier="MixedOrders">
  <Switch>
    <Case probability="0.8">
      <EventGenerator identifier="ProductA"/>
    </Case>
    <Case probability="0.2">
      <EventGenerator identifier="ProductB"/>
    </Case>
  </Switch>
</EventSimulation>
```

Figure 10. Simulation Model Artifacts

12 shows an example for an ECA set which filters out all events for stock level changes of product A, and sends them to the test environment.

ECA Set:

```
<ECASet identifier="OnlyProductAStockLevelChanges">
   <EventType type="StockLevelChanged"/>
   <Condition>
      //ProductName[.='ProductA']
   </Condition>
   <Action>
      <SendTestNotification/>
   </Action>
</ECASet>
```

Figure 12. ECA Sets for Discovering Event Patterns

Assertions. The SARI testing framework provides a set of assert functions that can be called within test scripts. Assert functions evaluate the data of events captured from collector services, and are called within test scripts. The following example of an assert function checks whether the attribute 'StockLevel' of all events for stock level changes is greater than 100.

```
AssertTrue(stockLevelChangedEvents,
           "//StockLevel > 100");
```

The events for the assert functions can be the result of discovered event patterns or they are directly retrieved from collector services. SARI further includes assert functions which can be applied to a single event or to a set of events. Additionally, there are assert functions for evaluating failure states of the process engine (e.g., AssertException(…), or AssertTransactionRollback(…)), which are useful for testing exceptional process behavior. The testing framework tracks the assertion results and reports them to a tester.

Test Scripts. For our testing framework, we decided to extend an existing unit testing framework for implementing and executing test scripts. SARI uses the test fixtures of the testing framework NUnit as a basis. We enhanced the test fixtures with the previously discussed capabilities, such as an event simulation, a communication infrastructure for interacting with a process engine, convenient access and control of mock objects, discovery and evaluation of event patterns as well as checking complex assertions and reporting test

Test Script (Setup):

```
void OnSetup() {
  EPM map = LoadMap(„ReplenishmentProcess.map");
  MockEventAdapter(map, „Order Management System");
  MockEventAdapter(map, „SCM System");
  MockEventService(map, „Reorder Service");
  TestCommunication.Deploy(map);
}
```

Figure 13. Initializing the Test Environment

Test Script (Scenario):

```
void ReorderProductATest() {
  // Register event pattern
  EventPattern pattern =
          LoadEventPattern(„ProductA.pat");
  TestCommunication.RegisterEventPattern(pattern);

  // Load and run simulation
  SimulationModel simModel =
          LoadSimulationModel(„Orders.sim");
  Event[] simEvents = RunSimulation(simModel);

  // Send simulation events to injection service
  TestCommunication.InjectEvents(simEvents,
          „Order Management System");

  // Wait for process execution
  TestCommunication.WaitForCollectorService(10,
          „Reorder Service");

  // Retrieve event pattern result
  Event[] result =
          TestCommunication.GetEventPatternResult(
          „OnlyProductAStockLevelChanges");

  // Check assertions
  AssertTrue(result.Length == 8);
  AssertTrue(result, "//StockLevel>100");
  ...
}
```

Figure 14. Test Scenario

results. By extending an existing unit testing, testers are able to define test suites which test the behavior of processes on various abstraction levels. The following two snippets of pseudo code show how the previously introduced artifacts are used within a test script.

The first snippet (see Figure 13) illustrates the initialization tasks for setting up the test environment. The snippet loads an EPM from the file system, replaces some components of the EPM with mocks and finally deploys the map. The snippet for the setup of the test environment is always called before a testing scenario is executed.

The second snippet (see Figure 14) shows a testing scenario, which tests the EPM whether the reordering of a certain product was correctly executed by the process engine. The following steps are performed in the test scenario:

1. An event pattern description is loaded from the file system and registered in the test environment.
2. A simulation model is loaded and used for generating events.
3. The simulated events are forwarded to the injection service which is a mock for the adapter of the order management system.
4. The test script waits until 10 events have been received by the collector service.
5. The events for the registered event pattern are retrieved from the testing environment.
6. The retrieved events are used for assertions.

As can be seen from our example, the test script is a central controller for the test scenario and uses various

elements of a process test specification. The testing framework automatically tracks the outcome of assertions and provides a tester with a summary of the test results. The illustrated tests are fully automated and can be easily included in test suites, which are performed on a daily basis. At the moment, testers have to use programming languages such as C# for writing tests. In the future, we want to allow testers to use higher level languages for writing tests.

VI. SARI TESTING VS. WS-BPEL TESTING

The concepts of our proposed testing framework are generic and can also be applied to other business process management platforms. One major challenge for building a testing framework for WS-BPEL processes is to find a flexible way for intercepting and managing requests to and from the process engine. The web services of a WS-BPEL process use the web service description language (WSDL) for defining an interface, which describes how to access a web service and what operations it will perform. A testing framework can use these interface descriptions for generating mocks that allow to inject or collect data transmitted to and from the process engine. Web services use the SOAP protocol for communication and exchange of messages which have to conform to an XML schema description. For simulating requests to a process engine, the testing framework has to able to generate input data that is valid for the XML schemas. Therefore, the testing framework has to extract any type definitions for web service requests from the process description or WSDL files. For implementing a test communication channel between the testing framework and the process engine, we can propose two strategies. On one hand, the WS-BPEL process description can be extended with additional activities for injecting and collecting process data, and on the other hand, the application programming interfaces (API) of the process engine can be used for this purpose. For the later case, process engines often support observer plugins, which allow extending a process engine with capabilities for testing. Table IV shows a summary of how the artifacts of our testing framework can be mapped to business process management systems based on WS-BPEL.

VII. CONCLUSION AND FUTURE WORK

With new paradigms for software development and software architectures companies are able to respond faster to changes in a business environment. In this paper, we introduced a testing framework that supports these paradigms by being able to test the correct behavior of running business solutions. We compared our approach with model-driven development, and showed how software tests can be used to make model assertions for business processes. We introduced process test specifications, which allow users to define simulation settings, test components, assertions and test scripts in order to make a business process testable. We applied the testing framework to the SARI system and outlined how the proposed concepts can be also used for the testing of WS-BPEL processes.

TABLE IV.
SARI TESTING VS. WS-BPEL TESTING

Testing Framework Artifact	SARI System	BPM System with WS-BPEL
Simulation Model	Simulation model describes sequences of events which are used as input for the SARI system. The structure for events is defined in an XML definition.	Simulation model describes a sequence of web service requests to/from source and target systems. The input data are messages for a web service, which are defined by an XML schema
Test Communication	Test communication is established by replacing event adapters with injection services and event services with collector services that are merged into the event processing map.	Test communication is established by adding WS-BPEL activities which allow receiving (→ injector service) simulation data and query captured process runtime data (→ collector service).
Mocks for Systems and Services	Replacing event adapters and event services with mock components.	Replacing existing WS-BPEL activities with mock activities is easy, since the interfaces for the invoked web service are available.
Event Patterns	The SARI system includes ECA (Event-Condition-Action) system service, which is used to notify business users about useful patterns in business processes. This service is reused to capture event patterns for testing purposes.	Since WS-BPEL processes do not include any mechanisms to evaluate complex situations with a business process, it is common to include a business rule management (BRM) system, which performs this task.
Assertions	Assert functions from an existing unit testing framework have been extended. A user can define assertions for event patterns or intermediary results of the event processing.	An existing (unit) testing framework can be extended for making advanced assertions within test scripts. Assert functions should be able to check WS-BPEL elements, such as changes to values of variables or the results of compensation actions.
Test Scripts	Implemented by extending an existing unit testing framework with a simulator, mocking services, a test communication infrastructure, an event pattern discovery service, and additional types of asserts.	Many BPM systems automatically generate WS-BPEL client proxies. These proxies can be further extended in order to facilitate the development of test scripts.

The work presented in this paper is part of a larger, long-term research effort aiming to making testing an integral part of the management of business processes. Part of this work is the development of a test bed for testing, simulating and benchmarking business process solutions. One of our key ideas to improve our current testing framework is to use a standardized language for the definition of process test specifications.

REFERENCES

[1] E. Albek, E. Bax, G. Billock, K. Chandy, and I. Swett, "An event processing language (epl) for building sense and respond applications," in Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 2005.

[2] K. Beck and D. Andres, Extreme Programming Explained, 2nd ed., Addison-Wesley Professional, November 2004.

[3] M. Beedle and K. Schwaber, Agile Software Development with Scrum, Prentice Hall, October 2001.

[4] A. Cockburn, Crystal Clear, Addison-Wesley, October 2004.

[5] S. Haeckel, Adaptive Enterprise: Creating and Leading Sense-And-Respond Organizations. Harvard Business School Press, 1999.

[6] Manifesto for Agile Software Development, http://www.agilemanifesto.org

[7] L. Meade, A. Presley, and J. Rogers, "Tools for engineering the agile enterprise," in Proceedings of the International Conference on Engineering and Technology Management (IEMC 96), IEEE, 1996.

[8] P. Hamill, Unit Test Frameworks, O'Reilly Media, Cambridge, 2004.

[9] M. P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In Proceedings of the Fourth International Conference on Web Information Systems Engineering, pages 3–12, December 2003.

[10] J. Schiefer, C. McGregor, Correlating Events for Monitoring Business Processes, in Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS), Porto, 2004.

[11] J. Schiefer, A. Seufert, Management and Controlling of Time-Sensitive Business Processes with Sense & Respond, International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA), Vienna, 2005.

[12] B. Selic, The Pragmatics of Model-Driven Development, IEEE Software 20 (5) (2003) 19-25.

[13] C. Stevenson, A. Pols, An Agile Approach to a Legacy System, 5th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004), 123-129, Garmisch-Partenkirchen, Germany, 2004.

[14] S. H. Kan, J. Parrish, D. Manlove, In-process metrics for software testing, IBM System Journal, Volume 40, Pages 220 - 241, 2001.

[15] W. T. Tsai, X. Bai, R. Paul, W. Shao, V. Agarwal, End-To-End Integration Testing Design, 25th Annual International Computer Software and Applications Conference, 2001.

[16] X. Bai, W.T. Tsai, R. Paul, T. Shen, B. Li, Distributed End-to-End Testing Management, 5th IEEE International Enterprise Distributed Object Computing Conference, 2001.

[17] D. C. Kung, P. Hsia, and J. Gao, Testing Object-Oriented Software, IEEE Computer Society Press, Los Alamitos, CA, 1999.

[18] Business process execution language for web services specification, http://www-128.ibm.com/developerworks /library/specification/wsbpel/, 2005.

[19] P. Hamill, Unit Test Frameworks, O'Reilly Media, Cambridge, 2004.

Dr. Josef Schiefer received his Ph.D in Information Systems from the University of Vienna and was a researcher in the Department of Software Technology at the Vienna University of Technology. He joined IBM's Thomas J. Watson Research Center where he continued his research in the areas of business process management, business process intelligence, and data warehousing. Today, Dr. Schiefer is Technical Director at Senactive Inc., a company offering software for real-time sense and respond solutions.

Gerd Saurer is student at the Vienna University of Technology and works as a software engineer at Senactive Inc., a company offering software for real-time sense and respond solutions. The primary focus of his studies includes software testing, agile software development methods and complex event processing. As a software engineer, he has an extensive background developing enterprise solutions in Java and C#.

Dr. Alexander Schatten received his Ph.D from Vienna University of Technology and works as a researcher at the Institute for Software Technology and Interactive Systems. He is leading the workgroup of Open Source Software at the Austrian Computer Society, and is researching in the field of event-based systems, software engineering and groupware. He also publishes on a regular basis for a broader audience in German and Swiss IT magazines on recent IT (middleware) trends.