Serdica J. Computing **5** (2011), 367–384

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

A FRAMEWORK FOR DESIGN-TIME TESTING OF SERVICE-BASED APPLICATIONS AT BPEL LEVEL*

S. Ilieva, V. Pavlov, I. Manova, D. Manova

Abstract. Software applications created on top of the service-oriented architecture (SOA) are increasingly popular but testing them remains a challenge. In this paper a framework named TASSA for testing the functional and non-functional behaviour of service-based applications is presented. The paper focuses on the concept of design time testing, the corresponding testing approach and architectural integration of the consisting TASSA tools. The individual TASSA tools with sample validation scenarios were already presented with a general view of their relation. This paper's contribution is the structured testing approach, based on the integral use of the tools and their architectural integration. The framework is based on SOA principles and is composable depending on user requirements.

1. Introduction. The popularity of software applications created on top of the service-oriented architecture (SOA) has been increasing greatly over

ACM Computing Classification System (1998): D.2.5.

Key words: software and services, testing Service-based Applications, testing framework, testing tools.

*The work reported in this paper was supported by a research project funded by the National Scientific Fund, Bulgarian Ministry of Education, Youth and Science, via agreement no. DOO2-182.

the last several years. Since SOA applications are composed of loosely coupled, business-level services, distributed over a network, we must test the end-to-end application, the individual services and their interfaces. Testing of Service-based applications (SBA) presents challenges due to their heterogeneity and lack of well-defined constraints, the business flexibility, the dynamic binding during execution and the reusability of the services. The following key problems in testing of SBA, based on the state-of-the-art survey, are identified:

1. Lack of a possibility for instrumentation of the Services which are not under our control.

2. Lack of functionality for automation of system behaviour testing under conditions of poor or unavailable communication channels with remote services.

3. Inability to test the service compositions because some of the components may not be available due to the parallel development of interdependent components.

In this paper we propose a TASSA framework that addresses the problems listed above with a focus on the concept and integration of the consisting TASSA tools.

The rest of the paper is organised as follows: Section 2, "Related Work", introduces current research efforts in the area of testing services and service-based applications. Section 3, named "TASSA Framework", presents the framework concept and testing approach. Section 4 presents the design-time TASSA tools with their main functionality and architecture. Section 5 concludes the paper and gives directions for future steps.

2. Related work. Research in the field of testing of services and service-based applications has been very intensive. There are approaches and tools for Fault injection [3, 11, 6, 17], Testing WS-BPEL [14, 24, 7, 9, 10, 23].

Kuk and Kim [8] proposed a framework for robustness testing of the service composition based on WSBPEL. It generates the virtual testing environment. Through this framework, the robustness of compositions of Web services against various errors and/or exceptional cases can be verified.

The project PLASTIC [16] proposes a whole platform which "enables robust distributed lightweight services in the B3G networking environment". The part of the platform called PLASTIC validation framework is of particular significance to our research [2]. It supports the validation of networked services with regard to both functional and extra-functional properties, and works in two

stages—off-line and on-line. The foreseen domains are e-business, e-health, e-voting and e-learning applications. One of the available tools named WS-Taxi is a WSDL-based Testing Tool for Web Services [1]. It can be used for automatic and controlled generation of valid and invalid instances, thus enabling the automated testing of I/O behaviour.

The project SOA4ALL [18] aims to "provide a comprehensive framework and infrastructure that integrates five complementary and revolutionary technical advances (SOA, Context management, Web principles, Web 2.0, Semantic Web) into a coherent and domain independent service delivery platform". The scope of the project is huge but the important achievements in terms of our research project are the proposed design of the Service Monitoring and Management Tool Suite [19] and SOA4All Testbeds specification and methodology [20]. The SOA4ALL approach to testing is related to functional testing (black box testing) and integration testing (any service invocation within a process or a system should be tested against any possible binding). The comprehensive specification and design offered by SOA4All could be a useful base for further development and implementation.

Our approach aims to address not only functional correctness but also other specified quality characteristics of SBAs such as robustness, performance, etc. The implementation of the framework is based on open source JBI-compliant ESB, such as Sun's openESB, which will be extended in two directions: extension of openESB's functionality and development of integrated testing tools. In this respect TASSA can also be seen as a framework where some of the abovementioned tools could be integrated for extension depending on the application domain and the context of the applications.

According to our knowledge currently there is no available open source testing suite with all the functionalities of the proposed complete TASSA framework. Rather, there are separate tools for testing on different layers (Business Process, Service Composition and Coordination, Service layer) of Service-based applications. The software developers, and particularly the application integrators, lack practical testing suites and corresponding testing methodologies. The TASSA framework addresses these needs, aiming to be both practical and complete.

3. The TASSA framework. The main goal of TASSA is to support the testing, validation and verification of both the functional and the nonfunctional behaviour of service-based applications at design time and at runtime. It consists of a methodology and a set of tools that can be used together with existing development environments of SBA. In this section, we describe the general

concept behind the TASSA Framework at design time.

3.1. TASSA concept. The proposed TASSA Framework comprises a set of tools that can be used together with existing SBA development environments. The TASSA tools complement the existing testing tools and can be used jointly to achieve end-to-end design time testing of SBA.

Addressing the problems of design-time SBA testing described in Section 1, our approach is the following:

- Provide a software tool (**Isolation Tool**) for the purposes of testing in order to temporarily remove dependencies of the software application under test from external services (services out of our control). This will allow the tester to control the service-returned results and pre-determine the possible routines in the process (problem 1), as well as to continue testing of the application even if a service is missing (problem 3).

- Judgement on the behaviour of a service can be quite demanding task. Therefore there is a need for an instrument (**Data Dependency Analysis Tool**) to advise the quality assurance engineer as to what results should be simulated in order to facilitate the consistency testing of particular business logic.

- Generation of input values (including randomization) for the different test cases (**Value Generation Tool**), which is needed by The Test Case Generation Tool.

- For the automation of system behaviour testing in case of drop off or poor quality of some of the communication channels, a tool (**Fault Injection Tool**) is provided, through which these cases are simulated in the ESB environment where we have full control (problem 2).

- Provide an instrument (**Test Case Generation Tool**) based on the two tools mentioned above (Isolation Tool and Data Dependency Analysis Tool) by which to analyse the business process in order to generate test cases covering all possible routines in the process.

Through those five tools, issues 1), 2) and 3) are solved and automation of the functional and the negative testing is achieved. According to our knowledge, such an overall approach for solving the above issues is currently not available.

Fig. 3.1 gives a general picture of the tools involved and their relation. The application to be tested is presented by the BPEL [4] description of the

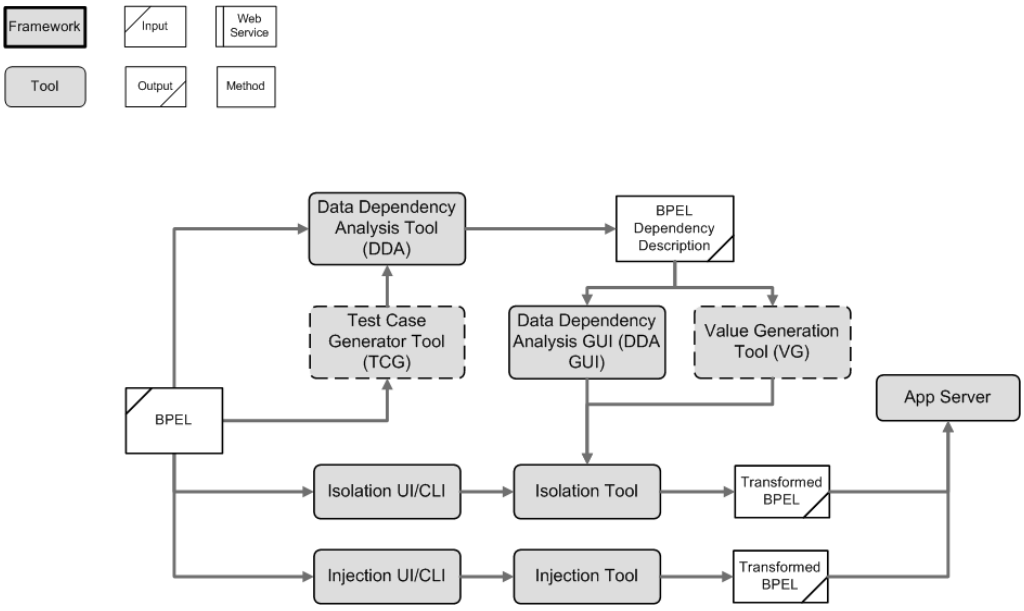

Fig. 3.1. TASSA Architecture

process, which in turn includes all the necessary WSDL and XSD descriptions of the operations called, the structures used and the messages exchanged.

The TASSA testing approach at design time includes the following steps:

1. Through the DDA tool (Covering Scenario Generation Algorithm) a set of paths (Path_N) and data related to those paths (Data_N) guaranteeing passing through them is defined. The defined pair Path+Data is called a testing scenario.

2. Based on previously defined criteria (Criteria_N) related to the concrete testing cycle, the subset of testing scenarios is identified. Those scenarios are used for validation and verification of the defined criteria. For that purpose the GQM (Goal Quality Metric) paradigm is applied for defining the goals and metrics for measuring the results. In practice each complex business process described by BPEL is decomposed to a set of simple BPEL processes (BPEL_N), which are executed explicitly (without possibilities for variants of their execution—for example the existence of *if-then-else* constructions). For each simple process the expected result is defined for the predefined input data.

3. This set of BPEL processes and the data, which guarantee passing through the paths (BPEL_N+Data_N), is notated as test cases N (Test_Cases_N) for the particular testing cycle. The Black-box technique for testing and assessment is applied.

4. The Isolation Tool supports the execution of positive test scenarios. It simulates the running of services which are not ready or are not available at the moment of executing the testing activities. The simulation is a well-known approach, but the new contribution is the use of a mechanism for isolation/substitution of the service call, instead of implementation with appropriately generated stubs.

5. Generation and execution of negative test scenarios is done by a fault injection mechanism implemented in the Injection Tool. The supported negative testing scenarios include:

   - Impossibility of sending or receiving a message—usually such a fault is caused by a failure in the communication channel, but with the support of the Injection Tool, this interruption is injected as the corresponding exception, under the control of the user.
   - Delay of sending or receiving a message—usually due to an overload in the communication channel that is hard to simulate in reality. The Injection Tool does it by invoking the original operation, waits for a specified number of seconds and returns the message to the BPEL process.
   - Message with syntax and structure errors caused by noise in the communication channel, currently injected as garbage, and easily controlled.

Messages with syntax errors in their data because of wrong business logic of particular web service are simulated with the help of the Injection Tool and the Value Generation Tool (WS-TAXI), which generate XML instances with random or predefined data values. In this way the Value Generation Tool changes the data values without any other intervention in the message or the partner service.

The details related to the implementation are described in the respective testing instruments.

6. Execution of positive and negative tests in an integrated testing environment

7. After the execution of each test case the actual and expected results are compared.

8. Based on the information obtained during the execution of the testing scenarios of the systems Behavioural patterns are defined. Those patterns are used for prevention or identification of problems by monitoring the software system in its real exploitation. The Behavioural patterns are in open database, which is being updated and extended during the exploitation process.

3.2. Use of TASSA. In the context of the TASSA framework two main strategy paths are envisioned:

- Development support—the focus of this strategy path is enabling efficient technological development of business process elements, their integration and providing methods and tools for quality measurement and control throughout these activities.

- Testing support—this strategy path captures the concepts and activities related to the TASSA framework from a quality assurance perspective. The focus in this case is on how to assess parameters related to functional and non-functional requirements of an SBA application based on BPEL abstraction at any stage of development or exploitation.

Related to the Testing support aspect the following types of testing are covered by the TASSA framework:

- Functional testing, where three main scenarios are included:
    - Path coverage according to functional requirements—testing the full branch coverage with shortest path length of the business process, to verify that it covers the functional requirements and that the expected results are met;
    - Strong data-dependent process—testing the full branch coverage with shortest path length of the business process with a different set of values, application of some check points and verification that the expected results are met;
    - Path coverage according to new functionality—testing the specific branch/es of the business process, in order to verify that the new functionality is appropriately implemented and that the expected results are met; testing the full branch coverage with shortest path length of

the process, to verify that the old functionality is not broken; comparing the test before and after the new, which has been implemented functionality.

- Reliability testing, where TASSA provides simulations of possible failures of unreliable service(s), and testing those paths in business processes that include branches with reliable services.
- Resilience testing, where TASSA provides simulations of possible failures of unreliable service(s), and testing those paths in business processes that include branches with unreliable services.
- Scalability testing, where the configuration changes are followed by testing the full branch coverage with least path length of the process, in order to verify that it will continue to work normally.

The abovementioned scenarios are further detailed and experiments based on real business processes are implemented, and are discussed in various papers. Based on the results from the abovementioned functional, reliability, resilience or scalability testing the further steps and strategies for SBA improvement must be undertaken by developers.

4. TASSA tools. In this section the TASSA design-time testing tools are presented.

4.1. Isolation tool. The Isolation Tool is central to the TASSA Framework. It provides the functionality needed to isolate the BPEL under test from outside dependencies. It works by transforming the BPEL so as to remove one or more dependencies and replace them with activities that are internal to the BPEL and can mimic the output of the original activity. The approach it follows is described in [5].

The tool is implemented as a web-service that provides methods for simulating these activities. For example, simulating the Invoke method accepts a BPEL, an XPath statement that points to the activity which needs to be replaced, and an array of assignment descriptors that will become the body of the Assign activity. The methods return the modified BPEL, which can be used to perform the necessary tests. The tool takes special care to transport essential attributes of the activity being simulated to the artificial activity and can handle both WS-BPEL 2.0 and IBM's proprietary version BPEL.

Apart from the actual web-service that performs the job, the framework includes two user interfaces (see Fig. 4.1): 1) a GUI realized as a Sun NetBeans

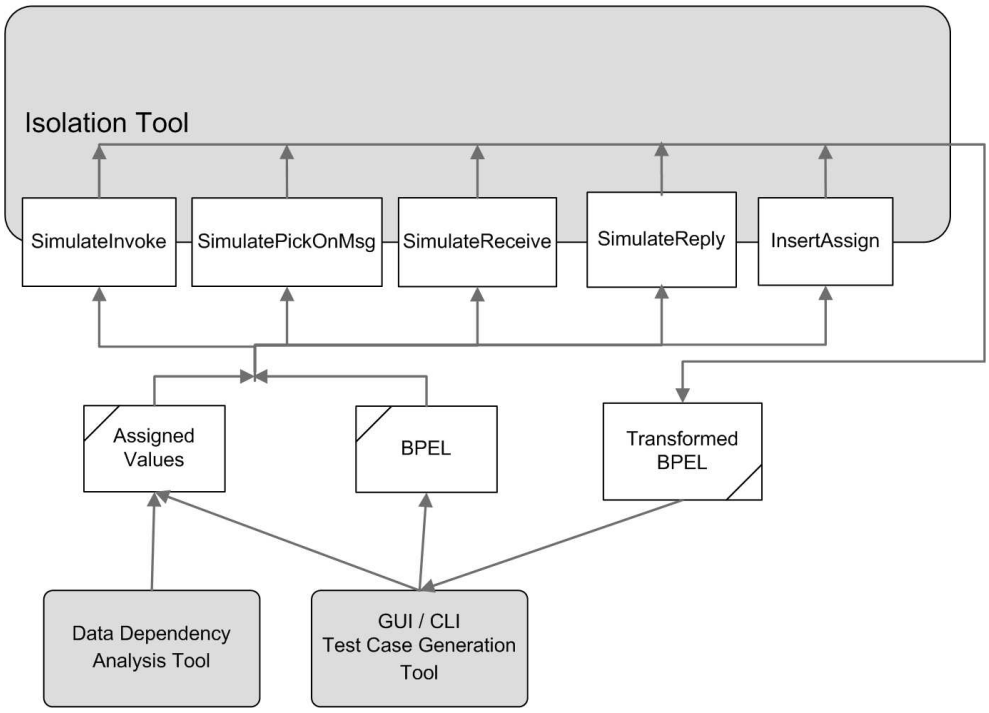

Fig. 4.1. Isolation Tool

IDE plug-in which allows for easy specification of the inputs; 2) a CLI that allows for batch processing and pipelining several calls.

By pipelining several calls to the tool's methods, one can completely isolate the BPEL from external dependencies. Moreover, since the user has control over the output of the simulated activities, he can guide the application into one execution path or another by setting proper values for the variables on which branch conditions occur. Finding such a sequence of branch conditions and the corresponding values is a hard task but can be automated and this is where the second TASSA tool can help.

4.2. Data Dependency Analysis tool. The Data Dependency Analysis Tool (DDAT) solves two tasks. First, for a given set of BPEL activities, the tool finds a path that goes through all activities in the set starting from one initial activity. Second, it finds all control activities on the discovered path and calculates the condition that should be met in order for the process to continue execution along the path.

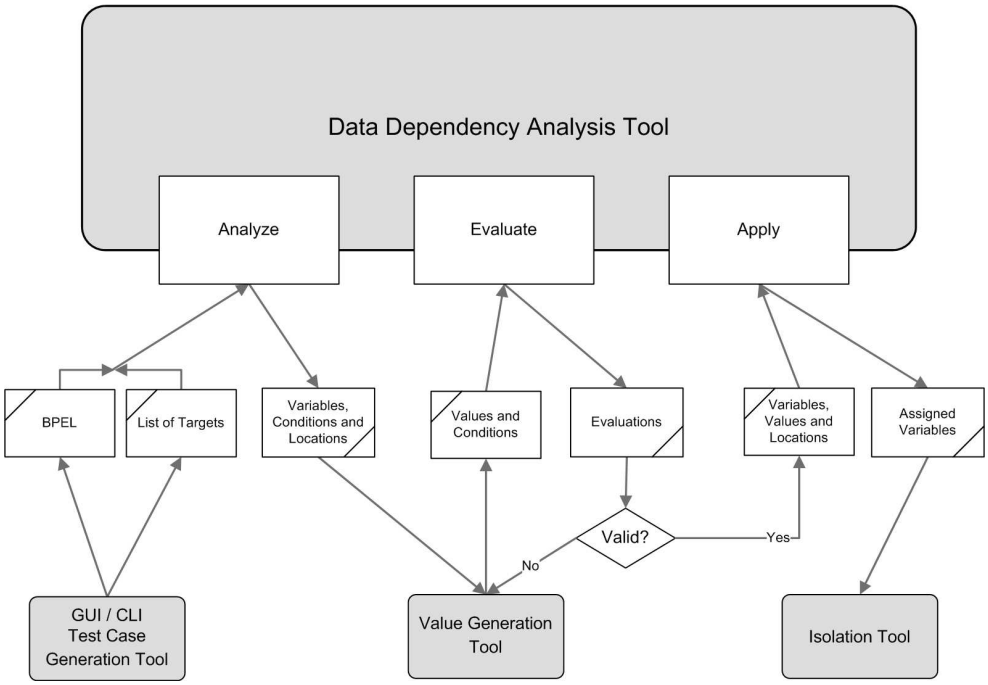

Fig. 4.2. DDA Tool

DDAT consists of the following components: Data Dependency Analysis Web Service (DDAWS), Graphical User Interface (GUI) and Command Line Interface (CLI). The architecture of DDAT is shown on Figure 4.2.

The approach and some experimental results are presented in [21, 22].

4.3. Value Generation Tool. The Value Generation (VG) Tool solves the following problem: given a condition in the form produced by the DDA Tool, provide values for the variables involved so as to satisfy the condition. The tool must support the generation of values for all XSD simple types and all types derived from simple types by means of constraints. The complex types are sequences of simple types and for the means of generation; each item in that sequence is treated as a standalone variable of a simple type. So the main task of VGT is to generate valid values for all fields of a given variable defined with XML Schema Definition (XSD). The generated data is applied in (1) creating of testing scenarios for the independent testing of web services (for generation of input data), (2) the integral testing of the BPEL process (for generation of input data) and (3) the testing of particular paths within the BPEL process

(value generation for the variables for corresponding conditions). The first two applications are well known, while the third one improves (adds new features to) the DDA tool and is specific for the TASSA framework.

The approach to solving the problem of generating test data in TASSA, is selecting an existing software tool that integrates into this test framework and serves its specific needs. An appropriate software tool for TASSA is WS-TAXI [1]. It is developed by a research team of the Software Engineering Research Laboratory at the ISTI—Istituto di Scienza e Tecnologie dell'Informazione A. Faedo in Pisa. WS-TAXI generates compliant XML instances from a given XML Schema by using well-known Category Partition technique [15]. From the XML schema definition, WS-TAXI analyses the types and frequency of occurrence of elements, also specifying the limitations. Using the information obtained and the schema's structure, TAXI generates a set of intermediate instances, combining the values of frequency of occurrence of the elements. The final instances are result of the intermediate with assigned value to each XML element. The values filled in the XML instances could be randomly generated or predefined. This way the values returned by the Value Generation Tool enable the user to put the business process under a variety of test cases. On one hand, the random data generation provides a randomly-distributed coverage of the BPEL response space, on the other hand – predefined, intelligent data tailored specifically to the values under test.

4.4. Injection tool. The set of tools depicted in Fig. 3.1 work together to help generate test cases that do a full path coverage testing of the BPEL. They can also help to isolate a specific activity, which for example blocks the testing process because it is not operational. On a different perspective, we want to verify what the impact is of a faulty communication infrastructure on the SBA under test.

The main task of the Injector tool is to simulate a random but valid combination of faults during message exchange in order to generate negative test cases. The possible situations that are simulated as of now are (1) overload of the communication channel that leads to a delay of sending or receiving a message, (2) failure of the communication channel that leads to impossibility of sending or receiving a message, (3) noise in the communication channel that leads to receiving a message with syntax and structure errors, and (4) wrong business logic of a particular web service that leads to sending or receiving a message with syntax errors in its data. The Injector tool takes as input a BPEL process under test, a list with failure parameters that describe the above situations and a string with values, which will be used in case (4). It returns a transformed BPEL

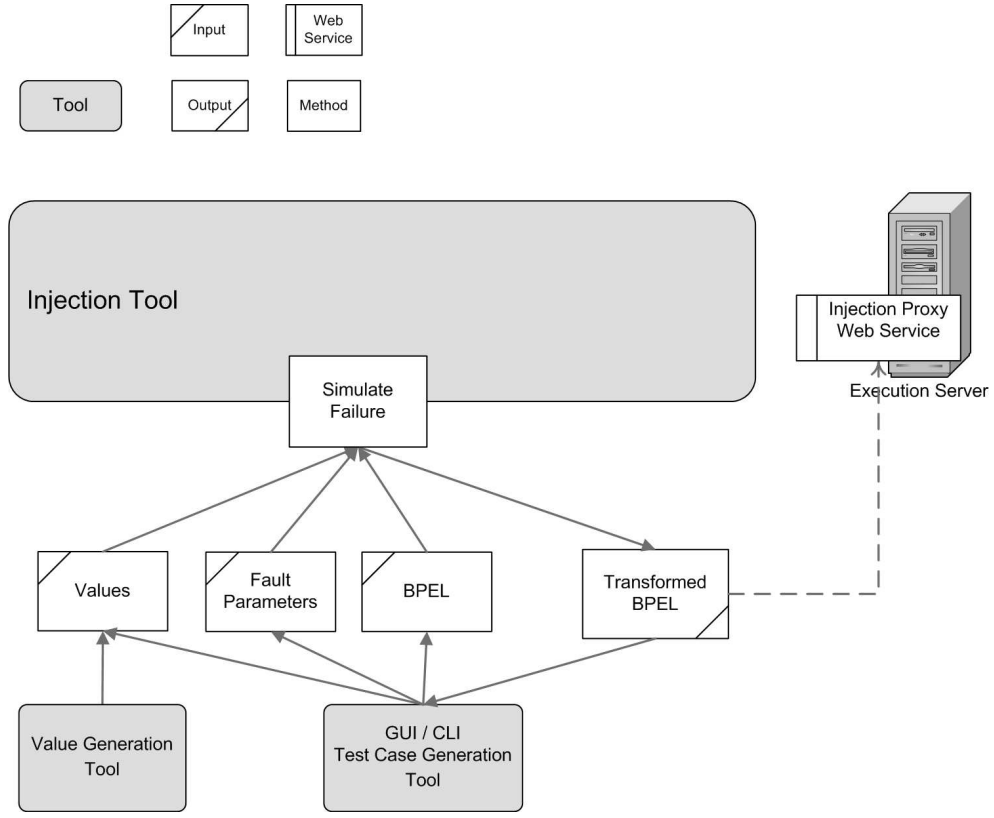

Fig. 4.3. Injector Tool

process with a call to a web service called Injection Proxy. The Injection Proxy Web Service is part of the TASSA framework and its main responsibility is to inject the faults described above at runtime.

The Injection Tool interacts with the user through a graphic user interface or command line interface to get the necessary input—BPEL, fault parameters and predefined values when necessary. On a greater level of automation the Test Case Generation Tool described in the next section and the Value Generation Tool could provide the input information.

The approach is presented in [12] and some experimental results in [13].

4.5. Test Case Generation tool. The Test Case Generation (TCG) Tool solves two main tasks—generation of testing scenarios for all possible execution paths in the business process (full path coverage) and management of the repository of test cases.

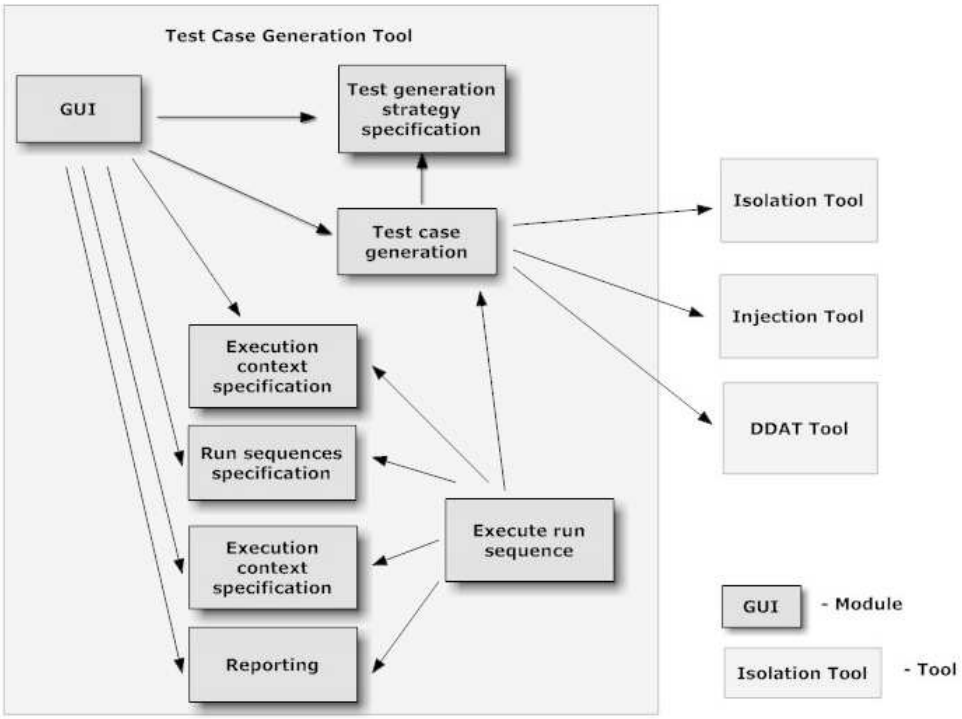

Fig. 4.4. Test Case Generation Tool

Task 1

The TCG Tool (Fig. 4.4) uses the Isolation, DDA and VG Tools in order to generate test cases that traverse the BPEL for all possible execution paths. In general the TCG Tool proceeds as follows:

- Convert the BPEL into internal representation;
- Using combinatorial algorithms, find all paths from the Initial Receive activity that initiates the process, to the final Reply activity that terminates the process;
- For each path, find a set of activities that belongs to this path only;
- Using the VG tool, generate values for the relevant fields that satisfy these conditions;
- Using the Isolation tool, instrument the BPEL, inserting corresponding Assign activities with the values from the previous step, in front of their

corresponding control activities, so as to make sure that the conditions are met;

- When executed, the modified BPEL will certainly take the chosen path;
- Continue with another path from the set of all possible paths;
- At the end, we will have a set of BPELs, each of them taking a predefined path of execution; their collective set covering all possible execution paths. Run the BPELs and verify by whatever means necessary whether the execution path behaves as expected.

Task 2

The test case repository is a database that keeps for each test scenario a set of attributes related to it, such as:

- A unique scenario identifier, given by the quality engineer (tester);
- Type of the scenario—one of the supported following testing scenario types:
  - Type 1: test of independent service (VG Tool, WS-TAXI);
  - Type 2: test of BPEL process isolated from external services (Isolation Tool);
  - Type 3: test of particular paths within BPEL process PEL (DDA Tool, Isolation Tool, TAXI);
  - Type 4: test in the circumstances when problems in communication channel appear (Injector Tool);
  - Type 5: test of all possible execution paths in the business process described by BPEL (TCG Tool);
- List of and pointers to the testing artefacts which are necessary for the execution of the corresponding scenario—for example modified BPEL files, input data for the instruments, etc.
- Expected results, which are compared with the real output data;
- History – when was the scenario used and what was the result;
- Rules for applicability of the scenario (for example the scenario X is not applied in version Y).

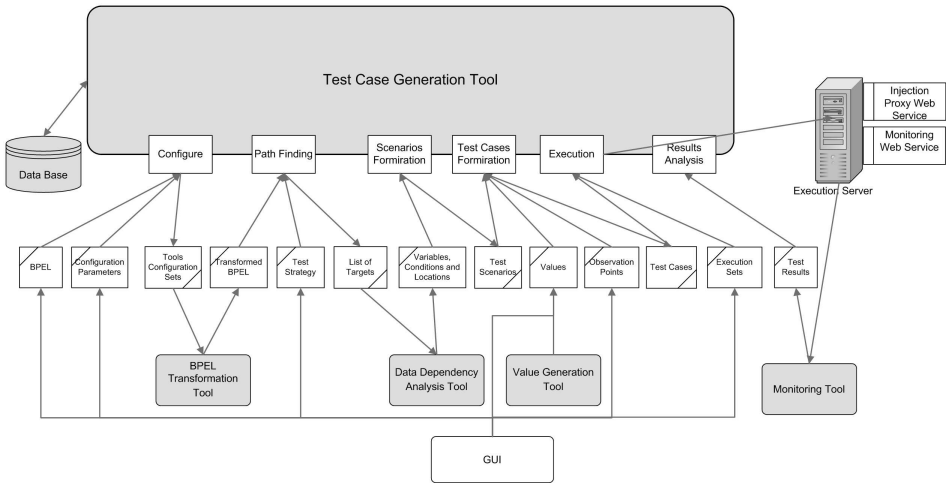

Fig. 4.5. TASSA Tools

The TCG tool could manipulate the above list of attributes by the following activities:

- Creation of a new testing scenario;
- Editing testing scenarios, including editions of different related data (artefacts, expected result, applicability rules, etc.);
- Deleting testing scenarios;
- Issue of one or more testing scenarios.

The TCG tool is implemented as a service-based application that uses the services provided by the rest of the tools. It is itself exported as a service and can be composed in other SBAs.

The integration of the TASSA tools with respective artifacts is shown in Figure 4.5. For a clearer presentation in the figure the Isolation and Injection Tools are united in the Tool named on the main function they are dealing with—BPEL Transformation Tool.

To solve all issues mentioned above the Test Case Generation Tool should work as a manager, to coordinate the work and the communication between all of the TASSA's tools, the storage space, the execution server and the user.

5. Conclusion. In this paper we presented, at a conceptual level, a complete practical testing suite for end-to-end design time testing of service-

based applications. The contribution of the proposed TASSA environment could be summarized in the following list of its advantages:

- An integrated set of tools cover completely all testing activities relevant to the Life Cycle of development of service-based applications;
- The implementation of the TASSA framework is based on open source solutions;
- It is technology-independent and generic, because it is based on SOA standards;
- It includes not only technological solutions but also methodology guidelines;
- Because of its nature TASSA can be extended and developed on demand.
- Also the testing approach was presented step by step and will be extended to testing methodology in the future work.

REFERENCES

[1] Bartolini C., A. Bertolino, E. Marchetti, A. Polini. WS-TAXI: A WSDL-based testing tool for web services. In: Proceedings of the International Conference on Software Testing Verification and Validation (ICST '09), Denver, Colorado, USA, 2009, IEEE Computer Society, 326–335.

[2] Bertolino A., D. Bianculli, A. Carzaniga, G. D. Angelis, I. Forgacs, L. Frantzen, Z. Gere, C. Ghezzi, A. Polini, F. Raimondi, A. Sabetta, A. Wolf. Test Framework Specification and Architecture. Technical Report Deliverable D4.1, PLASTIC Consortium, March 2007, IST STREP Project.

[3] Bessayah F., A. Cavalli, E. Martins. A Formal Approach for Specification and Verification of Fault Injection Process. In: Proceedings of the 2nd Int'l Conf. on Interactive Sciences, Seoul, Korea, November 2009.

[4] Web services business process execution language version 2.0. `http://docs.oasisopen.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, 2007

[5] Ilieva S., V. Pavlov, I. Manova. A Composable Framework for SBA Test Automation. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology (QUATIC'2010), 29 September–2 October 2010, Porto, Portugal, Print ISBN: 978-1-4244-8539-0, 286–291.

[6] Juszczyk L., S. Dustdar. Programmable Fault Injection Testbeds for Complex SOA. In: Proceedings of the 8th International Conference on Service Oriented Computing, San Francisco, USA, December 2010, 411–425.

[7] Karam M., H. Safa, H. Artail. An abstract workflow-based framework for testing composed web services. In: Proceedings of the International Conference on Computer Systems and Applications (AICCSA), 2007, 901–908.

[8] Kuk S., H. Kim. Robustness Testing Framework for Web Services Composition. In: Proceedings of the IEEE Asia Pacific Services Computing Conference 2009, 319–324.

[9] Li Z. J., W. Sun. BPEL-Unit: JUnit for BPEL Processes. Service-Oriented Computing—ICSOC, Lecture Notes in Computer Science, Vol. **4294**, Springer, 2006, 415–426.

[10] Li Z. J., H. F. Tan, H. H. Liu, J. Zhu, N. M. Mitsumori. Business-process-driven gray-box SOA testing. *IBM Systems Journal*, **47** (2008), 457–472.

[11] Looker N., B. Gwynne, J. Xu, M. Munro. Determining the Dependability of Service-Oriented Architectures. *International Journal of Simulation and Process Modelling*, **3** (2007), No 1–2, 88–97.

[12] Manova I., D. Manova, V. Pavlov, S. Ilieva, D. Petrova. Fault injection testing of web service business processes. *International Journal on Information Technologies and Security*, **2** (2011), 3–12.

[13] Manova D., I. Manova. S.Ilieva, D. Petrova-Antonova. faultInjector: A Tool for Injection of Faults in Synchronous WS-BPEL processes, In: Proceedings of the 2nd Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC 2011), Bratislava, Slovakia, 99–105.

[14] Mayer Ph., D. Lübke. Towards a BPEL unit testing framework. In: Proceedings of the workshop on Testing, analysis, and verification of web services and applications, 2006, 33–42.

[15] Ostrand T. J., M. J. Balcer. The Category-Partition Method for Specifying and Generating Functional Tests. *Comm. ACM*, **3**1 (1988), No 6, 676–686.

[16] Plastic project. `http://www.ist-plastic.org/`

[17] Reinecke Ph., K. Wolter. Towards a Multi-Level Fault-Injection Testbed for Service-Oriented Architectures: Requirements for Parameterisation. In: Proceedings of the SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems, Naples, Italy, 2008, doi: 10.1145/1596473.1596478.

[18] Service Oriented Architectures for All (SOA4All).
http://www.soa4all.eu/

[19] Service Oriented Architectures for All (SOA4All), deliverable D2.3.1 Service Monitoring and Management Tool Suite Design.
http://www.soa4all.eu/file-upload.html?func=startdown&id=17

[20] Service Oriented Architectures for All (SOA4All), deliverable D1.5.1 SOA4All Testbeds Specification and Methodology
http://www.soa4all.eu/docs/D1.5.1%20SOA4All%20Testbeds%20Specification%20And%20Methodology.pdf, 2009

[21] Spassov I., D. Petrova, V. Pavlov, S. Ilieva. An approach for Data Dependency Analysis of web service business processes. *Comptes rendus de l'Académie bulgare des Sciences*, **64** (2011), No 3, 405–412.

[22] Spassov I., D. Petrova, V. Pavlov, S. Ilieva. An approach for Data Dependency Analysis of BPEL processes. In: Proceedings of the Second International Workshop on Software Quality SQ (SQ 2011) within the International Conference on Computational Science and Applications (ICCSA 2011), Santander, Spain, June 20–23, 2011.

[23] Yuan, Z. Li, W. Sun. A graph-search based approach to BPEL4WS test generation. International Conference on Software Engineering Advances, ICSEA, Taiti, 2006, 14–22.

[24] Zheng Y., J. Zhou, P. Krause. An Automatic Test Case Generation Framework for Web Services. *Journal of Software*, **2** (2007), No 3, 64–77.

S. Ilieva
Institute of Information and Communication Technologies (IICT)
Bulgarian Academy of Sciences
Acad G. Bonchev Str., Bl. 25A
1113 Sofia, Bulgaria
e-mail: sylvia@acad.bg

V. Pavlov, I. Manova, D. Manova
Rila Solutions
Acad G. Bonchev Str., Bl. 27
1113 Sofia, Bulgaria
e-mail: vpavlov@rila.bg,
ilinam@rila.bg, denitsat@rila.bg

Received January 12, 2012
Final Accepted January 26, 2012