Service-oriented approach to integration testing in distributed systems

V.N.Fedotov

Software Engineering Department
Institute for System Programming
vnfedotov@acm.org

Abstract

Development of the service-oriented technologies has turned the market of integration platforms, becoming new challenge for IT experts worldwide. Key principles of SOA paradigm do not allow to apply the solutions tested on the client-server architecture. On the other hand, SOA offers its own set of tools to cope with majority of problems. In this paper it is shown how to effectively troubleshoot the distributed system by means of service-oriented approach.

1 Introduction

With development of communication technologies, business process automation becomes increasingly urgent task both for business, and for the state. Seen not so long ago just as a tool to reduce costs, now BPA is a key to survival in a changed world where internet and mobile services rule.

Majority of companies already has ERPs, CRMs, HRMs and many other applications managing their daily business. But do these applications automate company business processes? They are certainly providing some automation but it has nothing to do with business process, as they cannot interact with each other. So, BPA is only possible when there is a way to 'glue' applications together. In literature such way often called 'enterprise application integration' (EAI[1]).

There isn't a 'right' solution for application integration. As different integration concepts were developed, they didn't replace each other, instead they're competing until now. You still can find message-oriented middleware from early 90-s, integration brokers, and even CORBA, which is almost dead, because it didn't survive the competition with the newest trend - service-oriented architecture (SOA[2]).

While using many ideas from CORBA, SOA has taken completely different technological approach by using XML and J2EE to simplify development of integration components dramatically. In SOA these components are called 'services', as they mostly resemble web-services, but are arranged in a way defined by SOA guiding principles. These principles actually form SOA paradigm, as they distinct service-oriented architecture from bunch of web-services, just like OOP principles distinct object-oriented code from just some C++ code.

2 Enterprise as a Black Box

When thinking about how to make two applications interact with each other, you probably will come out with something like client-server approach, which is typical for Internet and various network applications.

Unfortunately, client-server architecture doesn't work in EAI as there is a tens of applications interconnected with each other in different ways by different protocols. In that case client-server approach requires each application to contain an adapter for every other application, causing serious flexibility and scalability issues.

These issues are the reason of using middleware solutions. Middleware platform are in the middle, obviously, of every interaction, mediating and routing messages between applications. For testing it means that instead of one interaction, you need to test two interactions, which isn't too bad. But SOA make matters worse, as it compose middleware platform from dozens of interconnected services. Now you need to test twenty or thirty interactions. As SOA solution becoming mature, its becoming more complex, with composition services and orchestrations, so testing efforts are only increasing.

Common solution to lower the efforts putted in integration testing is simple - don't do integration testing. Without advanced testing approach, this is the only way to cope with tight release schedule typical for SOA projects.

3 Test stubs as a solution

Of course there are better solutions. How to test integration in a distributed system - isn't a most recent question. It is also having a simple answer: instrument SUT in a such way so that interactions between its components became transparent. But there is actually quite a broad choice of technical approaches.

Logging is a most simple approach. Analysis of transactions journal provide all the necessary information about way that components interact. However logs can be pretty hard to reach, too big to read and it's impossible to analyze them automatically.

Testing tools traditionally offer a different solution - test stubs. Stubs form a virtual environment, surrounding each component and giving a possibility to inspect all external links of a component under test. However, stubs are way too invasive, as they are literally muffling external links, thus it is impossible to test entire business process. Also, test stubs are inaccessible from the test scripts, so all assertions must be contained within the test stubs themselves, thus test logic became sprayed over test environment. Because of that it is nearly impossible to support regression test sets in a virtual environment.

Various academical papers[3][4] propose instrumenting SUT by way of instrumenting its components to provide more information about interactions with other components and make that information accessible from test scripts. Unfortunately, such instrumentation requires a complete revamp of release process adopted by enterprise, thus making that approach inappropriate for most companies.

4 Alternative

As an alternative to the solutions described above we propose a new way to test application integration in a distributed system. Our solution offer similar approach, but entirely different technical implementation. In short it can be summarized as 'connect everything to ESB[5]'.

We propose to use ESB as a universal proxy for every interaction in a SUT, thus providing us with capability to monitor and control these interactions. Using of ESB also grants access for testing tools through JMX and event queues, which provide us a way to automate analysis of interactions inside the SUT and even modify messages on the fly to reach broader test coverage.

These are the key advantages of proposed solution:

- ESB provides transparency for all interactions between SUT components;
- test assertions are located in one place;
- tests can be easily automated;
- the minimum quantity of tests covers a maximum quantity of SUT components;
- thanks to a uniform configuration provided by ESB, the test environment becomes more flexible and controlled;
- testing process is focused on business process, so there is a minimum risks of occurrence of late integration defects.

5 Conclusion

In this paper we've shown a way to use service-oriented technologies for creation of flexible test environment, which allows to simplify integration testing of distributed systems by adding necessary transparency to interactions between components of SUT.

On the basis of the proposed approach it is planned to create the completed methodology of testing distributed systems based on service-oriented architecture, having presented original techniques of construction of test scenarios and carrying out a performance testing.

References

[1] Gregor Hohpe, Bobby Woolf. *Enterprise Integration Patterns.*

[2] OASIS Reference Model for Service Oriented Architecture 1.0
http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

[3] Cesare Bartolini, Antonia Bertolino, Sebastian Elbaum, Eda Marchetti. *Whitening SOA testing.*

[4] Youngkon Lee. *Double layered SOA test architecture based on BPA - simulation event.*

[5] David Chappell. *Enterprise Service Bus.*