Automatic Detection of SLS Violation Using Knowledge based Systems

Pedro Alípio, José Neves, and Paulo Carvalho

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
{pma,jneves,pmc}@di.uminho.pt

Abstract. Self-Management of multiservice network domains must include automatic detection of service quality violations in order to perform appropriate and timely responses, fostering the fulfillment of the service agreements committed with customers, or other ISPs. As knowledge based systems are capable of emulating human knowledge, they are presented here as convenient tools to specify heuristics for QoS violation detection. In this paper, the implementation of a service violation detection agent is described and an illustrative scenario of its use is also presented.

Key words: Service Level Specification, Knowledge based Systems, Quality of Service, Self-configuration Networks.

1 Introduction

Monitoring Service Level Agreements (SLAs) between Internet Service Providers (ISPs) and customers is becoming a very relevant topic as the number of companies using the Internet to perform their business activities increases (e.g. on-line banking services, auctioning, ticket reservation, Internet telephony, Internet TV and radio). Many of those services are tolerant to some performance degradation, while others are not tolerant at all. As a result, the disruption of the contracted Quality of Service (QoS) may have serious consequences on the customers activities. Under such circumstances, ISPs may be considered responsible and prosecuted. Therefore, in their own interest, ISPs must be sure that the QoS guarantees are being assured as a key component of SLA auditing procedures.

In the context of an SLA, the technical requirements of a service are specified as a Service Level Specification (SLS) which includes, among other information, the expected QoS. The most frequent QoS metrics used to express the service performance are throughput, one-way delay, interpacket delay variation and packet loss ratio. The target values for these metrics depend on each type of service and corresponding requirements, e.g., the expected QoS for an Internet Telephony service, consisting of inelastic traffic with fixed small packets transmitted at a Constant Bit Rate (CBR), is expressed by very low packet loss ratio, very low one-way delay and very low delay variation [1]. Other services do not require so strict QoS levels. For instance, a Web Server is highly tolerant to delay and its variation, but sensitive to loss.

Monitoring SLSs is the process of collecting statistical metrics about service's performance in order to verify if it meets the specified requirements. Our main concern is edge-to-edge SLS monitoring. Therefore, the service traffic has to be monitored at the entry and exit nodes of ISP domains, and then the resulting metrics verified against the information within SLSs. This process may be enhanced when performed autonomously using knowledge based technologies. Such technologies consist of knowledge specification through sets of production rules (**if** antecedents **then** consequents) and an Inference mechanism to perform the deductive process over factual information to reach out for conclusions. These conclusions may be one of the following: to assess the utilization of network resources, to carry out an automatic network reconfiguration process or to re-negotiate SLAs.

This paper has the following structure: related work is presented in Section 2; an overview of CLIPS knowledge based system shell is given in Section 3; the proposed SLS violation detection strategy is presented in Section 4; an example of an SLS violation scenario is given in Section 5 and the conclusions and future work are presented in Section 6.

2 Related Work

Detecting SLS violations is crucial for self-management networks pledged in assuring edge-to-edge QoS. The research community has been committed to SLS definition, monitoring and management [2,3,4,5,6,7,8,9,10,11,12]. Work is currently in progress

to establish service classes and their respective configuration in DiffServ networks [1], and on mapping SLS into PDBs (Per Domain Behaviors) [13]. These contributions, as well as the ones resulting from this work aim to assure a consistent forwarding path treatment for the services traffic in a network domain [14].

Most of these approaches use the Common Open Policy Service (COPS) [15] to map SLSs into network configurations. However, the COPS protocol does not include a knowledge base or any form of reasoning mechanisms. On these systems, SLS violations are detected by hard-coded conditions, and they are not flexible enough to support changes in the detection criteria or in the inclusion of new metrics. By using a general purpose rule based system, eg., the C Language Integrated Production System (CLIPS), a wider range of functionalities may be included improving the flexibility and applicability of the solution.

Although, in previous work, XML was used to specify service requirements [16], a XML rule description language such as RuleML[1] was not considered due to the following: (i) XML is very verbose and rules with a large number of conditional elements and consequents tend to be very difficult to read; (ii) CLIPS is easily extendable to support XML parsing and validation, database access, or to include a network configuration protocol to trigger short term actions; (iii) CLIPS is written in C and implements a forward chaining inference engine which is more efficient than XML rule based systems implementations such as Mandarax[2] which is written in Java and uses backward chaining.

3 An Overview of CLIPS

CLIPS was developed using the C programming language at NASA/Johnson Space Center aiming at high portability, low cost and easy integration with external systems.

CLIPS is a multiparadigm programming language providing support for rule-based, object-oriented, and procedural programming [17]. The inference engine algorithms and the knowledge representation provided by the rule-based programming language are similar, but even more powerful than those used in OPS5 production system [18]. CLIPS rules syntax is similar to rule languages such as ART, ART-IM, Eclipse, and Cognate. Only forward chaining is supported. The Object-oriented programming in CLIPS is called COOL (CLIPS Object Oriented Language), which combines features of common object-oriented languages, such as Smalltalk and Common Lisp Object System (CLOS), with some new ideas. The procedural programming language has features that are similar, among others, to C, PASCAL, ADA, but it is syntactically similar to LISP. CLIPS source code is available for multiple platforms.

4 Using CLIPS to Detect SLS Violations

To be possible to compare the values measured by the monitoring process with the QoS requirements, SLSs have to be specified in a structured way. An XML based specification was used for that purpose through XML Schema, containng the structure, elements and data types included in SLSs [16]. As XML is an universal language to be used in structured documents, XML SLSs may be used as a common ground for interoperability and heterogeneity within or across ISP domains. Although, many software applications tend to use XML, most of the knowledge based systems shells do not support it yet. Our solution involves transformation of XML SLSs into the public domain CLIPS through XSLT (eXtensible Stylesheet Language Transformations). In addition, knowledge based systems with XML support tend to be slower and the rules are often more verbose and more difficult to read.

In a network domain, services are often mapped into Classes of Service (CoS). In such case, traffic is commonly measured edge-to-edge for each service class. Otherwise, it is measured on a flow basis. In either case, no specific metric collection methods are assumed. Active or passive monitoring may therefore be used by the Measurement Agents (MAs).

Figure 1 illustrates the deployment of MA and SVDA agents in the boundaries of the ISP domain network. As showed, there may be a MA monitoring each node or each MA may collect information form several nodes or interfaces as MA1. MAs inform the SVDA periodically or when a anomalous situation occurs.

[1] http://www.ruleml.org
[2] http://mandarax.sourceforge.net

As SLSs may require different statistics types and monitoring periods, they may be expressed through rules obtained by preprocessing the SLS transformations. Therefore, two main rule sets need to be considered: a preprocessing rule set collecting higher level information from the resulting SLS XML transformations, and a second rule set containing rules to detect SLS violations. Whenever a violation is detected, the SVDA informs the self-management network decision agent, i.e., it will decide whether to reconfigure the network domain (short-term actions) or to renegotiate the SLA (long-term actions).

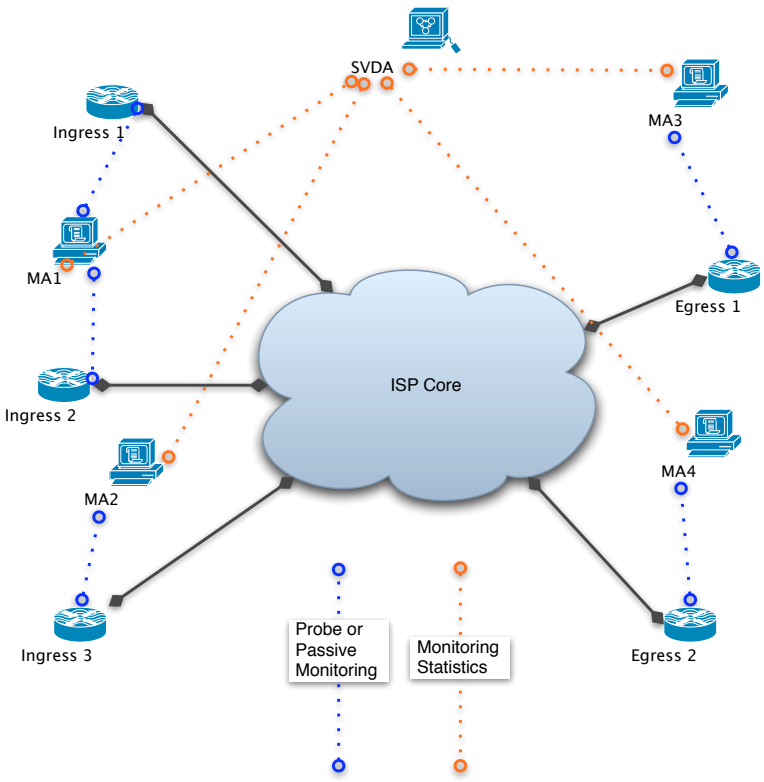

Fig. 1. SLS QoS Monitoring and Violation Detection Deployment.

4.1 The Monitoring System Architecture

The architecture of the monitoring system includes the following processing steps: (i) Validation of SLSs; (ii) management of SLSs; (iii) translation of SLSs into the Expert System Language; (iv) getting information from MAs and (v) the inference process. Figure 2 illustrates the whole process of service violation detection starting from the SLSs submission process.

After being submitted to the system, SLSs are validated. This step is accomplished by verifying if the SLS XML structure, elements and data types are written according to the XML Schema. Valid SLSs are passed to the SLS Translator which translates the XML SLS into the knowledge based system language. A record in a database is created by the SLS Management Module saving information about the active SLSs and their respective translation, to be added to the knowledge based system fact list.

The violation detection process starts whenever data from the MAs is received. As a result, the knowledge based system inference is activated and rules are fired over SLS facts and over facts build with the data collected by the MAs. Rule sets containing the service violation detection knowledge are also added to the inference engine. When a conclusion is reached, a response is issued. Responses may be either active or passive, i.e, they may be a message to the self-management network core or just an alarm message to the system operator.

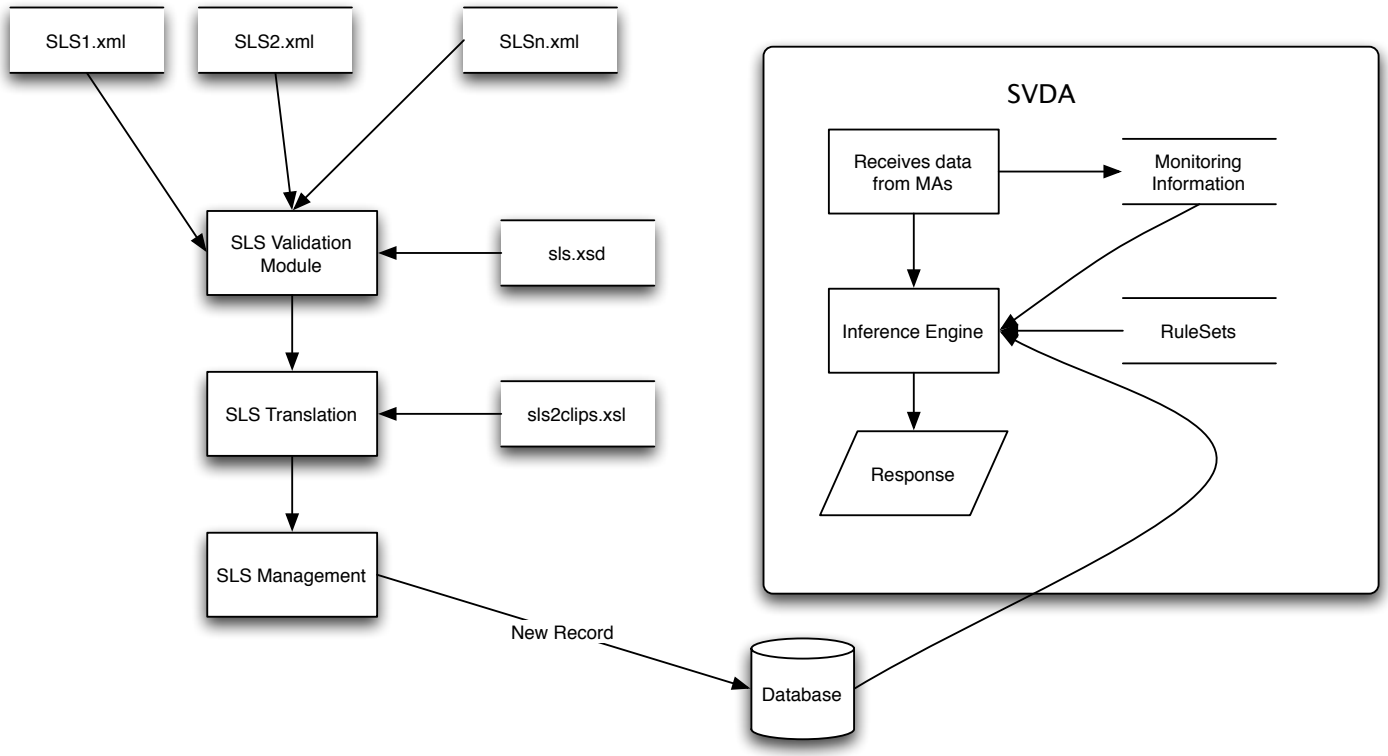

Fig. 2. SLS Violation Detection Process.

4.2 Implementation

CLIPS was used here because it has a set of functions for including new functionalities and extensions, and it is very well documented. Furthermore, it is a public domain open source knowledge based system shell. Consequently, it is possible to modify its code and recompile it.

The SVDA implementation includes several other software modules. The XML validation module was build using *libxml* [19]. The following code implements the SLSs validation function:

```
1 xmlSchemaParserCtxtPtr xmlsch_url = xmlSchemaNewParserCtxt("sls.xsd");
2 xmlSchemaPtr xmlsptr = xmlSchemaParse(xmlsch_url);
3 xmlSchemaValidCtxtPtr validxmlsch = xmlSchemaNewValidCtxt(xmlsptr);
4 xmlSchemaValidateDoc(validxmlsch,doc);
```

While the first line creates an XML Schema parse context for "sls.xsd", which contains the XML Schema file for SLSs, the second line parses the schema definition resource and build an internal XML Schema structure which can be used to validate XML SLS documents. The third line creates an XML Schema validation context based on the given schema. Finally, the forth line validates the document tree.

The transformation into CLIPS facts is accomplished by using *libxst* [20], according to the following code:

```
1 cur = xsltParseStylesheetFile((const xmlChar *) xsltfile);
2 res = xsltApplyStylesheet(cur, doc, params);
```

First line load and parses the XSLT stylesheet which in this case is "sls2clips.xsl". The second line applies the stylesheet to the document (doc). The variable *cur* is the current *xslt* file and parameters are passed to the stylesheet through the variable *params*.

SQLite [21] was used to store SLSs. *SQLite* is a small C library that implements a self-contained, embeddable, without configuration needs. CLIPS system was extended with functions to read, write and modify records in *SQLite* databases.

As the distributed MAs are not yet implemented, in the current version of the SVDA, data is read from a text file resulting from simulated scenarios using the Network Simulator 2 (NS2) [22]. These text files contain the monitored QoS metrics (one way delay, inter packet delay variation and packet drop ratio) for every combination of ingress nodes, egress nodes and CoS. In addition, per flow monitoring may also be used. Under such circumstances, metrics are collected for each combination of source, destination, and flow id. Each line is converted into CLIPS facts as they are read, according to the following rule:

```
1  (defrule mon1
2      (start-monitoring)
3  =>
4      (open ?*out-flow-file* rf "r")
5      (bind ?line (readline rf))
6      (while (not (eq ?line EOF))
7          (bind ?line (readline rf))
8          (assert-string (str-cat "(monitoring " ?line ")"))
9      )
10     (close rf)
11 )
```

This rule asserts the facts *(monitoring ?time ?i ?e ?cos ?drops ?delay ?jitter)* or *(monitoring ?time ?s ?d ?fid ?drops ?delay ?jitter)* on the agent knowledge base, for class based or flow based monitoring, respectively.

More than one hundred rules were created to preprocess SLS information, network topology and resources and monitoring. In other to simplify how the service violation detection is accomplished the following rules are considered:

```
1  (defrule exceeded-delay-detection
2          (monitoring ?instant ?e ?cos ?mdrops ?mdelay ?mjitter)
3          (dscp ?slsid ?cos)
4          (expected-delay ?slsid ?delay ?period ?quantile)
5          (first-read delay ?slsid ?start-time)
6          (within ?instant ?start-time ?period)
7          (test (> ?mdelay ?delay))
8          (exceeded delay ?slsid ?er)
9  =>
10         (assert (exceeded ?slsid (+ ?er 1)))
11 )
```

The first rule keeps track of delay violations within a measurement period. A value exceeding the maximum allowed delay may in fact not be considered a violation as quantiles may be specified in the SLS expected QoS section. A quantile is the percentage QoS violations allowed in a period. Line 3 looks for the CoS in which the service was previously mapped into. Line 4 gets the delay requirements from the SLS. Lines 5 and 6 are used to check if the measurement instant is within the period defined in the SLS for the one way delay parameter. Line 7 tests if the monitored value is greater than the SLS expected delay. Line 8 gets the last value of the exceeded values counter. If the rule is fired, the counter is increased by one.

```
1  (defrule total-reading-within-period
2          (monitoring ?instant ?i ?e ?cos ?mdrops ?mdelay ?mjitter)
3          (dscp ?slsid ?cos)
4          (expected-delay ?slsid ?delay ?period ?quantile)
5          (first-read delay ?slsid ?start-time)
6          (within ?instant ?start-time ?period)
7          fid<-(total-readings delay ?slsid ?tr)
8  =>
9          (retract ?fid)
10         (assert (total-readings delay ?slsid (+ ?tr 1)))
```

```
)
```

The second rule calculates the number of monitored events within a period. Lines 2 to 6 are equal to those in the previous rule. Line 7 gets the number of events within the period counter. If the rule matches, the counter is increased.

```
1  (defrule delay-violation-detection
2          (monitoring ?instant ?i ?e ?cos ?mdrops ?mdelay ?mjitter)
3          (dscp ?slsid ?cos)
4          (expected-delay ?slsid ?delay ?period ?quantile)
5          (first-read delay ?slsid ?start-time)
6          (within ?instant ?start-time ?period)
7          (exceeded delay ?slsid ?er)
8          (total-readings delay ?slsid ?tr)
9          (test (> ?er (* ?tr ?quantile)))
10 =>
11         (printout t "SLS " ?slsid " at " ?instant
12          " VIOLATION: delay(" ?mdelay ") > SLS delay(" ?delay ")" crlf)
13         (assert (violation ?slsid ?instant delay))
14 )
```

The third rule is the service violation detection rule. Lines 2 to 6 are equal to the previous rules. Line 7 gets the value of the exceed delay and line 8 gets the number of monitoring events within the period. Line 8 checks if the number of exceeded delay measurements is greater than the allowed quantile. When this occurs, a notification message is generated.

```
1  (defrule delay-period-restart
2          (monitoring ?instant ?i ?e ?cos ?mdrops ?mdelay ?mjitter)
3          (dscp ?slsid ?cos)
4          (expected-delay ?slsid ?delay ?period ?quantile)
5          fid1<-(first-read delay ?slsid ?start-time)
6          fid2<-(exceeded delay ?slsid ?instant ?er)
7          fid3<-(total-readings delay ?slsid ?tr)
8          (after ?instant ?start-time ?period)
9  =>
10         (retract ?fid)
11         (assert (first-read delay ?slsid ?instant))
12         (assert (exceeded delay ?slsid 0))
13         (assert (total-readings delay ?slsid 0))
14 )
```

The last rule is used to restart the counters when another period is in place.

5 Example of an SLS Violation Scenario

As an example, a simple scenario is presented. Consider a network domain with three nodes: two edge nodes (E1 and E2) and a core node (Core). In this example, E1 is the ingress node and E2 the egress node. All links have a maximum bandwidth capacity of 10Mbps and a propagation delay of 5ms. The domain accepts two SLAs and their corresponding SLSs, i.e, SLS1 and SLS2 specifying respectively, an Assured Forwarding (AF) and a Best Effort (BE) based service. At the ingress node, service SLS1 traffic is marked with DSCP 20 and SLS2 traffic is marked with DSCP 30 (Figure 3).

Table 1 presents the initial parameters used to test the SVDA. In this case, no congestion is induced on the network. As a result, no alarms are generated by the SVDA, as expected. In order to force an SLS violation, the initial traffic generator parameters are changed in order to force congestion. Source S2 throughput is modified to generate traffic at 10Mbps. Total traffic entering the network is 12Mbps, exceeding the link capacity of 10Mbps.

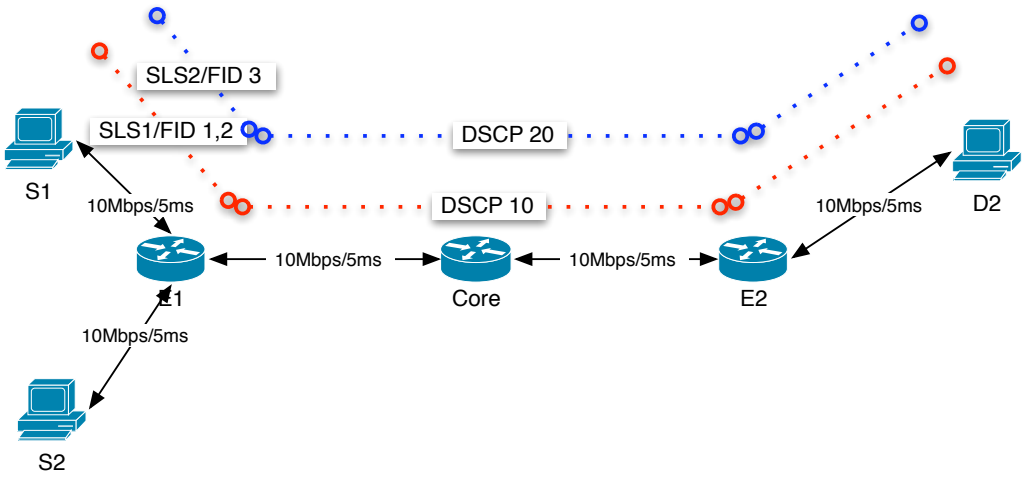

Fig. 3. Test Scenario for the SVDA.

Table 1. Initial test parameters.

Source	Destination	FID	SLS	Expected Delay	Expected Loss	Traffic	Period
S1	D1	1	SLS1	0.01s	0%	CBR 1Mbps	10s
S1	D1	2	SLS1	0.01s	0%	CBR 1Mbps	10s
S2	D1	3	SLS2	1s	10%	CBR 8Mbps	10s

In this case, the SVDA issued the following alerts:

```
SLS2 VIOLATION at 10s: loss(19%) > SLS loss(10%)
SLS2 VIOLATION at 20s: loss(20%) > SLS loss(10%)
SLS2 VIOLATION at 20s: loss(20%) > SLS loss(10%)
SLS2 VIOLATION at 20s: loss(20%) > SLS loss(10%)
```

As packet loss ratio is bigger than 10%, and according to the requirements of the expected QoS metrics in the SLS2, it is considered a violation. If the throughput of SLS1 is also increased to 10Mbps (5Mbps in each flow) than the result will be the following:

```
SLS2 VIOLATION at 10s: loss(51%) > SLS loss(10%)
SLS1 VIOLATION at 10s: loss(0.48%) > SLS loss(0%)
SLS1 VIOLATION at 10s: delay(0.0778) > SLS delay(0.01)
SLS1 VIOLATION at 20s: loss(48%) > SLS loss(0%)
SLS1 VIOLATION at 20s: delay(0.0778) > SLS delay(0.01)
                       ...
```

SLS1 is more restrictive, it does not tolerate any loss and the maximum one way delay is 0.01s. As SLS2 is more tolerant to loss and delay (1 second), only the expected loss ratio violation is detected.

6 Conclusion

Knowledge based systems are understood as symbolic systems, eligible to emulate human reasoning. Thus, besides detecting service QoS violations, this technology may be applied to other components of self-management network domains allowing to fulfill three main objectives: (i) simplify and reduce human intervention, mostly in repetitive network configuration tasks; (ii) fast and adaptive reconfiguration of the network; and (iii) enhanced and dynamic resource management in ISPs domains.

In this work, knowledge based systems have been used to analyze measurements from network service monitoring, to detect eventual violations of agreed QoS. This is a fundamental issue to allow the self-management of network services, as it may be used as a trigger to fire short or long term configuration actions, such as reconfiguring services at a network level (SLSs) or the renegotiating service requirements with the customer (SLAs).

As future work, several fundamental components required for self-management have to be studied and developed in order to allow building of fully autonomous ISP network domains. An ontology based approach is being considered to specify rules and data in the same model.

Acknowledgments

A PHD grant provided by Fundação para a Ciência e Tecnologia (SFRH/BD/17579/2004) is gratefully acknowledged.

References

1. Babiarz, J., Chan, K., Baker, F.: Configuration Guidelines for DiffServ Service Classes. Internet Draft (work in progress) (2004)
2. Morand, P., Boucadair, M., Levis, P., Egan, R., Asgari, H., Griffin, D., Griem, J., Spencer, J., Trimintzios, P., Howarth, M., Wang, N., Flegkas, P., Ho, K., Georgoulas, S., Pavlou, G., Georgatsos, P., Damilatis, T.: Mescal D1.2 - Initial Specification of Protocols and Algorithms for Inter-domain SLS Management and Traffic Engineering for QoS-based IP Service Delivery and their Test Requirements. Mescal Project IST-2001-37961 (2004)
3. A. Diaconescu, S. Antonio, M.E.S.R., Potts, M.: Cadenus D2.3 - Resource Management in SLA Networks. Cadenus Project IST-1999-11017 (2003)
4. M. Mellia, C. Casetti, G.M., Marsan, M.: An Analytical Framework for SLA Admission Control in a Diffserv Domain. in IEEE INFOCOM'03 (2003)
5. J. Chen, A. McAuley, V.S.S.B., Ohba, Y.: Dynamic service negotiation protocol (dsnp) and wireless diffserv. in ICC'02 (2002)
6. P. Bhoj, S.S., Chutani, S.: SLA Management in Federated Environments. Computer Networks, Vol. 35, No. 1 (2001)
7. P. Trimintzios, I. Andrikopoulos, G.P.C.C.D.G.Y.T.P.G.L.G.D.G.C.J.R.E., Memenios, G.: An Architectural Framework for Providing QoS in IP Differentiated Services Networks. in 7th IFIP/IEEE International Symposium on Integrated Network Management - IM'01 (2001)
8. Molina-Jimenez, C., Shrivastava, S., Crowcroft, J., Gevros, P.: TAPAS D10 - QoS Monitoring of Service Level Agreements. TAPAS Project IST-2001-34069 (2002)
9. Sevasti, A., Campanella, M.: Service Level Agreements Specification for IP Premium Service. Geant and Sequin Projects (2001)
10. S. Salsano, F. Ricciato, M.W.G.E.A.T.F.F.T.Z., Brandauer, C.: Definition and Usage of SLSs in the Aquila consortium. IETF draft: draft-salsano-aquila-sls-00.txt (work in progress) (2000)
11. Goderis, D., T'joens, Y., Jacquenet, C., Memenious, G., Pavlou, G., Egan, R., Griffin, D., Georgatsos, P., Georgiadis, L., Heuven, P.V.: Service Level Specification Semantics, Parameters, and Negotiation Requirements. Internet-Draft, <draft-tequila-sls-03.txt>, work in progress (2003)
12. Lamanna, D.D., Skene, J., Emmerich, W.: TAPAS D02 - Specification Language for Service Level Agreements. TAPAS Project IST-2001-34069 (2003)
13. Prieto, A.G., Brunner, M.: SLS to DiffServ configuration mappings. In: 12th International Workshop on Distributed Systems: Operations & Management. (2001)
14. Nichols, K., Carpenter, B.: Definition of Diff. Services Per Domain Behaviors and Rules for their Specification. IETF RFC 3086 (2001)
15. Boyle, J., Cohen, R., Durham, D., Herzog, S., Rajan, R., Sastry, A.: The COPS (Common Open Policy Service) Protocol. IETF RFC2748 (2000)
16. Alipio, P., Lima, S., Carvalho, P.: XML Service Level Specification and Validation. In: The 10th IEEE Symposium in Computers and Communications (ISCC 2005), IEEE Computer Society (2005) 975–980
17. Giarratano, J.C.: CLIPS User's Guide, Volume I - Basic Programming Guide. (2005)
18. Forgy, C.L.: OPS5 User's Manual. Technical Report CMU-CS-81-135, Carnegie Mellon University, Dept. of Computer Science (1981)
19. : libxml - The XML C parser and toolkit of Gnome. URL: http://xmlsoft.org (2005)
20. : libxslt - The XML C parser and toolkit of Gnome. URL: http://xmlsoft.org/XSLT (2005)
21. : The SQLite Website. URL: http://www.sqlite.org (2005)
22. : Network Simulator NSv2. URL: http://www.isi.edu/nsnam (2004)