Semantic Enabled Framework for SLA Monitoring

Kaouthar FAKHFAKH[1-2-3], Saïd TAZI[1-2], Khalil DRIRA[1,2]
[1]CNRS- LAAS
[2]Université de Toulouse; UT1, UPS, INSA, INP, ISAE; LAAS;
7 avenue du colonel Roche, F31077 Toulouse, France
{kchaari, tazi, khalil}@laas.fr

Tarak CHAARI[3] and Mohamed JMAIEL[3]
[3]National Engineering School of Sfax
ENIS-ReDCAD
Informatics and applied Mathematics department,
Route de la Soukra, B.P. W, 3038 Sfax, Tunisia
tarak.chaari@redcad.org
mohamed.jmaiel@enis.rnu.tn

Abstract--**Defining clear Quality of Service agreements between service providers and consumers is particularly important for the successful deployment of service-oriented architectures. The related challenges include correctly elaborating and monitoring QoS-aware contracts (called SLA: Service Level Agreement) to detect and handle their violations. In this paper, first, we study and compare existing SLA-related models. To address the insufficiencies of these models, we propose a complete, generic and semantically richer ontology-based model of Service Level Agreements. In this model, we use the SWRL language (Semantic Web Rule Language) to express SLA obligations. This language facilitates the SLA monitoring process and the eventual action triggering in case of violations. In a second step, we use our SLA model to automatically generate semantic-enabled QoS obligations monitors. The main algorithms that perform the monitoring process are presented in this article. We implement these algorithms in an automatically generated service-oriented architecture. Finally, we believe that this work is a step ahead to the complete automation of SLA management process.**

Keywords—Service Level Agreements; ontology-based model; SOA; SLA monitoring; QoS contracts

I. INTRODUCTION

Service Level Agreements (SLAs) have become very important in the information technology area of business firms. SLAs are used with increasing frequency in general application integrations, e-commerce, outsourcing and B2B deployments. As firms increased their outsourcing of IT services, SLAs become the primary management tool for governing the relationship among the provider and its consumers. The emergence of software as a service, especially Web service, has also spurred the development of service level agreements. As more business software moves to a Web delivery platform, SLAs became the primary tool that regulates the relationship between providers and consumers when they use software services.

Metrics like processing time, messages per hour, rejected transaction counts and queries per day are common examples of defined service qualities which may be measured either at end-points, or by an intermediary. These measurements are then typically compared by an enforcement process or application to the desired level. An action should be taken according to this comparison. This action can be simply gathering and reporting results, identifying and forwarding SLA violations, or changing service behavior based on current SLA conformance.

Monitoring of SLAs between providers of a service (for example on-line banking, auctioning, ticket reservation, etc.) and consumers is a topic that is gaining in importance for business success over the Internet. SLA monitoring involves the collection of statistical metrics about the performance of a service to evaluate whether the provider is delivering the level of QoS stipulated in a contract signed between the provider and the consumer. In this context, the monitoring and the management of SLAs and their related services are crucially important. Our work focuses on the required models and software tools to monitor the QoS obligations specified in these contracts and to react to the violations or failures in the system. In this paper, we focus on a generic ontology [1] development to assist the preparation of QoS contracts and to monitor the agreements and the specified obligations on these contracts. The choice of ontology is driven by its potential to facilitate the establishment of service level agreements between the different knowledge levels of service providers and consumers. In addition, ontology implementations, using open standards like OWL (Web Ontology Language) [2] and SWRL (Semantic Web Rule Language) [3], provide a common understandable language for machines and humans. They also facilitate the contract obligations expression and the necessary inferring to take the appropriate actions in case of violations.

In Section II of this paper, we start by defining the principles of the service level agreements, their structure, their establishment and their existing implementations. In Section III, we present the main SLA related existing models. In Section IV, we detail our service level agreement's generic model that we called *SLAOnt*. Then, in Section V, we explain how this model is used to monitor its obligation instances. We present also the simplified architecture and the main algorithms that perform the monitoring process. In Section VI, we present the SLA monitoring API (called *SLAOntAPI*) that we have developed to implement the monitoring algorithms. Before concluding, in Section VII, we give a simple instantiation example of our model and we show how we have monitored its obligations using our SLA monitoring prototype.

II. SLA PRINCIPLES

In this section, we present the definition of a Service Level Agreement (SLA), its structure, life-cycle and its main implementing languages that we can find in the literature.

A. *Definition: Service Level Agreements (SLA)*

Debusmann and al., in [4], define the term SLA as a contract that exists between consumers and their service provider, or between service providers. It records the common understanding about services, priorities, responsibilities, guarantee, and the quality level of the service according to all these parameters. For example, it may specify the levels of availability, serviceability, performance, operation, or other attributes of the service like billing and even penalties in the case of violation of the SLA.

SLA is also described in [5] as a "Contractual service commitment". An SLA is a document that describes the minimum performance criteria a provider promises to meet while delivering a service. It typically also sets out the remedial action and any penalties that will take effect if performance falls below the promised standard. It is an essential component of the legal contract between a service consumer and the provider."

B. *SLA structure*

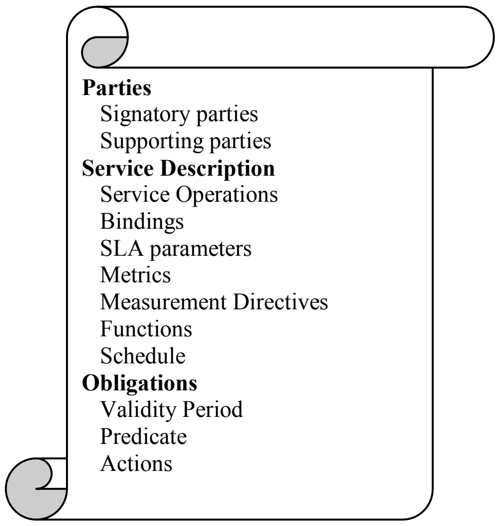


Figure 1. General SLA Structure

SLA is composed of three main sections as presented in figure 1. The first section contains the involved parties in the contract: the signatory parties (the service provider and its consumer) and the third parties that supervise SLA obligations. The second section presents the involved services description. This part contains the service operations, their input and output messages. For each service operation, one or more bindings may be specified. A binding is the transport encoding for the messages to be exchanged. It also contains the SLA parameters representing the QoS variables that will be used in the specification of the contract obligations. These parameters are based on metrics evaluated by measurement directives. Some functions can be used to aggregate multiple metric values. The last element (schedule) of this second part in the contract specifies the duration and the frequency of QoS measurements. The third section presents the contract obligations: their validity period indicating the time intervals for which a given SLA parameter is valid (for examples, business days, regular working hours or maintenance periods), the predicate that represents the conditions that specify these obligations and the actions to be taken when the contract is not respected.

C. *SLA life cycle*

Although the contracts are intended to formalize mutually accepted agreements by services providers and consumers, their establishment usually remains usually asymmetric and controlled by the providers. It includes several steps as shown in figure 2.

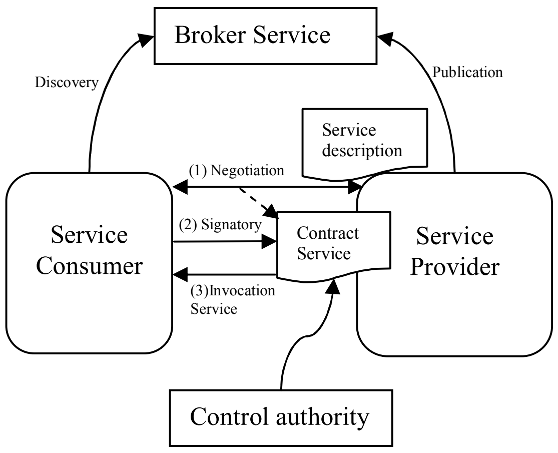


Figure 2. SLA Establishment

The service provider creates a contract model that defines the offered services and their associated constraints. Then, it publishes them at a given broker service. They also integrate the service's financial costs (included in the SLA obligations) as well as penalties in case of contract violation. The service consumer discovers this model from the broker service and selects the desired services and the contract instance. When the provider receives this instance, he checks it before its validation and sends it to the consumer. After a negotiation phase and when the two parties are in agreement, they sign the contract. After that, the consumer can invoke her/his the corresponding service. The specified obligations in the contract are constantly supervised by a controlling authority which is a third party that notifies the signatory parties when the contract is violated.

D. *SLA implementations*

Several languages were proposed to implement the SLA specifications. We can cite WSOL (Web Service Offerings Language) [6], GXLA [7], WSML (Web Services Management Language) [8], SLAang [9], Ws-Agreement

[10], Ws-Negotiation [11] and WSLA (Web Service Level Agreement) [12]. Among all these languages, the most successful contribution is the WSLA language created by IBM[1]. It is a flexible and extensible language based on XML Schema[2]. However, contract development remains a difficult task to achieve when using this language. In fact, providers and consumers don't have the same degree of knowledge and may not share the same language. In addition, the contract monitoring and its possible violation are difficult to establish due to some insufficiencies in the monitored QoS parameters description especially when they are composed by other elementary parameters. Consequently, we explore the existing SLA related models to find more structured and semantically richer descriptions of SLA obligations to ensure their automatic monitoring and management.

III. SLA-RELATED EXISTING MODELS

In this section, we present an analysis of some SLA related existing models. We notice a great interest in modeling the quality of services which is a principal element in the contract specification. In the following parts of this section, we present the main existing models in QoS specifications.

A. *OWL-QoS (Chen Zhou, Likang-Tien Chia, Bu-Sung Lee)*

OWL-QoS [13] is a QoS description model. It reuses OWL-S [14], the service description ontology standard. This model is characterized by its formal QoS specification, distribution and consumption. Unfortunately, it presents some insufficiencies: QoS metrics are instantiated without specifying how they will be measured and in what context they can be used. Moreover, the used approach is flawed in that it uses cardinality constraints to express bounds upon QoS properties. As the term cardinality suggests, this is actually a misuse of this OWL construct. A cardinality constraint puts constraints on the number of values a property can take, not on the values themselves. Even if the approach taken was valid, it also carries the limitation that it can only express bounds as positive integers (e.g., there is no simple way to say "availability> 0.999").

B. *QoSOnt (G. Dobson, R. Lock, I. Sommerville)*

QoSOnt [15] has much in common with other OWL ontologies [16] for web services. It contains links to OWL-S and concentrates on the metrics definition and on QoS requirements matching with metrics. As well as pointing the direction to the correct semantics for matchmaking, QoSOnt also correctly identifies that the value of a metric is only relevant in the correct scope (e.g.. network latency applies to a particular network route) and that metric has a "direction" e.g., the higher, the better. Initial attempts at representing how metrics combine when services are composed have also been made. Unfortunately, despite identifying the correct semantics for matching QoS with its metrics, QoSOnt uses a non standardized XML language losing many of the advantages of OWL [2].

C. *SL-Ontology (Steffen Bleul, Thomas Weise, Kurt Geihss)*

SL-Ontology [17] is another attempt at QoS modeling. It differentiates between the provider offers and customer demands. It presents the necessary elements of quality aware service discovery and the importance of integrating quality aspects in service integration. A description language needs flexibility for service level packages and service providing parties. It must also handle different terms in specifying QoS-Dimensions. In this scope, SL-Ontology specifies a part of measurement units transformations to address disparities between customers and suppliers languages. This resolution is specified only at the level of units in this model.

D. *WS-QoS (Tian, M., Gramm, A., Ritter, H., and Schiller, J.)*

WS-QoS [18] is a framework that uses a QoS-based ontology model for the dynamic Web services selection depending on the performance requirements and network bandwidth. This model is characterized by specific metrics that must be known in advance by all the services. It also uses a specific non-OWL XML language for metric description. Consequently, it loses the reasoning and the semantic inferences offered by the OWL language.

E. *FIPA QoS (M.B Alberto, G.V Marisol)*

FIPA [19] is another ontology-based model of QoS representation. It is complete, but unfortunately it remains too specific to the lower layers of the OSI model. This ontology also lacks an openly available implementation and links to OWL-S ontology. It has also been applied only in FIPA architecture and therefore it is not directly applicable in a web services environment.

F. *MOQ (HM. Kim, A. Sengupta and J. Evermann)*

MOQ [20] is another attempt of QoS modeling that defines QoS composite requirements but fails to suggest a mean to allow logical requirement combinations, only stating that if all sub-requirements are met then the composite is always satisfied. Unfortunately, the major drawback of MOQ is that it does not in itself seem to present an ontology, but only talks about the semantics of QoS ontologies in general. It doesn't use a vocabulary or taxonomy of QoS terms in its modeling and therefore it fails to address all of the issues that complete ontologies.

G. *Synthesis on existing QoS models*

We have made a comparative study between these various models. Table 1 presents a comparison of these models according to three criteria. The first criterion "Scope" illustrates the degree of completeness of each model by listing its main concepts. The second criterion "Implementation" shows if concrete examples were developed to validate these models. The third criterion

[1] http://www.research.ibm.com/wsla/

[2] http://www.w3.org/XML/Schema

"Automatic use facilities" illustrates the degree of information structuring into these models to facilitate their interpretation and their automatic use to monitor and manage service level agreements.

TABLE 1. QOS MODELS COMPARISON

	Scope	**Implementation**	**Automatic use facilities**
OWL-QoS [13]	Metric, Unit, Measurement functions, QoS Profile, agreements, Actors, Service	Partial implementations	QoS constraints as strings
QoSOnt [15]	Service Profile, QoS Profile, Metric, Unit, Actors, Measurement functions, Service	Partial implementations	Specific partial implementation language
SL-Ontology [17]	Metric, Unit,, QoS, Services	Partial implementations	for SLA establishment but not for monitoring
WS-QoS [18]	Metric, Functions, QoS, agreements, Actors	No OWL implementation	Specific implementation language
FIPA QoS [19]	Quality of Service Description, Rate Value, Probability Value, Transport Protocol Description	No OWL implementation	Specific implementation language
MOQ [20]	Qos Requirement, Traceability, Management	Provides a theoretical basis without implementations	Specific predicate interpretation

On the first criterion, the majority of the existing models have focused on the specification and the measurement of QoS. Few models are interested in establishing and managing the QoS contracts. Consequently, we usually have incomplete specifications to express the obligations of the involved actors in the contracts. On the second criterion, we looked for concrete examples that instantiate the existing models. We encountered various difficulties with the majority of them due to the insufficiency at the level of QoS obligation specifications in the contract. For example, the MOQ model is very abstract and lacks many concrete elements to be really implemented in real world instances. On the third criterion, specific and non standard implementations of some parts of all the described models make their automatic interpretation and monitoring difficult to establish. Ad-hoc solutions have to be developed to use these models.

All the models that we cited have advantages and relative limitations. Indeed, few existing models define the context concept in the quality of service (QoS); however, context is important to manage the contract lifecycle for QoS in an automatic way. In addition, some contributions (as WS-QoS) use specific XML formats for the full or partial implementation of their models. This may reduce the interest of using this ontology. In fact, the ontology offer inference possibilities and semantic interpretations when they are implemented using the OWL language. In addition, some models are either specific to a particular domain such as FIPA-QoS which is specific to the low layers of OSI model) or presenting various insufficiencies (like the lack of specifications of logical constraints in MOQ). Finally, all these models focus on the quality of service modeling without detailing the obligations and agreements between the involved actors. This last observation motivated us to develop an SLA model based on the advantages of the existing contributions.

Our contribution in this domain is to establish an ontology-based service level agreements (SLA) model. We made this choice to (i) facilitate the establishment of contracts between entities (suppliers and consumers) having different knowledge levels (ii) have a model offering rich semantics to be understood by humans and by machines (iii) use the semantic richness of SWRL rules in order to express SLA obligations and to easily infer and directly apply the necessary actions in case of violations and (iv) use their semantic richness to diagnose the causes of these violations.

IV. SLAONT: ONTOLOGY-BASED SLA MODEL PROPOSITION

Our model, that we called SLAont [21], defines an ontology describing various concepts and properties needed in a quality of service contract. Figure 3 presents the generic structure of this model. The root is the SLA concept. It represents the contracts class that can be instantiated from *SLAOnt*. This class is composed of the following concepts: Parties, Obligation and *ServiceDefiniton*. The first concept *Parties* defines the involved parties in the contract: the signatory and the supporting party. The signatory parties are generally the service providers and their consumers. The third parties provide the necessary entities for the quality of service measurement evaluation and monitoring. The second concept Obligation defines the quality of service obligations that have to be respected by the parties. These obligations are defined by service level objectives. Each objective is composed of predicates describing the QoS clauses that may cause the contract violation. The third concept *ServiceDefinition* describes the provided services that are concerned by these obligations. Our model uses the OWL-S [14] ontology to describe these services. This ontology is composed of three main parts: the service profile for advertising and discovering services; the process model, that gives a detailed description of the service operation; and the grounding that provides details on how to interoperate with a service, via messages.

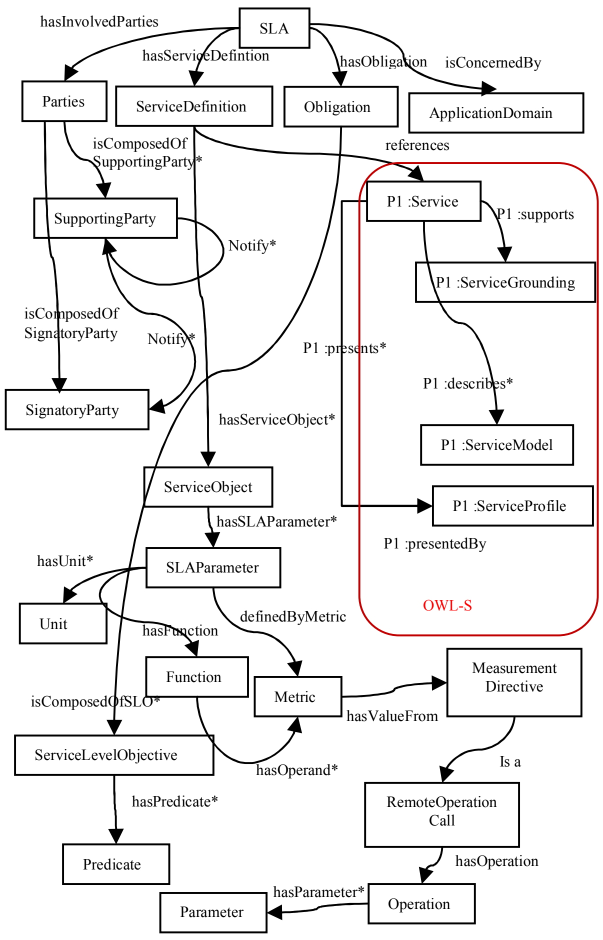


Figure 3. *SLAOnt*: general structure of a QoS contract

The contract is defined in a specific application domain. Thus, the concept *ApplicationDomain* is defined to describe the contract context. This concept is a reference to an external ontology describing the business domain of the involved service in the SLA. The necessary variables for defining the measured quality of service are modeled by the *SLAParameter* concept. *SLAParameter* is associated with one or more metrics (an aggregation of metrics), that define the quality of service parameters to be measured in the contract. These metrics, modeled by the metric concept, can be aggregated by a mathematical function or an algorithm defined in the Function concept. For example, the function can be an average or a computed percentage from a set of measurements.

The *SLAParameter* concept represents a variable that has a measurement unit (Unit) that can be seconds, minutes, percentages, etc. For each SLA parameter, the SLA designer has to specify the corresponding aggregation function to compute its values. These functions are described in the SLA instance. They can be composite and they are described as an abstract service providing an SLA parameter value and consuming metrics values or constants. Each function is defined by an implementing class and a list of operands corresponding to measurements of QoS metrics. These measurements are obtained using measurement directives *MeasurementDirective* corresponding to remote operation calls to retrieve the metrics values. We have modeled these calls by the *RemoteOperationCall* concept. It describes the operation to be called with its invocation protocol. Each operation is associated to a handler that represents the remote class that hosts the operation. In addition, we describe input parameters of the operation with the Parameter concept. This encapsulates the values to be passed to the remote operation for its invocation. The Predicate concept defines the QoS obligations that must be respected in the contract. Each predicate is expressed by an SWRL rule. Each rule is defined by a head part (swrl:head) and a body part (swrl:body). In the head part, we specify the actions that have to be taken when a violation is detected. The body part specifies the conditions that may cause a contract violation. Listing 1 gives a concrete example of an SLA predicate in SLAOnt. This rule sends a disseminate violation message to the signatory parties if the service response time is greater or equal to 100 milliseconds.

Listing 1. Response time less than100 ms evaluation rule

```
hasEvaluation (response_time, ?x) ^ swrlb: greater ThanOrEqual(?x, 100.0) →
slaont:disseminateViolation (response_time, ?x)
```

SLAOnt is a generic and rich model in terms of semantics. In fact, *SLAParameters* can be semantically composed of metrics according to a user defined aggregation functions. For example, an SLA parameter can describe the average response time of a service. In this case, the attached metric is response time and its aggregation function is the average function. Moreover, measurement directives are generic enough to evaluate any metric defined in this model. The model can also express dependencies between metrics, *SLAParameters* and QoS obligations. These dependencies can be statistical (aggregation to calculate average for example), logical (comparison with many thresholds) or semantic (parameters deduced by inference from other parameters). Finally, the semantic relationships between metrics, *SLAParameters* and QoS obligations offer an easy and reliable means to (1) evaluate them and (2) to produce inferences and reasoning in order to detect contract violations and even QoS degradations. In the next section, we present how we used SLAOnt to monitor QoS contracts.

V. MONITORING SLAONT INSTANCES

We used *SLAOnt* model to monitor the obligations defined in the SLAs. In this section, we present the simplified architecture and the main algorithms that perform the monitoring process. Figure 4 shows an overview of this monitoring process in our approach. It illustrates the main entities of our monitoring API. This API contains a monitoring main module that can deploy *SLAOnt* instances. For each metric, SLA parameter and obligation defined in an *SLAOnt* instance, the main module respectively generates a metric measurement service, an SLA parameter measurement service and obligation measurement service. In the remaining parts of this section, we detail the main functions and algorithms of these services.

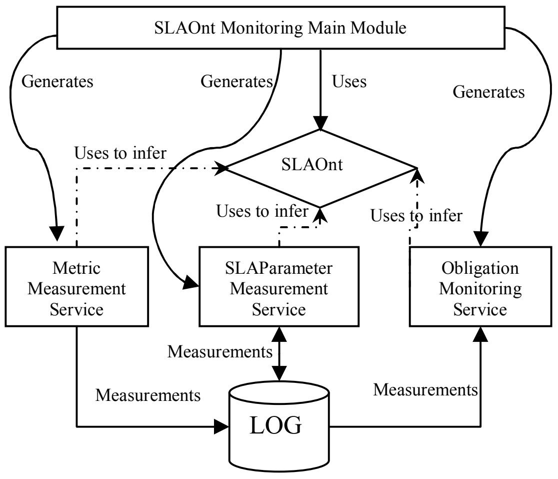

SLAOnt Monitoring Main Module
Generates
Generates
Uses
Generates
Uses to infer
SLAOnt
Uses to infer
Uses to infer
Metric Measurement Service
SLAParameter Measurement Service
Obligation Monitoring Service
Measurements
Measurements
Measurements
LOG

Figure 4. Simplified architecture of *SLAOnt* obligations monitoring

A. Metric measurement services

Metric measurement services provide metric values according to a frequency specified in the SLA. A metric measurement service is automatically instantiated by the *SLAOnt* monitoring main module for each metric defined in the SLA. This service invokes the measurement directive of the associated metric. Listing 2 presents the main functions of this service. First, it collects the metric name, the measurement directive and the measurement frequency of the associated metric. Then, it loops according to this frequency to invoke the measurement directive of the metric and to store the obtained value in a log file. These values will be used by the SLA parameter measurement services.

Listing 2. Metric measurement algorithm

```
MetricMeasurement(Metric metric)
{
  getMetricDetailsRule:="hasName("+ metric + ", ?metricName) ^
    hasValueFrom("+metric+",?measurementDirective)^
    hasMeasurementFrequency("+ metric +",?frequency) →
    query:select(?metricName, ? measurementDirective, ? frequency)";
(metricName, measurementDirective, frequency):= runRule(getMetricDetailsRule);
  Loop on frequency
  {
   measurement := invokeMeasurementDirective(measurementDirective);
   Log.store(metricName,measurement);
  }
}
```

B. SLA parameter measurement services

The SLA parameter measurement services apply aggregation functions on the metrics values to compute the QoS variables defined by the SLA parameter. An SLA parameter measurement service is automatically instantiated by the *SLAOnt* monitoring main module for each SLA parameter defined in the contract. For each SLA parameter, we have to specify the corresponding aggregation function to compute its values. The implementing class of each function will be called by the SLA parameter measurement algorithm after getting the necessary metric values to compute the function. This algorithm collects the different metrics associated with the SLA parameter. It also gets the computation frequency and the aggregation function of the SLA parameter. Then, it loops according to this frequency in order (i) to collect the last measured values of the metrics associated to the SLA parameter, (ii) to compute its aggregation functions and (iii) to add its value in the Log storage. This value will be used by the obligation monitoring services. Listing 3 presents the main functions of this service.

Listing 3. SLA parameter measurement algorithm

```
SLAParameterMeasurement(SLAParameter parameter)
{
  getMetricsRule := "hasFunction("+parameter+",?function) ^
    hasOperand(?function, ?metric) →query:select(?metric);
  metrics := runRule(getMetricsRule);
  getSLAParameterDetailsRule:= "hasFunction("+parameter+",?function) ^
    hasAggregationFrequency("+ parameter + ", ? aggregationFrequency)
    →query:select(?function, ?aggregationFrequency)";
  (function, aggregationFrequency):=runRule(getSLAParameterDetailsRule);
  Every aggregationFrequency do
  {
   Measurements:= ∅;
   For each metric in metrics do
   {
    Measurements.put(metric, Log.getLastValues(metric));
   }
   functionClass := loadFunctionClass(function) ;
   slaParameterValue := functionClass.call(Measurements);
   SLAOnt.setLastParameterValue(parameter, slaParameterValue);
   Log.store(parameter, slaParameterValue);
  }
}
```

C. Obligation monitoring services

The obligation monitoring services check the validity of each obligation defined in the contract. They are automatically instantiated by the *SLAOnt* monitoring main module for each Obligation defined in the SLA. It uses the computed values in the SLA parameter to check if they satisfy the specified conditions defined in the obligation. These conditions are defined as SWRL rules.

Listing 4. Predicate evaluation rule

```
hasEvaluation(average_response_time, ?x) ^ swrlb: greaterThanOrEqual(?x,
    100.0) → slaont:disseminate Violation(average_ response_time, ?x)
```

The evaluation of these conditions is simply an inference of these SWRL rules. The actions to be taken in case of violations can be directly applied in these rules like *disseminateViolation* in Listing 4.

Listing 5. Obligation monitoring algorithm

```
checkObligation(Obligation obligation)
{
getPredicatesRule:="isComposedOfSLO("+obligation+",?slo) ^ hasPredicate(?slo,
    ?predicate) → query:select(?predicate)";
Predicates: = runRule(getPredicatesRule) ;
For each p in Predicates
  {  getPredicateRule := "hasRule("+p+", ?rule) ^
     hasVerificationPeriodicity("+p+",?periodicity) →
     query:select(?rule,?periodicity)";
     (rule,periodicity) := runRule(getPredicateRule)
    Loop on periodicity
     {
      runRule(rule);
     }
  }
}
```

Listing 5 presents the main functions of this service. It starts by collecting the different predicates defined in the associated obligation. Then, every predicate evaluation periodicity (every sixty minutes for example), the inference engine computes the attached SWRL rule and executes the specified action to be taken in case of violation. For example, the SWRL of Listing 1, if the response time is greater than 100 ms, the action *disseminateViolation* is triggered. This action continuously reports all the detected violations and their causes to the involved parties in the SLA.

When the designer creates an *SLAOnt* instance, she/he can specify an execution order for the SWRL rules representing the SLA predicates. This order is ensured by a numbering sequence in the name of the rules that should be conflict free in order to produce relevant results. This conflict verification should be performed before the monitoring phase. The verification process is out of the scope of this work. Actually, we are working on a negotiation approach that generates SLAOnt instances with conflict free obligations.

In the next section, we present how we implemented these algorithms to develop a complete and a reusable monitoring API for *SLAOnt* instances.

VI. SLAOnt MONITORING API

In this section, we present the *SLAOnt* monitoring API (named *SLAOntAPI*) that we have developed to implement the algorithms presented in the previous section. Figure 5 shows the technical architecture of the monitoring process. In the lowest layer, ontologies used in the API are represented by their OWL files. Above this layer, the Xerces XML API[3] is used to read data from the owl file. Protégé OWL API[4] is used to handle owl data in the ontologies. Then, the SWRL Jess API[5] is used to make inferences and reasoning on the ontologies instances. In the upper layer, we have developed a JAVA Monitoring API (*SLAOntAPI*).

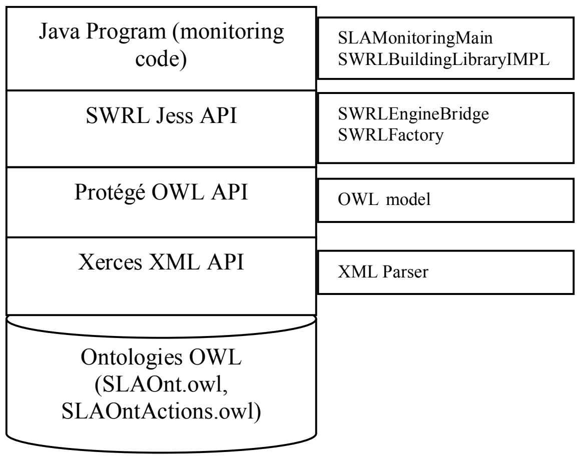


Figure 5. Technical architecture of the monitoring process

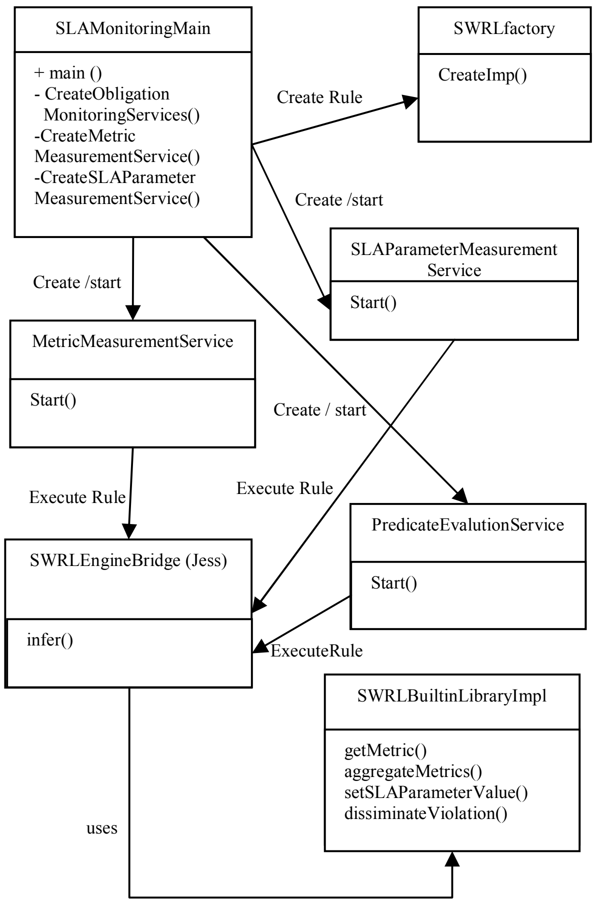


Figure 6. The *SLAOntAPI* class diagram

[3] http://xerces.apache.org/xerces-j/apiDocs/index.html

[4] http://protege.stanford.edu/plugins/owl/api/

[5] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab

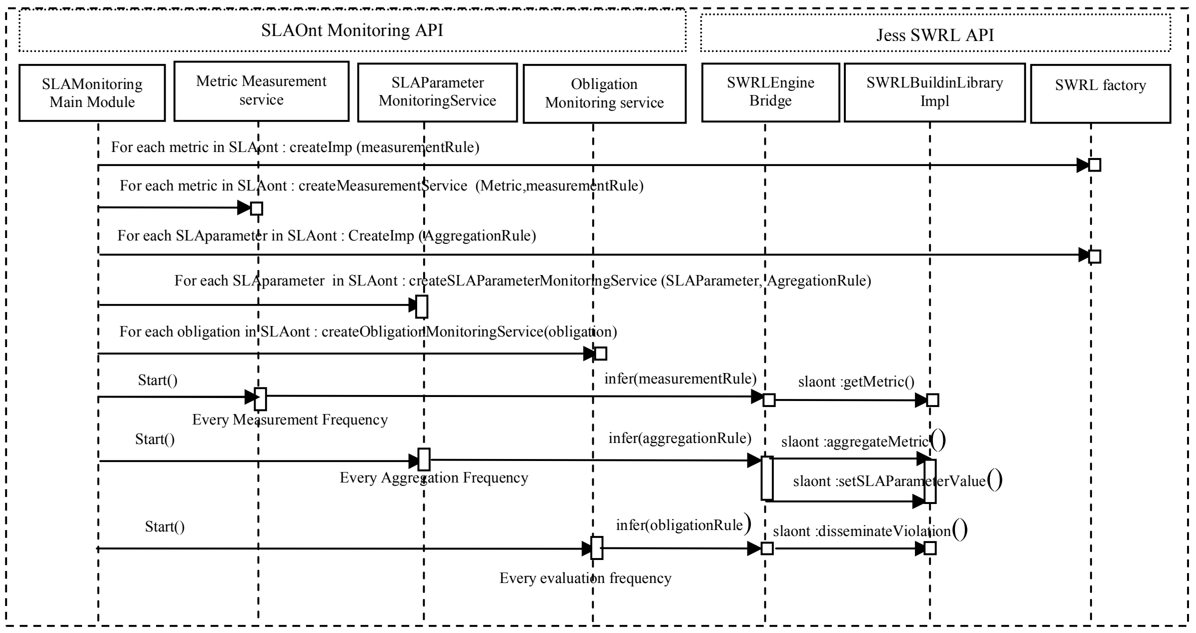

SLAOnt Monitoring API
Jess SWRL API
SLAMonitoring Main Module
Metric Measurement service
SLAParameter MonitoringService
Obligation Monitoring service
SWRLEngine Bridge
SWRLBuildinLibrary Impl
SWRL factory
For each metric in SLAont : createImp (measurementRule)
For each metric in SLAont : createMeasurementService (Metric,measurementRule)
For each SLAparameter in SLAont : CreateImp (AggregationRule)
For each SLAparameter in SLAont : createSLAParameterMonitoringService (SLAParameter, AgregationRule)
For each obligation in SLAont : createObligationMonitoringService(obligation)
Start()
infer(measurementRule)
slaont :getMetric()
Every Measurement Frequency
Start()
infer(aggregationRule)
slaont :aggregateMetric()
Every Aggregation Frequency
slaont :setSLAParameterValue()
Start()
infer(obligationRule)
slaont :disseminateViolation()
Every evaluation frequency

Figure 7. The *SLAOntAPI* Sequence Diagram

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.laas.fr/~kchaari/slaOntActions.owl#"
  xml:base="http://www.laas.fr/~kchaari/slaOntActions.owl">
  <owl:Ontology rdf:about=""/>
  <rdf:Property rdf:ID="inArgs"/>
  <owl:DatatypeProperty rdf:ID="minArgs"/>
  <owl:DatatypeProperty rdf:ID="maxArgs"/>
  <owl:DatatypeProperty rdf:ID="args"/>
  <swrl:Builtin rdf:ID="disseminateViolation">
    <minArgs rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</minArgs>
  </swrl:Builtin>
  <swrl:Builtin rdf:ID="getMetric">
    <args rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</args>
  </swrl:Builtin>
<swrl:Builtin rdf:ID="aggregateMetric">
    <minArgs rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</minArgs>
  </swrl:Builtin>
```

Figure 8. OWL file containing the actions to be triggered in SLA predicates

Figure 6 shows the class diagram of the *SLAOnt* API. The most important class in this API is the *SWRLBuiltInLibraryImpl* class. It contains the functions that will be directly invoked by the SWRL rules. This class offers the *getMetric* function which invokes the measurement directive of the specified metric and sends its value to the Log store. It also has an *aggregateMetrics* method that calls the aggregation function to compute SLA parameter values. Finally, it offers a *disseminateViolation* method that sends SLA violation messages to the signatory parties. These methods are defined as SWRL Built-ins which are predicates that can take on or more arguments.

Built-ins are analogous to functions in production rule systems. A number of core built-ins are defined in the SWRL specification. This core set includes basic mathematical operators and built-ins for string and date manipulations. These built-ins can be used directly in SWRL rules. User defined built-ins has to be declared in an external ontology. We have declared *getMetric*, *aggregateMetrics* and *disseminateViolation* built-ins in the ontology represented by an Owl file (slaOntActions.owl) as shown in figure 8.

Figure 7 shows the sequence diagram of our approach. For each metric in SLAOnt, the main program creates a measurement rule and a measurement service. For each *SLAParameter*, The main program creates a metric aggregation rule and a service to perform the aggregation. For each obligation in *SLAOnt*, the main program generates an obligation evaluation service.

VII. CASE STUDY: THE FLIGHT SLA EXAMPLE

To validate the service level agreements model and the developed monitoring API, we created an instance of *SLAOnt* model using the “protégé” tool. This instance consists in a simple agreement example between a provider of a flight booking service and its consumers. This service must provide an average response time less than 100 milliseconds for a certain class of clients. Figure 9 illustrates the *FlightSLA* instance in this example.

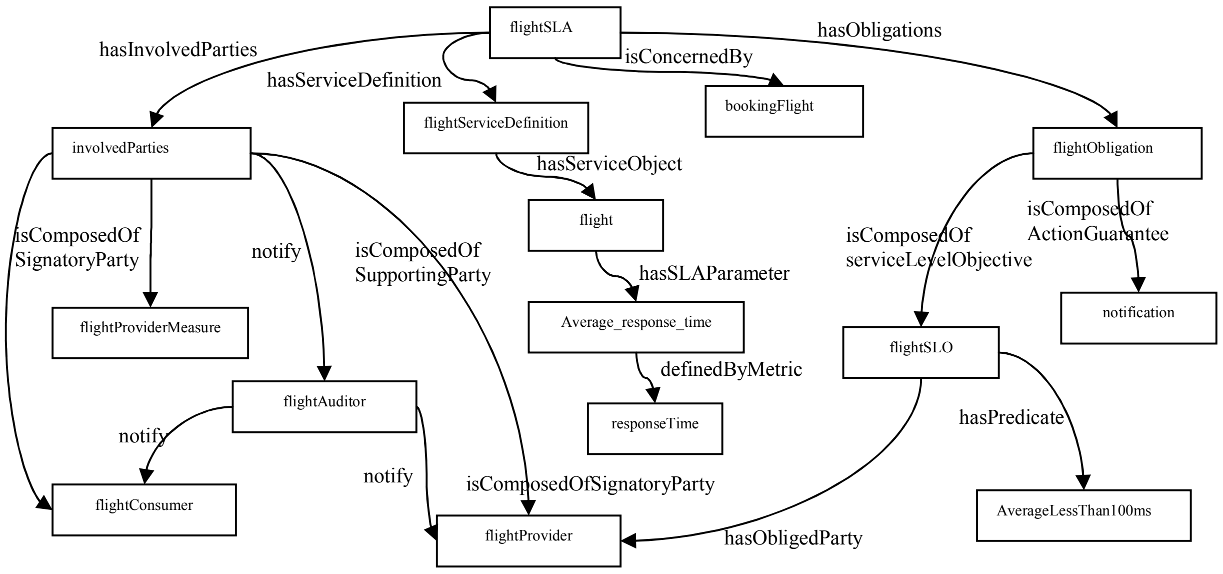


Figure 9. FlightSLA: An SLAOnt instance example

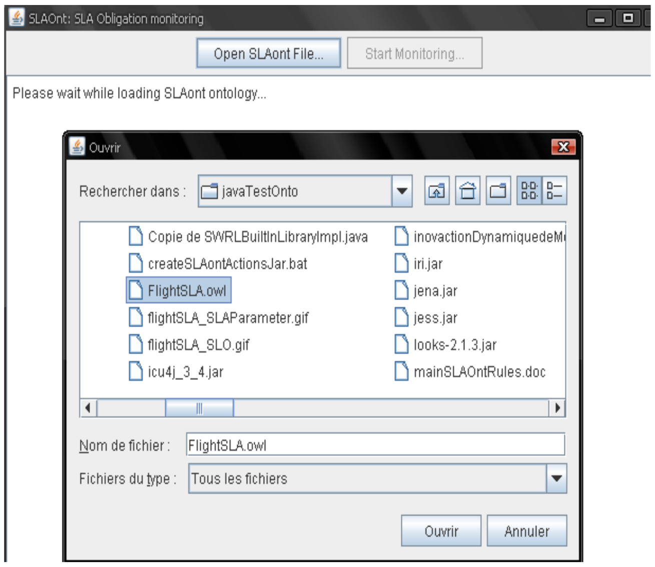
The service provider of this contract is named *FlightProvider* and his consumer is *FlightConsumer*. In this example, the third parties involved in this contract are: *FlightProviderMeasurement* and *FlightAuditor*. The *FlightProviderMeasurement* service must provide the response time measurements of the consumer requests. The *FlightAuditor* service must notify any violations of the contract to the signatory parties. The service provider must respect the objective defined by the *FilghtSLO* instance in the contract. This instance specifies the predicates that have to be satisfied in the contract. These predicates are defined in SWRL rules to facilitate the monitoring processes of the contract. The defined obligation in the *FlightSLA* example is shown in Listing 6.

Listing 6. Predicate evaluation rule

```
hasEvaluation(average_response_time, ?x) ^ swrlb: greaterThanOrEqual(?x,
100.0) → slaont:disseminate Violation(AverageLessThan100ms, "false", average_
response_time, ?x)
```

This rule verifies that the average response time of the monitored service is greater or equal to 100 milliseconds (body part of the rule). In this case, a violation message is disseminated to the signatory parties in the contract. These messages contain the parameter values that caused the violation. To perform the automatic monitoring process on this example, we loaded its owl file[6] in the monitoring API main module (figure 10).

Figure 11 shows the monitoring process of this example. To use the *SLAOntAPI*[7] with other SLA instances, the SLA designer should import the *SLAOnt* ontology[8] and creates the necessary instances of its main concepts. The actions to be taken in case of violations should be declared in an external ontology named *SLAOntActions.owl*. These actions should be implemented as SWRL Built-ins to work with our code. These built-ins are standard java code that can be easily personalized to manage the actions that should be taken in case of violations. Finally, the designer should save the created instance in an owl file and load it in our monitoring main module as shown in figure 10.



Please wait while loading SLAont ontology...

Copie de SWRLBuiltInLibraryImpl.java
createSLAontActionsJar.bat
FlightSLA.owl
flightSLA_SLAParameter.gif
flightSLA_SLO.gif
icu4j_3_4.jar
inovactionDynamiquedeM
iri.jar
jena.jar
jess.jar
looks-2.1.3.jar
mainSLAOntRules.doc

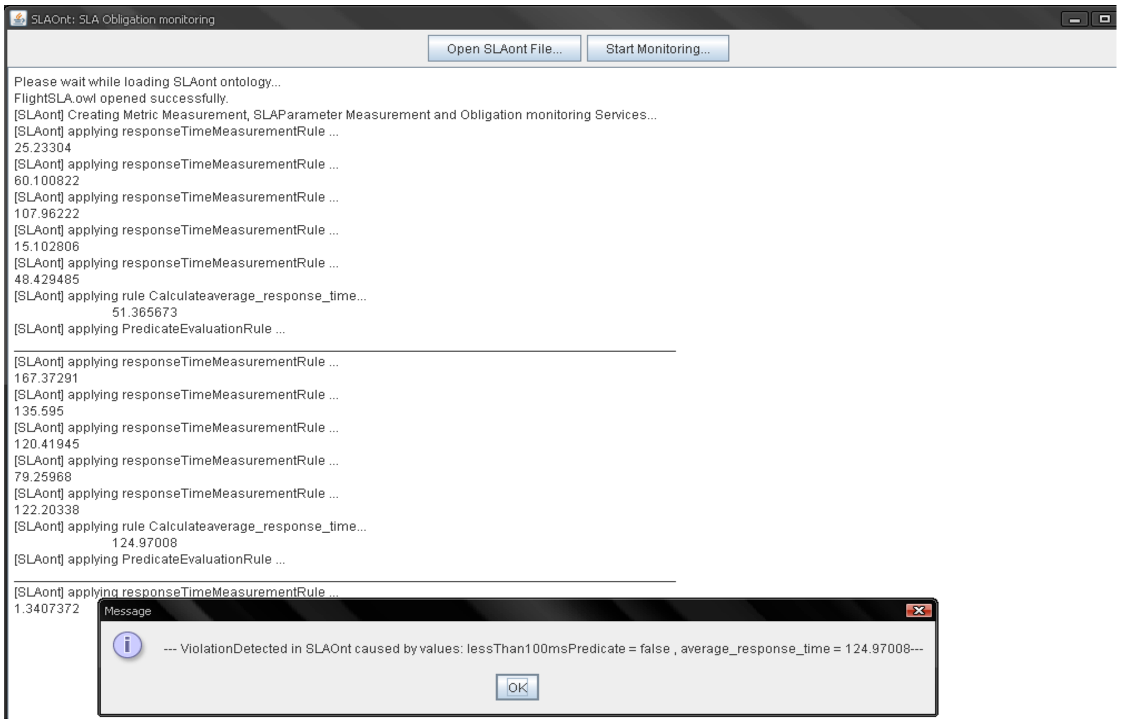
Nom de fichier : FlightSLA.owl
Fichiers du type : Tous les fichiers

Figure 10. SLAOnt instances loading interface

[6] http://www.laas.fr/~kchaari/slaOnt/FlightSLA.owl

[7] http://www.laas.fr/~kchaari/slaOnt/SLAmonitoring.zip

[8] http://www.laas.fr/~kchaari/slaOnt/SLAont.owl



```
Please wait while loading SLAont ontology...
FlightSLA.owl opened successfully.
[SLAont] Creating Metric Measurement, SLAParameter Measurement and Obligation monitoring Services...
[SLAont] applying responseTimeMeasurementRule ...
25.23304
[SLAont] applying responseTimeMeasurementRule ...
60.100822
[SLAont] applying responseTimeMeasurementRule ...
107.96222
[SLAont] applying responseTimeMeasurementRule ...
15.102806
[SLAont] applying responseTimeMeasurementRule ...
48.429485
[SLAont] applying rule Calculateaverage_response_time...
                51.365673
[SLAont] applying PredicateEvaluationRule ...
________________________________________________
[SLAont] applying responseTimeMeasurementRule ...
167.37291
[SLAont] applying responseTimeMeasurementRule ...
135.595
[SLAont] applying responseTimeMeasurementRule ...
120.41945
[SLAont] applying responseTimeMeasurementRule ...
79.25968
[SLAont] applying responseTimeMeasurementRule ...
122.20338
[SLAont] applying rule Calculateaverage_response_time...
                124.97008
[SLAont] applying PredicateEvaluationRule ...
________________________________________________
[SLAont] applying responseTimeMeasurementRule ...
1.3407372
```

```
--- ViolationDetected in SLAOnt caused by values: lessThan100msPredicate = false , average_response_time = 124.97008---
```

Figure 11. FlightSLA monitoring process

VIII. Conclusion and Future Work

The service oriented software engineering market requires QoS specifications by the services suppliers. Much effort has been invested in modeling QoS parameters to allow an automatic (or semi-automatic) selection of the services offering the best quality. We have explored several existing models in this domain. We have noticed the lack of a comprehensive and a generic model for the service level agreements specification and for their monitoring to detect possible violations. Therefore, we created an ontology that models these agreements (SLA) to facilitate the QoS contracts establishment between consumers and suppliers on one hand and to automate their management and monitoring on the other hand. In this paper, we presented the structure of our ontology-based model offering rich semantics to be understood by humans and by machines. In this model, we used the semantic richness of SWRL rules in order to express SLA obligations and to easily infer and apply the necessary actions in case of violations. Our second contribution in this work is the development of an API that guarantees the automatic monitoring of our SLA model instances. This API is based on an automatic generation of a service oriented architecture that gathers the measurements of the QoS defined in the SLAs. For clarity reasons, we presented a simple SLA example to illustrate the main principles of our SLA model and our monitoring API. We have tested this API on more complex examples concerning video streaming provider who offers two services: the first one to visualize film online and the second one for downloading films. In this example, a download time SLA parameter is monitored according to the video size and the client's throughput. Our API is scalable enough to handle a large number of metrics SLA parameter and obligations. In fact, their associated measurement services are instantiated dynamically in separate threads and can be distributed on many machines. We plan to use the monitored measurements to analyze and detect system degradations and to prevent SLA violations. Actually, we are working on a semantic-enabled negotiation framework to help the providers and their customers in establishing *SLAOnt* contracts. In a long term future work, we intend to propose corrective actions in case of QoS degradation. This issue will be very useful to evolve from the existing simple message notifications to corrective actions assistance.

Acknowledgements

This work has been partially funded by the ITEA2's UseNet (Ubiquitous M2M Service Networks) European project[9].

References

[1] R.Studer, R.Benjamins, D.Fensel. Knowledge engineering: principles and methods. IEEE Transactions on Data and Knowledge Engineering, 25 (1-2) pp.161-197, March 1998.

[9] https://usenet.erve.vtt.fi/

[2] Mc Guiness D. & ZSPLITZvan Harmelen F. (2004). OWL Web Ontology Language Overview, W3C Recommendation http://www.w3.org/TR/owl-features/

[3] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004.

[4] M. Debusmann, R. Kroger, and K. Geihs. "Unifying Service Level Management Using an MDA-based Approach". IEEE Network Operations and Management Symposium, pp.801-814, 2004.

[5] Procullux Media Ltd. http://looselycoupled.com/glossary/SLA. Last visited 15/01/09

[6] Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., and Ma, W. 2005. Management applications of the web service offerings language . Inf. Syst. 30, 7 (Nov. 2005), 564-586.

[7] Badis Tebbani, Issam Aib. "GXLA a Language for the Specification of Service Level Agreements". Autonomic Networking, 201-214, 2006.

[8] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F.Casati. "Automated SLA Monitoring for Web Services". IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, pp.28-41, 2002.

[9] J. Skene, D. Lamanna, and W. Emmerich. "SLAng: A Language for Service Level Agreements". Workshop on Future Trends in Distributed Computing Systems. IEEE Computer Society, 2002.

[10] A. Andrieux, A. Dan K. Czajkowski, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne,J. Rofrano, S. Tuecke, and M. Xu. "Web Services Agreement Specification (WSAgreement)". Specification draft, Global Grid Forum (GGF), September Version 09/2005.

[11] P. C. K. Hung, H. Li, and J-J Jeng. "WS-Negotiation: An Overview of Research Issues". Hawaii International Conference on System Sciences (HICSS), 2004.

[12] A. Keller and H. Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". Journal of Network and Systems Management, 11(1), 2003.

[13] Chia, Bu-Sung Lee. "QoS Measurement Issues with DAML-QoS". IEEE InterChen Zhou, Likang-Tien national Conference on e-Business Engineering (ICEBE'05) pp. 395 403.

[14] OWL-S. An OWL-based Web service ontology. <http://www.w3.org/Submission/2004/07/> November 2004.

[15] G. Dobson, R. Lock, I. Sommerville. "QoSOnt: a QoS Ontology for Service-Centric System", EUROMICRO Conference on Software Engineering and Advanced Applications, Porto, Portugal, Aug. 2005.

[16] C. Zhou, L. Chia, and B. Lee, "DAML-QoS Ontology for Web Services", Proceeding of the International Conference on Web Services 2004 (ICWS04), San Diego, California, USA, July 2004.

[17] Steffen Bleul, Thomas Weise, Kurt Geihss. "An Ontology for Quality-Aware Service Discovery". Special Edition Editorial: Engineering Design and Composition of Service-Oriented Applications, Computer Systems Science & Engineering, Volume 5, Number 21 – 2006.

[18] Tian, M., Gramm, A., Ritter,H., and Schiller,J. "Efficient Selection and Monitoring of QoSaware Web services with the WSQoS Framework". IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), Beijing, China, 2004.

[19] Foundation for Intelligent Physical Agents "FIPA Quality of Service Ontology Specification", Geneva, Switzerland, Nov. 2002.

[20] HM. Kim, A. Sengupta and J. Evermann. "MOQ: Web Services Ontologies for QOS and General Quality Evaluations", European Conference on Information Systems, Regensburg, Germany. May 2005.

[21] K. Fakhfakh, T. Chaari, S. Tazi, K. Drira, and M. Jmaiel. 2008. A Comprehensive Ontology-Based Approach for SLA Obligations Monitoring. In Proceedings of the 2008 the Second international Conference on Advanced Engineering Computing and Applications in Sciences - Volume 00 (September 29 - October 04, 2008). ADVCOMP 2008. IARIA. Published by the IEEE Computer Society Press, pp. 217-222.