# Envisioning SLO-driven Service Selection in Multi-cloud Applications

Abdessalam Elhabbash
Yehia Elkhatib
Gordon S Blair
{i.lastname}@lancaster.ac.uk
School of Computing and Communications
Lancaster University, UK

Yuhui Lin
Adam Barker
John Thomson
{yl205,adam.barker,j.thomson}@st-andrews.ac.uk
School of Computer Science
University of St Andrews, UK

## ABSTRACT

The current large selection of cloud instances that are functionally equivalent makes selecting the right cloud service a challenging decision. We envision a model driven engineering (MDE) approach to raise the level of abstraction for cloud service selection. One way to achieve this is through a domain specific language (DSL) for modelling the service level objectives (SLOs) and a brokerage system that utilises the SLO model to select services. However, this demands an understanding of the provider SLAs and the capabilities of the current cloud modelling languages (CMLs). This paper investigates the state-of-the-art for SLO support in both cloud providers SLAs and CMLs in order to identify the gaps for SLO support. We then outline research directions towards achieving the MDE-based cloud brokerage.

## CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Domain specific languages**.

## KEYWORDS

Cloud Computing, Cloud Modelling Languages, Domain Specific Language, Service Level Agreements, Service Level Objectives

**ACM Reference Format:**
Abdessalam Elhabbash, Yehia Elkhatib, Gordon S Blair, Yuhui Lin, Adam Barker, and John Thomson. 2019. Envisioning SLO-driven Service Selection in Multi-cloud Applications. In *IEEE/ACM 12th International Conference on Utility and Cloud Computing Companion (UCC '19 Companion), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3368235.3368831

## 1 INTRODUCTION

Cloud computing's major success over recent years has resulted in a significant expansion in the provider market and the number of services they provide. The growing expansion of this market, however, poses a challenge to its customers. Theory and practice are in agreement that such wide range of choice overwhelms customers [14], leading them to select suboptimal instances [21].

The problem of cloud service selection is application-specific, but provider guarantees are understandably broad and generic. Such guarantees come in the form of service level agreements (SLAs) that specify a set of service level objectives (SLOs) as commitments from the provider regarding *quantitative* aspects of their service [1]. When cloud services are functionally equivalent, SLOs play a focal role in the decision of selection. This demands thorough understanding of the SLOs *and* how they would affect an application. However, this problem is complex and, worse, variable across providers [15, 23]. Therefore, the current practice of manually inspecting SLAs to select services with matching SLOs is a cumbersome one.

We believe that a model driven engineering (MDE) approach would be appropriate to make it easier for cloud customers to identify the services that best suit their applications. Specifically, we envision that modelling languages should aid customers to model their applications' functional requirements and SLOs. An intermediary brokerage system could then utilise such models to select services that suit the application.

In the literature, a handful of cloud modelling languages (CMLs) have been proposed. A CML uses modelling concepts to raise the level of abstraction, enabling customers to describe their specific application needs. These could then be systematically matched against specific cloud service offerings. With this purview, we investigated the capabilities of the CMLs in terms of SLO modelling. Our investigation discovered the following significant shortcomings.

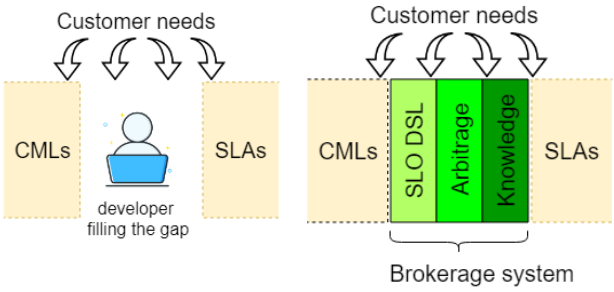
- Current CMLs provide limited support for SLO modelling, i.e. a large portion of SLOs that cloud customers need are **not supported** by current CMLs. This is because they have been designed mainly for modelling the structure and functional requirements of an application.
- There is a lack of post-deployment optimisation of the service selection based on the actual performance of the application. Recent studies show surprising results of inconsistent performance of promised cloud services [27].
- Some CMLs allow attaching non-functional requirement models to the functional ones they support. However, these are expressed using standards that were designed for providers to specify their SLOs, e.g. *WS Policy* [28]. We believe that SLO modelling should adopt the customer application perspective. Customers are more concerned about the actual performance levels of their applications rather than that the provider promises [14].

Therefore, we conclude that customer SLO specification using such standards is inadequate as they are primarily designed for cloud provider policies rather than customer operational requirements.

In addition, we also found that providers do not commit to many SLOs that customers might need. For example, there is no cloud *compute* service that specifies an SLO of 'mean time to recover'.


*UCC '19 Companion, December 2–5, 2019, Auckland, New Zealand*

ACM ISBN 978-1-4503-7044-8/19/12…$15.00
https://doi.org/10.1145/3368235.3368831




**(a) Current state of affairs** **(b) Envisioned approach**

**Figure 1: High-level view of the envisioned approach**

The above observations are based on a wide study we conducted to identify the limits of SLO specification. We examine the suitability of current CMLs in supporting different SLOs, finding several major deficiencies. It also identifies the various SLO constituents of major cloud provider SLAs, those specified in SLA standardisation documents, and those proposed in academic research efforts. Fig. 1a shows that with the current state of the art, cloud customers still need to manually fill the gap between CML capabilities and service SLAs. Our vision is to automate this manual approach through an intelligent MDE-based brokerage system as shown in Fig. 1b. As such, we distill our observations to draw a set of research directions that our envisioned MDE approach needs to follow in order to deliver assistance to cloud customer SLO-driven service selection. In summary, in this paper, we:

(1) Investigate the level of CML support for SLO modelling (§2);
(2) Explore the SLOs provided in the cloud market (§3);
(3) Identify the gap in the state of the art that developers are currently filling using manual methods (§4); and
(4) Provide insight into what research is needed to attain SLO-driven service selection using MDE methods, and comment on the obstacles in the road to such envisioned future cloud development (§5).

## 2 SLOs IN CLOUD MODELLING LANGUAGES

CMLs have long been proposed to raise the level of abstraction and increase the degree of automation in the development of cloud applications [10]. However, not all of them support SLO specification in the process of modelling cloud applications. In this section, we briefly introduce the CMLs that do and focus particularly on SLO specification capabilities. We surveyed the cloud computing literature and identified a set of ten CMLs (Table 1). We found that only four of them to either provide syntax for capturing SLOs or support attaching such annotations. We briefly introduce these CMLs.

### 2.1 Blueprint

Blueprint [26] provides concepts for representing service-based applications to facilitate deployment and migration on cloud services. The provided concepts also allow the representation of different cloud service offerings. Then both the application and service descriptions are submitted to an assumed marketplace that is meant to match them.With respect to SLO modelling, Blueprint does not provide concepts for capturing the service levels required by customers. However, this approach assumes that policy profiles can be attached to the Blueprint, and these can be written in any policy specification language such as WS-Policy [28], SLAng [22], etc.

### 2.2 TOSCA

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [12] is an OASIS standard for describing the structure of cloud applications. It provides an exchange format to describe the composite application components, their relationships, and functionalities. The description, called a service template, is to be realised by an orchestration tool that is able to parse and understand the service template and, accordingly, deploy application components on cloud services. Similar to the Blueprint approach, TOSCA does not provide concepts to capture SLOs. Instead, a policy profile is assumed to be attached to the service template.

### 2.3 MULTICLAPP

MULTICLAPP [19] introduces a UML-based profile to model cloud applications. The application components are represented as UML artefacts that can be annotated with deployment information. The intention is to create a provider-independent model of the application, which can then be transformed and customised into a provider-specific model using a transformation engine. Such engine, however, is not proposed in the MULTICLAPP paper.

In MULTICLAPP, the application is modelled as a set of Artefacts, each of which corresponds to an application component. Furthermore, QoS requirements can be modelled as UML-based *QoSParameter* expressions. Thereby, the specification of SLOs in MULTICLAPP is carried out by associating the application artefacts with the QoSParameter expressions. Each expression consists of three tagged values, namely: Property, Operator and Value.

### 2.4 GENTL

The GENeralized Topology Language (GENTL) [7] is another language to describe the topology of cloud applications. It borrows the modelling concepts of other languages like Blueprint and TOSCA for the sake of generalising the cloud application modelling process. The main motivation here, though, is to enable the mapping from other languages into a unified model so that additional information can be attached to the model in the form of annotations. The annotation scheme supported by GENTL enables users to attach information to the application topology itself. These annotations can be used to attach SLOs to the application topology components. The annotations are assumed to be used by the GENTL Environment to find services that satisfy the SLOs. However, it is not clear how the authors' claim of specifying QoS levels using annotations can be realised.

The main observation from the above is that current CMLs focus on structural and functional aspects of multi-cloud applications with limited support for customer-oriented SLO specification.

## 3 SLOs IN CLOUD SLAs

We now explore the SLO contents that are expected to exist in a cloud provider's SLA, and contrasts them to those that currently exist.

### 3.1 A cloud SLA content model

A cloud service agreement is a document that contains terms that govern the provided services [1]. These terms are typically specified by the provider and in some cases are negotiable. The part of this document that is concerned by the quality levels of the service is called the SLA. This part contains SLOs, which are quantitative characteristics of the cloud services. These are measured during service operation, and are obligations that the provider must adhere to.

**Table 1: An overview of the studied CMLs (in chronological order) and their SLO specification capabilities.**

| CML | Capability | SLO specification |
|---|---|---|
| **Blueprint [26]** | Supports | Attaching policy file |
| **MULTICLAPP [19]** | Provides | UML-based Notation |
| **TOSCA [12]** | Supports | Attaching policy file |
| **GENTL [7]** | Provides | Annotations |
| **MOCCA [25]** | n/a | n/a |
| **CloudMIG [18]** | n/a | n/a |
| **RESERVOIR [13]** | n/a | n/a |
| **StratusML [20]** | n/a | n/a |
| **CAML [11]** | n/a | n/a |
| **CloudML [17]** | n/a | n/a |

Considering the contents of cloud SLAs that relate to quantitative service levels, we aim to identify the main SLO areas that are expected in SLAs. For this purpose, we surveyed relevant ISO standards, NIST and ETSI publications, and well-cited SLA surveys [1, 6, 9, 15, 16]. As a result, we identified the following SLO areas. Table 2 presents a complete list of specific SLOs for each area.

- **Availability**: quantifies the degree to which service functionality is obtainable and useful in response to user requests. Any downtime, whether scheduled or failure-caused, contributes to the inability of users to access service functionality, reducing availability. Quantification can take many forms – one approach is to divide the downtime over a longer period of service time.
- **Performance**: is a general property that can be interpreted variously according to the concerns of the provider and the customer. It includes different components that can have alternative or collective SLOs to measure. For example, responsiveness of a cloud service can be measured using response time. Performance can also include other attributes such as machine capacity, scalability, etc.
- **Reliability**: reflects the ability to provide consistent functionality. It can include many aspects such as fault tolerance, data backup, and disaster recovery. For instance: providers can commit SLOs to set a limit for recovering a service after failure.[1] Other related SLOs pertain to frequency of making backups, the period of keeping backups, the time to restore data, among others.
- **Cost**: This property specifies the pricing model of the cloud service and the price that the customer pays per service usage. It also specifies the penalty rates that the provider pays to the customer in compensation if the service violates of any SLA items.
- **Termination of service**: relates to situations where the customer decides to stop using the service. Related SLOs such as the period after which the customer data and logs should be deleted from the provider's storage can also be set in the SLA.
- **Support**: As one of the motivations for adopting cloud computing is to remove the burden of maintaining physical infrastructure from the customer, providers are required to provide technical support for customers to resolve any emerging issues during service operation. Consequently, the cloud SLA can contain SLOs related to the support property including support hours, the Maximum First Support Response Time, and more.
- **Change management**: Changes to service features and functionalities may affect the customers that use that service. Therefore, some SLOs can be included to ensure that the customer will be prepared for such changes. An example is the Minimum Service Change Notification Period, which is the minimum time period the customer should be notified before a change is implemented.
- **Data management**: This property includes procedures related to how a provider should deal with customer data with respect to access, control, storage, etc. These procedures can generally be specified using qualitative measures. However, SLOs can also be used to specify quantitative objectives related to those procedures. For example, the Data Deletion SLO specifies time period during which the provider should delete the customer data upon request.

[1] The Recovery Time SLO specifies the acceptable time to restore service functionality after a severe failure, whereas the Recovery Point SLO specifies the amount of time to which data loss is acceptable before the occurrence of the failure.

## 3.2 SLO contents in current provider SLAs

We also studied the SLOs provided in the SLAs of six market leading cloud providers: Amazon AWS, DigitalOcean, Google Cloud, Joyent, Microsoft Azure [5], and Rackspace. The SLOs adopted are as follows:

- **Monthly Uptime Percentage (MUP)** is widely adopted to express availability. There are various ways to calculate this, however, depending on the service features and attributes. For example, for Microsoft Azure Virtual Machines the calculation of MUP depends on whether the VM is deployed as a single-instance or as an instance of an availability set, whereas for Microsoft Azure Storage MUP is a function of the Average Error Rate where the Error Rate is defined as *"the total number of Failed Storage Transactions divided by the Total Storage Transactions during a set time interval"*.
- **Monthly Availability Percentage** specifies the acceptable availability percentage of a cloud service in terms of the average error rate. The error rate is calculated as the percentage of failed requests in a time interval. This SLO is adopted in Azure Cosmos DB [5] and Amazon DynamoDB [2].
- **Maximum Processing Time** is defined as the time period after which a service will be considered unavailable if all requests to access that service return error or no response. It is used as a factor to calculate the availability of the service (i.e. MUP) in the SLAs of Microsoft Azure' Network Watcher, Storage, and Container Registry services [5].
- **Monthly Recovery Time Objective** specifies the acceptable average period of time beginning from the time a customer initiates a control transfer from an instance experiencing an outage to the time when the instance is running. This SLO is adopted only by the Microsoft Azure Site Recovery [5].
- **Monthly Consistency Attainment Percentage** measures the percentage of successful requests that do not exhibit consistency violation and thus achieve the required consistency level. It is adopted only by the Microsoft Azure Cosmos DB SLA [5].
- **Monthly Latency Attainment Percentage** specifies the acceptable percentage of successful requests experiencing latency greater than a certain threshold. It, too, is adopted only by Microsoft Azure Cosmos DB [5] to measure the latency for the database read, write, and configuration operations.
- **Monthly Throughput Percentage** specifies the acceptable percentage of requests that consume rates higher than the ones allocated by the service. Again, this SLO is adopted only by Microsoft Azure Cosmos DB [5].
- **Data Delivery Time** specifies the acceptable time from when the service receives customer data to when the customer receives data back from the service. This SLO is used only in the Google BigQuery Data Transfer Service [4], a data analytics service.
- **Time to Repair** specifies the acceptable time required to restore the functionality of a failed cloud server or cluster beginning from

**Table 2: Cloud SLA contents and their SLOs**

| Content | Sub-content | SLOs |
|---|---|---|
| Availability | n/a | • Availability, Interval-availability, Maximum Processing Time |
| Performance | Response time | • Maximum Response Time, Response Time Mean, Response Time Variance |
| | Capacity | • Maximum number of simultaneous connections, Disk space, CPU power, Memory size, Page view, Throughput, Bandwidth |
| | Elasticity | • Elasticity speed threshold, Elasticity Precision threshold |
| | Migration | • Migration Time |
| | Consistency | • Monthly Consistency Attainment Percentage |
| | Monitoring | • Monitoring Alert Notification Time |
| Reliability | Service resilience / fault tolerance | • Time to Service Recovery, Mean Time to Service Recovery, Maximum Time to Service Recovery, Number of Service Failures |
| | Customer data backup and restore | • Backup Interval, Retention Period for Backup Data, Number of Backup Generations, Backup Restoration Testing |
| | Disaster recovery | • Recovery Time Objective, Recovery Point Objective |
| | Migration | • Migration Notification Time |
| Cost | n/a | • Cost per Time Unit, Revenue per Request, Cost per kW, Penalty rate |
| Termination of service | n/a | • Data Retention Period, Log Retention Period |
| Support | n/a | • Support Hours, Service Incident Support Hours, Service Incident Notification Time, Maximum First Support Response Time, Maximum Incident Resolution Time |
| Change management | n/a | • Minimum Service Change Notification Period, Minimum Time Before Feature/Function Deprecation |
| Data management | Data Deletion | • Data Deletion Time |

**Table 3: SLOs currently adopted by cloud provider SLAs.**

| SLO | SLA content | Microsoft Azure | Amazon AWS | Google Cloud | Rackspace | Digital-Ocean | Joyent |
|---|---|---|---|---|---|---|---|
| **Monthly Uptime Percentage** | Availability | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Monthly Availability Percentage** | Availability | ✓ | ✓ | | | | |
| **Maximum Processing Time** | Availability | ✓ | | | | | |
| **Monthly Recovery Time Objective** | Reliability | ✓ | | | | | |
| **Consistency Attainment Percentage** | Performance | ✓ | | | | | |
| **Latency Attainment Percentage** | Performance | ✓ | | | | | |
| **Monthly Throughput Percentage** | Performance | ✓ | | | | | |
| **Data Delivery Time** | Performance | | | ✓ | | | |
| **Time to Repair** | Reliability | ✓ | | | | | |
| **Monitoring Alert Notification Time** | Reliability | | | | ✓ | | |
| **Migration Notification Time** | Reliability | | | | | | |
| **Migration Time** | Performance | | | | ✓ | | |
| **Disk space** | Capacity | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **CPU power** | Capacity | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Memory size** | Capacity | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Bandwidth** | Capacity | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

the failure detection time. This SLO is adopted only by Rackspace Cloud Server Hosts [3].

- **Monitoring Alert Notification Time** is the acceptable time to generate an alert to notify the customer of a violation of a predefined error condition exhibited by the cloud service. This SLO is adopted only by the Rackspace Cloud [3] for monitoring their database services.
- **Migration Notification Time** describes the acceptable time to notify customers if service migration is required. It is adopted only by Rackspace [3] when host server degradation is detected.
- **Migration Time** specifies the acceptable time duration to complete the migration of a cloud service. This SLO is adopted only by Rackspace [3].

Table 3 shows the SLOs adopted by each of the surveyed cloud providers. Note that the mark (✓) indicates that the corresponding SLO is provided in *at least one* of the provider's SLA, not necessarily in all.

In brief, the providers' perspective is to provide the SLOs they can guarantee for their 'unary' services. Providers do not commit to many SLOs that are required for cloud applications.

## 4 ANALYSIS OF THE PROBLEM SPACE

Our investigation of SLO specification in the cloud research literature, cloud platform SLAs, and CML capabilities has revealed several interesting observations, which we present and discuss in this section.

### 4.1 Lack of adopting SLOs

Clearly, there is a gap between the SLOs proposed by the cloud community and those provided in commercial SLAs. Among 39 SLOs proposed by the cloud community and listed in Table 2, only 16 appear in current SLAs as shown in Table 3. This imbalance is something that is well known in the cloud trade, and is partly behind the rise of DevOps: the realisation that software development is not sufficient and that there is a need to integrate it closely with operation management. What is remarkable, however, is the width of this chasm in a market that has been established for over a decade now.

Moreover, SLO support is surprisingly low for some of the major cloud providers. Amazon and Google are market dominators, but offer no more than basic uptime and availability guarantees. This is slightly better for other providers, such as Microsoft and Rackspace, but still not nearly enough given the needs as identified by the literature (§3.1). Furthermore, these latter providers only provide additional SLOs for *some* of their services. As a result, customer applications that require strong guarantees (e.g. reliability) will fail to find such guarantee for all cloud services.

### 4.2 Heterogeneity of SLO definitions in the SLAs

A key problem facing customer is the heterogeneity of SLO definition and calculation. In some cases, SLAs from different providers contain SLOs that have similar names but are calculated differently. This makes it difficult for customers to select services especially with the lack of interoperability between the provider SLAs and between the CMLs. As an example, both Microsoft Azure Cosmos DB and Amazon DynamoDB express the availability SLO as the Monthly Availability Percentage where availability is calculated in terms of the Error Rate, i.e. the percentage of failed requests in a time interval. However, Microsoft Azure Cosmos DB calculated the error rate in one-hour intervals whereas Amazon DynamoDB uses five minutes.

### 4.3 Lack of SLOs support in current CMLs

The lack of support for SLOs in current CMLs is obvious. Among the ten CMLs we studied, only four provide some kind of SLO specification support (Table 1). With respect to the CMLs that support SLOs, the following shortcomings are identified:

- **Complexity**: Specifying SLOs is predominantly done through XML schema. XML is verbose, with an angle-bracketed syntax that is complex and not easily human comprehensible. This complicates development and hinders the maintainability of cloud applications [8, 10].
- **Design perspective**: TOSCA and Blueprint do not provide concepts for modelling SLOs. Instead, they provide concepts for attaching policy files such as the WS-Policy and SLAng. The WS-Policy specification allows web services to publish their service levels, while SLAng allows providers to define their SLAs. In a nutshell, those specifications were designed taking the provider's perspective instead of the customer's. For example, Blueprint assumes the presence of a marketplace where providers publish their service descriptions and attach WS-Policy files.
- **Multi-cloud deployment**: The fact that some CMLs support the attachment of policies for specifying non-functional requirements using WS-Policy limits their capabilities of supporting SLO specification for multi-cloud applications. The reason is that the WS-Policy specification is designed for specifying non-functional properties of unary web services. This also applies to SLAng, which is designed for providers to specify policies for unary services.
- **Realisation**: An engine is required to process the SLOs after being specified in order to satisfy them by selecting the appropriate services. The engines developed to realise the above approaches are limited to matching the service offerings with the customer requirements. However, recalling that providers do not support many of the SLOs, such search engines will not find any service if the customer specified SLOs that no provider provide. This limitation can be addressed by an intermediary as discussed in Section 5.

## 5 RESEARCH DIRECTIONS & CHALLENGES

In this section we describe steps towards our vision of SLO-driven service selection, and comment on the main obstacles.

### 5.1 SLO domain specific language

A language is required to enable customers to model their application SLOs. It should provide a comprehensive syntax for capturing service level requirements, supporting all SLOs currently used by providers and those specified in industry standards. It should provide customers with a high level of abstraction, whereby they can specify SLOs for required cloud services regardless of the low level details of those services. Moreover, the language should supports applications in both single- and multi-cloud environments.

The challenge here is mostly technical in the form having a language design that is based on the following principles:

(1) **Customer-oriented.** The language should enable customers to specify their high-level operational requirements in a simple declarative syntax. The provided modelling concepts should be comprehensive for cloud application SLOs and not restricted to the ones supported in the cloud SLAs.
(2) **Independence**. To avoid vendor lock-in, SLO specification should be independent of cloud service specification. Furthermore, it needs to be independent of cloud application development technology and implementation details.
(3) **Abstraction**. Customers should be able to specify SLOs regardless of the required type of cloud service, e.g. SaaS, PaaS, etc.
(4) **Separation of concerns**. It should be possible to maintain and adapt isolated SLO specification at an application component level. For example, a load-balancing component's SLOs should be separate from those of a data storage element.
(5) **Mapping SLOs** High-level SLOs specified by users should be broken down to low-level ones, and then further mapped to the application component level. For example, the response time of a three-tier application consists of processing time for each layer.

### 5.2 Brokerage system

There is a need for a brokerage system to lie between the cloud customer and providers in order to understand the customer SLOs, select a matching infrastructure, and continually maintain application deployment. In order to realise such system, a number of requirements need to be considered:

(1) **Intelligent selection.** The problem of cloud service selection is clearly application-specific. Internal characteristics of the application, e.g. architecture and implementation, and external ones, e.g. workload, affect performance. Therefore, the selection of cloud services should go beyond the simple matching between application SLOs and service SLOs. This requires novel benchmarking techniques to identify and extract information that is relevant to application performance. As such approach could

get expensive, here is where novel machine learning techniques could help in making this approach practical.

(2) **Post-deployment optimisation.** Provider services exhibit performance characteristics that are not fully defined, but could be important to any real-world optimisation decision. In addition, application characteristics may change at run-time. These challenges demand an adaptation strategy to continually optimise the selection. This includes actively monitoring cloud service performance and costs to dynamically change deployment to satisfy a multi-objective optimisation scenario. Such a solution is likely to consist of both dynamic search strategies, and an accumulation of knowledge about cloud workloads via predictive modelling. Collectively, this can be used to intelligently manage the lifecycle of customer applications and fully satisfy modelled SLOs.

Realising such brokerage as a third party is not trivial. First, it raises **legislative** issues due to the position it stands as intermediary between customers and providers. Those issues include SLA violation penalties, intellectual property and contractual issues, privacy and data protection rights, and green computing regulation, among others. These issues need to be carefully assessed and, consequently, a framework is required to manage customer-broker and broker-provider contracts in order to mitigate any attached risks.

The second major challenge is **economic feasibility**. From the customer perspective, the value that a broker adds is the ability to make cost-effective service selection decisions. This added value is not completely appreciated yet, thus economic models to compare broker-assisted and broker-less decision making are needed to evaluate the cost-effectiveness of using the broker. Meanwhile, from the broker's perspective, economic models are needed to compare the economical consequences of the alternative decisions and alternative decision making techniques to maximise the profit of the broker and thus achieve **economic sustainability**.

### 5.3 Knowledge management

The matching of cloud services to customer needs requires continual processes to acquire, represent, utilise, and maintain knowledge about the different parts of the system. Knowledge required to inform service selection can be broken down into three levels:

(1) **Provider SLOs**, i.e. the 'book values' obtained from SLAs.
(2) **Service behavior** observed by gathering low-level metrics.
(3) **Real application performance** as perceived by users, hence map to high-level application requirements.

A main challenge here relates to the **dynamism** of the cloud market: although provider landscape is relatively stable, service offerings change from time to time. More importantly, the performance levels of many cloud services is in a constant state of flux particularly due to the number of hidden features at play [24]. This dynamism makes attaining knowledge a 'moving target' that evolves over time. Continuous evaluation and adaptation of the learned models is necessary to verify that they represent a true state of the system. Another related challenge is the **trust** of both accumulated knowledge and the decisions made using it.

## ACKNOWLEDGMENTS

This work is supported by the Adaptive Brokerage for the Cloud (ABC) project, UK EPSRC grant EP/R010889/1 and EP/R010528/1.

## REFERENCES

[1] 2016. *Information technology – Cloud computing – Service level agreement framework.* Standard ISO/IEC 19086-1:2016(E). International Organization for Standardization.
[2] 2018. Amazon DynamoDB Service Level Agreement. https://aws.amazon.com/dynamodb/sla/. Accessed: 2018-10-30.
[3] 2018. Cloud Service Level Agreement. https://www.rackspace.com/information/legal/cloud/sla. Accessed: 2018-10-30.
[4] 2018. Google Prediction API and Google BigQuery SLA. https://cloud.google.com/bigquery/sla. Accessed: 2018-10-30.
[5] 2018. SLA for Azure. https://azure.microsoft.com/en-us/support/legal/sla/.
[6] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. 2010. Conceptual SLA framework for cloud computing. In *DEST*. 606–610. https://doi.org/10.1109/DEST.2010.5610586
[7] Vasilios Andrikopoulos, Anja Reuter, Santiago Gómez Sáez, and Frank Leymann. 2014. A GENTL Approach for Cloud Application Topologies. In *ESOCC*. 148–159.
[8] Greg J Badros. 2000. JavaML: a markup language for Java source code. *Computer Networks* 33, 1 (2000), 159 – 177.
[9] Salman A. Baset. 2012. Cloud SLAs: Present and Future. *SIGOPS Oper. Syst. Rev.* 46, 2 (Jul 2012), 57–66. https://doi.org/10.1145/2331576.2331586
[10] Alexander Bergmayr, Uwe Breitenbücher, Nicolas Ferry, Alessandro Rossini, Arnor Solberg, Manuel Wimmer, Gerti Kappel, and Frank Leymann. 2018. A Systematic Review of Cloud Modeling Languages. *ACM Comput. Surv.* 51, 1, Article 22 (Feb 2018), 38 pages. https://doi.org/10.1145/3150227
[11] Alexander Bergmayr, Uwe Breitenbücher, Oliver Kopp, Manuel Wimmer, Gerti Kappel, and Frank Leymann. 2016. From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA. In *CLOSER*. https://doi.org/10.5220/0005806900970108
[12] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. 2014. TOSCA: Portable Automated Deployment and Management of Cloud Applications. In *Advanced Web Services*. 527–549. https://doi.org/10.1007/978-1-4614-7535-4_22
[13] Clovis Chapman, Wolfgang Emmerich, Fermín Galán Márquez, Stuart Clayman, and Alex Galis. 2012. Software architecture definition for on-demand cloud provisioning. *Cluster Computing* 15, 2 (01 Jun 2012), 79–100. https://doi.org/10.1007/s10586-011-0152-0
[14] Cloud Standards Coordination (CSC). 2016. *CSC Phase 2: Cloud Computing Users Needs – Analysis, conclusions and recommendations from a public survey.* Special Report 003 381 V2.1.1. ETSI. 12–19 pages. http://csc.etsi.org/phase2/UserNeeds.html
[15] Frederic J de Vaulx, Eric D Simmon, and Robert B Bohn. 2018. *Cloud computing service metrics description.* Technical Report 500-307. NIST.
[16] Funmilade Faniyi and Rami Bahsoon. 2015. A Systematic Review of Service Level Management in the Cloud. *ACM Comput. Surv.* 48, 3, Article 43 (Dec 2015), 27 pages. https://doi.org/10.1145/2843890
[17] Nicolas Ferry, Franck Chauvel, Hui Song, Alessandro Rossini, Maksym Lushpenko, and Arnor Solberg. 2018. CloudMF: Model-Driven Management of Multi-Cloud Applications. *ACM Trans. Internet Technol.* 18, 2, Article 16 (Jan 2018), 24 pages. https://doi.org/10.1145/3125621
[18] Sören Frey and Wilhelm Hasselbring. 2011. The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. *Int'l Journal on Advances in Software* 4, 3–4 (2011), 342–353.
[19] Joaquín Guillén, Javier Miranda, Juan Manuel Murillo, and Carlos Canal. 2013. A UML Profile for Modeling Multicloud Applications. In *ESOCC*. 180–187.
[20] Mohammad Hamdaqa and Ladan Tahvildari. 2015. Stratus ML: A Layered Cloud Modeling Framework. In *IC2E*. 96–105. https://doi.org/10.1109/IC2E.2015.42
[21] Cinar Kilcioglu, Justin M. Rao, Aadharsh Kannan, and R. Preston McAfee. 2017. Usage Patterns and the Economics of the Public Cloud. In *WWW*. 83–91. https://doi.org/10.1145/3038912.3052707
[22] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. 2003. SLAng: a language for defining service level agreements. In *Workshop on Future Trends of Distributed Computing Systems*. https://doi.org/10.1109/FTDCS.2003.1204317
[23] Jason Lango. 2014. Toward Software-defined SLAs. *Commun. ACM* 57, 1 (2014), 54–60. https://doi.org/10.1145/2541883.2541894
[24] Philipp Leitner and Jürgen Cito. 2016. Patterns in the Chaos — A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Transactions on Internet Technology* 16, 3, Article 15 (Apr 2016), 15:1–15:23 pages. https://doi.org/10.1145/2885497
[25] Frank Leymann, Christoph Fehling, Ralph Mietzner, Alexander Nowak, and Sscahram Dustdar. 2011. Moving Applications to the Cloud: An Approach based on Application Model Enrichment. *Int'l Journal of Cooperative Information Systems* 20, 3 (2011), 307–356. https://doi.org/10.1142/S0218843011002250 arXiv:https://doi.org/10.1142/S0218843011002250
[26] Dinh Khoa Nguyen, Francesco Lelli, Yehia Taher, Michael Parkin, Mike P. Papazoglou, and Willem-Jan van den Heuvel. 2011. Blueprint Template Support for Engineering Cloud-Based Services. In *Towards a Service-Based Internet*. 26–37.
[27] F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair. 2016. Daleel: Simplifying cloud instance selection using machine learning. In *NOMS*. 557–563. https://doi.org/10.1109/NOMS.2016.7502858
[28] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Umit Yalçinalp. 2007. *Web Services Policy 1.5-Framework.* Technical Report. W3C.