elpub2002 Proceedings — J. A. Carvalho et. al. (Eds.) — VWF Berlin, 2002

Continuous On-Line Validation of Web Services

André Rocha[1], Giuseppe Valetto[2], Elio Paschetta[2], and Seppo Heikkinen[3]

[1] Portugal Telecom Inovação, Porto, Portugal
ACRocha@ptinovacao.pt
[2] Telecom Italia Lab, Turin, Italy
{Giuseppe.Valetto, Elio.Paschetta}@tilab.com
[3] Elisa Communications, Tampere, Finland
Seppo.Heikkinen@elisa.fi

Abstract. The main objectives of the work herein presented were to define and build a platform for workflow-based continuous on-line validation of systems and to study its application to an e-business marketplace website implemented with the emerging Web Services technology, by many considered the new revolution in the world wide web context. This work took place in the form of one of the case studies developed in the Eurescom project P1108: "Workflow-based on-line validation of complex component based Internet services".

1 Introduction

The main purpose of Eurescom project P1108 – "Workflow-based on-line validation of complex component based Internet services" – was to study the use of continuous on-line validation techniques with deployed component-based services, as opposed to the use of traditional off-line, pre-deployment validation methods. Several key motivating factors were identified: component-based techniques are the key to the development and provisioning of services; validation is the key to quality of services; high-quality service provisioning is the key to medium and long-term competitive advantage.

Part of this work consisted in defining and building (by using mainly freely-available open-source tools) a platform for continuous on-line validation of systems (COLV in the remainder), which was then applied to a set of different target systems: an active firewall, a multi-channel instant messenger system and an e-business marketplace implemented with the Web Services technology. This paper only reports the lessons learnt from the latter case study.

A travel agency service, accessible through a web interface, is the scenario for the marketplace site. Flight and hotel reservations are the two kinds of functionality made available by the travel agency (other possible types of services would be for instance car rental or cruise reservation). The travel agency component works as a marketplace, in the sense that, instead of implementing flight and hotel reservations on its own, it requests them to other pre-existing and specialised service providers, mediating and combining them to come up with the composed product requested by the user. Therefore, specific web services for flight and hotel reservation are also part of the system. In the remaining sections of this paper several issues are discussed. Section 2 addresses the Web Services technology itself, the proposed architecture for a COLV platform and the reasons why the latter is relevant to the former. Section 3 describes how the COLV infrastructure was applied to the travel agency prototype mentioned above and evaluates the outcome of such a case study (how easy it was, what benefits were achieved, etc.). The final conclusions are reported in section 4.

2 Technological Background and Motivation

This section introduces the relevant technological concepts involved. Some brief introduction to Web Services is given in the first subsection, followed by a presentation of the concepts and technologies behind the COLV platform (section 2.2). The last part deals with the motivation and benefits of combining these two technologies.

2.1 Web Services

The term Web Services refers to a group of standards, which define a communication framework between Internet applications, which allows them to find other applications and services and use them in a standard, well-defined way. As such, Web Services concept is supposed to make application integration and composition of new services easier and more dynamic. To promote widespread acceptance, the concept is based on open standards like SOAP, XML, WSDL and UDDI. Implementation of those standards as a set of APIs contribute to hiding the implementation details of a service, allowing its remote invocation independently from any hardware, software or programming language concerns. This allows Web Services applications to be loosely coupled, component-oriented and cross-technology implementations [1].

Figure 1 shows the high-level model of Web Services. In the heart of things is UDDI (Universal Description, Discovery and Integration) that can be seen as a repository framework where different parties can publish information about themselves and the descriptions for the services they offer [2]. This information is then

publicly searchable and can be used to find suitable business partners and services. So, in simple terms, UDDI can be viewed as analogous to phonebook "yellow pages". WSDL (Web Services Description Language) is an XML-based language for describing service features which include functions of service, location of service and method signatures for invoking the service through its programmatic interface [3]. So, in essence, WSDL is used to define and publish the interface with which the communication is to be taking place. The actual messaging between different parties is achieved with the use of SOAP (Simple Object Access Protocol) which is an XML-based messaging protocol [4]. Simply put, SOAP acts as an enabler of remote procedure calls for the specified interface. SOAP could be used with a variety of underlying network protocols but at the moment HTTP is the preferred one.

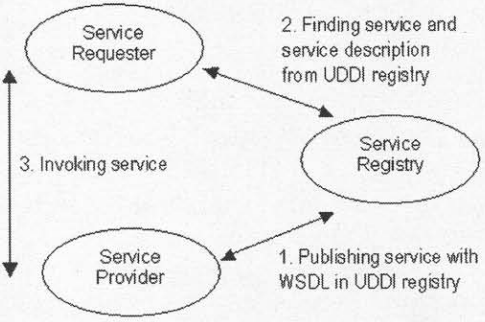

Fig. 1. *High level model of Web Services*

2.2 Continuous On-Line Validation

Continuous On-Line Validation (see for example [10] for further details) has the goal of verifying and keeping under control at all times critical functional and extra-functional parameters of a running software application or service (the COLV target), by putting in place a completely automated closed control loop that is superimposed on the target system and remains orthogonal to it.

COLV may be exploited in a variety of contexts, ranging from application monitoring, to on-line testing, to service optimisation, through on-the-fly (re)configuration of components and whole services, to controlled application roll out and redeployment, to on-demand service assembly and coordination, and more.

COLV relates to both application management and application coordination, but substantially departs from the current state of the art in those disciplines: with respect to the former, because of its stress on active control and automation thereof; with respect to the latter because it advocates decoupling and externalising the

coordination features rather than embedding them in any form within the target application.

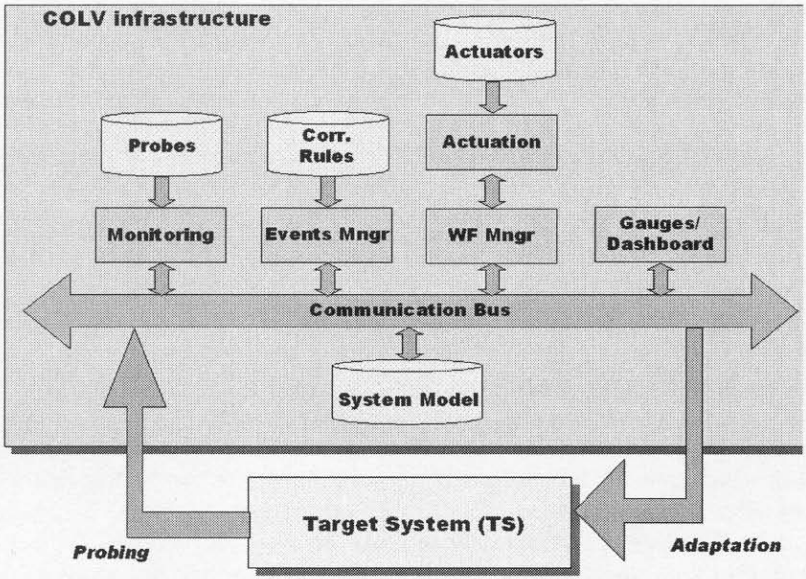

Fig. 2. *COLV platform high level architecture*

The conceptual architecture for a general-purpose COLV platform that we present is a derivation of [11] and includes a number of cooperating, loosely coupled subsystems (see Figure 2). In particular: a *monitoring* subsystem that collects and issues information about the behaviour of the running target system, via a set of probes which instrument the target code; an *events manager* subsystem, which semantically interprets and correlates the information collected by multiple probes on multiple components, and outputs high-level notifications, thus providing *gauges* for the running target system; a *data communication and manipulation* subsystem, that is, a bus that transports the information among the other subsystems, while massaging and formatting uniformly the data potentially coming from multiple heterogeneous probing sources; a dynamic *modelling* subsystem capturing and making available knowledge about the target system, its state and its COLV-relevant characteristics; an *actuation* subsystem, which can intervene on the target system and components thereof, adapting or repairing its architecture, configuration and behaviour; a *decision and coordination* subsystem (a workflow manager suitable for orchestrating system-to-system interactions) in charge of issuing and orchestrating actuators according to codified adaptation processes that predicate on the system

model; a *report and control dashboard* that allows to visualise gauges and to manage the COLV platform.

2.3 Relevance of COLV to Web Services

Even though Web Services provide an attractive framework for creating a wide range of services, they still lack standardised management functions, which are obviously vital in the business domain. The service itself becomes more compelling to the customers if some assurance about the quality of service can be given. In this sense, customers might find it more attractive and reliable to use services that are under continuous validation.

Crucial functional parameters like availability and response capacity can be improved by automatically monitoring certain sensitive points of the system and, when necessary, taking corrective measures upon their values.

In such a competitive and global environment like Web Services, in which service providers and mediators (such as marketplaces) dynamically discover the services they need and use them to compose a final product, quality of service (QoS) is something which greatly depends on third parties. In order to assure its own QoS, one must also look after the quality of the whole service chain. Logically, it is clearly unfeasible to try to manually monitor all the web services in that chain; COLV can therefore play an important role, since it allows to automatically notice any problems in a single service block and take automated actions to prevent them from further degrading the overall QoS.

With COLV, the management staff can also have a clear and uniform view in the dashboard of the whole service chain, enabling to recognise what is happening in the system. This decreases the response time in case something requires human intervention and is far better than having the customer reporting something wrong in the system and then trying to figure out what might be the point of failure.

COLV can be effectively used for other kind of management purposes as well. Statistics about the service usage can give input to the business planning processes to enhance the cost-effectiveness and the flow of transactions. It can also give valuable information about how customers use the service and what sort of errors they make. This could be used for usability enhancements.

3 Application of COLV to a Web Services System

In this section, the application of the COLV platform addressed in 2.2 to the travel agency web service scenario is discussed, firstly by describing how that task was accomplished and then by evaluating the achieved results.

3.1 Description

Before specifically getting to the integration of the COLV platform with the travel agency system issue, it is convenient to better describe the latter one. Figure 3 represents its architecture.

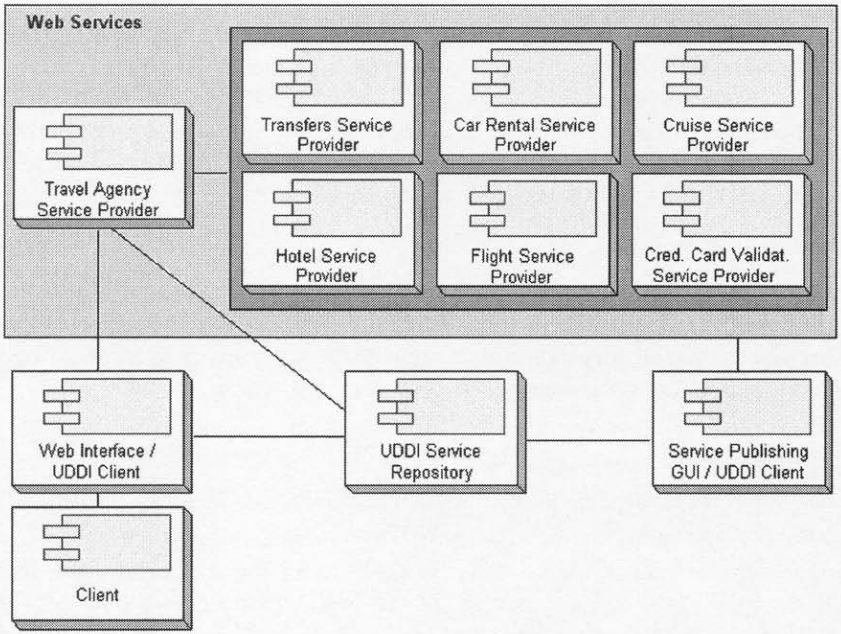

Fig. 3. *Travel agency system architecture*

As already mentioned in section 1, the travel agency web service does not implement itself the functionality that will be provided to the end user, but instead sub-hires other specialised providers in order to compose its service: hotel and flight reservation, car rental and so on. For this reason, several types of web services are represented in Figure 3. The travel agency web service is accessed via a web interface, where the user may specify his request. Note that there is absolutely no contact between the web interface (and thus the end user) and the so-called specialised web services; only with the travel agency.

As an example of the system operation, imagine that the travel agency receives a request for hotel and flight reservation, with a given set of parameters: dates, location, etc. The workflow in this situation would be the following: the travel agency web service would query the UDDI server for registered flight and hotel reservation providers, then contact each one of them to know about their availability to

provide the specified service and finally send the various possibilities found to the web interface.

The component to which the COLV platform was applied is the travel agency service provider. In the remainder, one will refer to it as the target system (TS).

Instrumentation of Target System. The first step in the integration process was to enable the TS to inject raw information about itself into the communication bus of the COLV platform. Among the various ways of performing this, source code instrumentation was the one chosen: a probing technology, Active Interface Development Environment – AIDE [6], was used to automatically add instrumentation statements at the entrance and exit points of each method of some of the TS classes, after what these classes were re-compiled.

Probes. Having the necessary classes instrumented, some code was necessary to define which probing points would actually be inserting information into the platform and in which form they would do it. A wide array of probes, covering several different issues of a Web Services system, was implemented:

- user login/logout notification;
- success/failure in finding all the providers needed to compose service;
- number of providers found in the UDDI repository, per service type;
- for each contacted service provider: availability of the communication link and WSDL description file, correctness of the WSDL, occurrence of failure on invocation via SOAP, response time;
- unavailability of the UDDI server;
- success/failure on starting/stopping a Tomcat server [9].

This set of probes enables the system, for instance, to keep a record of the number of currently logged in users and to continuously collect valuable information about the specialised web services (flight, hotel, car rental, etc.). These probes provide all the basic information about the TS, which then feeds the workflow engine and the dashboard for activity visualisation.

Also, some pro-active functionality was provided to the COLV system, by developing a periodic probe which, executed with fixed time intervals, posts a "normal" search request to the travel agency web service (flight and hotel reservation, for instance), as if some user had manually done it in the web interface. This way, the travel agency is triggered to contact the necessary service providers and the COLV platform has the chance to detect any of the above-mentioned failures (unavailability of UDDI server, WSDL not found, etc.) in that process.

Gauges. Based on the information coming from the referred probes, some higher-level measures were derived:

- number of users currently logged in the system (already mentioned);
- number of search/purchase requests (absolute and average per time unit);
- rate of service providing failures of the specialised services (overall and per service type).

Among these, only the first gauge was actually used in the on-line validation workflow; the remaining are business/technical measures computed for visualisation purposes, only.

Actuators. Actuators provide means for the COLV platform to intervene in the TS. The actuators developed were the following:

- starter/stopper of Tomcat servers;
- reconfigurator of UDDI server (changes the selected UDDI server);
- database client for collecting web services failures and response times;
- launcher of the periodic probe.

Workflow. Based on the implemented probes, gauges and actuators, it was possible to develop a set of workflow rules, which were intended to achieve one of the main goals of the COLV platform: to correct failure situations, repair and adapt the system on the fly and thus improve the quality of service of the running TS. The variety of application domains already seen in the probes and actuators list mainly, is obviously reflected here. The workflow engine was programmed to:

- launch and shut down Tomcat instances serving the travel agency web service and web interface, according to the number of logged in users;
- switch from the currently selected UDDI server to another, when the former is unavailable;
- collect web services failures and response times information in a database;
- schedule the triggering of the periodic probe.

3.2 Evaluation of Results

This section highlights several features of the evaluation process carried out on the results of our case study.

Optimisation of the Target System. Optimisation of the TS was achieved in various different ways.

Firstly, the existence of several Tomcat servers, deploying the travel agency web service and web interface, and intended to be launched and stopped by the COLV platform according to the number of logged in users. Although yet not actually configured due to time constraints, a simple Apache web server on top of these would have been sufficient to balance the load between them and therefore improve the response time of the system.

On a different domain, the switching between the available UDDI servers when the currently selected is unreachable, obviously improves the availability time of the system: the UDDI server is a vital component and without it the travel agency is simply unable to provide any kind of functionality to the end user.

The information about the specialised web service providers continuously collected over time, either when a user makes a request or when the periodic probe is executed, enables the travel agency web service to previously evaluate the quality of each contacted service provider (i.e., based on its past history, extrapolate what level of QoS could be expected from a given service provider). This classification may then be shown in the web interface or used in some (semi-) automatic providers filtering process, prior to displaying the set of possibilities in the interface.

Learning Curve of the Platform Components. Some of the platform components used have a high peak at the beginning of their learning curve, which then generally tends to reasonably low values. This is the case of the tool used for TS instrumentation, AIDE [6], and of the workflow engine, Cougaar [5]. Also, some problems have arisen due to the "under development" state of the event processing tool, Xues [7]. Nevertheless, its use was quite straightforward.

Performance Impact. The performance tests carried out revealed that the emission of probes in the COLV platform communication bus represents an overhead of about 16% in the travel agency response times. This number is thought to be quite acceptable, given the type of TS and, specially, the human nature of the systems which make use of it. ApacheBench [8] was the tool used for these tests.

Source Lines of Code. Very few lines of source code had to be written in order to insert the monitoring and actuating points into the TS: around 3% of its total lines of code for the former and 7% for the latter.

4 Conclusions

The three main subjects of this paper consist in a brief technological background on Web Services, the major principles, components and proposed architecture for

Continuous On-Line Validation and a report of how, and with which results, the COLV platform was applied to a Web Services case study.

One has effectively assembled and put to use a prototype of an infrastructure for continuous on-line validation, which provides enough features and flexibility to enable its application to many different types of target systems. This platform remains orthogonal to the system and implements a completely automated closed control loop upon it.

Furthermore, the case study described in this paper made evident that crucial functional parameters of a Web Services application, like availability, response capacity and overall quality of service, can be put under control and improved by using a COLV platform to automatically monitor and actuate certain sensitive points of the system: UDDI server availability, response times and failures occurred with third-party web services, number of logged in users, etc. All this was achieved at a reasonably low cost and effort: probes insertion represents an overhead of 16% in the response times of the travel agency and, in order to deploy these monitoring points, about 3% of the total lines of code of the target system were sufficient. However, some of the platform components initially demand significant learning effort.

As a bottom line, we regard the use of COLV techniques as both relevant and promising in achieving manageability, QoS assurance and – as a consequence – business advantages in the emerging Web Services arena.

Acknowledgements

One would like to thank all the participants in Eurescom project P1108, in particular Vesa Huotari for his insights onto this paper.

References

1. Kreger, H., "Web Services Conceptual Architecture", 2001. `http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf`
2. Universal Description, Discovery and Integration technical references. `http://www.uddi.org/specification.html`
3. Web Services Description Language (WSDL) 1.1 – W3C Note. `http://www.w3c.org/TR/wsdl`
4. Simple Object Access Protocol (SOAP) 1.1 – W3C Note. `http://www.w3c.org/TR/SOAP`
5. Cognitive Agent Architecture – Cougaar. `http://www.cougaar.org/`
6. Active Interface Development Environment – AIDE. `http://www.cs.wpi.edu/~heineman/dasada/`
7. XML Universal Event Service – XUES. `http://www.psl.cs.columbia.edu/xues/`
8. ApacheBench (Benchmarking Tool). `http://www.apache.org/`

9. The Jakarta Project – Apache Tomcat Server. http://jakarta.apache.org/tomcat/
10. Dasada Home Page. http://www.if.afrl.af.mil/tech/programs/dasada/
11. Kaiser, G., Gross, P., Kc, G., Parekh, J., and Valetto, G., "An Approach to Autonomizing Legacy Systems", in Workshop on Self-Healing, Adaptive and Self-MANaged Systems, June 2002.