SOFTWARE – PRACTICE AND EXPERIENCE
*Softw. Pract. Exper.* 2012; **42**:501–518
Published online 27 July 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1104

# A dependency-aware ontology-based approach for deploying service level agreement monitoring services in Cloud

Amir Vahid Dastjerdi[1,*,†], Sayed Gholam Hassan Tabatabaei[2] and Rajkumar Buyya[1]

[1]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, VIC 3010, Australia*
[2]*Advanced Informatics School (AIS), Universiti Teknologi Malaysia (UTM), International Campus, Malaysia*

SUMMARY

Cloud computing offers virtualized computing elements on demand in a pay-as-you-go manner. The major motivations to adopt Cloud services include no upfront investment on infrastructure and transferring responsibility of maintenance, backups, and license management to Cloud Providers. However, one of the key challenges that holds businesses from adopting Cloud computing services is that, by migrating to Cloud, they move some of their information and services out of their direct control. Their main concern is how well the Cloud providers keep their information (security) and deliver their services (performance). To cope with this challenge, several service level agreement management systems have been proposed. However, monitoring service deployment as a major responsibility of those systems have not been deeply investigated yet. Therefore, this paper shows how monitoring services have to be described, deployed (discovered and ranked), and then how they have to be executed to enforce accurate penalties by eliminating service level agreement failure cascading effects on violation detection. Copyright © 2011 John Wiley & Sons, Ltd.

Received 14 April 2011; Accepted 5 June 2011

KEY WORDS: service level agreement; Cloud computing; monitoring; Semantic Web Service

## 1. INTRODUCTION

Cloud computing has transferred the delivery of IT services to new level that brings comfort of traditional utilities such as water, electricity to its users. Buyya and colleagues highlighted main aspects of Cloud namely as dynamic scalability, and service level agreement (SLA) negotiability. According to their definition [1], "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service providers and consumers." Major motivations to adopt Cloud services include reasonable price as they are offered in economy of scale, and transferring responsibility of maintenance, backups, and license management to Cloud service providers.

However, one of the key challenges that holds back businesses from adopting Cloud computing services, even if they found it cost effective is that, by migrating to Cloud, they move some of their information and services out of their direct control [2]. The main concern is how confidentially the Cloud providers keep their information (security) and with which quality they deliver their services (performance). To cope with this challenge, service level agreement (SLA) has been introduced. SLA contract, which is signed by both parties includes Quality of Service (QoS) requirements and penalties in case QoS requirement is not met by providers. However, SLA is not sufficient to ensure Cloud reliability. For example, if a business has critical Web application deployed on Cloud and it

*Correspondence to: Amir Vahid Dastjerdi, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, VIC 3010, Australia.
†E-mail: amirv@csse.unimelb.edu.au

fails, thousands of dollars might be lost. Nevertheless, according to most SLA contracts, they only give a penalty as much as a portion of the deployment fee. Therefore, the responsibility cannot be transferred to the Cloud service providers by SLA. Instead, an efficient runtime monitoring services have to be deployed to validate the SLA and enforce the penalties. Moreover, Theilmann *et al*. [3] cited that to realize the vision of dynamic service provisioning the whole process of monitoring has to be automated. Therefore, automating monitoring service deployment is the main objective of this work.

We consider a Cloud service chain, which includes services such as virtual unit (infrastructure as a service), appliance, and software as a service. In such an environment where various providers involved in satisfying user requirements, we face a set of difficulties in SLA monitoring. Firstly, existence of different SLA offer, counter offer, and contract templates makes it difficult to discover necessary monitoring services that have required capabilities to monitor service level objectives in SLAs. Therefore, [4], creating a standard model for describing SLAs in different layers of Cloud has been considered as a major challenge in this area of research. In this research, this matter has been addressed by a form of semantic SLA, which brings a common language and understanding to all parties involved in service provisioning of Cloud.

Next, in the Cloud environment, there are dependencies between different layers' service performances. It means that if one of the lower-layer services (infrastructure layer) is not functioning properly, it can affect performance of higher-layer services. Whereas SLA dependency has been considered by several works [4], no practical approach has been presented to model the dependencies among services. Consequently, this paper shows how dependency knowledge can be modeled using semantic technology, and how that knowledge can be used in discovery of monitoring services and SLA failure detection. The major contributions of the paper can be summarized as follows:

- Proposing a Semantic SLA, which can be understood by all parties including providers, requestors, and monitoring services. The Semantic SLA contract is defined in a way, which can be used as a goal for discovery of necessary monitoring services.
- SLA dependency modeling using Web Service Modeling Ontology (WSMO) to build knowledgebase, which can be exploit to eliminate effects of SLA failure cascading on violation detection.
- Proposing an architecture for service deployment in Cloud that is also capable of discovery, ranking, and coordination of monitoring services.
- Offering algorithms for discovery and ranking of monitoring services while considering users' preferences on reliability, budget, and legal constraints.

The rest of the paper is organized as follows: In the next section, a brief introduction to necessary concepts related to the paper is given. In Section 3, we describe motivation scenario, in which necessary monitoring services are required for monitoring SLA contracts where set of quality of service dependencies exist among services. We next offer an architecture for service deployment in Cloud, which is capable of Cloud and monitoring service deployment and management. This section is considered as a heart of the paper where we present the discovery and ranking algorithm for monitoring services and offer a way to model dependency knowledge and apply them to reduce the number of false alarms. Later in Section 5, we show effectiveness of our approach by feasibility and scalability tests. The comparison between our work and related works is given in Section 6. Finally, Section 7 summarizes the major contributions and draws the conclusion.

## 2. BACKGROUNDS

In this section, concepts related to our approach, that is, Cloud services and monitoring layers, Web Service Modeling Ontology and Service Level Agreement Management are described.

### *2.1. Cloud service and monitoring layers*

As Emeakaroha *et al*. [5] and Theilmann *et al*. [3] cited, Cloud service can be classified in different layers (as depicted in Figure 1), which will lead to hierarchical structure of SLA contracts between



DOI: 10.1002/spe

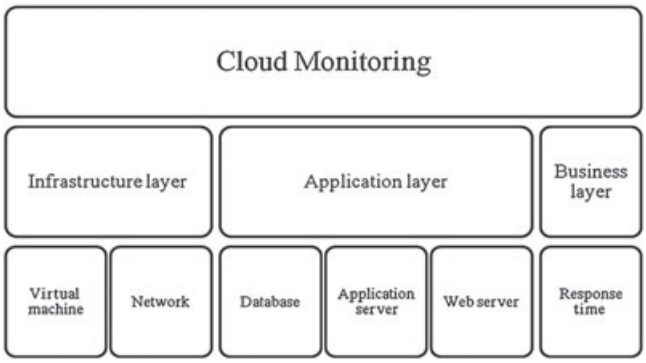


Figure 1. Cloud monitoring service layers.

different supply chain partners. In our scenario, three layers, namely infrastructure, application, and business layers are considered, which are defined as follows:

1. Infrastructure layer services: In this layer Cloud service providers offer virtual units, which are resources that have been virtualized and can be a virtual computer, database system, or even a virtual cluster. Monitoring criteria for infrastructure as a service provider is typically going to include network uptime and server uptime. Some other QoS metrics include network (such as bandwidth availability), system performance, support response time, server deployment latency, CPU utilization, and RAM monitoring.
2. Application layer services: To overcome application installation challenges such as root privilege requirements and library dependency, virtual appliance technology is adopted in Cloud environment. Virtual appliances, a set of virtual machines including optimized operating systems, pre-built, pre-configured, ready-to-run applications and embedded appliance specific components are emerging as a breakthrough technology to solve the complexities of service deployment. Virtual appliances are proved to provide a better service deployment mechanism [6]. Therefore, they are adopted as a major Cloud component functioning in application layer [7]. The role of appliance providers is providing a ready-to-run software package with predictable behavior.
3. Business layer service: This layer answers end-users business requirements by providing services implemented on top of appliance and infrastructure layer. As shown by Comuzzi *et al*. [8], service level objectives (SLO) in this layer can be modeled by a set of business rules. End-user satisfaction (for example in terms of response time) [2] is another important metric, which can be monitored in this layer.

We consider Cloud services, which are located in different layers, with performance dependencies. It means that if one of the lower-layer (infrastructure layer) services in a chain is not functioning properly, it can affect higher-layer services performance.

### *2.2. Web Service Modeling Ontology*

WSMO [9] defines a model to describe Semantic Web Services (WSs), based on the conceptual design set up in the Web Service Modelling Framework [10]. WSMO identifies four top-level elements as the main concepts [11]:

- Ontologies provide the (domain specific) terminologies used and is the key element for the success of Semantic WSs. Furthermore, they use formal semantics to connect machine and human terminologies.
- Web services are computational entities that provide some value in a certain domain. The WSMO Web service element is defined as follows:
  - Capability: This element describes the functionality offered by a given service.



DOI: 10.1002/spe

  - Interface: This element describes how the capability of a service can be satisfied. The Web service interface principally describes the behavior of Web Services.
- Goals describe aspects related to user desires with respect to the requested functionality, that is, they specify the objectives of a client when consulting a WS. Thus, they are an individual top-level entity in WSMO.
- Mediators describe elements that handle interoperability problems between different elements, for example, two different ontologies or services. Mediators can be used to resolve incompatibilities appearing between different terminologies (data level), to communicate between services (protocol level), and to combine Web services and goals (process level). Besides these main elements, non-functional properties such as cost, location, and reliability are used in the definition of WSMO elements that can be used by all its modeling elements.

### *2.3. Service Level Agreement Management*

Service Level Agreement Management (SLAM) can be defined as a process of management of four major phases:

1. Development of SLA template: SLA templates are created by Clouds to represent their service capabilities. They stored in a repository, which is queried by brokers for discovery of required Cloud services. Next, the templates are used for negotiation on the quality of services. And finally, an SLA contract will be achieved if two parties can reach an agreement on a set of QoS values (which are called SLOs).
2. SLA negotiation: to find the most suitable Cloud service providers, a fast and effective negotiation technique has to be applied. Policy-based [12] and time-dependent [13] are among the most popular negotiation techniques.
3. Service provisioning and deployment: in this phase, the selected service has to be configured properly to be able to deliver its SLOs in SLA contract.
4. Monitoring and violation reporting: deployment of services, which are capable of monitoring SLA contracts have to be completed in this phase. Then, monitoring services have to be executed to evaluate Cloud services based on SLO. When a monitoring service detects that a service is not performing according to its SLA contract, the SLA violation has to be reported.

## 3. MOTIVATING SCENARIO

Amir, the IT administrator of an e-Business website, is required to deploy necessary monitoring services. The e-business services are implemented on top of a LAMP appliance from VMware and hosted on Amazon EC2. In addition, the e-Business application requires access to SalesForce.com as illustrated in Figure 2.

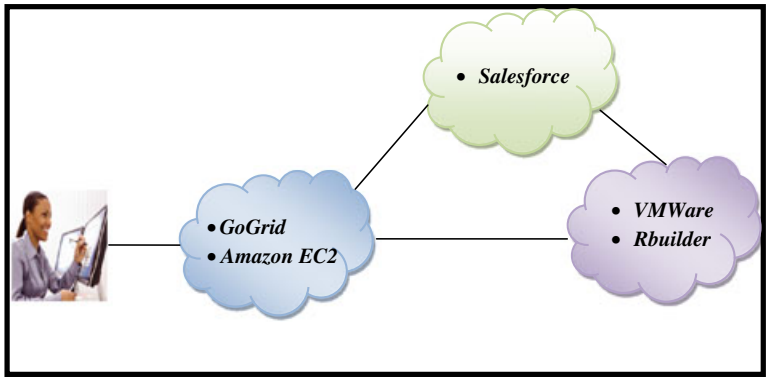


Figure 2. Multilayer cloud service environment.



DOI: 10.1002/spe

In one hand, he has advertised monitoring services in repository, and on the other hand he has signed SLAs, which have to be monitored. Therefore, required monitoring services should be deployed based on the information given in SLA contracts and both Cloud user and provider preferences (regarding cost and reliability of the monitoring services). Because his traditional IT monitoring tools simply cannot monitor varied components, he has decided to use third party monitoring services. He prefers to deploy monitoring services with the highest reliability and the least price. As he intends to automate the whole process of monitoring deployment and execution, he faces the following challenges:

1. The first problem is how to discover required monitoring services using SLA contract when not all parties (Monitoring Service providers and Cloud service providers) are using the same terminology for describing capabilities of monitoring services and QoS criteria which has to be monitored in SLA.
2. The second challenge is how he can rank the discovered monitoring services to increase reliability and save on cost. If we assume he can overcome the first two problems, still the following challenge exists.
3. Imagine the monitoring service which is responsible for monitoring Salesforce services, reports a violation of SLA, hence Amir has reported the violation to Salesforce. Salesforce investigated the violation report and replied back stating that their service during that time was fully functional. Therefore, Amir has checked the monitoring service logs and has found that the root cause of the SLA failure was Amazon EC2 low throughput. Now he faces a new challenge:
   - How could he model services performance inter-dependencies in his system?
   - How could he eliminate SLA failure cascading effects on violation detection?

## 4. CLOUD SERVICE DEPLOYMENT ARCHITECTURE

This section along with introduction of the architecture components shows how SLA contracts and monitoring capabilities are modeled in a heterogeneous environment such as Cloud. In addition, we offer a model for describing Cloud service dependencies to address the issue of SLA failure cascading. First an overview on the architecture is given, and then each component is explained individually except the monitoring service manager, which will be investigated in detail in Subsection 4.2.

As illustrated in Figure 3, in phase ①, a service requestor specifies certain requirements such as hardware specifications like CPU, storage, and memory. A client request may just describe some of the required resources, for example, only CPU and storage. In this situation, default values for other requirements are assigned by the portal. These default values are presented by the portal and could be assigned according to the software requirements and previous requested virtual units of users. In phase ②, software requirements used as an input for discovering the best suited appliances among various repositories of virtual appliance providers [14–16], which is named as virtual market place [17]. Simultaneously, hardware requirements are used by ontology-based matchmaker in discovery component [18] to search for required virtual units. After that negotiator subcomponent starts negotiating with discovered service providers on QoS criteria such as price based on the preferences. During phase ③, once the negotiation is completed the selection component uses user quality of service preferences to select the best virtual appliance and virtual unit offer combination. Then enforceable SLA will be signed by both parties, and the obtained Web service contract [19] is kept in the SLA contract repository. Phase ④ deals with building the Open Virtualization Format (OVF) package and its metadata based on discovered virtual appliances from external appliance providers. Finally in phase ⑤ the monitoring service manager discovers and selects the required monitoring services based on user preferences (cost and reliability) for each SLA contract in the repository. Moreover, the Alert center is listening to all the coming alerts from the monitoring services and uses the dependency knowledge to filter alerts caused as a result of SLA failure cascading.



DOI: 10.1002/spe

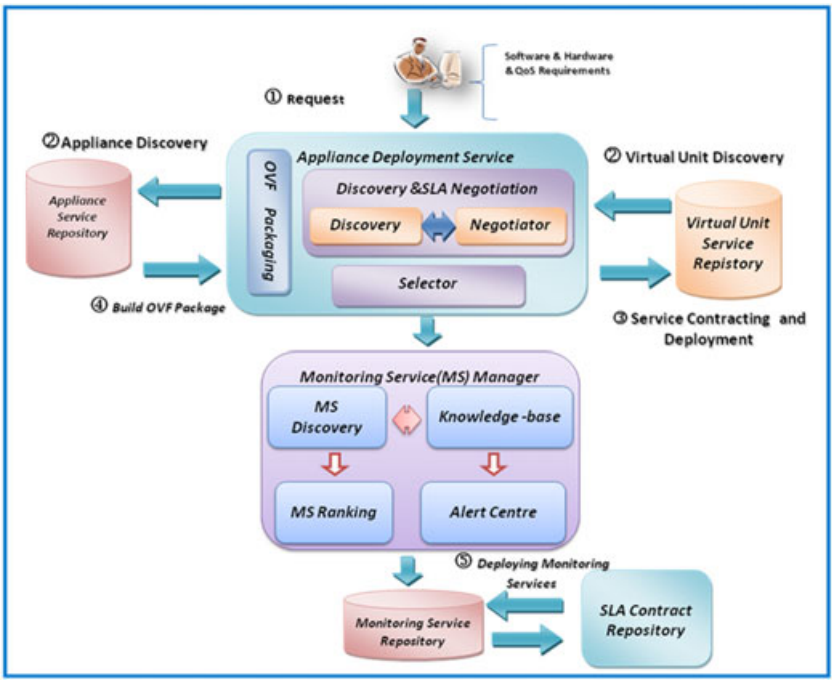


Figure 3. Cloud service deployment architecture.

### *4.1. Architecture components*

*4.1.1. Appliance and virtual unit repository.* It allows Cloud providers to advertise their virtual units and appliances. The advertisement contains descriptions of their features, costs, and the validity time of the advertisement. From a standardization perspective, a common metamodel that describes those services has to be created. However, because of lack of standard, we have developed our own metamodel based on previous standards in this area using WSMO, which was explained in our previous work [18].

*4.1.2. Appliance deployment service.* The component's first responsibility is to discover Cloud services, which have capability to satisfy user software and hardware requirement. The details of the discovery algorithm are given in our previous work [18]. Once the candidates are identified by the discovery process, Cloud services negotiator component negotiates with all candidate appliance and virtual unit providers in parallel. When negotiation ends, the selection component chooses the service providers based on their SLA offers and QoS preferences of the user. After selection, the SLA contract will be created, which has to be monitored. After that, selected images have to be packed based on a standard format. The OVF standard is chosen for this purpose, which has been largely adopted by the industry (VMware, Citrix, and rPath) [20]. The OVF structure can be used to describe which software and from which appliance provider can be used. The possibility of using external references in OVF helps us to add disk images from different appliance providers and enhance flexibility of our approach.

*4.1.3. Service level agreement contract repository.* As explained in Section 3, in a heterogeneous environment such as Cloud, it is difficult to enforce syntax and semantics of services, their quality of services, and consequently SLA contracts. Therefore, applying symmetric attribute–based on matching between SLA contracts and monitoring services is impossible. In order to tackle the problems, the first step is creating a QoS ontology for Cloud services that provides common understanding for QoS among all parties involved in service provisioning. For this purpose (as it is shown



DOI: 10.1002/spe

in Figure 4), we extended WSML to support description of Cloud service QoS. The Figure shows how QoS such as availability, throughput, and their measurement unit ontology can be described using WSML. In the next step, the Cloud QoS ontology is applied to model SLA contracts semantically as illustrated in Figure 5. Consequently, monitoring service discovery can perform semantic matching using defined terms in ontology. The ontology-based discovery can increase the level of flexibility and automation, allowing the two parties to use their own terminology as long as it is related to the commonly understood conceptual model. All the SLAs are stored in the SLA repository to be used as a goal for discovery of required monitoring services. For example, Figures 5(a) and (b) show that SLA needs a monitoring service, which can monitor the bandwidth, storage, and availability of a Cloud service and, if the set of described obligation is not met by the Cloud service, it has to raise an alert.

*4.1.4. Monitoring service repository.* There are traditional server monitoring services that can be used to likewise monitor cloud services. There are also vendor specific monitoring services like EC2 CloudWatch [21]. In addition, there are third party independent Cloud monitoring services like Cloudstatus [22], cloudkick [23], and others, which all advertise their capabilities of Cloud service monitoring in the repository.

For the monitoring service to be discovered, they have to expose their monitoring capabilities in a uniform way as discussed earlier. Monitoring service description contains its monitoring capability [8] and its non-functional properties. We have modeled monitoring services using WSMO in our system. As it is shown in Figure 6, the capability of monitoring services identifies the Cloud services and their QoS criteria, which can be monitored. In addition, non-functional properties of monitoring services such as price, reliability, and location have been considered in modeling. The non-functional properties of monitoring service are exploit by ranking component to rank them

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.example.org/ontologies/Cloud_Service_QoS#",
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }

ontology Cloud_Service_QoS

concept Stockholder
concept Provider subConceptOf Stockholder
concept Customer subConceptOf Stockholder
concept ApplianceProvider subConceptOf Provider
concept VirtualUnitProvider subConceptOf Provider
concept MonitoringService subConceptOf Provider


concept MeasurementUnit
concept Incoming_Bandwidth subConceptOf MeasurementUnit
concept Incoming_Bandwidth subConceptOf MeasurementUnit
concept Outgoing_Bandwidth subConceptOf MeasurementUnit
concept MemoryUnit subConceptOf MeasurementUnit
concept TimeUnit subConceptOf MeasurementUnit
instance GB memberOf MemoryUnit
value hasValue "GB"
instance Kbps memberOf Incoming_Bandwidth
value hasValue "Kbps"
instance Kbps memberOf Outgoing_Bandwidth
value hasValue "Kbps"
instance Sec memberOf TimeUnit
value hasValue "Sec"

concept ApplianceQoS
concept ApplianceReliability subConceptOf ApplianceQoS
concept ApplianceAvailability subConceptOf ApplianceQoS
concept ResponseTime subConceptOf ApplianceQoS
concept Throughput subConceptOf ApplianceQoS

concept VirtualUnitQoS
concept StartTime subConceptOf VirtualUnitQoS
concept VirtualUnitAvailability subConceptOf VirtualUnitQoS
concept Latency subConceptOf VirtualUnitQoS
concept VirtualUnitReliability subConceptOf VirtualUnitQoS
concept RAM subConceptOf VirtualUnitQoS
concept CPU subConceptOf VirtualUnitQoS
concept Storage subConceptOf VirtualUnitQoS
concept Bandwidth subConceptOf VirtualUnitQoS
```

Figure 4. QoS ontology.



DOI: 10.1002/spe

(a)

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { up _"http://www.wsmo.org/ontologies/nfp/upperOnto.wsml#",
 dc _"http://purl.org/dc/elements/1.1#",
 req _"http://www.example.org/contract_goal.wsml#",
 oblg _"http://www.wsmo.org/ontologies/nfp/Obligations.wsml#",
 CSQ _"http://www.example.org/ontologies/Cloud_Service_QoS.wsml#",
 ConOnt _"http://www.example.org/contract_ontology.wsml#"
}
/************************
* Contract GOAL
************************/
goal _"http://www.example.org/contract_goal"

   nonFunctionalProperties
    up#nfp hasValue up#hasObligations
    up#hasObligation hasValue req#DefinitionObligations
    up#hasViolation hasValue oblg#DefinitionViolations
   endNonFunctionalProperties

   importsOntology {CSQ#Cloud_Service_QoS,
   CloudServicesDependencies}

   capability _"http://www.example.org/contract_goal#capl"
       effect
       definedBy
       [ [
           [ [ POBox hasValue "P.O.Box 218",
               City hasValue "Yorktown, NY 10598, USA"] memberOf Contact and
             [ Location hasValue "USA",
               TrustedServices hasValue {"CloudHarmony", "CloudStatus", "Nimsoft",
"Monitis", "Hypertic"}
             ] memberOf ProviderMonitoringConstraint
           ] memberOf ServiceProvider [hasName hasValue "Amazon"] and
           [ [ Street hasValue "30 Saw Mill River RD",
               City hasValue "Hawthorne, NY 10532, USA"] memberOf Contact and
             [ Location hasValue "USA",
               TrustedServices hasValue {"Nimsoft", "Monitis", "Hypertic"}
             ] memberOf CustomerMonitoringConstraint
           ] memberOf ServiceCustomer [hasName hasValue "customer"]
         ] memberOf Parties and
         [ hasInstance hasValue "small",
           hasPrice hasValue "$100"
         ] memberOf ServiceDefinition [ hasName hasValue "EC2"]
       ] memberOf SLA [ hasName hasValue "ServiceAgreement1"]
```

(b)

```
/************************
* Contract Ontology
************************/
ontology _"http://www.example.org/contract_ontology#"
    instance Cloud_Service_QoS memberOf CSQ#Cloud_Service_QoS
      CSQ#MemoryUnit hasValue ?MemoryUnit
      CSQ#Incoming_Bandwidth hasValue ?Incoming_Bandwidth
      CSQ#Outgoing_Bandwidth hasValue ?Outgoing_Bandwidth
      CSQ#TimeUnit hasValue ?TimeUnit
      ...
    axiom DefinitionObligations
       definedBy
        hasObligation(CSQ#hasBandwidth, ?BandwidthSupportValue):-
        CSQ#hasBandwidth[value hasValue ?BandwidthSupportValue] and
         ?BandwidthSupportValue <60 and ?BandwidthSupportValue >50.
        hasStorage hasValue "100".
        hasVirtualUnitAvailability hasValue "98".
/************************
* Contract Web Service
************************/
webService _"http://www.example.org/contract_webService#"
  interface
   orchestration
       //to be done
```

Figure 5. (a) SLA contract goal, (b) SLA contract ontology.

based on user preferences. For example, as illustrated in Figure 6, the monitoring service is capable of monitoring Cloud virtual unit services, which are located in the USA and from specific providers (EC2, GoGrid, and RackSpace), moreover it could only monitor the CPU, bandwidth, memory, availability, and throughput of the services. It is worth mentioning that semantically described monitoring services are using QoS ontology depicted by Figure 4, which provides shared understanding of QoS criteria to all parties in SLA management phases.



DOI: 10.1002/spe

```
/*************************
* Monitoring Service
*************************/
webService _"http://www.example.org/VirtualUnitMonitoringService/Nimsoft"

   importsOntology {CSQ _"http://www.example.org/ontologies/Cloud_Service_QoS.wsml#",
   VU _"http://www.example.org/ontologies/Virtual_Unit.wsml#"}

capability _"http://www.example.org/VirtualUnitMonitoringService/Nimsoft#Capl"
   nonFunctionalProperties
    VU#Price hasValue ?price
    VU#Location hasValue ?loc
    CSQ#Reliability hasValue ?reliablity
   endNonFunctionalProperties
      sharedVariables {?band, ?CPU, ?memory, ?availability, ?throughput}
      effect
       definedBy
       [
         [ VU#Location hasValue "USA",
           VU#Providers hasValue {"Amazon", "GoGrid", "RackSpace"}
         ] memberOf ProviderSupported and
         [ CSQ#Bandwidth hasValue ?band,
           CSQ#CPU hasValue ?CPU,
           CSQ#Memory hasValue ?memory,
           CSQ#Availability hasValue ?availability,
           CSQ#Throughput hasValue ?throughput
         ] memberOf QoS_Criteria
       ] memberOf MonitoringService [ hasName hasValue "Nimsoft"]
```

Figure 6. Ontology-based monitoring service modeling.

### *4.2. Monitoring service manager*

The MSM component is responsible for discovery, selection, and coordination of third party monitoring services. The main objective of this component is automating the process of deploying necessary third party monitoring services by applying semantic service technology WSMO. It consists of several subcomponents such as discovery, ranking, knowledge base, and alert center, which will be explained in following subsections.

*4.2.1. Knowledge base: Cloud service dependency knowledge modeling.* In our work, the service dependency [24] defined as a relationship between one service and one or multiple services where if a service A is dependent on service B, performance of a service A in one or multiple QoS criteria can be affected by service B. The QoS dependency of a Cloud service can be defined formally as follows:

Cloud service dependency: let CS $= \{S_1, S_2, \ldots, S_n\}$ be a set of Cloud services in various levels. Then service dependency can be defined as Equation (1):

$$Cloudservicedependency : \{S_d, S, QC_d, QC, EQCP\} \tag{1}$$

Where: $S_d$ is a service such that its performance in QoS criteria of $QC_d$ depends on the performance of QoS criteria of $QC$ of service S. In addition, $EQCP$ is expected QoS criteria performance from S. It means the service $S_d$ is able to perform according to its SLA on $QC_d$ if S can perform as at least equal to $EQCP$ on $QC$. It is worth mentioning that dependency rules in the knowledgebase are transitive. For example if $S_1$ performance depends on $S_2$ and $S_2$ performance depends on $S_3$, therefore knowledgebase deduces that $S_1$ performance also depends on $S_3$. Knowledge regarding dependencies is particularly significant for:

- Discovery of monitoring service, which will be discussed in Subsection 4.2.2.
- Prevention of false positives because of SLA failure cascading, which will be discussed in Section 4.2.4.

We present an approach for modeling dependencies as a foundation for adapting multilayer Cloud services. Description logics (DLs) is known as formalism for representing knowledge. Consequently, DL languages such as WSML-DL are considered as a heart of the knowledge representation system. After representing dependency of services as knowledge, it can be queried by WSML-reasoner for the reasoning purpose as explained in Subsection 4.2.4. An example of the modeling is shown in Figure 7, where the appliance service availability is dependent on virtual unit availability.



DOI: 10.1002/spe

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.example.org/ontologies/CloudServicesDependencies#",
     dc _"http://purl.org/dc/elements/1.1#",
     up _"http://www.wsmo.org/ontologies/nfp/upperOnto.wsml#",
     CSQ _"http://www.example.org/ontologies/Cloud_Service_QoS.wsml#",
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }
/************************
* Dependencies Ontology
************************/

ontology CloudServicesDependencies

   nonFunctionalProperties
    up#nfp hasValue up#hasDependency
   endNonFunctionalProperties

   importsOntology {CSQ#Cloud_Service_QoS}

// Dependencies Rules

    axiom DefinitionDependencies
       nonFunctionalProperties
        dc#description hasValue
          "Note: the appliance service availability is dependent on virtual unit
availability"
       endNonFunctionalProperties
       definedBy
          ?hasDependency memberOf CSQ#ApplianceAvailability
implies
?hasDependency[up#hasDependency hasValue CSQ#VirtualUnitAvailability].
```

Figure 7. Ontology-based dependency modeling.

*4.2.2. Monitoring service discovery.* As described in Section 4.1, we have deployed all the monitoring services specifications as well as SLA contracts based on WSMO Semantic WSs technology to provide semantic searching and discovery of appropriate monitoring services. Figure 6 demonstrates the monitoring services modeling based on WSMO. Moreover, an example of how to model SLA contract goals based on WSMO is shown in Figures 5(a) and (b). Therefore, we can find out appropriate monitoring services, that fulfil a given SLA contract goal using a discovery solution. In order to realize the discovery component of our proposed architecture, a monitoring service discovery algorithm (Algorithm 1) has been presented. This algorithm applies five matching operations namely Exact, PlugIn, Subsumption, Intersection, and NonMatch in response to the requested SLA contract goal. The definitions of these matching types can be found in [18].

As shown in Figure 3, the *SLA contract repository* is responsible to keep all the information regarding the existing SLA contract goals ($\mathcal{G}_{\text{exist}}$) together with their matching monitoring services and matching types. In the monitoring service discovery algorithm, if each of the existing SLA contract goals can be matched with the requested SLA contract goal ($\mathcal{G}_{\text{user}}$), then the discovery mechanism will not continue anymore, and the already existing goal that consists of the matched monitoring services and the matching type will be sent as the result.

In case of no match between $\mathcal{G}_{\text{user}}$ and $\mathcal{G}_{\text{exist}}$, the discovery mechanism is applied on all available monitoring services (w) located in monitoring service repository ($W$) to find out the matched monitoring services with the $\mathcal{G}_{\text{user}}$. In that case, the candidate monitoring service must be a member of trusted services, which are supported by both service providers and customers. If the aforementioned conditions are satisfied, then the matchmaker process will be executed to find out the matching type between monitoring services and the $\mathcal{G}_{\text{user}}$. In that case, two set of elements of $\mathcal{G}_{\text{user}}$ and w are compared with each other, as follows:

1. $\Omega(\mathcal{G}_{\text{user}})$ with QoS criteria of the monitoring service in which $\Omega(\mathcal{G}_{\text{user}})$ is a collection of obligations of the $\mathcal{G}_{\text{user}}$ and QoS criteria of Service_Definition ($\mathcal{G}_{\text{user}}$), in which other services performance are dependent on. Here is where we need dependency knowledge for discovery. When we are going to discover a monitoring service for an SLA contract of a Cloud service, the monitoring service not only has to be capable of monitoring all QoS criteria described in SLA contracts but also has to monitor QoS criteria of the Cloud service in the contract, in which other services are dependent on. In this way, if ascendant services cause an SLA failure, the monitoring service has the record of that, and using the dependency knowledge box, the root cause of the problem can be detected.



DOI: 10.1002/spe

**Inputs:** $\mathcal{G}_{user}$ is the SLA contract goal, $\mathcal{G}_{exist}$ is set of existing SLA contract goals, $\mathcal{W}$ is set of monitoring services with WSMO format

1: **for all** $\mathcal{G} \in \mathcal{G}_{exist}$ // $\mathcal{G}$ is an existing SLA contract with WSMO format
2: **if** $\mathcal{G}_{user} \equiv \mathcal{G}$ **then**
3: **return** ($\mathcal{G}$ , matchtype($\mathcal{G}$))
4: **else**
5: **for all** $w \in \mathcal{W}$ **do**// $w$ is a monitoring service with WSMO format
6: **if** $Mw \in (T_p(\mathcal{G}_{user}) \sqcap T_c(\mathcal{G}_{user}))$ **and** $\mathrm{P}(\mathcal{G}_{user}) \in \mathrm{P}w$ **then**
// $Mw$ is the monitoring service name, P is the service provider
// $T_p(\mathcal{G}_{user})$ and $T_c(\mathcal{G}_{user})$ are trusted services of the provider and the customer
7: $\Omega(\mathcal{G}_{user}) \equiv \mathrm{DC}(\mathcal{G}_{user}) \sqcup \mathcal{O}(\mathcal{G}_{user})$
// DC is QoS criteria of Service_Definition ($\mathcal{G}_{user}$) which other services perfomance are dependent on
// $\mathcal{O}$ is set of obligations in the SLA contract
8: **if** $\Omega(\mathcal{G}_{user}) \equiv QoS(w)$ **and** $N(\mathcal{G}_{user}) \equiv Nw$ **then**
// QoS is set of Quality of Service criteria and $N$ is set of non-functional properties
9: **return** ($w$, *Exact*)
10: **else if** $\Omega(\mathcal{G}_{user}) \sqsubseteq QoS(w)$ **and** $N(\mathcal{G}_{user}) \sqsubseteq Nw$ **then**
11: **return** ($w$, *PlugIn*)
12: **else if** $QoS(w) \sqsubseteq \Omega(\mathcal{G}_{user})$ **and** $Nw \sqsubseteq N(\mathcal{G}_{user})$ **then**
13: **return** ($w$, *Subsumption*)
14: **else if** $\neg(\Omega(\mathcal{G}_{user})) \sqcap QoS(w) \sqsubseteq \bot$) **and** $\neg(N(\mathcal{G}_{user}) \sqcap Nw \sqsubseteq \bot)$ **then**
15: **return** ($w$, *Intersection*)
16: **end if**
17: **end for**
18: **end if**
19: **end for**
20: **return** ($\mathcal{G}$, *NonMatch*)

Algorithm 1. Monitoring service discovery ($\mathcal{G}_{user}$, $\mathcal{G}_{exist}$, $\mathcal{W}$).

2. Non-functional properties of the $\mathcal{G}_{\text{user}}$ and w must be compared with each other (reliability, and cost).

*4.2.3. Monitoring service ranking.* QoS properties for ranking of monitoring services are cost, and reliability which will be defined as follows:

1. Reliability: one important comparison factor for monitoring services is reliability. For example, initially, we have evaluated pingdom [25] and monitis [26] for monitoring Cloud services. However, we quickly realized that these services triggered many false positives because they only do single-node outage verification and, occasionally, those monitoring nodes themselves experienced network issues that caused them to inadvertently trigger outages. Panopta [27] on the other hand, does triple-node outage verification before triggering an outage, which provides a much more reliable availability metric. In order to measure the reliability of monitoring services, a formula is developed as shown in Equation (2).
2. Cost: monitoring service cost is a non-functional requirement of a user who wants to deploy required monitoring services. In our problem, minimization of deployment cost is considered as the objective of users. Besides, there are different kinds of pricing for monitoring services in the market, for example EC2 offers Cloudwatch for free for a 5-min interval monitoring with any EC2 instance. Other services charge per server or per service. For example, Cloudkick charges $99/month for six servers/month. Panopta charges per service monitored at around $1.5/service/month.

In the selection phase, based on user QoS preferences, all discovered services are ranked and the top service in the ranked list is returned. As discussed in the previous section, reliability and cost are considered as QoS criteria of monitoring services. We first offer an approach to measure reliability of monitoring services, and then show how monitoring services can be ranked by applying



DOI: 10.1002/spe

the Analytical Hierarchy Process (AHP) approach [28]. There are several third-party Cloud monitoring services existing in the market, which allows users some degree of flexibility in choosing the service that best suits their preferences. One very important factor to consider while comparing monitoring services is their reliability, which can be measured based on monitoring services' false reports generated for each monitored QoS criteria as shown in Equation (2). The false positive is referring to the frequency with which the monitoring service reports violation of SLA in error. And the false negative is the frequency with which the monitoring service fails to detect violation of SLA, which actually happened.

$$Reliability = \sum_{c=1}^{k} I_c \left( W_{nc} \times \frac{total\ number\ of\ detection}{false\ violation\ detection} + W_{pc} \frac{total\ number\ of\ detection + number\ of\ violation\ not\ detected)}{number\ of\ violation\ not\ detected} \right) \quad (2)$$

Where $\begin{cases} I_c \text{ is the relative importance of criterion c.} \\ W_{pc} \text{ is the relative importance of false positive rates compared with false negative rates} \\ W_{nc} \text{ is the relative importance of false negative rates compared with false positive rates} \end{cases}$

The reason for introducing weights for false positive and negative rates is to allow users giving higher weights to the rates and criteria which are more important to them. To capture weights in the equation and rank the monitoring services, AHP has been applied, which is one of the most applied methods in the multiple criteria decision making category. The AHP method was suggested by Satty in 1998 [28] and is based on pairwise comparison of criteria to determine their weights in utility function. The major contribution of AHP is to convert subjective assessments of relative importance to numerical values or weights. The methodology of AHP is based on pairwise comparisons of the criterion by asking the question of "how important is criterion Ci compared with criterion Cj?" The answer (which could be one which shown in Table I) to the question determines weights for criteria. Figure 8 depicts the process of choosing the best monitoring service provider when two criteria (cost and reliability) are considered.

After the pairwise comparison, as Figure 8 shows relative importance has been assigned to each criterion. In the next step, similar questions have to be asked to evaluate the performance scores for monitoring services on the subjective criteria. And then, based on the results of this phase, we can rank alternatives and select the one with the highest rank. We have adopted an open decision maker software to our project, and result of the ranking for the discovered services is demonstrated in Figure 11.

*4.2.4. Violation reporting.* This subcomponent is in charge of violation detection and reporting to enforce accurate penalties by elimination of SLA failure cascading effects on violation detection. In order to do that, as shown in Algorithm 2, if it receives violation reports on performance of a service ($S_d$) on a QoS criteria ($QC_d$) from deployed monitoring services (line 2 in Algorithm 2), it queries the knowledgebase as illustrated in Figure 9 to check whether the service performance for that criteria depends on other service's performance or not. For example, as it can be seen in Figure 9, the system queries the knowledgebase to acquire which services can affect Salesforce availability. If it is dependent, then it checks the performance of the QoS criteria of the ascendant service; if

Table I. Major scale of pair-wise comparisons.

| *Scores* | *Response to the question* |
|---|---|
| 1 | Equal importance or preference. |
| 3 | Moderate importance or preference of one over another. |
| 5 | Strong or essential importance or preference. |
| 7 | Very strong or demonstrated importance or preference. |
| 9 | Extreme importance or preference. |



DOI: 10.1002/spe

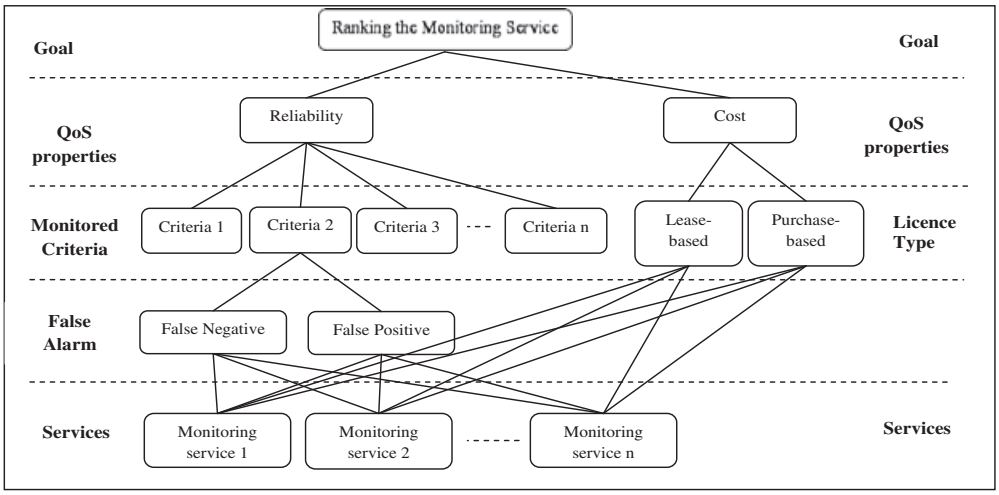


Figure 8. Applying AHP for ranking monitoring services.

1: False_alarm=flase;
2: Catch VR
3: $S_d \leftarrow$ VR. S
4: $QC_d \leftarrow$VR.QC
5: For all $S_a$ in DS do // DS is a set of all Deployed services
6: If exist $(S_d, S_a, QC_d, QC_a, EQCP\}$ in knowledge-based then AS$\leftarrow S_a$
7: Endif
8: Endfor
9: For each $S_a$ in AS do
10: If ( $QC_a$ is worse than EQCP) then
11: False_alarm=true;
12: Endif
13: Endfor

Algorithm 2. False_alarm_detection.

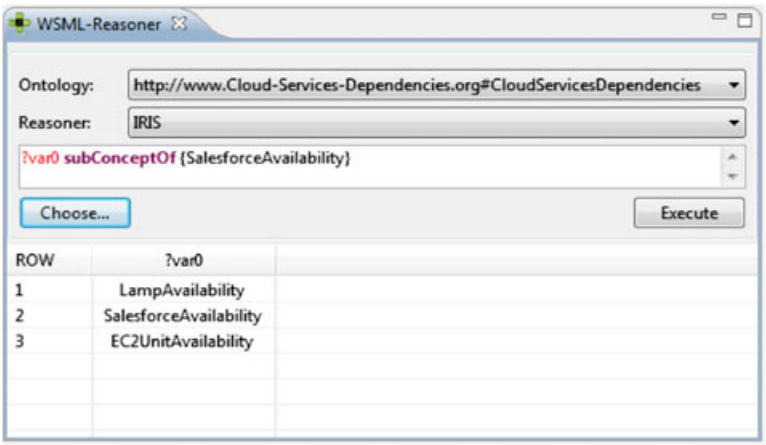


Figure 9. Dependency reasoning for salesforce availability.



DOI: 10.1002/spe

it is not as equal or better than *EQCP*, then it derives that the reported violation is false(line 9-13 in Algorithm 2), and instead, it is reporting the root cause of the problem and all the consequent effects.

## 5. PERFORMANCE EVALUATION

Experiments in this section were run on a system with Intel i3-350-M, 2.27-GHz processor and 4 GB of RAM. The main experiments are: (i) a feasibility test for the a case study with one SLA contract and five monitoring services in the repository and (ii) measuring the deployment time (discovery and ranking execution time) for variation in number of monitoring services in the repository.

### *5.1. Monitoring services discovery for case study*

For the case study, the SLA contract in Figures 5(a) and (b) is considered to be the goal, and set of five monitoring services with different QoS parameters and capabilities were imported to the repository. The summary of the capabilities are shown in Table II. Figure 10 illustrates the output of discovery algorithm. As Figure 10 illustrates, three monitoring services as monitis, hyperic, and nimsoft could match (with match type of *subsumption*) the requirements. The other two monitoring services were not discovered because Cloudharmony does not have the capability to monitor the SLA contract, and Cloudwatch was not in the trusted list (as shown in Figure 5(a)) of both the Cloud and the user. Then in the ranking phase, as it is shown in Figure 11, monitoring services are ranked using AHP and nimsoft; monitoring service could score higher when reliability criteria is more important to the user.

Table II. Monitoring services in the repository for the case study.

| Monitoring service | QoS monitoring capabilities | Service monitoring capabilities | Location |
|---|---|---|---|
| Monitis | CPU, memory, storage, load, processes, availability | EC2, GoGrid | USA |
| Hyperic | CPU, memory, storage, availability, bandwidth | EC2, GoGrid | USA |
| Nimsoft | CPU, memory, storage, availability, bandwidth | EC2, GoGrid | USA |
| Cloudharmony | Availability, throughput | EC2, GoGrid, RackSpace, Flexiscale | USA |
| Cloudwatch | CPU, memory, storage | EC2 | USA |

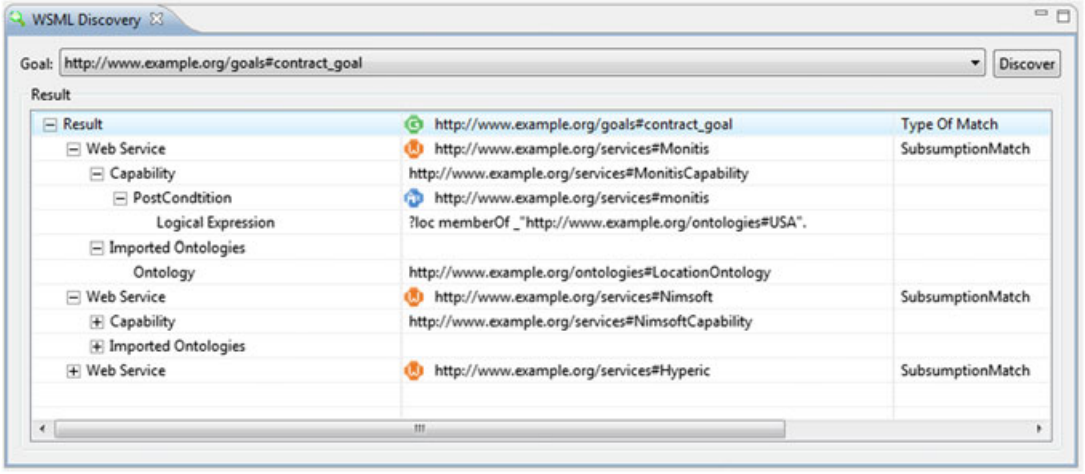


Figure 10. Monitoring service discovery algorithm validation for the case study.



DOI: 10.1002/spe

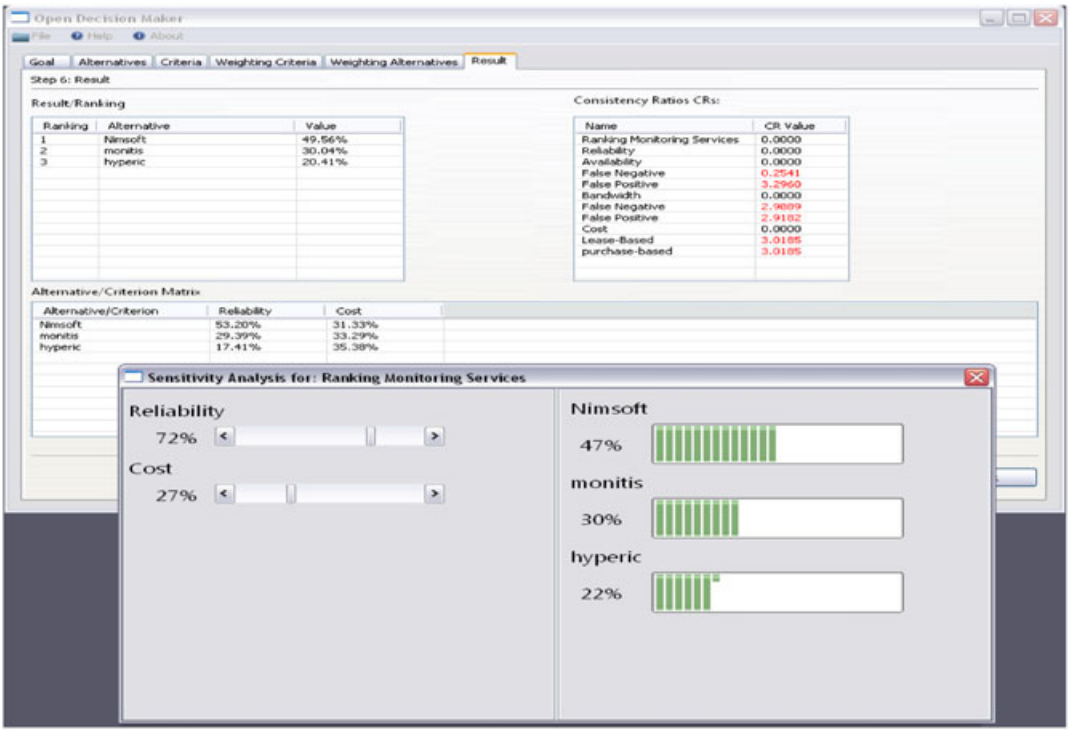


Figure 11. Ranking algorithm validation for the case study.

### 5.2. *Deployment time measurement*

In these experiments, we increased the number of monitoring services in the repository to investigate the scalability of our approach in terms of execution time. Then, for each case of different numbers of monitoring services, we repeated the experiment 20 times in a way that each time the SLA contract QoS types were randomly altered. As it is illustrated in Figure 12, the mean of deployment time, which is the sum of discovery and ranking time, has increased from less than 3 s when there are only 30 offers in the repository to almost 10 s when the number of offers increased to 120. Therefore, although our approach causes semantic-based matching overhead, still it can discover and rank services in reasonable time even for large number of services in the repository.

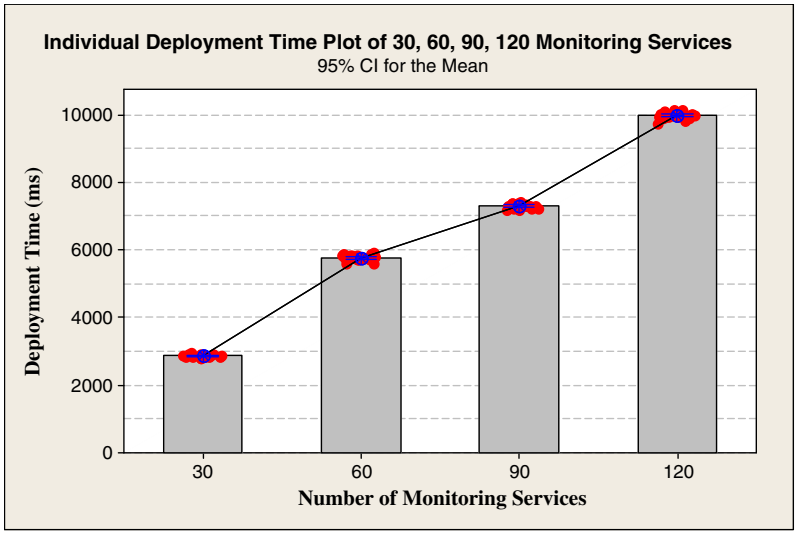


Figure 12. Execution time for monitoring service discovery and ranking.



DOI: 10.1002/spe

## 6. RELATED WORK

Winkler *et al*. [24] propose an approach for the automated management of SLAs to support composite services. In that approach, the explicit knowledge about a set of dependencies to automate the tasks of negotiation and renegotiation of SLAs, and the handling of SLO violations are taken into account. The authors propose a formal model for dependency representation and a set of algorithms for automating SLA management tasks, namely the validation of SLAs during negotiation, and the evaluation of renegotiation requests, and SLO violations to achieve their approach. The paper focuses on SLA management for composite services with an explicitly modeled workflow description whereas our approach is deployed based on a semantic-based modeling for services, that is, WSMO to facilitate the interoperability problems as well. Furthermore, our approach provides a deployment strategy for a set of appropriate monitoring services to support the SLA contract, which was not addressed by Winkler *et al*.

Bodenstaff *et al*. [29] propose an approach called MoDe4SLA to monitor dependencies between SLA when managing composite services. In that case, different types of dependencies between services and the impact that services have on each other are analyzed during development phase. Moreover, bilateral monitoring results can be combined with analyzed dependencies and impacts of the composite service. In contrast with our approach, no support is provided by the authors for discovering the monitoring services by means of semantic Web and QoS criteria. And in addition, the approach has not provided an inter-Cloud language to share common understanding regarding the QoS criteria and their measurement units.

Kotsokalis *et al*. [4] offer a generic architecture, which can be used across different domains and use cases to support SLA management and tries to address all phases of the SLA management lifecycle, from negotiation and establishment to termination, with respect to the existence of SLA interdependencies. The goal is to ensure that operations of existing services would not be affected. In contrast with our approach, no mechanism is provided for semantic annotation of monitoring services and the SLA contract goals.

Emeakaroha *et al*. [5] propose a framework called LoM2HiS to provide a mapping for low-level resource metric to high-level SLA parameters. The proposed framework has aimed to develop an infrastructure for autonomic SLA management and enforcement. The framework is able to detect possible SLA violation threats based on predefined threat thresholds and announce the enactor component to prevent the threats. Although, the multi-level SLA issues have been discussed in the work, no solution for modeling dependency knowledge was proposed.

Theilmann *et al*. [3] discusses the need for multi-level SLA management approaches in order to fuel the next step towards a service-oriented economy. In that case, the authors propose a conceptual architecture, which consistently bridges and mediates the various views of stakeholders and business/IT layers. However, no implementation is provided to verify the applicability of the proposed architecture.

In summary, compare to related efforts, our approach is unique in the following aspects:

- It has brought monitoring service deployment (discovery and selection) and SLA validation together while it is enriched with semantic technology for being highly flexible and enabling inter-cloud language for SLA management.
- In addition, it offers ranking methods for monitoring services based on user preferences on reliability and cost, which with the best of our knowledge has not been investigated before.
- Another novel contribution of the work is consideration of dependency knowledge in discovery of monitoring services and violation detection.

## 7. CONCLUSIONS AND FUTURE WORKS

Automating the process of SLA management in Cloud needs an approach for deployment of required monitoring services. In heterogeneous environment such as Cloud, the discovery of monitoring services cannot be done using syntax-based matching of advertised monitoring services in



DOI: 10.1002/spe

repository and signed SLAs. Consequently, this work proposed an ontology-based approach for modeling monitoring capabilities and SLA contracts to semantically match them and avoid low recall caused by lack of common QoS understanding. This paper started with modeling of QoS ontology using WSMO and continued with using the QoS ontology in SLA and monitoring services description to build inter-Cloud language for monitoring service deployment. In addition, the paper discusses the effects of QoS dependencies among services on generation of false SLA violation report; therefore, it tackled the problem by deploying proper monitoring services and filtering violation reports using dependency knowledge. We tested our approach on a case study and the result shows the efficiency and effectiveness of the proposed work. In addition, the computational overhead of the semantic discovery for the repository has been measured for various numbers of monitoring services. The Result shows the approach is scalable and the deployment can be accomplished in a reasonable time. Future works can investigate the approaches for dependency-aware Cloud services composition and their effects on the composition reliability, deployment time, precision and recall.

## REFERENCES


1. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 2009; **25**(6):599–616.
2. Jones D. *The Definitive Guide to Monitoring the Data Center, Virtual Environments, and the Cloud*. Nimsoft: Campbell, CA, 2010.
3. Theilmann W, Winkler U, Happe J, de Abril IM. Managing on-demand business applications with hierarchical service level agreements. *Future Internet - FIS 2010 - Third Future Internet Symposium*, Berlin, Germany, September 20–22, 2010.
4. Kotsokalis C, Yahyapour R, Gonzalez MAR. SAMI: The SLA Management Instance. In *2010 Fifth International Conference on Internet and Web Applications and Services*. IEEE, 2010; 303–308.
5. Emeakaroha VC, Brandic I, Maurer M, Dustdar S. Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments. In *2010 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, 2010; 48–54.
6. Sun C, He L, Wang Q, Willenborg R. Simplifying service deployment with virtual appliances. *IEEE International Conference on Services Computing*, 2008.
7. Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing 360-degree compared. *IEEE Grid Computing Environments Workshop*, 2008.
8. Comuzzi M, Spanoudakis G. Describing and verifying monitoring capabilities for service based systems. *Proceedings of the CAiSE 2009 Forum*, Amsterdam, Netherlands, June 2009.
9. WSMO Working Group. Available from: http://www.wsmo.org.
10. Fensel D, Bussler C. The Web Service Modeling Framework WSMF. *Electronic Commerce: Research and Applications* 2002; **1**(2):113–137.
11. Fensel D, Lausen H, Polleres A, de Bruijn J, Stollberg M, Roman D, Domingue J. *Enabling Semantic Web Services: The Web Service Modelling Ontology*. Springer-Verlag: New York, 2006.
12. Zulkernine F, Martin P, Craddock C, Wilson K. A policy-based middleware for web services SLA negotiation. In *Proceedings of the IEEE International Conference on Web Services (ICWS'08), industry track*. IEEE CS Press: Beijing, China, September 23–26, 2008.
13. Faratin P, Sierra C, Jennings NR. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems* 1998; **24**(3–4):159–182.
14. Virtual Appliance Marketplace. Available from: http://www.vmware.com/appliances/learn/overview.html.
15. Wilson M. Constructing and managing appliances for cloud deployments from repositories of reusable components. *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, 2009. Workshop: https://www.rpath.org/ui/.
16. Comuzzi M, Pernici B. A framework for QoS-based Web service contracting. *ACM Transactions on the Web* 2009; **3**(3):1–52.
17. 3Tera AppStore. Available from: http://www.3tera.com/AppStore/.
18. Dastjerdi AV, Tabatabaei SGH, Buyya R. An effective architecture for automated appliance management system applying ontology-based cloud discovery. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2010; 104–112.
19. rBuilder Online. Available from: http://www.rpath.com/corp/wind-down.
20. Open Virtualization Format White Paper, June 2009. Available from: http://www.dmtf.org/standards/published_documents/DSP2017_1.0.0.pdf.
21. Amazon CloudWatch, monitoring for AWS cloud resources. Available from: http://aws.amazon.com/cloudwatch/.
22. Enterprise Monitoring and Management for Cloud Services. Available from: http://www.hyperic.com/products/Cloud-monitoring.html.
23. Cloudkick Cloud management toolkit. Available from: https://www.cloudkick.com/.




DOI: 10.1002/spe

24. Winkler M, Springer T, Schill A. Automating composite SLA management tasks by exploiting service dependency information. In *2010 Eighth IEEE European Conference on Web Services*. IEEE, 2010; 59–66.
25. An uptime and performance monitoring toolkit. Available from: http://www.pingdom.com.
26. Monitis, All-in-One Monitoring Platform. Available from: http://portal.monitis.com.
27. Panopta. Advanced Server Monitoring. Available from: http://www.panopta.com.
28. Saaty TL. *Fundamentals of Decision Making and Priority Theory with the Analytic Hierarchy Process*, Vol. 6. RWS Publications: Pittsburgh, PA, 1994.
29. Bodenstaff L, Wombacher A, Reichert M, Jaeger MC. Monitoring dependencies for SLAs: The MoDe4SLA approach. In *IEEE International Conference on Services Computing, 2008. SCC '08*, Vol. 1. IEEE, 2008; 21–29.



DOI: 10.1002/spe