Comprehensive Explanation of SLA Violations at Runtime

C. Müller(1), M. Oriol(2), X. Franch(2), J. Marco(2), M. Resinas(1), A. Ruiz–Cortés(1), M. Rodríguez(2)
(1)University of Seville, LSI, Seville (Spain), ISA research group, http://www.isa.us.es/
{cmuller,resinas,aruiz}@us.es
(2)Universitat Politècnica de Catalunya, Barcelona (Spain), GESSI research group, http://www.essi.upc.edu/~gessi/
{moriol,jmarco}@lsi.upc.edu, {franch,marcr}@essi.upc.edu

***Abstract*— Service Level Agreements (SLAs) establish the Quality of Service (QoS) agreed between service-based systems consumers and providers. Since the violation of such SLAs may involve penalties, quality assurance techniques have been developed to supervise the SLAs fulfillment at runtime. However, existing proposals present some drawbacks: (1) the SLAs they support are not expressive enough to model real-world scenarios, (2) they couple the monitoring configuration to a given SLA specification, (3) the explanations of the violations are difficult to understand and even potentially inaccurate, (4) some proposals either do not provide an architecture, or present low cohesion within their elements. In this paper, we propose a comprehensive solution, from a conceptual reference model to its design and implementation, that overcomes these drawbacks. The resulting platform, SALMonADA, receives the SLA agreed between the parties as input and reports the explanations of SLA violations in a timely and highly understandable way. SALMonADA performs an automated monitoring configuration and it analyses highly expressive SLAs by means of a constraint satisfaction problems based technique. We have evaluated the impact of SALMonADA over the resulting service consumption time performance. The results are satisfactory enough to consider SALMonADA for SLA supervision because of its low intrusiveness.**

***Keywords*-service level agreement; SLA; monitoring; analysis; violation detection and explanation; QoS.**

I. Introduction and Motivation

Service Level Agreements (SLAs) establish the Quality of Service (QoS) agreed between service-based systems consumers and providers, as well as penalty/reward clauses to be applied when the SLA is violated. Detecting such violations may be complicated, consider for instance the following case.

The Amazon Simple Storage Service (AmazonS3) guarantees a *monthly uptime percentage* equal to or greater than 99.9% in its SLA[1] including a clause to reward their consumers against a lack of service. However, Amazon requires the customer to proof this violation by sending an email within ten business days after the end of the billing cycle in which the errors occurred. Proving the violation demands the customer to compute the monthly uptime percentage by subtracting from 100% the average of the *error rates*[2].

This Amazon scenario illustrates the need of having techniques to supervise the fulfillment of SLAs [1]. In response to this need, several quality assurance proposals have been formulated. They deal with aspects such as: *violation detection*, either at runtime [2]–[8] or testing time [9], [10]; *violation notification* either by *push* (i.e. notifications to subscribed clients as soon as violations are detected at runtime) [4], [5], [11] or *pull* strategies [2], [8], [12]; and *violation explanation* [13], [14]. The possible adoption of these proposals in real-world scenarios is greatly influenced by the following factors:

1) Which SLAs are supported. Usually, real-world SLAs (e.g. AmazonS3 SLA) describe the parties obligations in natural language and they may comprise complex elements such as: (1) conditional terms subject to a precondition (i.e. if the precondition holds, then the term applies), for instance, AmazonS3 SLA offers a data durability of 99.99% only if the client choose a cheaper reduced redundancy storage; (2) optional or alternative terms (i.e. a set of terms that can be chosen by the customers), for instance, AmazonS3 offers a set of alternative support plan terms.
2) How the interaction with the monitor is performed. In order to detect violations we need to monitor the QoS offered by the providers at runtime (i.e. specific values for monitorable service properties such as the service availability or response time). The interaction with the monitoring techniques [15] that gather such QoS monitoring results, requires both: (a) to configure the monitor with the location of the service, and its monitorable service properties; and (b) to decide how the QoS monitoring results are specified. For instance, in the case of AmazonS3, the SLA determines the need for monitoring and obtaining a specific value for the monthly uptime percentage for each billing cycle in order to detect a possible violation.
3) How the violations are detected and explained. The SLA violations are detected by checking the agreed QoS against the obtained QoS monitoring result. And, ideally, the explanation of violations needs to be both comprehensive and timely. By comprehensive, we mean to provide a user-friendly and accurate violation report including not only the violated terms, but also the violation causes and even possible effects, in a form that is easy to understand by humans. For instance, in the AmazonS3 scenario, a comprehensive explanation

[1]http://aws.amazon.com/s3-sla/
[2]internal server errors divided by the requests during each five minute period

would require a report including that the violated term is "monthly uptime percentage equal to or greater than 99.9%" by a monthly uptime percentage of 92% measured in a specific monthly billing cycle. By timely, we mean to communicate the violation as soon as it happens, i.e. when the QoS monitoring result has just been retrieved at runtime and it is checked against the SLA to detect and explain the violations.

4) Which architecture is proposed. In order to ease the maintainability and adaptability of the system, the architecture should comprise loosely coupled and highly cohesive elements separating monitoring from analysis.

As shown in Sec. II, current proposals do not satisfactorily deal with the characteristics above. Some of them may be completely coupled to a particular SLA notation that are not able to deal with all the aspects required in real-world scenarios, others couple the monitoring configuration to a given SLA specification, and most (if not all) provide very basic information when a violation occurs. Furthermore, not all proposals provide architectures with a clear separation of concerns between monitoring and analysis.

The goal of this work is to design and implement SALMonADA, a service-based system to monitor and analyse SLAs in order to provide timely and comprehensive explanations of violations. SALMonADA main features are:

1) specification of a wide range of SLA structures and complex elements based on the use of WS–Agreement [16].
2) automated monitoring configuration through the analysis of the SLAs but without coupling to a given SLA specification.
3) powerful detection and explanation of the SLAs violations by means of a Constraint Satisfaction Problem (CSP)-based technique. A comprehensive and timely explanation of the SLA violations is notified either by push or pull strategies.
4) low coupling in its service-oriented architecture that supports the independent evolution of the core monitoring and analysis components, or even their independent substitution when moving from one problem domain to another.

The impact of SALMonADA over the resulting service consumption time performance and its scalability have been evaluated over real services. We analysed the results in alternative deployment scenarios and we state that SALMonADA has a low intrusiveness compared to the benefits it provides. Moreover, we have developed a web client as a front-end to try it online.

The paper is organised as follows. The related work is evaluated in Sec. II. The conceptual reference model of SALMonADA is detailed in Sec. III, while the details of its design and development are included in Sec. IV. Section V provides information about the CSP-based technique to analyse the SLA fulfillment. Section VI provides information about how SALMonADA checks the QoS monitoring result against the SLA to provide timely and comprehensive explanations. In turn, Section VII evaluates the impact of SALMonADA over the resulting service consumption time performance and scalability. Finally, Sec. VIII concludes the paper with a discussion of our contributions.

Table I
ANALYSIS OF THE RELATED WORK

	Functionality				Architecture	
PROPOSALS	**Supported SLAs**	**Monitor config.**	**Monitoring results**	**Explanation of violations**	**Architecture elements**	**Architecture structure**
WSLA [2,3]	General purpose. Not H-U. ★★☆	Automatic. Coupled to the SLA. ★★☆	Through API. ★☆☆	Detection and partial explanation Not H-U. ★★☆	Monitor and Analyser separated. ★★★	CBS ★☆☆
Comuzzi, Kotsokalis [4]	Particular. ★☆☆	Automatic. Decoupled from the SLA. ★★★	Through API. ★☆☆	Detection. ★☆☆	Monitor and Analyser separated. ★★★	SOA ★★★
Michlmayr et al. [5]	Particular. ★☆☆	Automatic. Coupled to the SLA. ★★☆	Through query lang. ★★☆	Detection. ★☆☆	Monitor and Analyser separated. ★★★	SOA ★★★
Raimondi et al. [20]	General purpose. Not H-U. ★★☆	Automatic. Decoupled from the SLA. ★★★	Through log. ★☆☆	Detection. ★☆☆	Monitor and Analyser in 1 component. ★☆☆	CBS ★☆☆
Sahai et al. [17]	Particular. ★☆☆	Automatic. Coupled to the SLA. ★★☆	Through a formal model. ★★★	Detection and partial explanation Not H-U. ★★☆	Monitor and Analyser separated. ★★★	CBS ★☆☆
Palacios et al. [9]	General purpose. Not H-U. ★★☆	Automatic. Coupled to the SLA. ★★☆	[N/A] ☆☆☆	Detection. ★☆☆	[N/A] ☆☆☆	[N/A] ☆☆☆
Di Penta et al. [10]	Particular. ★☆☆	Automatic. Coupled to the SLA. ★★☆	[N/A] ☆☆☆	Detection. ★☆☆	[N/A] ☆☆☆	[N/A] ☆☆☆
SLA@SOI [11]	General purpose. Not H-U. ★★☆	Automatic. Decoupled from the SLA. ★★★	Through API. ★☆☆	Detection and partial explanation. Not H-U. ★★☆	Monitor and Analyser separated. ★★★	SOA ★★★
TRUSTCOM [12]	General purpose. Not H-U. ★★☆	Automatic. Coupled to the SLA. ★★☆	Through API. ★☆☆	Detection and partial explanation. Not H-U. ★★☆	Monitor and Analyser separated. ★★★	SOA ★★★
Mahbub, Spanoudakis [13,14]	General purpose. Not H-U. ★★☆	Automatic. Coupled to the SLA. ★★☆	[N/A] ☆☆☆	Detection and explanation. Not H-U. ★★☆	Monitor and Analyser separated. ★★★	CBS ★☆☆
Comuzzi, Spanoudakis [19]	General purpose. Not H-U. ★★☆	Automatic. Decoupled from the SLA. ★★★	Through API. ★☆☆	Detection. ★☆☆	Monitor and Analyser separated. ★★★	SOA ★★★
our proposal SALMonADA	**General purpose. H-U. ★★★**	**Automatic. Decoupled from the SLA ★★★**	**Through a formal document. ★★★**	**Detection and explanation. H-U. ★★★**	**Monitor and Analyser separated. ★★★**	**SOA ★★★**

H-U: human understandable

II. RELATED WORK

Several service-based systems quality assurance proposals that aggregate monitoring and analysis facilities can be found in the literature. To conduct the search of the related work, we have revised the most relevant conferences and journals in the area, selecting those papers that were scoped in the field of SLA monitoring and analysis. Furthermore, we have increased the results by adding relevant papers obtained from experts in the field. Table I summarizes the results of this study.

We have examined the selected papers under the four factors described in the introduction. The first three factors fall into the functionality of the proposed solution, whereas the fourth factor falls into its architecture.

Functionality. Considering the three factors for functionality identified in the introduction, we focus on the following issues: (1) Which SLAs are supported, (2a) How the information to configure the monitor is specified, (2b) How the QoS monitoring result is specified and (3) How the violations are explained.

Architecture. The issues arising from this factor are: (4a) Which architectural elements are needed and (4b) How the architectural elements are structured.

We analyse these issues below:

Which SLAs are supported. The proposals fall into one of the following categories:

- Ad-hoc SLA notation [4], [5], [10], [17]. The supported SLAs include ad-hoc information without considering a general-purpose structure or notation. In addition the proposals are not able to deal with all the aspects required on some real scenarios. For instance, they do not support conditional terms subject to preconditions, expressive Service Level Objectives (SLOs), and optional or alternative terms to specify agreement variants [18]
- General-purpose SLA notation [2], [3], [9], [11]–[14], [19], [20]. The supported SLAs consider a general-purpose structure and/or notation. Specifically, in proposals such as [9], [13], [14], [19] the SLAs support the general-purpose structure proposed in the WS–Agreement specification [16], a highly flexible and widespread SLA notation. An advantage of WS–Agreement over other SLA proposals is that it supports the aforementioned aspects that are necessary to model agreements of real scenarios, namely: (1) optional or alternative terms to specify agreement variants, and (2) expressive SLOs that can be guarded by a qualifying condition (QC) to specify conditional terms. However, WS–Agreement just provides a general-purpose schema that must be extended up to eight different points with an internal sublanguage. More specifically, two of the eight points need to be necessarily extended to become a fully-fledged language, namely the service description terms (SDT) that defines the service functionality, and the SLOs. This causes that a system that can deal with a particular WS–Agreement notation is not able to deal with other WS–Agreement notations. Moreover, the sublanguages used in [9], [13], [14], [19] are not neither general-purpose to be easily mapped to each other, nor human understandable. Other works support SLAs specified with WSLA [2], [3] and SLA* [11] that include general-purpose structures and notations. However, such notations are based on XML schemas even for the SLOs assertions (being recursive in the case of SLA*) and thus, they are not as human understandable as they should be.

How the information to configure the monitor is specified. We find the following situations:

- Automatic, coupled to the SLA. Some approaches include a mechanism to automatically configure a monitor from the SLA [2], [3], [5], [9], [10], [12]–[14], [17]. However, in these solutions, the monitor can only be used for a concrete SLA specification, and if this specification changes, the monitor must be modified as well due to its high coupling.
- Automatic, decoupled from the SLA. There are some works [4], [11], [19], [20] which decouple the SLA from the monitor by translating automatically the SLA to another document which includes the information required for monitoring.

How the QoS monitoring result is specified. Some approaches do not describe how the monitoring results are reported [9], [10], [13], [14]. From those that describe it, some provide a log file [20] or an Application Programming Interface (API) [2]–[4], [11], [12], [19] to access the monitoring results. Since these logs and APIs are not standarized, they present significant differences to each other. The lack of a standard prevents the possibility to easily change the monitor and also it requires the analyser to be compatible with the monitor's API or log. A more effective solution is found in [5] where the authors propose a query language to access the measurements of the monitor. However, in this solution, the monitor must deal with that query language. A more advanced solution is also found at [17], where the authors propose a model to store the results, however such model is not explicitly described.

How the violations are explained. The proposals fall into one of the following categories:

- Just detection. Some works detect SLA violations without explaining the cause [4], [5], [9], [10], [19], [20].
- Partial explanation. Other approaches provide the SLOs that have been violated [2], [3], [11], [12], [17], leading to a *partial explanation* of the violation as an SLO could be violated for several reasons that are not detailed.
- Precise explanation but not human-understandable. In [13], [14] the authors provide an accurate violation explanation that identifies the violated SLOs and the monitoring results which caused the violation. However, they use Event-Calculus to express both the conditions and monitoring results, which as the authors state, is not user-friendly.

Which architectural elements are needed. We find the following situations:

- The proposal does not include an architecture [9], [10].
- The proposal includes an architecture where monitoring and analysis are performed in the same component [20].
- The proposal implements the QoS monitoring and SLA analysis in two separated components with different responsibilities, increasing the cohesion and reusability [2]–[5], [11]–[14], [17], [19].

How the architectural elements are structured. We classify the approaches into:

- Component-Based Systems (CBS) [2], [3], [13], [14], [17], [20]. The components are specific of the system, and no details regarding deployment or protocols used are described, which makes them difficult to reuse or replace.
- Service-Oriented Architecture (SOA) [4], [5], [11], [12], [19]. This architecture adds the capability of deploying the different constituent services in a distributed manner, adapting or replacing them, in a highly cohesive and loosely coupled system.

As a summary of this state of the art (see Table I), we can observe that in the functional part, the proposals cover satisfactorily just one or even none of the four identified issues. Improving this situation is the aim of our work. Our solution takes all the features mentioned and either: (1) cover the issues to the same degree of the best proposal of the state of the art or improves the existing proposals by introducing a new strategy, as it will be described in the following sections. This improvement on functionality is accompanied by optimal architectonic decisions (highly decoupled SOA solution).

Our solution includes the usage of platforms from our previous works, SALMon [21] and ADA [22]. The main

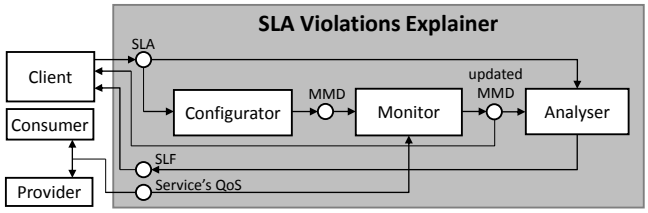


Figure 1. Conceptual Reference Model for SLA violations explainers

goals and initial features they had before this approach are as follows: ADA is a platform whose main goal is to check the consistency of an SLA and compatibility between SLA offers and SLA demands. In turn, SALMon is a monitoring platform whose main goal is to gather the QoS of web services and check simple conditions for several activities. None of these platforms succesfully cover the aforementioned issues on their own. However, under the SALMonADA platform, both platforms have been extended to fullfill the required functionality. ADA has been extended to support detection and explanation of violations, and SALMon has been extended to support automatic monitoring configuration and to provide monitoring results in a formal document. A detailed description of these platforms, including their enhancements in this approach are described in Sec. IV-C for SALMon and Sec. IV-D for ADA.

III. The SALMonADA Conceptual Reference Model

This section presents the conceptual reference model of the SALMonADA platform. This model introduces the relevant human and software agents that participate in the platform, and the data that they need to interchange. Its purpose is to provide a high-level view of the platform before going into the architectural and technological details. The conceptual reference model is shown in Fig. 1, using the SAP-TAM notation [23]. It includes the following agents:

Client: is the user of the platform. The client goal is to retrieve the explanations of SLA violations and/or the monitoring results. To fulfill such a goal the client has the responsibility of providing the SLA to monitor. The role of client may be played by the service consumer, the service provider or even a third party interested in monitoring the assessment of the SLA.

Configurator: is the agent that configures the monitor with the information included in the SLA. Thus, it decouples the SLA (a contractual specification understood by SLA-dependent agents) from the Monitor, by generating from the SLA a Monitoring Management Document (MMD), which is a specification of the monitoring directives to configure a monitor (see Sect. IV-B for more details about its structure).

Monitor: is the agent responsible of monitoring the interaction between the provider and the consumer according to the monitoring directives given in the MMD. The Monitor obtains the measured metrics from such an interaction and updates the monitoring results in the MMD right after each consumer request.

Analyser: is the agent that checks whether the monitoring results of a service, available in the updated MMD, is compliant or not with the agreed QoS included in the SLA. It ultimately produces the explanations of violations structured in a document designed for this purpose, the Service Level Fulfillment (SLF).

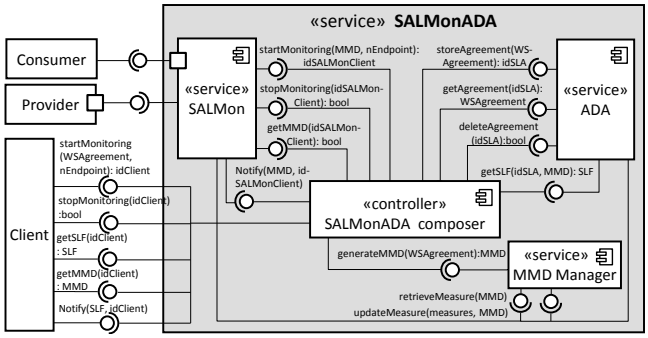


Figure 2. Architectural Model of SALMonADA

As a summary, the conceptual reference model shows a clear separation of concerns on the management of the SLAs, the MMDs and the SLFs, independent of the concrete technologies used, that are described in the next section.

IV. The SALMonADA Platform

In this section we present the details of SALMonADA platform which can be viewed as an instantiation of the conceptual reference model presented above. The platform is able to monitor and analyse expressive SLAs specified with WS–Agreement. SALMonADA has a decoupled architecture that integrates into a service-based system two previously existing systems which in turn, have been extended to realize this project: the SALMon monitor [21] and the ADA analyser [22]. Such an architecture is shown in Fig. 2. The core component of SALMonADA is its *composer* that provides the external interface and controls the execution flow of the system keeping *SALMon* and *ADA* decoupled from each other. Moreover, SALMonADA also comprises the *MMD Manager* service, which is used to generate and manipulate the MMDs independently of the underlying structure of such documents.

In the following subsections, we provide more details about these SALMonADA components. We focus on the internal architectures and responsibilities.

A. The SALMonADA composer

The SALMonADA composer is the component that presents the external interface to the client and controls the execution flow of the system. It also orchestrates the composition of ADA and SALMon to: (1) extract the monitoring information to be included in an MMD from an SLA, (2) monitor SLAs, and (3) analyse SLAs. The composer follows the low coupling design principle, so that it is possible to replace the monitor, the analyser, or the MMD manager without affecting the other elements of the platform. This implies that also the formats of SLAs and MMDs can be changed with minimal impact.

Thus, the SALMonADA composer controls the information flow needed for the different constituent services such as: (1) which client is interested in knowing which SLA violations; (2) the MMDs obtained from the SLAs through the MMD

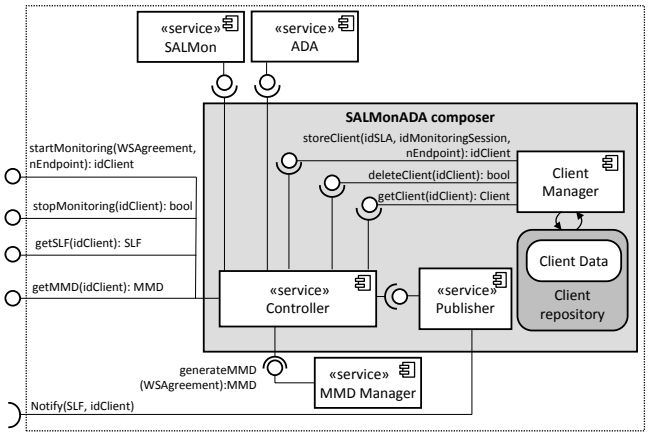


Figure 3. Architectural Model of the SALMonADA composer

```
Agreement – SALMonADA–complaint ADA SLA
  Context:
    Initiator: IneedSLAAnalysisCorp.
    Responder: ADA Tool of ISA Group, as ServiceProvider,
    ExpirationTime: 2014–01–01T00:00:00
All
 ADA–SDT: Service Description for ADAService
 //Web service information
  Description = ADA is an analysis tool for WS–Ag. docs,
  Domain = Analysis tool,
  wsdlURL = http://www.isa.us.es:8081/ADAService?wsdl
  endpoint = http://www.isa.us.es:8081/ADAService?wsdl
  operations: checkDocumentConsistency, getMetricFile,
   explainNonCompliance, xmlToWSAg4People, wsag4PeopleToXML
    ... more operations are included in the real SLA ...

 SP1: Service Property for ADAService
  AverageResponseTime – measured by metric:Float,
   //average response time of a given operation
  GeneralResponseTime – measured by metric:Float,
   //average response time of all operations
  AverageAvailability – measured by metric:Percent
   //It is the average service availability

 GTgeneralAvailability: Guaranteed by Provider
  SLO: AverageAvailability >= 95

 GTgeneralResponseTime: Guaranteed by Provider
  SLO: GeneralResponseTime <= 1.5

 GTexplainNonComplianceRespTime: Guaranteed by Provider
  Scope: explainNonCompliance,
  QC: GeneralResponseTime >= 0.5,
  SLO: AverageResponseTime < 2

 GTgeneralResponseTimeRelations: Guaranteed by Provider
  Scope: checkDocumentConsistency, xmlToWSAg4People,
         wsag4PeopleToXML, getMetricFile,
  QC: GeneralResponseTime >= 0.5,
  SLO: AverageResponseTime <= GeneralResponseTime
 ... more guarantee terms are included in the real SLA ...
```

Figure 4. Main terms of the ADA SLA

Manager; (3) the MMD with the monitoring results obtained by the monitor service; and (4) the pair SLA-MMD with the monitoring results that is required by the analyser service. Moreover, the composer provides the interface to supply to the clients both, the SLF information, and the MMD with the monitoring results to be aware of the QoS evolution at runtime.

The architectural model of the SALMonADA composer is depicted in Fig. 3[3] and comprises the following elements:

Controller: it provides the external interface to the client and controls the execution flow of the system by communicating with the SALMon and ADA services. It is also responsible to interact with the other components of the composer.

Client Manager: is responsible for storing and retrieving the different clients of SALMonADA from the client repository. Each client has associated one or more WS–Agreement documents to monitor.

Publisher: This component implements the Reference Data Observer SOA pattern [24]. In SALMonADA it notifies the SLF to any subscribed service when a violation occurs.

B. The MMD Manager service

The MMD Manager is one of the components that delimits the kind of SLA supported by SALMonADA because it generates the MMD from a given SLA. SALMonADA aims to support a general-purpose SLA specification in order to report the SLF of as many SLAs as possible. Although any other MMD Manager service can be developed supporting any SLA specification, we choose to support the WS–Agreement specification due to its flexibility. As stated in Sec. II, the WS–Agreement schema must be extended at least with two sublanguages. In our proposal we use general-purpose sublanguages [18] to make it easier to map from other sublanguages and SLA notations to our supported SLA. Specifically, we support specifications of: (1) the service by means of a simple attribute-value pairs within SDTs, and (2) the SLOs by means of an assertion language that is described in Sec. IV-D. Examples of both sublanguages are included in the SLA of Fig. 4 that comprises several SLOs relating three service properties of the ADA analysis service: the average service availability (`AverageAvailability`); the average response time of all service operations (`GeneralResponseTime`); and the average between several response time measures of a given service operation (`AverageResponseTime`). In this sense, note that the two last SLOs of the SLA guarantee a certain response time depending on the operations that are in the scope of the guarantee term. In addition, such terms are conditional terms because they are guarded by a qualifying condition (QC). That is, the SLOs are only evaluated if the QCs holds. In this case, the QCs holds when the `GeneralResponseTime` is more than 0.5 seconds.

The MMD Manager receives the WS–Agreement document and generates the corresponding MMD. Fig. 5 provides an excerpt of the MMD obtained from the WS–Agreement document of Fig. 4 and it depicts our selected MMD structure as well as which SLA information is included in each and every of its XML elements. Thus, the MMD structure is divided into four elements: (1) the *web service information* element which includes the information needed to invoke the service, that is extracted from the service description terms of the SLA; (2) the *monitor configuration* element which comprises the monitoring time interval, which starts monitoring from the generation of the MMD until the SLA expiration time; (3) the *service metrics* which are the service properties used within the SLOs whose scope is the whole service (i.e. to all service operation in the SLA); and (4) the *operation metrics* which

[3]Note that rounded squares denote stored data and repositories in SAP-TAM

```
<MonitoringManagementDocument>
 <WebServiceInformation name="SALMonADA-complaint...">
 <!--extracted from the service description term-->
  <description>ADA is an Agreement Document Analysis
               tool for WS-Agreement documents</...>
  <domain>Analysis tool</...>
  <wsdlURL>http://www.isa.us.es:8081/ADAService?wsdl</.>
  <endpoint>http://www.isa.us.es:8081/ADAService</...>
  <operation name="checkDocumentConsistency">
   <soapAction>checkDocumentConsistency</...>
  </...>
  <operation name="explainNonCompliance">
   <soapAction>explainNonCompliance</...>
  </...>
  <!--more operations were included-->
 </WebServiceInformation>

 <monitorConfiguration>
  <globalPeriodInit>2013-05-18T18:02:38</...>
  <!--starting monitoring time-->
  <globalPeriodEnd>2014-01-01T00:00:00</...>
  <!--extracted from the expiration time-->
 </...>

 <!--QoS attributes of the whole service:-->
 <serviceMetric>
  <metric>AverageAvailability</...>
  <localPeriodInit>2013-05-18T18:02:38</...>
  <localPeriodEnd>2014-01-01T00:00:00</...>
 </...>
 <serviceMetric>
  <metric>GeneralResponseTime</...>
  <localPeriodInit>2013-05-18T18:02:38</...>
  <localPeriodEnd>2014-01-01T00:00:00</...>
 </...>

 <!--QoS attributes of specific operations:-->
 <operationMetric opName="explainNonCompliance">
  <metric>AverageResponseTime</...>
  <localPeriodInit>2013-05-18T18:02:38</...>
  <localPeriodEnd>2014-01-01T00:00:00</...>
 </...>
 <operationMetric opName="checkDocumentConsistency">
  <metric>AverageResponseTime</...>
  <localPeriodInit>2013-05-18T18:02:38</...>
  <localPeriodEnd>2014-01-01T00:00:00</...>
 </...>
 <!--more operation metrics were included-->
</MonitoringManagmentDocument>
```

Figure 5. Excerpt of the MMD obtained from document of Fig. 4.

are the service properties used within SLOs whose scope is one or more specific operations but not to all of them. For instance, `AverageResponseTime` is an operation metric for the explainNonCompliance and checkDocumentConsistency operations, among others; and `AverageAvailability` is a service metric obtained from the `GTgeneralAvailabity` term. The information stored in the MMD is always accessed or modified through the MMD Manager. Therefore, the MMD structure is completely independent of the system. To achieve this, we have considered it as a variability point at design level and hence any other MMD Manager service could be used.

C. The SALMon service

SALMon is a service-based system aimed at monitoring the QoS of web services. The main features of SALMon are (see [21] for details): it may operate on any available technology (SOAP-based, RESTful, etc.) with minor and localised changes; it may interoperate easily with other frameworks (e.g., self-adaptive service-based systems [25] or cloud infrastructures [26]); it is easily extensible to monitor new metrics; it combines passive monitoring and on-line testing [15].

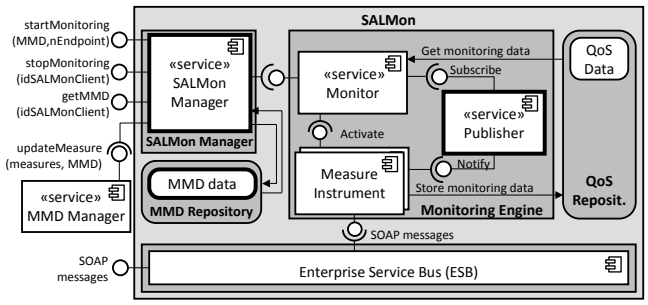


Figure 6. Architectural Model of the extended SALMon

In the context of SALMonADA, SALMon has been enhanced with new components and services (see elements with a thicker frame in Fig. 6): the SALMon Manager service, the MMD repository and the Publisher service. The existing components of SALMon have also been adapted to interact with the new services. We describe below each module:

SALMon Manager: is responsible to (1) store the MMDs in the repository, (2) configure the Monitoring Engine and (3) update the MMD when new measured metrics are retrieved. A measured metric is a metric with its value in a specific timestamp (see Fig. 7). The SALMon Manager uses the MMD Manager whenever it requires to get or store the monitoring results to the MMD. In such a way, it can be extended to support different MMD structures.

Monitoring Engine: is responsible to monitor the services. The Monitor service creates and manages the Measure Instruments. Each Measure Instrument is responsible to gather a specific metric such as response time or availability (metrics of the service from the client's perspective) and store the results in the QoS Repository. The module also includes the Publisher to notify when a new measured metric is obtained.

Enterprise Service Bus (ESB): all requests and responses are sent through this communication channel, which in turn, feeds the Measure Instruments with the intercepted messages.

QoS Repository: stores the measured metrics. To reduce storage consumption, it saves only the last measurements, the average values and the number of invocations.

The generated MMD with the monitoring results is shown in Fig. 7. Notice that the measured metrics related to time are defined in milliseconds, as a convenient unit to express the monitoring results. The platform performs afterwards the required unit conversions for the SLA analysis.

Finally, it is important to address performance and privacy issues during monitoring. The former is evaluated in depth in the experimental results in Sec. VII. The latter, which is strongly affected by the service policies and the deployment configuration, is discussed here considering the different factors involved. On secure services, the ESB does not interfere with the security level in the communications. However the issue arises for non-secure services dealing with sensitive data. For such a reason, the body of the message is never stored. Moreover, the ESB can be deployed on either the client or provider side under their management, so any sensitive data can be encrypted using WS-Security before forwarding the message to the Measure Instruments.

```
<MonitoringManagementDocument>
 ...
 <serviceMetric>
  <metric>AverageAvailability</...>
  <localPeriodInit>2013-05-18T18:02:38</...>
  <localPeriodEnd>2014-01-01T00:00:00</...>
   <measure>
    <value>100</value>
    <timeStamp>2013-05-18T18:02:38</timeStamp>
   </measure>
 </...>
 ...
 <operationMetric opName="explainNonCompliance">
  <metric>AverageResponseTime</...>
  <localPeriodInit>2013-05-18T18:02:38</...>
  <localPeriodEnd>2014-01-01T00:00:00</...>
   <measure>
    <value>3421</value>
    <timeStamp>2013-05-18T18:02:38</timeStamp>
   </measure>
 </...>
 ...
</MonitoringManagmentDocument>
```

Figure 7. Excerpt of the MMD of Fig. 5 with the monitoring results.

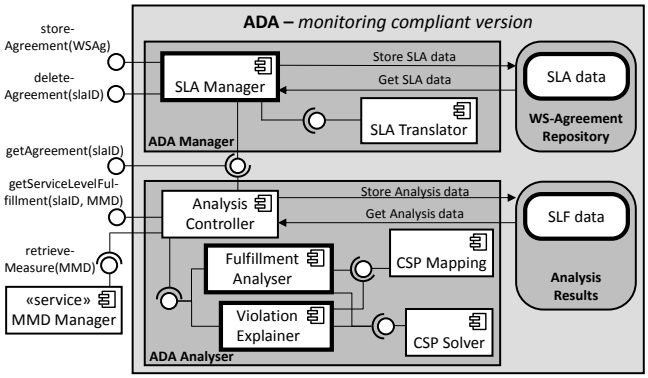


Figure 8. Architectural Model of the monitoring-compliant ADA.

```
Unfulfillment Explantion:
  Violated terms: GTexplainNonComplianceRespTime
                  (AverageResponseTime < 2 s.)
   explainedBy: AverageResponseTime = 3421 ms.
```

Figure 9. SLF for the violation of the SLA (Fig. 4) by the MMD (Fig. 7)

D. The ADA service

ADA is an *Agreement Document Analysis* framework aimed at extracting useful information from agreement documents [22]. It has been developed based on our previous theoretical works on applying the *constraint satisfaction problem* (CSP) [27] paradigm to the automated procurement of web services [28]. The main features of ADA [29] are: (1) *ready-to-use* by detecting and explaining conflicts within and between WS–Agreement documents [29], [30]; (2) *functional suitability* by supporting the analysis of expressive WS–Agreement documents with conditional, optional, or alternative terms, term scopes, arithmetic-logic expressions inside SLOs, et cetera [18]; (3) *understandability* by supporting a plain-text notation [18] that makes reading and writing WS–Agreement documents easier for humans; (4) *interoperability* through a triple distribution model (Java library, OSGi[4] service, and web service); and (5) *CSP solver independence.*

Similarly to SALMon, ADA has been enhanced with new components (see elements with a thicker frame in Fig. 8): the SLA Manager, several analysis facilities to detect and explain violations at monitoring, and repositories for SLAs and SLFs. The existing ADA components have also been adapted to interact with other components of the SALMonADA framework. ADA components are grouped into two modules with the following responsibilities.

ADA Manager: is responsible for SLA storage and retrieval from the repository; as well as the translation between several SLA models to a WS–Agreement-based normalised one using XML that ADA is able to analyse.

ADA Analyser: is responsible for: (1) analysing the fulfillment of the WS–Agreement document given the monitoring results stored in the corresponding MMD; (2) the creation of violation explanations when a violation is detected; and (3) the storage and retrieval of the SLF information. Such analysis is performed by means of a CSP solver tool (see Sec. V for more details) that supports the following assertion language:

[4]www.osgi.org

$P ::= P\ op_L\ P \mid T$**,** predicate, where $op_L \in \{\wedge \mid \vee \mid \neg \mid \Rightarrow \mid \Leftrightarrow\}$
$T ::= E\ op_C\ E$**,** term, where $op_C \in \{ = \mid \neq \mid > \mid \geq \mid < \mid \leq \}$
$E ::= E\ op_A\ E \mid var \mid lit$**,** expression, where op_A is an algebraic operator defined on the domain of variables and literals

As a result, we work with an assertion language inside WS–Agreement documents that is not just expressive but also easy to understand by non-technical users (cf. SLOs of Fig. 4). Moreover, as Fig. 9 shows, the reported SLF is also understandable because the violated terms are associated with the violating monitored values, both expressed in a human understandable way.

V. A CSP-Based Technique to Explain Violations

The detection and explanation of violations, which is the ultimate goal of this paper, is necessarily based on the alignment of the WS–Agreement document, which expresses the requirements on the service, and the MMD document, which collects monitoring information at runtime. To implement this alignment, we need to provide semantics to both documents and then define the concept of violation and the procedure to get explanations. In our technique, the semantics is defined through semantic mappings. Under this view, the elements of the documents, that are considered as source models are mapped into a target domain whose semantics have been formally defined [31]. The main advantage of semantic mappings is that they enable the usage of techniques, preferably automated, which are specific to the target domain in order to infer properties in the source models [32].

In our case, we have chosen constraint satisfaction problems (CSP) [27] as the target domain. Solving problems by means of CSPs has been a research topic in Artificial Intelligence for years. In short [33], a CSP is a three–tuple of the form (V, D, C) where V is a set of variables, D is a set of domains, and C is a set of constraints. Each variable $V_i \in V$ has a finite domain $D_i \in D$. Each constraint in C applies to a subset of the variables, and restricts the combination of values that

those variables can take at the same time. A solution to a CSP is an assignment of a value to each variable such that all the constraints are satisfied. Consider, for instance, the CSP: $(\{a,b\},\{[0,2],[0,2]\},\{a+b<4\})$. Then, $(a=2,b=0)$ is a possible solution since it verifies that $2+0<4$, whilst $(a=2,b=2)$ is not a valid solution.

The reason for selecting CSP is twofold. On the one hand, the MMD and the most significant part of an WS–Agreement document are sets of constraints over service properties and, therefore, CSPs can be used to model the detection and explanation problem in a very natural way, as we showed in previous works [28]–[30], [34]–[36]. On the other hand, there is a plethora of CSP solvers available that support a wide range of constraints and can be used to automatically analyse WS–Agreement documents in an efficient manner.

Our proposal to interpret the SLA fulfillment using CSPs is based on checking that the monitored metrics values are a possible solution for the CSP mapped from the SLA.

The proposed CSP-based technique to detect and explain SLA unfulfillments takes into consideration that those WS–Agreement terms whose scope is a specific service operation are only affected by monitored metrics measured while such operation is being executed. For example, the `AverageResponseTime` attribute of a term whose scope is the `explainNonCompliance` operation, would be checked for fulfillment with the average of monitored response time of `explainNonCompliance` operation. To consider this in our system, the original WS–Agreement document Δ is separated into several views Δ_{op} by the service operation scope. Each view is a WS–Agreement document by itself restricted to both: the set of terms whose scope is a specific service operation; and such terms whose scope are all service operations (i.e. such terms that do not specify any scope such as `GTgeneralAvailability` of Fig. 4). In the case a term whose scope includes more than one service operation, but not all of them, it would be included in as many views as scoped service operations. For instance, the term `GTgeneralResponseTimeRelations` of Fig. 4 would be included in four views, one for each scoped operation.

In general, a WS–Agreement view Δ_{op} defines some guarantees for a set of service properties whose domain has been previously defined. WS–Agreement views can be modelled in a CSP by means of the semantic mapping $map(\Delta_{op})$ that is summarised in Table II, whereas Fig. 10 shows the CSP mapped from the `explainNonCompliance` view ($\Delta_{explainNonCompliance}$) of the ADA SLA included in Fig. 4.

Concerning MMDs, the original MMD M is also separated into several views M_{op} by the monitored service operation. Thus, each view comprises the QoS monitoring result that includes metric–measure pairs of a unique service operation[5], and those affecting all service operations. Table III summarises the MMD semantic mapping $map(M_{op})$, whereas Fig. 11 shows the CSP mapped from the monitoring results of the ADA SLA corresponding to the `explainNonCompliance` operation included in Fig. 7.

[5]Note that the measure units must match for a given scale (e.g. time) in the service properties of the SLA and the MMD. If they do not match, a conversion formula should be applied for each CSP constraint, as proposed in some QoS ontologies for Web Services [37]. Fig. 15 includes different measure units for the `AverageResponseTime` property: seconds in the SLA and milliseconds in the MMD.

Table II
TERMS MAPPING FROM WS–AGREEMENT VIEW TO CSP.

WS–Agreement Element	CSP Mapping

`Agreement...`
`Context:...`
`Responder/Initiator:...`
	This information is not mapped into the CSP

`name: Service Property`
`propertyName`
`– measured by metricDefinition`
	$V \leftarrow V \cup$ `propertyName`
	$D \leftarrow D \cup$ *domain*(`metricDefinition`)
	$C \leftarrow C$

`name: Service Description` `SDTExpr`	This information is not mapped into the CSP

`name: Guaranteed by x`
`Scope: ServiceOperation',`
`QC: QCExpr,`
`SLO: SLOExpr`
	$V \leftarrow V$
	$D \leftarrow D$
	$C \leftarrow C \cup$ (`QCExpr`⇒`SLOExpr`)

`name: Guaranteed by x`
`Scope: ServiceOperation',`
`SLO: SLOExpr`
	$V \leftarrow V$
	$D \leftarrow D$
	$C \leftarrow C \cup$ (`SLOExpr`)

Table III
MONITORING QOS DATA MAPPING FROM MMD VIEW TO CSP.

Measure	CSP Mapping
`metric = value` `for ServiceOperation'`	$V \leftarrow V \cup$ `metric` $D \leftarrow D \cup$ *domain*(`metric`) $C \leftarrow C \cup$ (`metric = value`)

In order to carry out the detection and explanation of SLA unfulfillment, we need to use a pair of analysis techniques that have been widely used in CSPs:

- $solve(V,D,C)$ tries to find all CSP solutions. To this end, many heuristics and techniques have been developed to solve CSPs in an efficient manner [38], [39].
- $explain(V,D,C)$ tries to provide an explanation when such solution is not possible [40]. This explanation is a minimal set of constraints $c \in C$ that makes impossible to find a solution that satisfies C. For instance, the CSP $(\{a,b,d\},\{[0..2],[0..2],[0..2]\},\{a+b<1,a>0,d>1\})$ is not satisfiable, and its possible explanations c are either $\{(a+b<1)\}$ or $\{(a>0)\}$.

On the basis of these operations and the previously described map functions for the SLA and MMD, we may trace back in order to infer the SLA fulfillment with the MMD measures, providing the precise semantics for detecting the unfulfillment and their explanation, as follows. Let $map(\Delta_{op}) = (V,D,C)$ and $map(M_{op}) = (V',D',C')$

$$unfulfillment(\Delta_{op}, M_{op}) \Leftrightarrow solve(V \cup V', D \cup D', C \cup C') = \varnothing$$

In the example above, the WS–Agreement view $\Delta_{explainNonCompliance}$ is not fulfilled by the MMD measures $M_{explainNonCompliance}$, due to the lack of solutions for the joint CSP generated.

In this case, an explanation like such provided in Fig. 9 would be reported using the following formula:

$$unfulfillment_{exp}(\Delta_{op}, M_{op}) = explain(V \cup V', D \cup D', C \cup C')$$

V = { AverageResponseTime, GeneralResponseTime, AverageAvailability }
D = { [1..∞], [1..∞], [1..100] }
C = { AverageAvailability>=95, GeneralResponseTime<=1.5, GeneralResponseTime>0.5 ⇒ AverageResponseTime<2 }

Figure 10. CSP generated from WS–Agreement view: $\Delta_{explainNonCompliance}$.

V = { AverageResponseTime, GeneralResponseTime, AverageAvailability }
D = { [1..∞], [1..∞], [1..100] }
C = { AverageAvailability=100, GeneralResponseTime=1.7, AverageResponseTime=3.421}

Figure 11. CSP generated from MMD view: $M_{explainNonCompliance}$.

The algorithm is applied subsequently to the different views of both documents and trace the CSP constraints back to the corresponding WS–Agreement term or MMD measure. An example of the final result is provided in Fig. 9 and in Sec. VI.

VI. SALMonADA in Use

SALMonADA is designed and developed to support push and pull interaction styles with its clients. Thus, a client, based on its own needs, may choose the approach that best fits them. Independently of the selected approach, a client shall start the use of SALMonADA by subscribing as such a client. Similarly, the client shall stop the SALMonADA monitoring at the end of the interaction, to be unsubscribed as client.

As the sequence diagram of Fig. 13 shows, the client starts the monitoring process by providing a WS–Agreement document to monitor its fulfillment. In case of push approach, the client shall also provide the endpoint where the notification is awaited. Independently of the approach used, *ADA* stores the WS–Agreement document. This document is also sent to the *MMD Manager* for it to generate the MMD. Such MMD will store the information to configure the monitor and all the measured metrics obtained while monitoring the service agreed in the WS–Agreement document. Afterwards, a monitoring session is started in *SALMon* providing the endpoint where the monitoring result must be notified to update the MMD after a service consumption. Finally, a client identifier (`clientID`) is generated to denote the specific monitoring session for this SLA. If several SALMonADA clients wanted to monitor a unique SLA, for instance the service consumer and provider to be informed about the SLA fulfillment, a different monitoring session would be started and thus, a different `clientID` would be returned.

When a client wants to be unsubscribed from SALMonADA, it provides its `clientID` to stop its monitoring process. The WS–Agreement document and the monitoring session is removed from SALMonADA by *ADA* and *SALMon* if there is no other client monitoring the same SLA, otherwhise SALMonADA will keep them while in use by these other clients. In any case, the `clientID` is removed from the system.

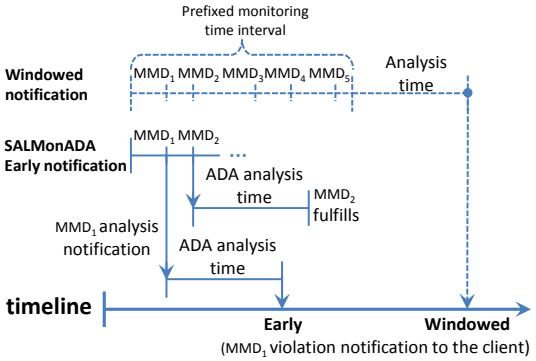


Figure 12. Early analysis notification vs. windowed analysis notification.

A. Push Interaction Approach

The *push approach* is the most convenient way to interact with SALMonADA due to the push nature of its service monitoring and analysing. In this sense, as Fig. 12 shows, the platform incorporates an *early analysis notification* that supports the analysis of the SLA fulfillment as soon as new measured metrics have been updated in the MMD. Thus, the SLF notification is sent to the client without any further delay than the analysis time, in contrast with windowed proposals [13] that get periodically the monitored values in a prefixed time interval to analyse them later. Assuming a similar analysis time, the difference between both approaches is higher when the violation affects to a measured metric at the beginning of the prefixed time interval (see MMD_1 in Fig. 12). In the case that the violation affects to a measured metric at the end of the prefixed time interval, the notification will be delivered at the same time in both approaches. As the sequence diagram of Fig. 14 depicts, once the client has started to monitor, the *provider service* included in the reported WS–Agreement document is monitored by the *SALMon* service. Next, the MMD created from the monitored WS–Agreement document is sent to the *MMD Manager* with the measured metrics to be updated. Finally, the new MMD is notified to the *SALMonADA composer* that sends it to the *ADA* service to analyse the service level fulfillment of the corresponding WS–Agreement document (cf. Sec. IV-D for more details). Then, the client is notified about the WS–Agreement document fulfillment by means of the SLF. If the WS–Agreement is not fulfilled, both the specific violated WS–Agreement terms and the violating metrics, are included as violation explanation. Note that SALMonADA supports the same endpoint acting as different clients, for instance, one of them to get the SLF, and another to store reputation analytics of the service consumer and provider, or even to perform self-adaptation strategies.

For instance, let us suppose that SALMonADA is monitoring the WS–Agreement document of ADA depicted in Fig. 4. That document specifies that some operations have a higher priority and are required to be faster than the average response time of the different methods of the service (*AverageResponseTime* <= *GeneralResponseTime*). If there was a violation, SALMonADA would report an explanation right after detecting it. The explanation would identify if the

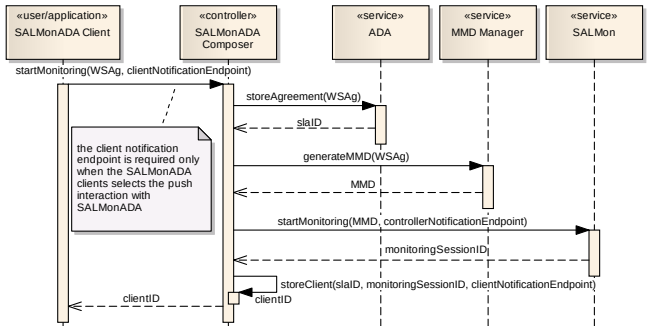


Figure 13. A SALMonADA client starts monitoring.

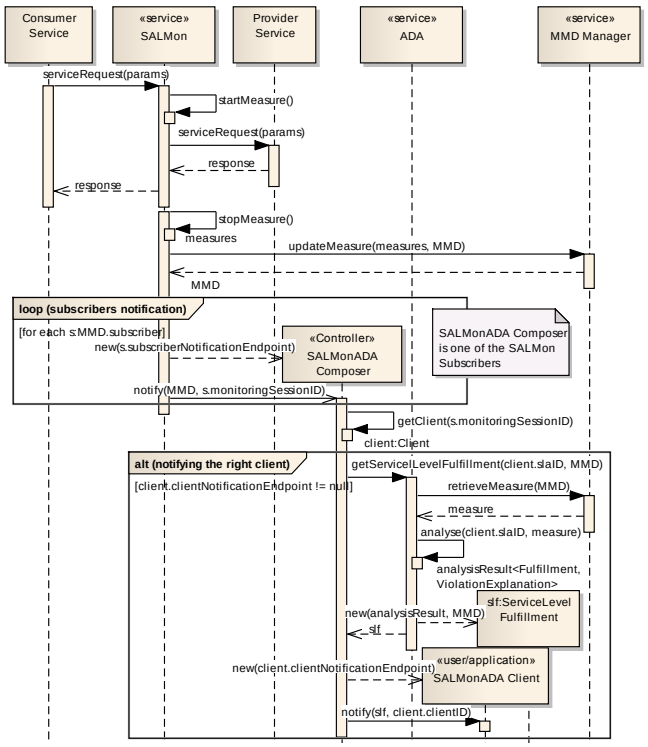


Figure 14. Push SALMonADA approach.

violating metric is either *AverageResponseTime* or *GeneralResponseTime* because a simple identification of the violated term is not enough to grasp the violation cause. Similarly, SALMonADA supports the explanation of violations of more expressive SLOs. For instance, the provider may guarantee a different average response time limit for the slower service operations, depending on the general response time of the service, as follows: ((*GeneralResponseTime >= 0 AND GeneralResponseTime < 2*) *IMPLIES* (*AverageResponseTime < 3*)) *AND* ((*GeneralResponseTime >= 2 AND GeneralResponseTime <= 4*) *IMPLIES* (*AverageResponseTime < 5*)).

B. Pull Interaction Approach

The *pull approach* allows the client to actively request the results of SALMonADA for either: the current MMD with the most recent monitoring results obtained by *SALMon*; or the current SLF of the WS–Agreement document analysed by *ADA*. The former document is obtained by invoking the

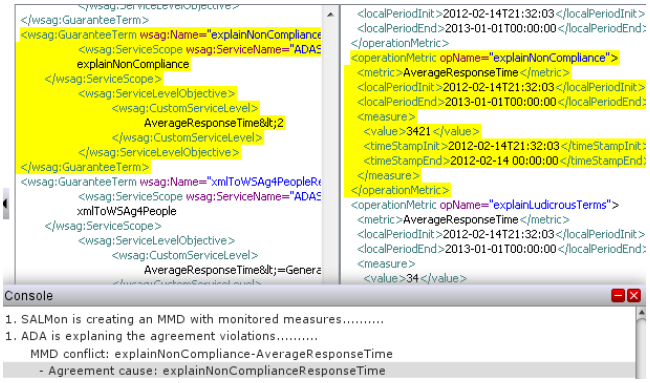


Figure 15. Reporting a violation with the SALMonADA client

getMMD method, whereas the latter is through the getServiceLevelFulfillment method. In both cases, the client is required to provide the `clientID` as input.

As usual in pull approaches, if the consumer(s) do not invoke the service, there is not new monitoring information and thus, it is possible that the client gets the same monitoring information in consecutive MMD requests.

For demonstration purposes, we have implemented a web application[6] as a SALMonADA client in order to specify or upload the WS–Agreement documents to monitor, execute SALMonADA and receive the results. In this web application, we have introduced the WS–Agreements of ADA and SALMon themselves. By monitoring the SLAs of these services, we assess on the one hand, the functionality of SALMonADA, and on the other, the non-functional aspects of its main components. Such a SALMonADA client uses the pull interaction because the user press the corresponding interface controls to get the MMD and the analysis results. Moreover, as part of the demonstration and to assure that the service subject of the SLA is being requested, we have simulated the consumers that execute ADA and SALMon services.

With the client identifier, the SALMonADA client can get, at any time, the MMD, check if a violation has occurred, and in such a case, receive an explanation of the violation.

As Fig. 15 depicts, the web application highlights as violation explanation that the *AverageResponseTime* of *explainNonCompliance* operation is the violating metric because it was measured as *3421* milliseconds, while the guarantee term obligates the provider to respond in less than *2* seconds. As stated in Sec. V, SALMonADA handles different measurement units and the required unit transformation is performed to evaluate the conditions.

VII. Performance and Scalability Evaluation

In this section we evaluate both the performance and scalability of SALMonADA. Particularly, we focus on the overhead introduced by including SALMonADA within the consumer-provider interaction, and the maximum number of service request it is able to handle without incrementing such overhead. To do so, we first introduce how the components of its architectural model (see Fig. 2) affects the performance:

[6]The SALMonADA web application can be tried at www.isa.us.es/ada.source/SLAnalyzer/. A screencast is available at gessi.lsi.upc.edu/salmon/ada/

SALMonADA composer, MMD Manager and ADA: We have implemented two strategies to avoid them introducing overhead: (1) they are executed concurrently, without interfering the service consumption, because the response of the provider is returned to the consumer before analysing the SLA fulfillment (see Fig. 14), and (2) they are deployed in a different location from the monitored service, and hence they do not share the same resources (e.g. CPU, RAM, etc.). Consequently, although these components might be time-consuming (e.g. the analysis of the SLA), they do not introduce an overhead over the response time of the monitored service.

SALMon: The ESB Apache Synapse included in SALMon adds a low overhead while handling the HTTP messages. The ESB has a non-blocking HTTP transport and multi-threaded mediation, which as we measured, results in a negligible 1 - 3 ms overhead. Nevertheless, in our approach, there are three possible locations where SALMon can be deployed: at the server side, at the client side, or in an intermediate server (i.e. in the middle). Depending on the location, the overhead experienced by the consumer varies.

If SALMon is placed at the server or client side, there is an overhead on the resources due to the execution of the monitoring components. However, this overhead can be easily compensated by adding more resources.

If SALMon is placed in the middle, it does not produce an overhead on the resources of the client or server side. However, the deployment of SALMon in an intermediate server adds a network delay from Internet Service Providers due to the redirection of the messages. In this scenario, SALMonADA is not responsible of the overhead introduced by the network, but under heavy usage the components of SALMon might experience a bottleneck. To quantify it, we evaluate by means of an adequate benchmark, (1) the response time overhead under normal operating conditions (i.e. one invocation at a time), and (2) the maximum throughput SALMonADA is able to handle without incrementing such overhead.

a) Setting up the experiment: To perform the evaluation, we invoke a set of real services, and compare the response time by invoking the services both directly and through SALMon. The agents involved in the experiment are the monitored services, the client and SALMon.

To obtain a set of representative services, we started from a list of 393 services available in a public repository[7]. Then we applied the following criteria: (1) We first considered the most recently submitted services under the assumption that recent services are more likely to be available and running than older services. Considering the length of the list, we established as threshold the 1/3 of the complete list. (2) From the resulting 131 services, we removed those ones falling into any of the following situations: were not available, were payment services, required registration or didn't have stateless operations, resulting in 23 services. (3) We tested these 23 services and removed those ones that had errors in their descriptions (WSDL), or that gave faulty results in their functionality when invoked, resulting in a final list of 11 services from 8 different service providers, deployed on their respective servers (i.e. out of the control of the experiment), see Appendix A for the full list.

[7]http://www.xmethods.net/ve2/Directory.po

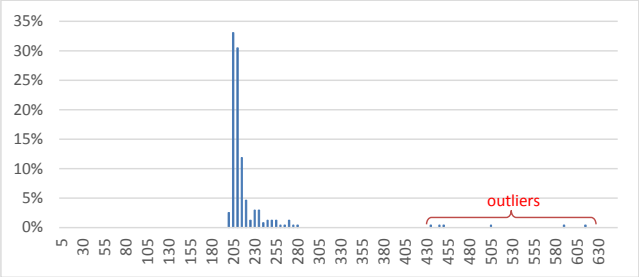


Figure 16. Response time distribution of ISBN service

The client that invokes those services has been deployed in Seville in an Intel i7 of 2,20 GHz, 8 GB of RAM, a download speed of 13,92 Mbps and upload speed of 8,15 Mbps.

SALMon has been deployed in Barcelona in a dedicated server Intel 2,6Ghz, 6GB RAM, a download speed of 56,57 Mbps and upload speed of 15,71 Mbps.

b) Conducting the experiment: We first conducted the experiments under normal conditions executing 100 service calls per each service in both direct and redirected forms by using synchronous calls (i.e. one invocation at a time). Then, we conducted asynchronous calls to test the maximum throughput of the system starting from a throughput of 1 invocation per second to a throughput of 100 invocations per second. We performed 100 invocations per each throughput in both direct and redirected forms. Some services have limitations to be tested under these circumstances (e.g. restrictions on the number of concurrent invocations) and the scalability have been analysed on the services that didn't have these limitations (6 of the 11 services).

c) Analysing the results: One key issue regarding the analysis of the results is dealing with outliers (e.g. network failures that increase the response time of an invocation). Commonly used methods to deal with outliers require that the data follow a Gaussian distribution [41]. However, from the experiment results we have observed that response times do not follow a Gaussian distribution, but an exponentially modified Gaussian or inverse Gaussian distribution. For instance, the distribution of the response time of one of the monitored services is depicted in Fig. 16. As shown, the population grows rapidly on the left-hand side and decreases slowly on the right-hand side in the form of a tail. Those elements that are far away from the mean are considered outliers. To deal with these outliers, we followed the methods described and evaluated by Ratcliff for dealing with response time outliers [42]. Although Ratcliff studied response time of people in the field of psychology, the results can be applied to any model that follows the inverse Gaussian distribution. According to Ratcliff, we will not compute directly the average response time (which is not a robust estimator in front of outliers), but we will use two other robust estimators, namely, the inverse transformation and removing outliers at a standard deviation distance. The first estimator consists on applying the inverse response time (1/R) on each individual invocation, calculate the average, and then invert the result. The second

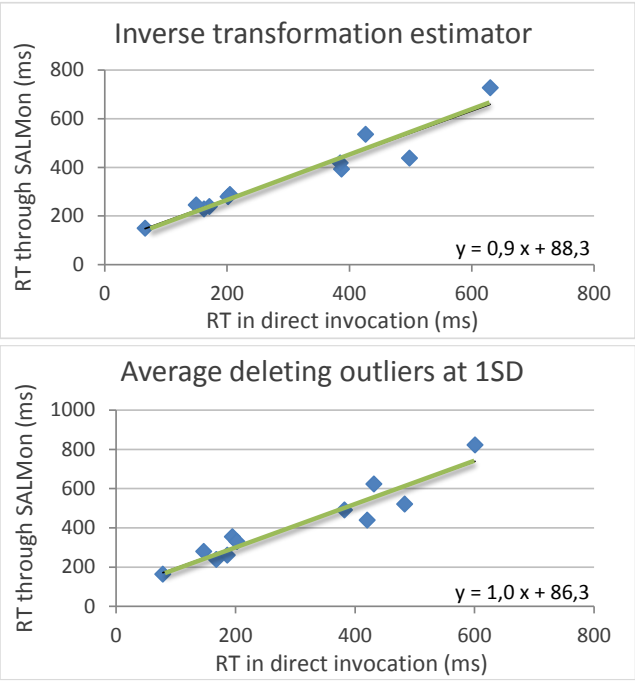


Figure 17. Response time of the service with SALMon with respect to direct invocation

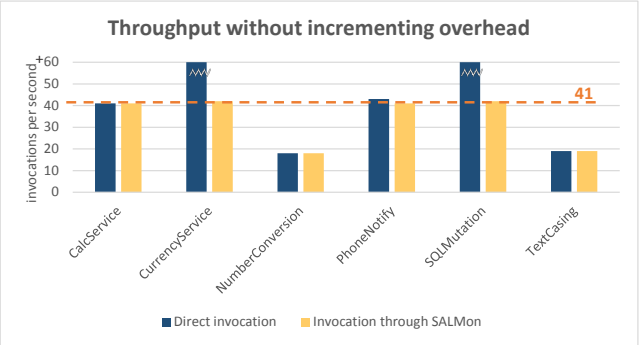


Figure 18. Throughput without incrementing response time

estimator consists on calculating the average response time after removing the outliers at a standard deviation distance. We computed these methods over the invocations on each service for both directed and redirected invocations. As a result, we got two robust estimators per each service. We applied these estimators to the response time of direct and redirected invocations in order to calculate the response time overhead introduced in the service interaction by the deployment of SALMon in the middle. We decided to relate the two parameters with a linear interpolation curve fitting method with the aim of obtaining mathematical functions approximating the response time overhead. Fig. 17 shows the obtained functions for each of the two applied robust estimators methods, which are: $y = 0.9x + 88.3ms$ and $y = 1.0x + 86.3ms$.

Then, using the same estimators, we calculated what is the maximum throughput without incrementing such overhead. As shown in the results of Fig. 18, the maximum throughput of redirected invocations is 41 invocations per second. Above this number, the overhead response time raises significantly. Notice that some monitored services present a lower maximum throughput, but as shown, it is because they are unable to process such amount of invocations (either direct or redirected). We conclude that the response time overhead of the deployment of SALMon in the middle has a constant value betweeen 86 and 89 ms, and its maximum throughput is 41 invocations per second.

We must remark that this overhead does not interfere with the SLA analysis, because the monitored response time corresponds to the real response time of the service. Nevertheless, as the consumer of the service experiences this delay, it is worthy to mention some deployment strategies to mitigate any concern. On the one hand, deploying SALMon at the client or server side is suitable for service consumers that require extremely fast response times and need to avoid the 86-89 ms overhead caused by the network delay. Moreover, as discussed before, this deployment is also suitable for services dealing with sensitive data. On the other hand, for other types of services, we argue that a deployment in the middle is preferred, since this solution is less intrusive to both the client and the provider server, as it does not require the installation of the monitor in their infrastructures. Moreover, if a throughput higher than 41 requests per second is required, multiple instances of the ESB can be replicated in different servers under the same SALMon platform.

VIII. Conclusions and Discussion

In this paper we have presented a solution to monitor and analyse SLAs in order to provide timely detection and comprehensive explanations of their violations. Such information is really appealing for decision-making activities performed at runtime. For instance, the consumers of AmazonS3 scenario mentioned in Sec. I would be benefited from using SALMonADA because they are able to decide if they have to claim for a reward or not due to the service level fulfillment information provided. Other scenarios that may benefit from our proposal are the renegotiation of SLAs or the adaptation of SBSs. Our solution addresses satisfactorily the different issues identified in Sec. II:

- SALMonADA's supported SLAs are general-purpose because they follow the WS–Agreement [16] structure, completed with general-purpose sublanguages [18]. Moreover, the used notation is more human understandable than such proposed in [2], [3], [9], [11], [13], [14], [19] making it easier to be managed by human users.
- The SALMonADA platform is able to extract automatically from the SLA the information needed to configure the monitor in order to detect violations. This is done through a document, the Monitoring Management Document (MMD), which supports decoupling the SLA structure from the monitor service.
- SALMonADA uses the MMD itself to store the monitoring results coming from the monitor. The advantage of our approach is to have the MMD as the unique document that centralise all monitoring-related information without coupling to a specific API or query language.

- We introduce the Service Level Fulfillment (SLF) as a document to explain accurately the violations by identifying explicitly the violated terms and the violating monitored results. The explanations are computed through the application of a powerful CSP-based mechanism. SALMonADA supports reporting such SLF information either as soon as a violation is detected (push), or when the client requests it (pull).
- SALMonADA's architecture keeps the monitor and analyser services decoupled from each other allowing thus independent evolution and eventually selective substitution. The MMD and SLF documents support this decoupling.
- The organization of SALMonADA as a SOA makes the interoperability of the platform with other tools easier.

Moreover, we have performed an evaluation of the impact that SALMonADA has over the performance of the service consumption. The evaluation has been performed over real services using suitable estimators for response time to evaluate both its overhead and scalability. Although a low overhead is added either in the resources consumption or in the service response time, we expose how to mitigate it either by adding more resources, or deploying SALMonADA in alternative locations. Despite such an overhead, the client and the service provider are benefited from using our proposal. On the one hand, the client gets the real QoS of the service, which is required for several activities, such as self healing, claiming rewards due to SLA penalties, et cetera. On the other hand, the service provider can manage its services with the knowledge of the QoS, which is helpful to take appropriate decisions such as adding more resources to a specific service, renegotiating the SLA, et cetera.

As Future work we plan to study how to integrate SALMonADA on different self healing systems as such proposed in [13], [19]. In doing so, we will demonstrate that SALMonADA can be easily deployed with other monitors and analysers, as well as supporting other MMDs and SLAs.

ACKNOWLEDGMENT

This work has been partially supported by: S–Cube, the European Network of Excellence in Software Services and Systems; the European Commission (FEDER); the Spanish Government under the CICYT projects SETI (TIN2009–07366), TAPAS (TIN2012–32273) and ProS–Req (TIN2010–19130–C02–01); and by the Andalusian Government under the projects THEOS (TIC–5906) and ISABEL (P07–TIC–2533).

REFERENCES

[1] M. P. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, eds., *Service Research Challenges and Solutions for the Future Internet - S-Cube*, vol. 6500 of *LNCS*, Springer, 2010.

[2] A. Keller and H. Ludwig, "Defining and monitoring service level agreements for dynamic e-business," in *Proc. of the 16th USENIX System Administration Conference*, no. November, 2002.

[3] A. Keller and H. Ludwig, "The WSLA Framework : Specifying and Monitoring Service Level Agreements for Web Services," *Network*, vol. 11, no. 1, pp. 57–81, 2003.

[4] M. Comuzzi and C. Kotsokalis, "Establishing and monitoring SLAs in complex service based systems," *Web Services*, 2009.

[5] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Comprehensive qos monitoring of web services and event-based sla violation detection," in *4th Int. Workshop on Middleware for Service Oriented Comp.*, 2009.

[6] B. Pernici, "Adaptation of web services based on QoS satisfaction," *Service-Oriented Computing*, pp. 65–75, 2011.

[7] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of service-based applications based on process quality factor analysis," in *ICSOC Workshops*, vol. 6275 of *LNCS*, 2009.

[8] O. Moser and F. Rosenberg, "Non-intrusive monitoring and service adaptation for WS-BPEL," in *Proc. of the 17th Int. Conf. in WWW*, pp. 815–824, 2008.

[9] M. Palacios, J. Garcia-Fanjul, J. Tuya, and C. De La Riva, "A Proactive Approach to Test Service Level Agreements," *5th Int. Conf. on Software Engineering Advances*, pp. 453–458, 2010.

[10] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno, "Search-based testing of service level agreements," in *Proc. of the 9th Conf. on Genetic and evolutionary computation*, pp. 1090–1097, ACM, 2007.

[11] SLA@SOI, "Reference architecture for an sla management framework (whitepaper of deliverable d.a1 a framework architecture)," 2011.

[12] M. Wilson et al., "Trustcom framework v4," 2007.

[13] K. Mahbub and G. Spanoudakis, "Monitoring *ws-agreement* s: An event calculus-based approach," in *Test and Analysis of Web Services*, pp. 265–306, Springer, 2007.

[14] G. Spanoudakis and K. Mahbub, "Non-intrusive monitoring of service-based systems," *International Journal of Cooperative Information Systems*, vol. 15, no. 3, pp. 325–358, 2006.

[15] S. Benbernou, L. Cavallaro, M. Sahid-Hacid, R. Kazhamiakin, G. Kecskemeti, J-L. Poizat, F. Silvestri, M. Uhlig, B. Wetzstein, "State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs (S-Cube deliverable PO-JRA-1.2.1)," 2008.

[16] A. Andrieux et al., "Web Services Agreement Specification (WS-Agreement) (v. gfd-r.192)," 2011. OGF - Grid Resource Allocation Agreement Protocol WG.

[17] A. Sahai, V. Machiraju, M. Sayal, A. P. A. v. Moorsel, and F. Casati, "Automated sla monitoring for web services," in *13th IEEE International Workshop on Distributed Systems: Operations and Management*, 2002.

[18] C. Müller, A. Durán, M. Resinas, A. Ruiz-Cortés, and O. Martín-Díaz, "Experiences from building a ws–agreement document analyzer tool," Tech. Rep. ISA-10-TR-03, http://www.isa.us.es/, Jul 2010.

[19] M. Comuzzi and G. Spanoudakis, "Dynamic set-up of monitoring infrastructures for service based systems," pp. 2414–2421, ACM, 2010.

[20] F. Raimondi, J. Skene, and W. Emmerich, "Efficient online monitoring of web-service slas," in *16th ACM SIGSOFT Int. Symposium on Foundations of software engineering*, (New York, NY, USA), ACM, 2008.

[21] M. Oriol, X. Franch, J. Marco, and D. Ameller, "Monitoring adaptable soa-systems using salmon," in *Workshop on Service Monitoring, Adaptation and Beyond (Mona+)*, pp. 19–28, 2008.

[22] C. Müller, M. Resinas, and A. Ruiz-Cortés, "A Framework to Analyse WS–Agreement Documents," in *4th Workshop on Non-Functional Properties and SLA Management in SOC (NFPSLAM-SOC'10)*, 2010.

[23] S. AG, *Standardized Technical Architecture Modeling: Conceptual and Design Level. Version 1.0.* SAP, 2007.

[24] T. Erl, *SOA Design Patterns*. Prentice Hall PTR, 1st ed., 2009.

[25] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl, "Usage-based online testing for proactive adaptation of service-based applications," in *COMPSAC*, pp. 582–587, IEEE Comp. Soc., 2011.

[26] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, M. Rodríguez, O. Mercè, A. Marosi, J. Marco, and X. Franch, "Enhancing federated cloud management with an integrated service monitoring approach," *Journal of Grid Computing*, pp. 1–22, 2013.

[27] E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, 1995.

[28] A. Ruiz-Cortés, O. Martín-Díaz, A. Durán, and M. Toro, "Improving the Automatic Procurement of Web Services using Constraint Programming," *I. J. on Cooperative Information Systems*, vol. 14, no. 4, 2005.

[29] C. Müller, M. Resinas, and A. Ruiz-Cortés, "Automated Analysis of Conflicts in WS–Agreement Documents," *in press Transactions on Services Computing*, 2012.

[30] C. Müller, M. Resinas, and A. Ruiz-Cortés, "Explaining the Non-Compliance between Templates and Agreement Offers in WS-Agreement*," in *Proc. of the 7th Int. Conf. on SOC*, pp. 237–252, 2009.

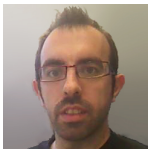
[31] D. Harel and B. Rumpe, "Meaningful modeling: What's the semantics of "semantics"?," *IEEE Computer*, vol. 37, no. 10, pp. 64–72, 2004.

[32] J. Rivera, E. Guerra, J. de Lara, and A. Vallecillo, "Analyzing rule-based behavioral semantics of visual modeling languages with Maude," in *Software Language Engineering*, LNCS, pp. 54–73, 2009.

[33] A. Swearngin, B. Choueiry, and E. Freuder, "A reformulation strategy for multi-dimensional csps: The case study of the set game," 2011.

[34] O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, D. Benavides, and M. Toro, "Automating the Procurement of Web Services," in *Proc. of The 1st Int. Conf. on Service-Oriented Computing*, LNCS, pp. 91–103, 2003.
[35] O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, and C. Müller, "An approach to temporal-aware procurement of web services," in *Proc. of The 3rd Int. Conf. on Service-Oriented Computing*, pp. 170–184, 2005.
[36] C. Müller, A. Ruiz-Cortés, and M. Resinas, "An Initial Approach to Explaining SLA Inconsistencies," in *Proc. of the 6th Int. Conf. on Service-Oriented Computing (ICSOC)*, pp. 394–406, 2008.
[37] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A qos-aware selection model for semantic web services," in *Proc. of The 4th Int. Conf. in Service-Oriented Computing* (A. Dan and W. Lamersdorf, eds.), vol. 4294 of *LNCS*, pp. 390–401, Springer Berlin / Heidelberg, 2006.
[38] F. Laburthe, N. Jussien, G. Rochart, H. Cambazard, C. Prud'homme, A. Malapert, and J. Menana, "Choco constraint solver web site," 2010. http://www.emn.fr/z-info/choco-solver/index.html.
[39] IBM, "IBM ILOG web site," 2013. http://www-01.ibm.com/software/websphere/ilog/.
[40] U. Junker, "Quickxplain: preferred explanations and relaxations for over-constrained problems," in *Proc. of the 19th Int. Conf. on Artifical intelligence*, pp. 167–172, AAAI Press, 2004.
[41] D. Hawkins, *Identification of outliers*. Chapman and Hall London, 1980.
[42] R. Ratcliff, "Methods for dealing with reaction time outliers.," *Psychological bulletin*, vol. 114, pp. 510–532, Nov. 1993.

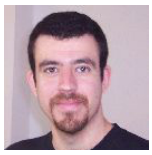
Carlos Müller is a Research Assistant and member of the Applied Software Engineering Group (ISA, www.isa.us.es) at University of Sevilla, Spain. His current research line includes service oriented computing, specifically the automated analysis of service level agreements and the application of such analysis at SLA design and monitoring.



Marc Oriol is a PhD student in Computer Science and member of the GESSI research group at the at Universitat Politècnica de Catalunya (UPC), Spain. He obtained a Msc in Computing from this University. His current research lines include Service-Oriented Computing, Quality-of-Service and monitoring.

Xavier Franch is Associate professor and Head of the GESSI research group at the Universitat Politècnica de Catalunya (UPC), Spain. He obtained his PhD and Msc in Informatics from this University. His current research lines include Service-Oriented Computing, Requirements Engineering, Software Quality and Software Architecture, among others.

Jordi Marco is Associate professor and member of the GESSI research group at the Universitat Politècnica de Catalunya (UPC), Spain. He obtained his PhD and Msc in Computing from this University. His current research lines include Service-Oriented Computing, conceptual modelling, container libraries, and computer graphics.



Manuel Resinas is a Lecturer at the University of Sevilla, Spain. He obtained his PhD in Computer Science from this University. His current research lines include analysis and management of service level agreements, business process compliance, and process performance management. Previously, he worked on automated negotiation of service level agreements.

Antonio Ruiz-Cortés is Associate Professor and Head of the Applied Software Engineering Group (ISA, www.isa.us.es) at University of Sevilla, Spain. He obtained his PhD in Computer Science from this University. His current research lines include service oriented computing, software product lines, and business process management.

Marc Rodríguez is an undergraduate student and PAS researcher of the GESSI research group at the Universitat Politècnica de Catalunya (UPC), Spain. His current research lines include Service Oriented Computing, Quality-of-Service and monitoring.