ACTA TECHNICA CORVINIENSIS – Bulletin of Engineering
Tome V (Year 2012). FASCICULE 1 [January–March]. ISSN 2067-3809

[1.] Dušan OKANOVIĆ, [2.] Milan VIDAKOVIĆ, [3.] Zora KONJOVIĆ

SERVICE LEVEL AGREEMENT XML SCHEMA FOR SOFTWARE QUALITY ASSURANCE

[1.] FACULTY OF TECHNICAL SCIENCES, NOVI SAD, SERBIA

ABSTRACT: *In order to assure that the software service levels required by the service consumer are met by the service provider, constant monitoring and verification of the software is required. We propose a new XML schema for defining service level parameters. In documents based on this schema we define parts of application to be monitored, which metric is going to be used and what are expected values. We present the DProf tool for constant monitoring of software performance. The system is implemented in Java, but, with minor modifications, it can be used for .NET applications.*
KEYWORDS: *continuous software monitoring, service management, service level agreement*

INTRODUCTION - SERVICE LEVEL AGREEMENTS

To determine whether the quality of service and service level agreements are on a satisfactory level, it is necessary to monitor software in its operational stage and environment. While new, previously unknown, errors can show up, it is a common phenomenon for software performance and quality of service to degrade over time [16], too. Software testing, debugging, and profiling in development environments hardly allow to detect errors and unpredicted events that can occur after the software is deployed and used in its production environment.

Service level agreement (abr. SLA) [1] is usually a part of an agreement between service consumer and service provider. Based on this document, service provider is obliged not only to provide service, but also to provide certain quality level of the service, too. SLAs specify permanent monitoring and verification of IT service levels. It specifies metrics to be used, service management and reactions to agreement breaches. It also contains time constraints, e.g. period of validity of contract and frequency of measurements.

The life cycle of SLA [13] begins with the agreement definition. It is then passed to the service provider. Within service provider organization, duties are assigned, and monitoring phase can begin. During this phase, SLA parameters are monitored and data is gathered. This data is analyzed and used for 1) detection of violation of SLA and 2) service level improvement. After data analysis, SLA is revised, and the whole process continues from the beginning. Graphic representation of this process' cycle is shown in Figure 1.

To determine how software behaves under production workload, continuous monitoring of that software is a valuable option. Continuous monitoring of software is a technique that provides a picture of the dynamic behavior of software under real usage, but often results in a large amount of data. In the process of the analysis, the obtained data can be used to reconstruct architectural models and perform their visualization (e.g., employing UML).

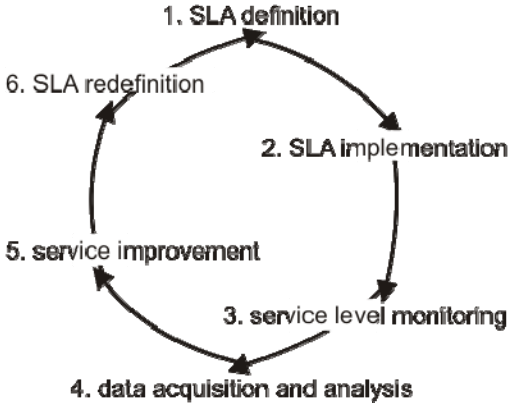

Figure 1. SLA lifecycle

In the development phase, software developers usually utilize tools such as debuggers and profilers. Although they provide a picture of the software behavior, they typically induce a significant performance overhead – something which is unacceptable for production use.

In order to check if software performance is in compliance with SLA, we have developed the DProf system. It performs continuous monitoring of software and analyses gathered data. Based on this data and our DProfSLA XML schema, DProf can find which part of application is not in accordance with SLA. This reduces time needed by developers to identify the source of the problem and to solve it. DProfSLA schema based documents are used to define required service-levels in various metrics. System is extensible so that users can define their own metrics and implement measuring techniques.

The rest of the paper is structured as follows. Chapter 2 presents related work in the field. In chapter 3 we present DProfSLA XML schema, while in chapter 4 we give short description of our DProf system. Fifth

chapter gives an example of how DProf can be used, while chapter 6 provides conclusion and guidelines for future work.

RELATED WORK - EXISTING SLA STANDARDS

Related work focuses on existing standards for SLA documents definition and monitoring tools.

In order to automate service level management process, SLAs must be defined in machine-readable format. As shown by Tebbani et al. [15], there are only few formal SLA specification languages. SLAs are usually written in some spoken language. Authors propose GXLA – XML specification for GSLA (**G**eneralized **S**ervice **L**evel **A**greement). GSLA is defined by authors as a contract signed between two or more parties, which is designed to create a measurable common understanding of each party role. A role presents the set of rules which define the minimal service level expectations and obligations the party has. GXLA is a XML schema which implements GSLA information model. GXLA is composed of following sections: schedule (temporal parameters of contract), party (models involved parties), service package (an abstraction used to describe services) and role (as described). Creation and use of GXLA allows automation of service-management process.

WSLA described in [6] is XML based and it is used to specify service levels for web services. WSLA document defines interested parties, metrics, measuring techniques, responsibilities and courses of action. Authors state that every SLA (and WSLA, too) contain 1) information regarding agreeing parties and their roles, 2) SLA parameters and measurement specification and 3) obligations for each party.

Paschke et al [11] performed categorization scheme for SLA metrics with the goal to support the design and implementation of automatable SLAs.

Standard elements of each SLA are identified and shown in: technical (service descriptions, service objects, metrics and actions), organizational (roles, monitoring parameters, reporting and change management) and legal (legal obligations, payment, additional rights, etc.). Authors categorized service metrics in accordance with standard IT objects: hardware, software, network, help desk and storage. SLAs are grouped according to their intended purpose, scope of application or versatility (using categorization by Binder [2]).

According to this categorization, DProfSLA documents are operation level documents (by intended purpose) to be used in-house (by scope of application). By versatility categorization, they belong to standard agreements. The schema provides subset of elements defined by already existing GXLA or WSLA, and documents can be translated into these schemas using XSLT.

SOFTWARE MONITORING

Study shown in [12] shows that, while service levels and performance of applications are of critical importance in practice, application level monitoring tools are rarely used.

Java application monitoring tools are usually developed using either JVMTI/JVMPI [4, 5] or aspect-oriented programming (AOP) [7].

JVMTI and JVMPI APIs require knowledge of C/C++ in addition to Java, and also yield significant overhead [16].

COMPASS JEEM [10] can be used to monitor JEE applications, but every application layer needs different set of probes. Tools developed by Briand et al. [3] can be used only for UML diagram reconstruction, and it cannot be used for monitoring of web-services.

There are also commercial application monitoring tools, such as DynaTrace and JXInsight.

AOP is used for instrumentation of code. Separation of concerns allows for monitoring code to be separated from application code. There are several monitoring tools based on AOP. The Kieker framework [16] is a framework for continuous monitoring and analysis of all types of Java applications. It uses aspects to define and implement monitoring probes. The HotWave framework [17] tool allows run-time reweaving of aspects and creation of adaptive monitoring scenarios, but it is still in development phase.

The DProf system presented in this work is based on the Kieker framework [1] and the JMX technology [14]. It can be used for adaptive and reconfigurable continuous monitoring of JEE applications, as presented in this paper. Use of Kieker grants low overhead, and separation of monitoring code from application code by using the AOP. JMX is used for controlling of monitoring process at run-time.

Together with DProfSLA schema, DProf system can be used to monitor how SLA is executed and where problems occur.

DProfSLA Xml SCHEMA

Monitoring process goals are defined using a special XML schema – DProfSLA schema. Schema is specified in accordance with categorizations and existing schemas shown in related work.

Root element of this schema is shown in Figure 2.

The root element (DProfSLA) has three sub elements:

- Parties (parties in the agreement),
- Trace (call-traces to be monitored) and
- Timing (time constraints of this agreement).

The Parties element represents interested parties in the agreement. This element is presented in Figure 3.

The Parties element has two sub elements: Provider (representing service provider) and Consumer

(representing service consumer). Both of these sub elements contain contact data regarding service provider and service consumer, respectively – i.e. interested parties in this agreement. Each sub element is represented using the OrganizationType (Figure 4) complex type.

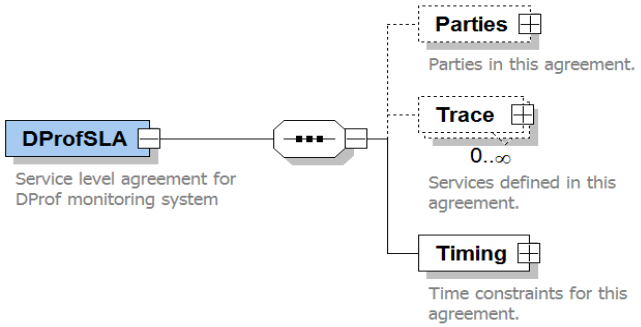

Figure 2 Root element of the DProfSLA schema

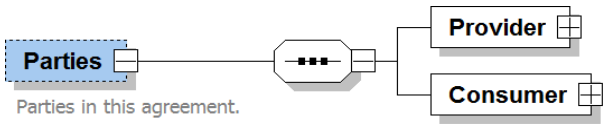

Figure 3. Parties element in the DProfSLA schema

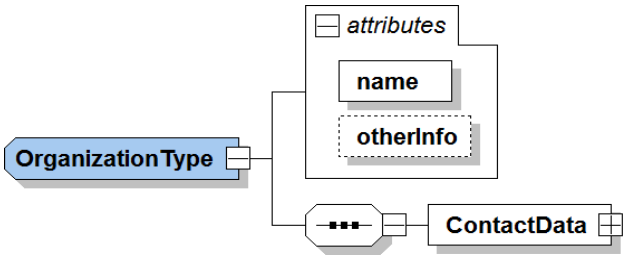

Figure 4. OrganizationType complex type defined in the DProfSLA schema

The OrganizationType element contains the following attributes:

- *name (organization name) and*
- *otherInfo (some other information regarding that organization).*

Contact information for that organization is stated in the ContactData sub element which is presented using the ContactDataType (Figure 5) complex type.

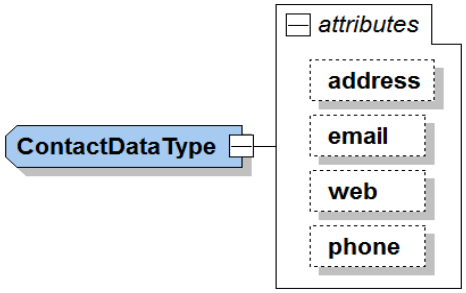

Figure 5. ContactType complex type defined in the DProfSLA schema

ContactDataType contains (optional) attributes for address, e-mail address, web address and contact phone for each interested party in the agreement.

The Trace element (Figure 6) represents performance information for one call trace. It is of the TraceType complex type.

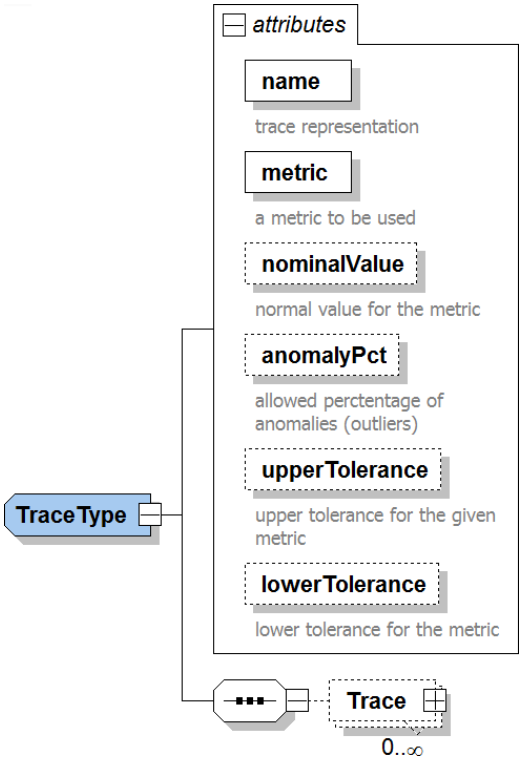

Figure 6. TraceType complex type defined in the DProfSLA schema

The Trace element has two mandatory attributes. The name attribute is used to specify a part of application to be monitored. String representation of a call tree is used for this. The metric attribute specifies which metric is used, i.e. which aspect of application performance is going to be monitored. Sub elements of the Trace element can be other Trace elements, e.g. methods that are invoked from other (parent) method, described in parent Trace element.

Furthermore, there are four optional attributes for specification of expected performance values in designated metric. The nominalValue represents expected average value, while the upperTolerance and lowerTolerance are maximal and minimal average values in designated metric, respectively. The anomalyPct is used to define allowed number of extreme values in obtained results.

The Timing element (Figure 7) is used to specify time constraints for this agreement. Sub elements StartTime and EndTime define period to which this document applies. Both times are in milliseconds (XML schema long values), starting from midnight, January 1, 1970 UTC (as in Java specification).The SamplingPeriod element denotes time (in milliseconds, long values) between two analyses of obtained results.

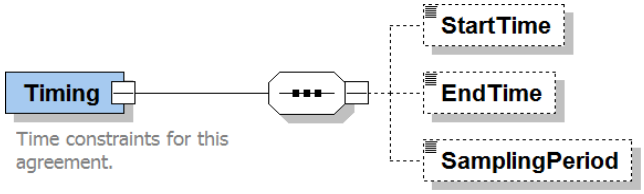

Figure 7. Timing element defined in the DProfSLA schema

DProfSLA MONITORING SYSTEM

In order to continuously monitor software applications we have developed the DProf monitoring system. It is mainly designed for continuous monitoring of JEE applications. With minor modifications it can be used for applications developed for other platforms.

This system is based on Kieker framework for continuous monitoring and analysis of software systems. We have developed additional components in order to allow changing of monitoring parameters while the application is still running.

The Kieker framework consists of the Kieker.Monitoring and the Kieker.Analysis components. The Kieker.Monitoring component collects and stores monitoring data. The Kieker.Analysis component performs analysis and visualization of this monitoring data.

The component diagram of Kieker framework is shown in Figure 8.

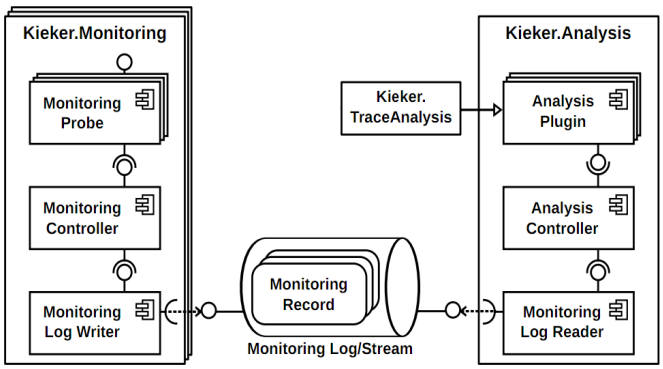

Figure 8. Component diagram of DProf monitoring system

The Kieker.Monitoring component is executed on the same computer where the monitored application is being run. This component collects data during the execution of the monitored applications. The Monitoring Probe component is a software sensor that is inserted into the observed application and takes various measurements. For example, Kieker includes probes to monitor control-flow and timing information of method executions. Probes are most commonly implemented using AOP aspects, and additional probes can be added to support different measurements (e.g. for adding support for new metrics). Monitoring Log Writers store the collected data, in the form of Monitoring Records, in a Monitoring Log. The framework is distributed with Monitoring Log Writers that can store Monitoring Records in file systems, databases, or JMS queues. Additionally, users can implement and use their own writers. The Monitoring Controller component controls the work of this part of the framework.

The data in the Monitoring Log is analyzed by the Kieker.Analysis component. A Monitoring Log Reader reads records from the Monitoring Log and forwards them to Analysis Plugins. Analysis Plugins may, for example, analyze and visualize gathered data. Control of all components in this part of the Kieker framework is performed by the Analysis Controller component.

The DProf system uses Kieker's infrastructure for data acquisition, but with some additional components. Architecture of DProf system and its connection to Kieker is shown on Figure 9.

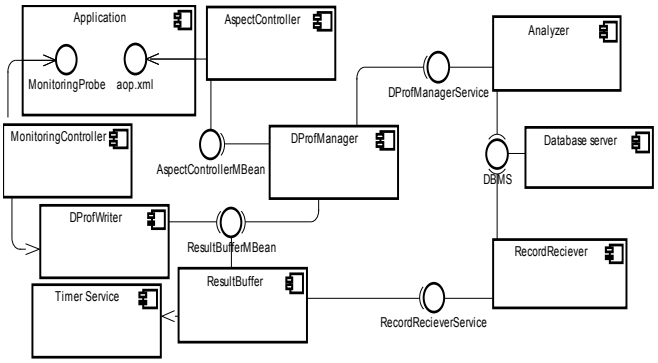

Figure 9. Component diagram of DProf monitoring system

The DProfWriter is the new Monitoring Log Writer. It sends Monitoring Records to the ResultBuffer component. The ResultBuffer sends data (periodically or on demand) to the RecordReceiver component, which, in turn, stores data into the database. This combination of ResultBuffer, RecordReceiver and database plays the role of the Monitoring Log.

Received data is periodically analyzed by the Analyzer component. The Analyzer can create new monitoring parameters (based on data analysis) and send these parameters to the DprofManager component. The DProfManager component passes these parameters to the ResultBuffer component (if the command requires change in data sending period) or to the AspectController component (if the command requires change in aspects or join points).

While using the DProfManager and these additional components we can change monitoring parameters at run-time. This allows us to reduce monitoring overhead by shutting off of monitoring in some parts of software, and obtain more accurate results. Setting of new parameters can be performed either manually, by a person in charge or by the Analyzer component. The Analyzer component, provided with document based on the DProfSLA XML schema, can check if service levels read from gathered data, are not in accordance with SLA and which part of the software causes this.

More detailed information about DProf system and in depth explanation of its architecture can be found in [8].

Since the RecordReceiver component is designed as a web-service, this component can be used for receiving monitoring records from application developed for platforms other then Java/Java EE. In order to use this system with some other platform, such as .NET, all we need is Kieker and DProfManager implementation in

.NET. Although it seems complicated, using existing AOP and JMX implementations for .NET, this can be reduced to rewriting required components in corresponding programming language.

MONITORING AND EVALUATION OF A SAMPLE APPLICATION

The Case study of our solution will be described on the JEE application shown in [9]. It is a simple software configuration management (SCM) application, based on EJB and JAX-WS. The DProf was configured to monitor memory usage during execution of a method that creates organizations (OrganizationFacade.createOrganization(...)) and methods invoked from this method (OrganisationFacade.checkName(...) and City.getId(...) methods). Maximal values for memory usage during executions of these methods are given in the DProfSLA document. Measurement of memory usage in monitoring probes was performed using JMX platform MemoryMXBean.

The analysis of the obtained data will be performed every 12 hours (43200000ms). First, only createOrganization() method is monitored and then, if there is deviation from values in DProfSLA, only methods invoked from this one are monitored. If there is a deviation from SLA values in one of these methods, that particular method needs to be reengineered. If there's no problem with any of them, parent method – createOrganization() – needs reengineering.

Classes from kieker.*, java.* and javax.* packages are not monitored – we only look for problems in this application classes.

Listing 1. shows a part of DProfSLA document.

```
<DProfSLA>
    <Parties>
        <Provider name="...">...</Provider>
        <Consumer name="...">...</Consumer>
    </Parties>
    <Trace metric="memory"
        name="[{OrganisationFacade.createOrganisation,[{
        OrganisationFacade.checkOrgName,[]},
        {City.getId,[]}]" upperTolerance="450000000">
        <Trace metric="memory"
            name="[{Organisation.checkOrgName,[]}]"
            upperTolerance="35000000"/>
        <Trace metric="memory" name="[{City.getId,[]}]"
            upperTolerance="40000000"/>
    </Trace>
    <Timing>
        <SamplingPeriod>43200000</SamplingPeriod>
    </Timing>
</DProfSLA>
```

Listing 1. DProfSLA document for this example

Obtained results were analyzed by the Analyzer, and they show increased memory consumption during the execution of the createOrganization(...) method (averagely 404519341B ≈ 385MB – Figure 10).

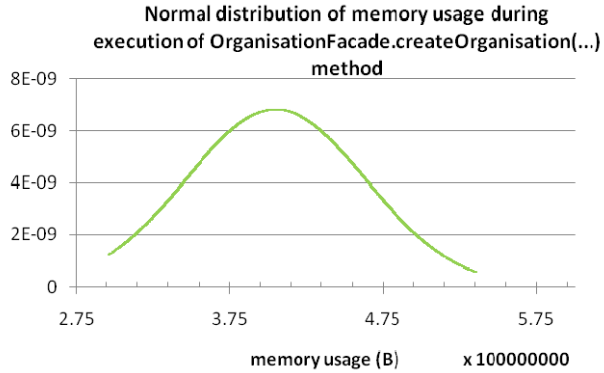

Figure 10. Memory usage during execution of OrganizationFacade.createOrganization(...) method

To find the source of the problem, the Analyzer component changed monitoring parameters and disabled monitoring of createOrganization(...) method. Monitoring of methods invoked from this method was turned on.

New set of data is shown on Figure 11.

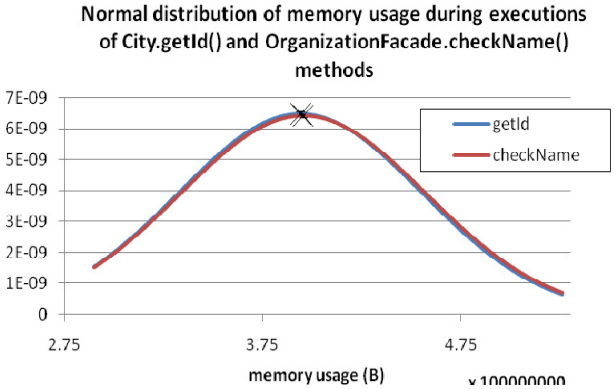

Figure 11. Memory usage during execution of OrganizationFacade.checkName(...) and City.getId() methods

Analysis of gathered data, twelve hours after previous analysis, showed that checkName(...) method consumes more memory then expected by SLA (averagely 394070676B ≈ 375MB).

Based on these results, it can be said that checkName (...) method requires refactoring in order to be optimized and in accordance to the SLA.

CONCLUSIONS

In this paper we presented a XML schema for creating SLA documents and extensible system for continuous monitoring of applications and automatic evaluation of software against expected values, defined in SLA – DProf. Using this system we can search for problems in honoring an agreement between service provider and consumer.

The system can gather data on application execution, compare these results with expected results and find which part of application causes deviations. Expected values are defined in a document based on DProfSLA XML schema. The schema is designed with existing SLA schema standards (such as GXLA and WSLA) and with categorizations of these schemas in mind. Its main use is for standard intra-organizational

agreements, but it can be used for inter-organizational agreements, too. The system supports various metrics and additional metrics can be added as needed.

As a proof-of-concept, the DProf system was used for monitoring of memory usage of one SCM application based on EJB and web services technologies.

Future work on this system will focus on implementation of the DProf Analyzer component as Kieker.TraceAnalysis module and improvement of integration of the DProf component into the Kieker distribution. We will also work on extending of the system with additional monitoring probes for different and more complex measurements.

ACKNOWLEDGEMENT

The research presented in this paper was supported by the Ministry of Science and Technological Development of the Republic of Serbia, grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models.

REFERENCES

[1] BENYON, R.: Service Agreements: A Management Guide, Van Haren Publishing, Netherlands, 2006.

[2] BINDER, U.: Ehevertrag für IT Dienstleistungen, Infoweek 34(4), 2001.

[3] BRIAND, L. C., LABICHE, Y., LEDUC, J.: Toward the reverse engineering of UML sequence diagrams for distributed Java software, IEEE Transactions on Software Engineering 32(9), 2006, pp. 642–663.

[4] Java Virtual Machine Profiler Interface. Available at: http://download.oracle.com/javase/1.4.2/docs/guide/jvmpi/jvmpi.html,(accessed: 11 June 2011)

[5] Java Virtual Machine Tool Interface. Available at: http://download.oracle.com/javase/6/docs/technotes/guides/jvmti (accessed: 11 June 2011)

[6] KELLER, A., LUDWIG, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, Journal of Network and Systems Management, 11(1), 2003, pp. 57-81.

[7] LADDAD, R., JOHNSON, R.: Aspectj in Action: Enterprise AOP with Spring Applications, Manning Publications, USA, 2009.

[8] OKANOVIĆ, D., VAN HOORN, A., KONJOVIĆ, Z., VIDAKOVIĆ, M.: Towards Adaptive Monitoring of Java EE Applications, In Proceedings of 5th International Conference on Information Technology – ICIT, Amman, Jordan, 2011.

[9] OKANOVIĆ, D., VIDAKOVIĆ, M.: One Implementation of the System for Application Version Tracking and Automatic Updating", In Proceedings of the IASTED International Conference on Software Engineering, ACTA Press, pp. 62–67, 2008.

[10] PARSONS, T., MOS, A., MURPHY, J.: Non-intrusive End-to-end Runtime Path tracing for J2EE Systems, IEEE Proceedings - Software 153(4), pp. 149–161, 2006.

[11] PASCHKE, A., SCHNAPPINGER-GERULL, E.: A Categorization Scheme for SLA Metrics, In Proceedings of Multi-Conference Information Systems (MKWI 2006), Passau, Germany, 2006.

[12] SNATZKE, R. G.: Performance survey 2008, 2008. Available at: http://www.codecentric.de/export/sites/homepage/__resources/pdf/studien/performance-studie.pdf (accessed: 11 June 2011)

[13] STURM, R., MORRIS, W.: Foundations of Service Level Management, Sams, Indianapolis, USA, 2000.

[14] SULLINS, B. G., WHIPPLE, M. B.: JMX in Action, Manning Publications, USA, 2002.

[15] TEBBANI, B., AIB, I.: GXLA a Language for the Specification of Service Level Agreements, Springer Lecture Notes in Computer Science 4195/2006, pp. 201-214, 2006.

[16] van HOORN A., ROHR M., HASSELBRING W., WALLER J., EHLERS J., FREY S., KIESELHORST D. Continuous Monitoring of Software Services: Design and Application of Kieker Framework, Technical Report TR-0921. Department of Computer Science, University of Kiel, Germany, 2009. available at: http://www.informatik.uni-kiel.de/uploads/tx_publication/vanhoorn_tr0921.pdf (accessed: 11 June 2011)

[17] VILLAZON, A., BINDER, W., ANSALONI, D., MORET, P.: HotWave: Creating Adaptive Tools with Dynamic Aspect-Oriented Programming in Java, In the Proceedings of the 8th International Conference on Generative Programming and Component Engineering (GPCE '09), ACM, pp. 95–98, 2009.

ACTA TECHNICA CORVINIENSIS – BULLETIN of ENGINEERING

ISSN: 2067-3809 [CD-Rom, online]