Testing of Service Oriented Architectures – A practical approach

Schahram Dustdar, Stephan Haslinger

[1] Distributed Systems Group, Vienna University of Technology
dustdar@infosys.tuwien.ac.at
[2] UCS GmbH, Vienna, Austria
stephan.haslinger@ucs.at

Abstract. Service Oriented Architectures (SOAs) have recently emerged as a new promising paradigm for supporting distributed computing. Web services, as well as integration-packages relying on message oriented middleware are special substitutes for this kind of architecture. However, even if a lot of work has been focused on how to build such systems less work has been invested in how testing and monitoring such systems could be done. It is obvious that cyclic testing in the development phase of such architecture can help to identify vulnerabilities early in the design phase which leads to a stable deployment of the system. Testing in distributed systems is very challenging and automated test tools can help to reduce the development costs enormously. In the next years vast investment will be made in integrating systems, as a lot of companies have the need to integrate their systems to establish more flexible workflows. In this paper we will propose an approach as to how automatic testing for SOAs can be done. We will introduce a Meta language in XML, which allows defining test cases for services. A service can be a Web service, as well as an adaptor for a messaging-system, such as Tibco. We define a generic Meta language, but predict that maybe some cases are not covered, as our research for automatic testing of SOAs is still in progress. This paper focuses on a real life prototype implementation called SITT (Service Integration Test Tool). SITT is designed in a way that it can also be used as a monitoring system for SOAs. It has the possibility to test and monitor if certain workflows between multiple service endpoints really behave as described with the XML Meta language. We will also explain a feature called online and batch testing. Online testing means that tests are going on in real time; in contrast batch testing has the possibility to test service endpoints not connected to SITT, which will be often the case when integrating systems with external departments or companies. This paper shows how SITT is designed and we will present its features by introducing a real-world application scenario from the domain of Telecommunications providers, namely "Mobile Number Portability". This paper focuses on SITT as a test tool, how SITT can be used as a monitoring system is not part of this paper.

Keywords: Service Oriented Architecture, Automatic Testing, Web services

1 Introduction

Service Oriented Architectures (SOAs) have recently emerged as a promising paradigm for supporting distributed computing. SOAs are often used in intra-enterprise-integration, such as Message Oriented Messaging systems, as well as in inter-enterprise-integration, where Web services are utilized. As the systems are often asynchronous, testing can be very challenging. Testing of such systems therefore, often goes hand in hand with setting up test systems performing some message exchanges and to analyse the results. This is very time consuming and inefficient, as manual intervention is needed. An ideal state would be if such tests could be performed automatically by just pressing a button of a test system and getting back a report of the performed tests and results. This would help to advance the quality and reliability of the system and would help to find errors in the system earlier in the development phase. It is obvious that such a test system always has the need for adapting it to special test scenarios. In this paper we present one approach how a deployment of such a test system could look like.
The contribution of this paper is a description of a test system for SOAs (SITT), as well as a Meta language in XML, which is used to describe the test cases for SITT. The goal is to give the implementer a powerful tool, so that testing can be done in early stages of the development cycle. The concept will be designed in a way that the test system can be also used for monitoring the system once it is deployed. The reconfiguration from the test system to a monitoring system should be as easy as possible. The advantage of this concept is that the test system can function as a monitoring system just by doing little reconfiguration. As our discussion with developers working in the field of EAI showed, many of them argued that it is not always possible for them to set up an efficient monitoring tool once the product is deployed, as the development cycles in some areas is very low. Time for setting up monitoring is often not planned. If we think about telecom companies, the development cycles are sometimes just a few weeks for launching a new product to the market. We can imagine that in some cases time for testing and monitoring has not the highest priority in the project plan. With an automated testing tool it is obvious that such disadvantages can get corrected.

2 The Need for Testing

This section provides an introduction to current integration techniques and testing. It helps to understand which decisions led us to the investigation of testing for SOAs. The production of a high quality software product requires application of both defect prevention and defect detection techniques. A common defect detection strategy is to subject the product to several phases of testing such as unit, integration and system testing. These testing phases consume significant project resources and cycle time. As software companies continue to search for ways for reducing cycle time and development costs while increasing quality, software-testing processes emerge as a prime target for investigation [15]. In average software testing needs from 40% to 85% of the whole development life cycle [15, 10]. Even if testing is very cost

intensive, surveys showed that testing in every stage of the development cycle reduces the whole cost for a system. Testing can help to find errors in the system at an early stage and therefore time and cost intensive rectification of defects can get reduced. An experiment showed that developers are much more efficient when using automated test. Here a short abstract of a survey:

> *The experiment indicates that the tool has a statistically significant effect on developer success in completing a programming task, without affecting time worked. Developers using continuous testing were three times more likely to complete the task before the deadline than those without. Most participants found continuous testing to be useful and believed that it helped them write better code faster and 90% would recommend the tool to others. The participants were more resilient to distraction than we had feared and intuitively developed ways of incorporating the feedback into their workflow [16].*

In some areas automated testing of software parts is much more efficient and easier than manual testing, even if we like to point out that this is not always the case. Sometimes the configuration for automated tests needs more time than a manual test would do. The current state in software development is that testing is widely recognised as a key success factor, but that automated tests are rarely used. The problem often is that tools for automated testing are seldom integrated in software development IDE´s. JUnit integrated with the Eclipse-IDE is one of the creditable exceptions. As our research showed, there are tools for automated testing on the market but most of them focus on unit and regression testing. There is less research on how distributed objects, spread around several different machines can be tested. When we work in the area of SOA we face the problem that testing can be sometimes very crucial, extremely time consuming, and error-prone. Sometimes it needs more time to set up a test environment and to analyse the results than to write the source code itself. Whenever a change in one of the source code parts arises a new test has to be done and very often there is no stable test environment. This then leads to the problem of setting up the whole test environment for every single change. In such cases the investment for testing can explode. There are no reliable surveys which provide information on how much was invested in integration technique in the last years, but IDC [17] predicted that in 2003 50 billion dollars will be invested into software covering integration and there is an upward forecast for the next years. We assume that a large proportion of this investment was due to manual testing the components. If testing can be automated in a desirable way this would be a big economy of time and money without decreasing the quality of the software. The approach we are presenting in the following sections tries to show how a test tool could be designed to automate tests for SOAs. This paper is structured as follows. In Section 3 we will present an example, which shows the complexity of testing in the field of SOAs. In Section 4 we will then present the design concept of the tool. Section 5 gives an overview what can be tested. The XML Meta language can be found in the appendix [21].

3 Example – Mobile Number Portability

To highlight the challenge of testing SOAs we will first present an example and will then describe how a test suite could assist in testing such a system. For this we are using an example out of the telecom world, namely "Mobile Number Portability". Mobile Number Portability (MNP) is enforced by the European Union on its member states. Every Telco in Europe has to give his customers the possibility to change the service provider without losing his current telephone number [19]. Telecom companies have to set up a very flexible and reliable system when exchanging customer data for MNP. We presented an approach for this real world scenario in [19]. Assume Telco A has to port out a customer and therefore has to sent a request (portCustomer_Request()) to every one of its national partners, which we assume are four. To fulfil a successful "port out" each of these four partners has to acknowledge (portCustomer_Response()) the request within a certain timeframe, assumed to be three hours. After the successful receipt of the four acknowledgements the business process of Telco A proposes that the 'port out'-request has to be published (Customer_successfully_ported()) to the internal messaging bus and that at least the SAP system has to receive it within 2 hours after acknowledgement. Figure 1 provides a graphical overview of this example.

If we assume to test such a scenario manually, then it is obvious that a person would have to check some kind of log entries if the test succeeded or not. This can be very time consuming and besides this it is imaginable that such a test may not be done during the day, as it is likely that developers are working on the system.

This scenario is a good example for showing the need of testing workflows. Here a lot of endpoints are working together to fulfil a certain goal. SITT assists the tester to test such workflows effectively.

Tests familiar to these are needed very often in the set up phase of such a particular system and besides this often other project relevant problems are putting pressure on the test and development team. To set up such a scenario with the tool we will present in this paper requires some time and specific knowledge of the test person, but only consumes a little percentage of time and money as continuous manual tests would do. Once the workflow is set up properly the test can be done automatically and the results will be presented in a report. The tests can also be done scheduled over the night, therefore not disturbing daily work.

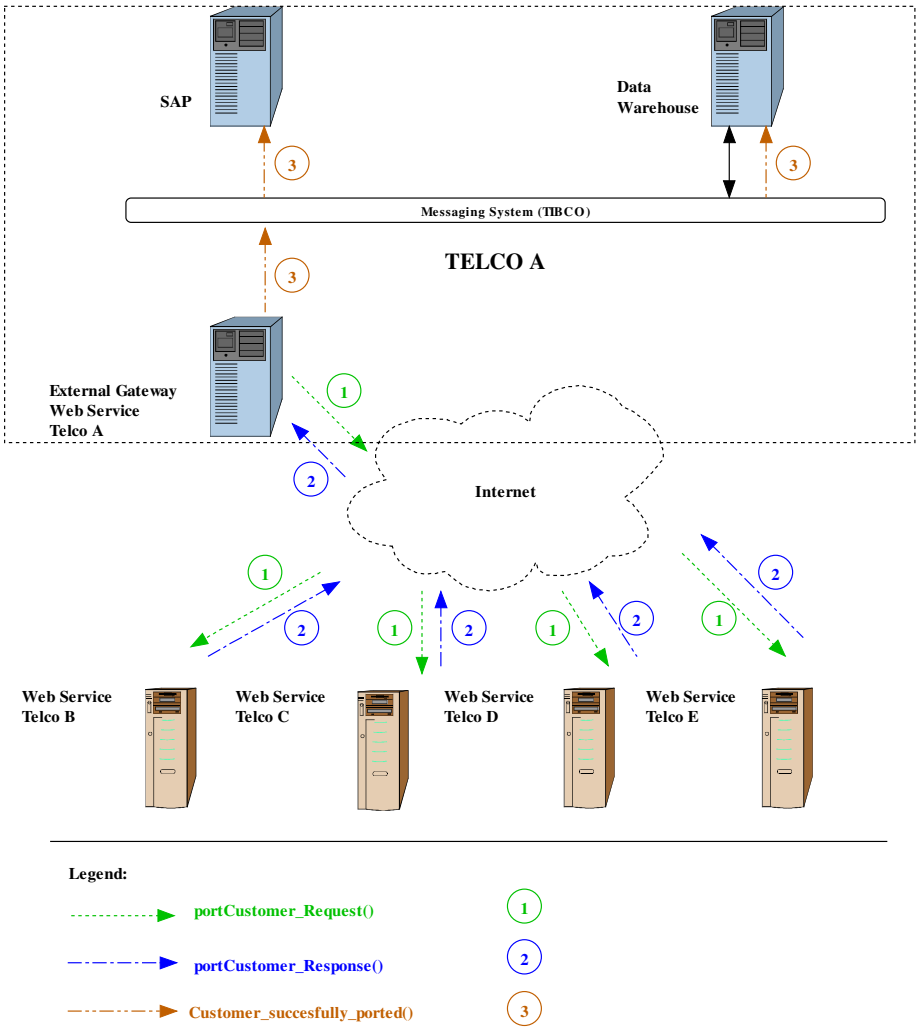

Fig. 1: Example Mobile Number Portability

4 Design Concept

In this section we present the overall design for our test tool, called SITT. SITT is an acronym for “Service Integration Test Tool”. The service endpoints can be Web services as well as adaptors to any kind of messaging product, such as “Tibco Rendezvous”. Hence, whenever we talk about a service, we are not just talking about one service endpoint, instead, we refer to a set of service endpoints working together to fulfil a certain goal. This concept is also known as “Service Orchestration”. The idea behind SITT is to test services and their workflow by analysing the message flows. To fetch every message a service endpoint receives or sends, the message has

to be written to a log file in a standardized manner. In the paper we will always refer to messages which get exchanged between service endpoints, but of course also function calls are eligible to test. The input of the log files is then read by a so-called "Test Agent" (TA) and is then sent to the "Master Agent" (MA). We refer to software as agents, but like to point out that we do not refer to any agent paradigm. After a certain time span it is possible to analyse if every message and their replies where in the right order and fulfilled conditions like "response time" and so on. With this analysis a deep insight of the messaging system can be achieved. It is also possible to analyse if a messaging system is working properly to a Service Level Agreement (SLA). Sometimes such SLAs include conditions like "a certain message has to be answered by the receiver within x seconds". All test scenarios will be stored in a database for later retrieval and traceability. In the next subsections we will explain in detail the design of SITT.

The test suite in the first phase uses 2 different kinds of XML sources. One XML source is used to hold configuration data for the MA and the TAs. The other XML source holds information about the test strategy used by the test daemon. The Appendix [21] gives detailed information about the Meta language and also provides sample XML files for configuration and testing by focusing on the MNP example from Section 3.

4.1 Idea and Configuration

Figure 2 shows a graphical overview of SITT. Every service endpoint involved writes the appropriate information to a log file. The information is read by the TA and is then sent to the MA over a socket connection. Figure 2 shows that on Host1 two service endpoints are running. Both are writing to the same log file. This is not necessary. If the service endpoints would use different log files then two TAs would have to run on Host1, one for every specific log file. If we think about a test suite for SOAs maybe one idea could be that the test suite itself sends messages, which are stored before. This would of course enhance the features of the test suite dramatically. But how could this be solved? One way could be to store messages in a file, or in the database and the test suite would have control of a certain service endpoint to distribute the messages. This testing of workflows could be tested much easier, as we will show later. We thought about a solution where SITT takes control over a certain service endpoint, which would function as an initiator for test cases, but after some design studies we found out that it is not practicable to create a common test suite with such a feature.

Nevertheless, SITT can assist testers efficiently. It also has the possibility of recording test cases. This means that once a test case is executed and all the information from the TAs is received, such a test case can function as a template. Whenever the same test is run again the results can be matched to the appropriate template and SITT analyses if the test succeeded or failed. Recording of tests has one drawback, namely that tests matching to a template have to provide exactly the same results in every field the TAs collect from the log file. Just the timestamp can vary. In Figure 2 all TAs connect independently to the MA, which follows a point-to-point structure. If we would have to test a system with a huge amount of services, for instance 1000, then this could lead to a performance problem. For the first phase of

SITT, where we will use to suite to gather information about the usefulness of such a tool, this is acceptable. In a future enhancement this problem may be solved by replacing the point-to-point connections with a bus structure.

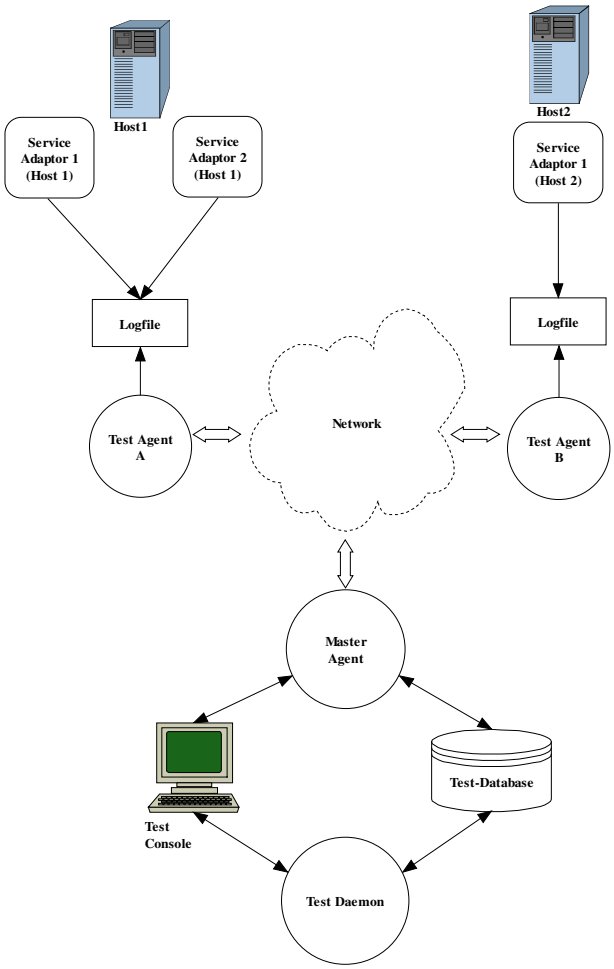

Fig. 2: Overview SITT

4.2 Test Agents

Test agents (TA) are programs, which run on the same machine as a service endpoint. Their purpose is to parse the log files the service endpoints are producing and to send this information to the Master agent (MA). The MA then stores the messages into the test database. In certain time intervals (normally every few seconds) messages from the test database are analysed from the test daemon against the predefined test behaviour described with the XML Meta language. After the TAs are started, an initialisation is done, which means that the TA connects to the MA via a socket over a

certain port. The port is of course configurable. Afterwards the TA sends an initialisation-message to the MA, which makes it possible for the MA to know which TA is online and which not. The initialisation-message contains the name of the TA which is unique. With this information the MA retrieves configuration information for the TA from the configuration repository, which is an XML-File. We will present the structure of this XML-file in the appendix [21]. The MA then sends back the configuration information for the TA. After the TA acknowledges the configuration, the TA is online and configured for the test. This has to be done for every TA, which is involved in testing.
It is obvious that the TA can be written in any programming language as long as the TA keeps with the protocol the MA expects. This makes the design very flexible as new TAs can be written in any preferable programming language.

4.3 Online Testing

Online testing means that testing is done in real-time with TAs. The test daemon frequently analyses the information the TAs send back to the MA and which are stored in the database. This testing mode can be used if every TA is able to open a connection to the MA. In some scenarios it is possible that the TA is not able to open a connection to the MA, e.g., due to the security policy or when the machine where the service is located, is not in the scope of the same department or company. In such cases online testing alone is not suitable enough. To eliminate this drawback SITT also has a possibility of Batch Testing.

4.4 Batch Testing

In certain cases it is not possible to test all involved service endpoints online via a direct connection of the TAs to the MA. In such cases SITT has the possibility to transfer the log files directly (via ftp, rcp, and so on) to the machine where the MA is running or to any other machine from where a connection to the MA can be opened. Then a TA is started. This extracts the important messages from the log file and sends them to the MA. In order the MA stores the information equivalent to the normal message exchange, in the test database. Afterwards the test daemon starts to analyse the test output. Batch testing is very powerful in inter-enterprise integration. In Section 3 we showed an example where batch testing is needed, as the Telco B, C, D, and E do not use SITT. Batch testing is a very powerful method to debug and trace system behaviour when an external system seems to have an error and the error cannot be analysed with the reply messages from the external system. One precondition is that the log files have a well-known structure. In few instances this will be true when working with external partners. In such cases, a special tool has to clear the exchanged log files to the fixed structure.

4.5 Log Files

SITT relies on log files with a fixed structure. The developer of the service endpoint has to take care that the service is producing valid log files according to the structure presented in this section. We assume that for every message exchanged, which means for every retrieved or sent message, one entry in the log file can be found. Every entry has to include the following:

Field Name	Short Value Field Name [1]	Value Description	Mandatory/ Optional
Unique Pattern	UP	'SITT-MESSAGE:'	M
MessageID	MID	A unique ID for the message. In the first phase SITT needs a fixed MessageID for every message; in a future enhancement this drawback will be solved by introducing algebraic functions for calculating MessageIDs.	M
Timestamp	TS	Format: YYYY-MM-DD HH24:MI:SS The timestamp has to be in Greenwich Mean Time (GMT)	M
ServiceName	SN	A unique name of the service endpoint.	M
MessageString	MS	The message itself as ascii-string 'NA' if no MessageString is available.	O
DirectionIndicator	DI	'S': if the service was sending a message 'R': if the service was retrieving a message	M
MessagePriority	MP	A valid priority index for the service if available: E.g.: JMS provides a 10 level priority (0-9), where 9 is the highest priority and 0 the lowest. 'NA' if there is no MessagePriority or the priority is not known.	O
Retrieved from	RF	ServiceName of the sender if available and DirectionIndicator is R. When using a publish-subscribe service the value is fixed 'PS'. 'NA' if the ServiceName of the sender is not known and it is no publish-subscribe service.	M
Sent to	ST	ServiceName of the recipient if available and DirectionIndicator is S. When using publish-subscribe service the value is fixed 'PS'. 'NA' if the ServiceName of the receiver is not known and it is no publish-subscribe service.	M
MessageSubject	MS	Subject of the message. 'NA' if no subject is available. The subject is important for publish-subscribe services.	O

Table 1: Log File Structure

Table 1 shows a structured log file for SITT. SITT should function as a generic test suitable for a wide range of SOAs, e.g., for Web services, adaptors for messaging systems and so on. The log file structure was designed in a way to cover most of the

[1] The Short Value will be used by the 'Message Exchange Protocol' between the TA and the MA.

aspects needed in testing workflows for such services. If a value is mandatory it is marked with an “M” in the column “Mandatory/Optional”, if it is obligatory it is marked with an “O”. Not every service can provide a meaningful value for every field and therefore not every possible test scenario can be run for every service. We would like to point out that the structure was chosen to have the possibility to test as many features of SOAs as possible, even if the structure is not exhaustive. There might be cases where additional attributes seem to be suitable.

5 Test Facilities & Test Strategy

Before we come to the XML Meta language used by SITT, we will discuss the test strategy of SITT, mainly which functions can be tested. SITT shows our first attempt of what can be tested in SOAs, but is not exhaustive. The design is built to plug in new test possibilities easily. The main focus in the first release is to test workflows.
We provide a detailed description showing which test cases, nearly useful for every SOA, can be tested, their meaning and their relevance.

5.1.1 Predetermined Workflow

The most powerful test possibility is to test if predetermined workflows are processed in the right way as determined by the designer or tester of the system. This feature will be used to extend SITT as a monitoring tool for messaging systems. This is a research project and the results will be released soon.
SITT will have the possibility to test simple workflows between service endpoints. Our work showed that this is the most challenging part of a test suite for messaging systems and we believe that test systems in the future have to focus on this stronger to be successful on the market.
The XML files covering the example from section 3 can be found in the appendix [21]. At the first sight it looks complicated to build such a XML file but we would like to point out that this drawback will be solved by introducing a GUI for setting up such test cases. In future our research will also cover if and how orchestration languages, such as BPEL, could be extended to be used by SITT.

5.1.2 Message Delivery

Many messaging systems are built in a way that they guarantee message delivery, e.g., using a messaging bus. Even if this is guaranteed it makes sense to test if a message really got delivered to certain recipients. If the messaging system does not support reliable message exchange, e.g. a Web service, this test case is even more relevant, as there is no mechanism for reliable message delivery. If we are talking about point-to-point connections, the sender knows who is the recipient of a certain message, in contrast to a publish-subscribe scenario. We assume that the designer of a system normally knows which services are subscribing to a special subject. If the designer likes to test if every service actually received certain messages this

information has to be provided in the XML Source for test scenarios. It can test if a service endpoint really sent or retrieved every single message. We will provide the possibility to indicate a timeframe in which the message has to be sent or received. Every message occurring after the timeframe will be not indicated as sent or received.

5.1.3 Message Ordering

Messages sent by a message producer with the same message priority and delivery mode and on the same topic in the case of pub/sub messaging style, must be delivered in the same order as it was sent [2].

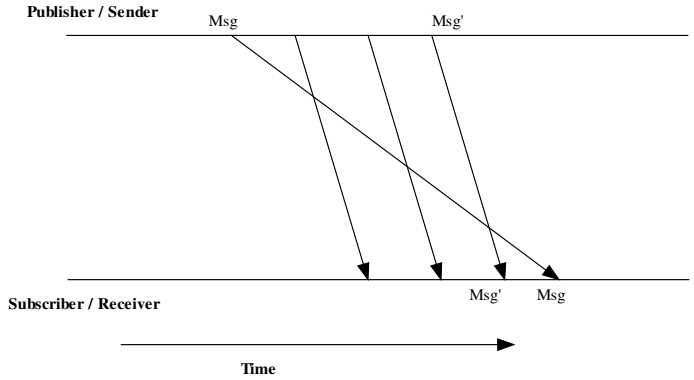

Fig. 3: Message Ordering

Figure 3 shows different messages sent from a publisher / sender to a subscriber / receiver. Even we would suppose Msg to be delivered before Msg' this is not the case, as Msg has a lower priority than Msg'. In the first implementation of SITT we will not provide a possibility to test messaging order if the messages have different priorities. In such a case SITT will fail. Our research is focusing on how the impact of message priority can be tested and there are some ideas which can be implemented, but nevertheless, the setup in the XML Meta language for those scenarios would be very challenging for the tester and we think that the usability of the test tool would decrease.

5.1.4 Performance / Throughput

Performance is used very often in software development without explaining the meaning of it and leaving the user just with a clue. In SITT performance covers following aspects:

- Messages sent by service endpoint / Timeframe
Number of messages a service endpoint sent in a specified timeframe.

- Messages received by service endpoint / Timeframe
Number of messages a service endpoint received in a specified timeframe.

- Messages sent by a group of service endpoints / Timeframe
Any preferred endpoints can be grouped by a service name to a particular group. The number will function as an indicator. This indicator makes sense if some service endpoints are designed to function in standby mode, where one service endpoint can take over the work of the other in case of any failure.

- Messages received by a group of service endpoints / Timeframe
Similar to "Messages sent by a group of service endpoints", this time indicating received messages.

These measures can be used to test if SOAs adhere to certain SLAs.

5.1.5 Reliability

Reliability gives an indicator how stable the system works through predetermined workflows.

#Successful workflows / #of all tested workflows

Every workflow is equitable, which means that the system makes no distinction in the importance of a workflow.

Example:
20 times "Workflow 1" and 25 times "Workflow 2" are tested. Workflow 1 ends successfully 13 times and Workflow 2 ends successfully 4 times. The reliability indicator in this case would be:

17 / 45 = 0.37

This means that 37% of all tested workflows ended successfully. In this example Workflow 2 seems to be a complicated workflow, which fails more often. Maybe this behaviour is something the test person knows and which cannot be changed easily. In such a test environment the reliability of the whole system would be very low, even if it the system is quite stable and just one service endpoint is not functioning properly. To give the tester a better granularity for this indicator the reliability indicator could be introduced for every single workflow or for groups of workflows. This will be a future enhancement of SITT.

5.2 Test Daemon

Every test possibility introduced in 5 can be tested with SITT. The information for the tests are collected by the TAs and sent to the MA, which then inserts the information into the test database. The design of the database will be not covered in this paper. After the collection of all information the test daemon runs and analyses if the test succeeded or failed according to the test description in XML. Statistical information

like performance and reliability can be retrieved by the user whenever needed. There will be a possibility to retrieve the information with simple group functions, such as "getting the count of all messages sent by service endpoint A for one particular day".

6 Related Work

There is some research in the field of testing services, which is mainly focusing on testing Web services.AWSST (Automatic Web Services Testing Tool) [24] is a framework, which has the possibility to make load/stress testing and authorization testing. Coyote [22] tests services in a way that test scenarios are generated from the WSDL description of the Web service. The idea behind these concepts is more or less to invoke the services and to test the behaviour. In [23] a concept is presented how WSDL could be extended to perform black box testing. In [25] a new compile-time analysis that enables a testing methodology for white-box coverage testing of error recovery code (i.e., exception handlers) in Java web services, using compiler directed fault injection, is presented. These concepts can help to reduce the complexity of testing Web services very much and lead to a better deployment and robustness of services. In contrast SITT can be used to test if certain workflows are executed in the right order and has the advantage that the service endpoint can be any type of service adaptor, not just a Web service.

7 Conclusions

In this paper we presented an approach for automated testing of Service Oriented Architectures. To show the effectiveness of such an automatization, we develop a tool called SITT and expect to have a stable version until end of 2004. Nevertheless, our research showed that highly complex services are very hard to test and that testing consumes a lot of money and time. By automatizing these manual and repetitive tests a tool can assist the test team in a highly effective way. Automatic testing does not mean that there is no work for setting up tests and that testing can then be seen just as a simple process at the end of development. In fact setting up the test cases for a tool consumes also some time and needs experience of the test team, but once the test cases are in place they can be repeated whenever needed and the test results are much easier to interpret as they are in contrast to manual testing. As SITT is still in development there may be some slight changes in the XML Meta language, as we may find that some attributes are failing or that some elements do not provide the reasonable information needed.

8 Future Work

The next steps in our work are to finalize and stabilize our current prototype implementation of SITT and to investigate how such a tool increases the effectiveness of a development cycle and how it is accepted by a test team. Furthermore we will concentrate on setting up a monitoring system for SOAs based on SITT. Additionally,

we are researching how message ordering with messages of different priority can be tested reasonably. We are also working on how analysing the message itself can increase the usefulness of the test suite and we will introduce a feature finding messages belonging together by introducing algebraic functions to calculate a further MessageID on base of a still known MessageID. Besides this, we are focusing on generating valuable test cases easier, as this is now done manually by the test persons. Further we will introduce a GUI for creating test cases more comfortably. In our further research we will investigate if orchestration languages, such as BPEL, could be utilized for describing test scenarios for SITT. Besides this we will extend SITT with proxy functionality, so that testing for Web services can be done by analysing the messages, which are exchanged between the service endpoints.

9 References

1. http://expect.nist.gov/
2. Dean Kuo, Doug Palmer: "Automated Analysis of Java Message Service Providers". Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001.
3. Karim Baina; Boualem Benatallah; Fabio Casati; Farouk Toumani: "Model-Driven Web Service Development". Procs of CAiSE 2004. Riga, Latvia. June 2004.
4. Daniela Grigori, Fabio Casati, Malu Castellanos, Umesh Dayal, Ming-Chien Shan, Mehmet Sayal. Business Process Intelligence. Computers in Industry to appear.
5. Boualem Benatallah, Fabio Casati, Farouk Toumani, Rachid Hamadi. Conceptual Modeling of Web Service Conversations. Procs of CAiSE 2003. June 2003.
6. Tevfik Bultan, Xiang Fu, Richard Hull, Jianwen Su: „Conversation Specification: A New Approach to Design and Analysis of E-Service Composition". The Twelfth International World Wide Web Conference. Budapest, Hungary. 20-24 May 2003.
7. James E. Hanson, Prabir Nandi, Santosh Kumaran: "Conversation Support for Business Process Integration". Sixth International ENTERPRISE DISTRIBUTED OBJECT COMPUTING Conference (EDOC´02). Lausanne, Switzerland. September 17 - 20, 2002.
8. Olaf Henninger; Miao Lu; Hasan Ural: "Automatic Generation of Test Purposes for Testing Distributed Systems". Formal Approaches to Software Testing: Third International Workshop, FATES 2003. Montreal, Quebec, Canada. October 6th, 2003.
9. Vladimír Mařík, Luboš Král, Radek Mařík: "Software Testing & Diagnostics: Theory & Practice". SOFSEM 2000: Theory and Practice of Informatics: 27th Conference on Current Trends in Theory and Practice of Informatics. Milovy, Czech Republic. November/December 2000.
10. Vinod Suvarna, Mahesh Chandra: "Estimation Technique for Automated Testing". IBM Whitepaper.
11. James B. Michael, Bernard J. Bossuyt, Byron B. Snyder: "Metrics for Measuring the Effectiveness of Software-Testing Tools". 13th International Symposium on Software Reliability Engineering (ISSRE´02). Annapolis, Maryland. November 12-15,2002.
12. McKinsey & Company: "EAI – elementarer Treiber der künftigen Wettbewerbsposition". Management & Consulting 1 / 2001. online at: http://bto.mckinsey.de
13. Meta Group Deutschland GmbH: "e-business und Enterprise Application Integration: Der Schlüssel zum e-Erfolg". online at: http://www.metagroup.de

14. Jessica Chen: „On Using Static Analysis in Distributed System Testing“.Engineering Distributed Objects: Second International Workshop, EDO 2000. Davis, CA, USA. November 2000.
15. James S. Collofello, Zhen Yang, Derek Merrill, Ioana Rus, and John D. Tvedt: “Modeling Software Testing Processes”. submitted to the International Phoenix Conference on Computers and Communications(IPCCC'96), 1996.
16. David Saff, Michael D. Ernst: “An experimental evaluation of continous testing during development”. In ISSTA 2004, Proceedings of the 2004 International Symposium on Software Testing and Analysis, (Boston, MA, USA), July 12-14, 2004.
17. http://www.idc.com
18. Antionia Bertolino, Paola Inverardi, Henry Muccini: “Formal Methods in Testing Software Architecture”. Formal Methods for Software Architectures: Third International School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures, SFM 2003, Bertinoro, Italy, September 22-27, 2003.
19. Schahram Dustdar, Stephan Haslinger: “Service-oriented Modeling with UML – A Case Study”. Submitted to International Journal of Cooperative Information Systems (IJCIS) – “Special issue on Service Oriented Modeling”, 2004.
20. Martin Fowler: “Patterns of Enterprise Application Architecture”. Addison Wesley, 2003.
21. http://www.infosys.tuwien.ac.at/staff/sd/NODE_Appendix_for_SITT_V1.0.pdf
22. W. T. Tsai, Ray Paul, Weiwei Song, Zhibin Cao: “Coyote: An XML-Based Framework for Web Services Testing”. 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02). Tokyo, Japan. October 23-25, 2002.
23. W. T. Tsai, Ray Paul, Yamin Wang, Chun Fan, Dong Wang: **“**Extending WSDL to Facilitate Web Services Testing”. 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02). Tokyo, Japan. October 23-25, 2002.
24. Ying Li, Minglu Li, Jiadi Yu: “Web Services Testing, the Methodology, and the Implementation of the Automation-Testing Tool”. Grid and Cooperative Computing: Second International Workshop, GCC 2003. Shanhai, China. December 7-10. 2003.
25. Chen Fu, Barbara Ryder, Ana Milanova, and David Wannacott: "Testing of Java Web Services for Robustness". To appear in Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2004), July, 2004.