ICIT 2015 The 7th International Conference on Information Technology
doi:10.15849/icit.2015.0119 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

A Domain-Specific Language for Service Level Agreement Specification

Renata Vaderna, Željko Vuković, Dušan Okanović, Igor Dejanović
Faculty of Technical Sciences
University of Novi Sad
Novi Sad, Serbia
{vrenata, zeljkov, oki, igord}@uns.ac.rs

***Abstract*—In order to perform continuous monitoring, SLA document between interested parties has to be signed. These documents should be in machine readable format in order to automate monitoring process. On the other hand, it would be beneficial if it is human readable, too. This way, it is easier to perform configuration and maintenance of monitoring subsystem. Building up on our previous work, in this paper we present DProfLang. DProfLang is a domain specific language for defining SLAs, that are both human and machine readable.**

Keywords—SLA, continuous monitoring, Domain-Specific Languages

I. INTRODUCTION

Requirements that certain software has to fulfill are usually agreed between interested parties before the start of implementation. There are two types of requirements: functional and non-functional. Ensuring that software fulfills functional requirement means that it will "do what it is expected to do." On the other hand, implementation of non-functional requirements means that the software will "do what is expected, but in a certain way." It is important to stress that while performance measurements can be performed during the development phase, it is only under production workload that we can retrieve realistic software performance data. There are often bugs that take a lot of time to manifest themselves [1], and this kind of time is not available during development. In contrast to profiling and debugging, when performing continuous monitoring we measure application performance parameters under production workload.

There is a wide array of nonfunctional requirements and metrics that can be used to quantify them. Some commonly used are response time, availability, security, robustness, memory footprint, CPU time. These parameters are usually referred to as software performance and are specified in an additional document that follows the initial agreement between the parties. This document is called Service Level Agreement (SLA). It can contain functional requirements, ways of measuring their fulfillment, referent values, ways of processing these values, and whom to contact if something goes wrong, either with the obtained values or the measuring process itself.

In our previous works [2, 3], we have described the DProf system for adaptive continuous monitoring. It is based on the Kieker monitoring framework [4], and it monitors application performance using monitoring probes. These probes are inserted into software using AspectJ or some other tool [5], and collect monitoring data, while the application is running. Adaptation of the monitoring process allows for reduction of monitoring overhead. This is done by turning monitoring off in the call tree [6] branches that show no discrepancy between the obtained values and values specified in SLA.

SLA for the DProf system is an XML document based on the DProfSLA XML schema [2]. Since XML is a machine readable format, but not well suited for human use [20], in this paper we propose a new language - DProfLang - for monitoring goals definition. The domain specific language that we propose in this paper has the advantage of being both human and machine readable, thus allowing easier maintenance of monitoring configuration, while being well suited for monitoring automation.

The remainder of this paper is as follows. Chapter 2 shows XML schema that we currently use. In chapter 3, grammar of the new language is shown. Chapter 4 shows how to translate a document from DProfSLA format into DProfLang. Chapter 5 presents related work, while in the last section we draw conclusions and outline for the future work.

II. DPROFSLA

Root element of DProfSLA XML schema is shown in Fig. 1. It has three subelements:

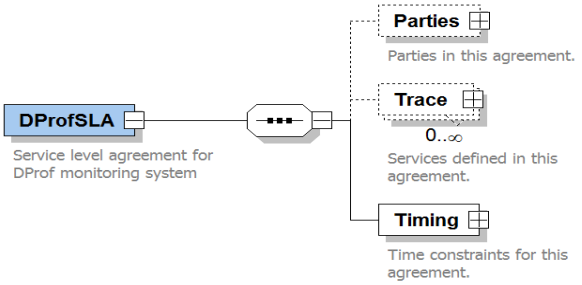

Fig. 1. Root element of DProfSLA XML schema

doi:10.15849/icit.2015.0119 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

Parties element is simple and is used to designate interested parties and their roles in the execution of the agreement.

Timing element specifies the agreement's time constraints - the start and the end of the monitoring process, and the frequency of checkups.

Trace element (of *CallTreeNode* type - Fig. 2) is used to specify which part of the application is monitored and how the obtained data is processed. In essence, every trace element relates to one node in a call tree, i.e. a method call.

For designating call tree nodes we use attribute *name* in *CallTreeNodeType* and syntax shown in [2]. For the call tree in Fig. 3, we have the DProfSLA document from Listing 1.

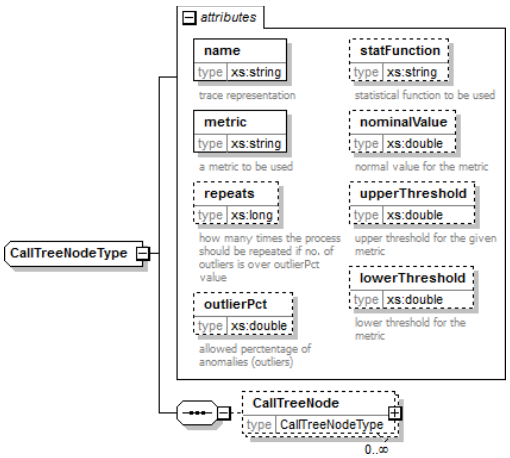

Fig. 2. Call tree node representation in DProfSLA XML schema

A node is represented with class and method name, followed by names of methods that are invoked from it. In this example, we monitor execution times, calculate averages, and compare those values to the specified upper threshold.

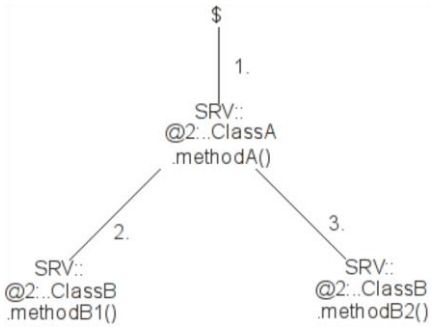

Fig. 3. An example of call tree

```
<DProfSLA>
<Parties><Provider name="Org1" />
  <Consumer name="Org2" /></Parties>
<CallTreeNode metric="avgExecutionTime"
  name="ClassA.methodA,[{ClassB.methodB1,[]},
  {ClassB.methodB2,[]}]" upperThreshold="350">
 <CallTreeNode metric="avgExecutionTime"
    name="[{ ClassB.methodB1,[]}]"
   upperThreshold="150"/>
 <CallTreeNode metric="avgExecutionTime"
    name="[{ ClassB.methodB2,[]}]"
   upperThreshold="150"/>
</CallTreeNode>
<Timing>
    <SamplingPeriod>600000</SamplingPeriod>
</Timing>
</DProfSLA>
```

Listing 1. DProfSLA XML for the example shown in Fig. 3.

As stated in the introductory chapter, the use of XML provides the possibility of automation of the monitoring process, since XML is machine readable. However, the use of DSL would allow human readability, while retaining machine readability.

III. DProfLang Language Grammar

DProfLang DSL is implemented using textX [7] meta-language and library for DSL development in Python programming language. From a single language description (grammar) textX builds a parser and a meta-model (i.e. abstract syntax) for the language.

textX grammar consists of a set of rules which define each language construct and will be translated to Python classes during Abstract Syntax Tree (AST) construction. Each rule also defines the syntax of the language element.

In Listing 2 a part of DProfLang grammar is presented. From this grammar textX will create the meta-model presented in Fig. 4. BASETYPE hierarchy is a part of the built-in textX type system.

```
DProfModel:
    'SLA' name=STRING description=STRING
    'parties' '{'
        parties+=Party
    '}'
    timing=Timing
    call_node=CallNode
;
CallNode:
    'cnode' name=STRING '{'
    // Inherited from parent node.
    // The root node must specify it.
        ('metric' metric=Metric)?
        // These are optional as there are
        // default values specified.
        ('repeats' repeats=INT)?
        ('outlier_percentage'
              outlier_percentage=INT)?
        ('stat_func' stat_func=StatFunc)?
        ('nominal_value' nominal_value=FLOAT)?
        ('lower_treshold' lower_threshold=FLOAT)?
        'upper_treshold' upper_treshold=FLOAT
        nodes*=CallNode
    '}'
;
```

Listing 2. A part of the DProfLang grammar in textX

doi:10.15849/icit.2015.0119 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

The DProfModel rule is the root of the meta-model. Instances of these classes have the following attributes:

- *name* – is the name of the SLA agreement,
- *description* – is an optional description given as a string,
- *parties* – is a list of the involved parties,
- *timing* – is an interval specifying when the monitoring will be applied,
- *call_node* – is the root of the call tree node hierarchy.

DProfLang meta-model instance is a Pyton object which is capable of parsing and instantiating DProfLang models written as DSL textual specifications.

Listing 3 shows an example of a DProfLang agreement of the DProfSLA document from Listing 1. It is obvious that the readability and comprehensibility is vastly improved with the DSL approach.

A. *Transformation From DProfSLA to DProfLang*

In order to integrate the new language with our previous work, we have developed two code generators. The first

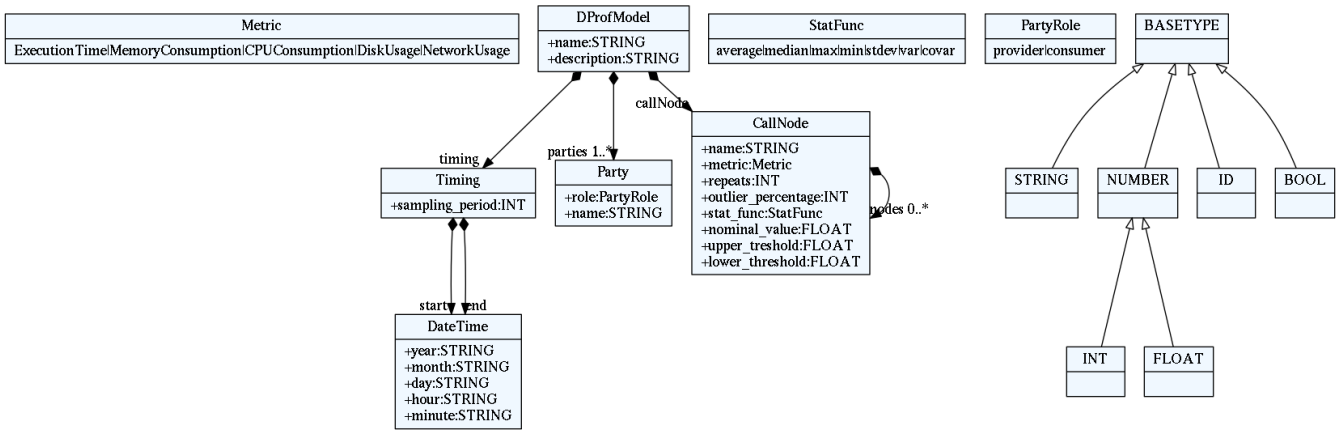

Fig. 4. DProfLang textX meta-model

CallNode rule defines a node in a call tree node hierarchy and specifies monitoring parameters such as: used metric, repeats and outlier percentage, nominal value, upper and lower threshold. This rule uses composite pattern, as each node can contain other nodes which are specified by the assignment nodes*=CallNode. textX assignment operator '*=' will match zero or more right-hand-side rules and each instance will be appended to the left-hand-side attribute.

```
SLA "Example"

parties {
    provider "Org1"
    consumer "Org2"
}

timing {
    sampling_period 600000
}

cnode "ClassA.methodA" {
  metric ExecutionTime
  stat_func Average

  upper_treshold 350

  cnode "ClassB.methodB1" {
    upper_treshold 150
  }
  cnode "ClassB.methodB2" {
    upper_treshold 150
  }
}
```

Listing 3. An example of SLA specification written in DProfLang

generator loads DProfSLA document in the original XML format and outputs the agreement in the new DSL format. The second one does the reverse job - it parses the agreements in DProfLang format and provides XML based DProfSLA document.

For code generation, Jinja2 template engine [8] for Python has been used. A template engine is a piece of software that combines a data model with a template specification to produce a textual output. In our case data model is based on DProfLang meta-model. Two templates have been used: DProfSLA XML template and DProfLang DSL template. Instantiating data model from DProfLang DSL is supported through textX, since it automatically constructs the model from the grammar. In order to support XML we had to develop a procedure that builds data model out of DProfSLA XML.

IV. Related Work

SLAs must be defined in machine-readable format to allow automatic service level management. Tebbani et al. [9] have already shown that only a few formal SLA specification languages exist. Usually, SLAs are written in some informal language, which is not acceptable for automation of the process. Therefore, authors propose Generalized Service Level Agreement language - GSLA. A GSLA document is a contract between interested parties that is designed to create a measurable common understanding of each party's role. The role is a set of rules which defines the service level expectations and obligations the party has. To specify GSLA in machine readable format, GXLA XML schema has been

doi:10.15849/icit.2015.0119 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

proposed. Sections of GXLA documents are as follows. Schedule section contains temporal parameters of the contract. Party section models involved parties. Service package is an abstraction that is used to describe the services and previously mentioned roles. By using GXLA the service management process can be automated.

For web service SLAs, WSLA [10] can be used. It is also XML-based. Similarly to GSLA/GXLA, WSLA documents define the involved parties, metrics, measuring techniques, responsibilities, and courses of action. The authors state that every SLA language, such as WSLA, should contain 1) information regarding the agreeing parties and their roles, 2) SLA parameters and a measurement specification, as well as 3) obligations for each party.

SLAng [11] is a language for specifying SLAs based on the Meta Object Facility [12]. It can use different languages to describe constraints, e.g., utilizing OCL [13] or HUTN [14].

The WS-Agreement specification language [15] has been approved by the Open Grid Forum. It defines a language that can be used by service providers to offer services and resources, and by clients to create an agreement with that provider.

Paschke et al. [16] propose to categorize SLA metrics in order to support the design and implementation of SLAs that can be monitored and enforced automatically. Standard elements of each SLA are categorized as: technical (service descriptions, service objects, metrics, and actions), organizational (roles, monitoring parameters, reporting, and change management), and legal (legal obligations, payment, additional rights, etc.).

According to this categorization, our DProfLang documents are operation-level documents intended to be used in-house. By versatility categorization, they belong to standard agreements. As was the case with DProfSLA schema documents, we do not need all of the features of the described schemas. DProfLang is specifically designed to be used with the DProf system. Our documents provide a subset of the elements defined by GXLA or WSLA. A transformation of SLA documents between DProfLang and the mentioned schemas could, for example, be performed using appropriate generators.

Aside from XML, an SLA can be specified using domain specific languages. Most of them are AOP based, like DiSL [17], Josh [18] or Scope [19]. The problem with using AOP is that they are very platform specific. The use of a true DSL for SLA specification allows for writing of human readable documents that can be translated into instrumentation for any platform.

V. CONCLUSION

In this paper we have shown a new language for instrumentation specification. The advantage of this approach over the use of XML is that the SLA documents written with DProfLang are human readable. This allows for easier maintenance of monitoring system and better overall control over monitoring process. In contrast to the use of AOP and AOP-like tools, our approach is platform independent. Whichever the underlying platform might be, DProfLang SLA document will be translated into instrumentation for the underlying platform.

DProfLang is implemented in textX meta-language which enables easy language grammar and meta-model modifications thus facilitating its evolution. To enable integration with our pre-existing XML based solution we have also implemented a translator from XML to the new DSL and vice versa.

Our future work will focus on development of instrumentation generators for different platforms. As DProf and Kieker use AspectJ instrumentation, our first step is to develop instrumentation generators for AspectJ. After that, our work will include generators for DiSL and .NET AOP frameworks.

ACKNOWLEDGMENT

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] M. Grottke, K. S. Trivedi. “Fighting Bugs: Remove, Retry, Replicate, Rejuvenate,” IEEE Computer, v.40, n. 2, 2007, pp. 107-109.

[2] D. Okanović, A. Van Hoorn, Z. Konjović, M. Vidaković, “SLA-Driven Adaptive Monitoring of Distributed Applications for Performance Problem Localization,” Computer Science and Information Systems, vol. 10, no. 1, 2013, pp. 25-50.

[3] D. Okanović, A. van Hoorn, Z. Konjović, M. Vidaković, “Towards Adaptive Monitoring of Java EE Applications”, Proceedings of the 5th International Conference on Information Technology - ICIT. Amman, Jordan, 2011, CD.

[4] A. van Hoorn, W. Hasselbring, J. Waller, “Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis,” Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012), Boston, USA, 2012, pp. 247-248.

[5] D. Okanović, M. Vidaković, “Evaluation of Alternative Instrumentation Frameworks,” Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days, Stuttgart, Germany, 2014, pp. 83-90.

[6] W. Binder, J. Hulaas, P. Moret, “Advanced Java Bytecode Instrumentation,” 5th International Symposium on Principles and Practice of Programming in Java, Lisboa, Portugal, 2007, p. 135-144.

[7] textX [Online] https://github.com/igordejanovic/textX (January 2015)

[8] Jinja2 [Online] http://jinja.pocoo.org/docs/dev/ (January 2015)

[9] B. Tebbani, I. Aib, “GXLA a Language for the Specification of Service Level Agreements,” Lecture Notes in Computer Science, v. 4195. Springer-Verlag, Berlin Heidelberg New York, 2006, p. 201-214.

[10] A. Keller, H. Ludwig, “The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services,” Journal of Network and Systems Management, vol. 11, no. 1, 2003, pp. 57-81.

[11] D. Lamanna, J. Skene, W. Emmerich, “SLAng: A Language for Defining Service Level Agreements,” Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computer Systems (FTDCS '03), IEEE Computer Society, San Juan, Puerto Rico, 2003, pp. 100-107.

[12] Meta Object Facility (MOF) 2.0 Core Specification. OMG. [Online] Available: http://www.omg.org/spec/MOF/2.0 (current September 2011)

doi:10.15849/icit.2015.0119 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

[13] Object Constraint Language (OCL) 2.0. OMG. [Online] Available: http://www.omg.org/spec/MOF/2.0 (January 2015)

[14] Human Usable Textual Notation (HUTN) Specification. OMG. [Online] Available: http://www.omg.org/spec/HUTN/index.htm (January 2015)

[15] N. Oldham, K. Verma, A. Sheth, F. Hakimpour, "Semantic WS-agreement partner selection," 15th International Conference on World Wide Web. ACM, Edinburgh, Scotland, UK, 2006, pp. 697-706.

[16] A. Paschke, E. Schnappinger-Gerull, "A Categorization Scheme for SLA Metrics," Multi-Conference Information Systems (MKWI 2006), Passau, Germany, 2006, pp. 25-40.

[17] L. Marek, A. Villazón, Y. Zheng, D. Ansaloni, W. Binder, Z. Qi, "DiSL: a Domain Specific Language for Bytecode Instrumentation," 11th Annual International Conference on Aspect-Oriented Software Development (AOSD '12), 2012, pp. 239-250.

[18] S. Chiba, K. Nakagawa, "Josh: an Open AspectJ-Like Language, ". AOSD'04, ACM, 2004, pp. 102–111.

[19] T. Aotani, H. Masuhara, "Scope: an AspectJ Compiler for Supporting User-Defined Analysis-Based Pointcuts," AOSD'07, ACM, 2007, pp. 161–172.

[20] T. Parr, "Humans should not have to grok XML; Answers to the question 'When shouldn't you use XML?'", IBM DeveloperWorks, 2001