An Approach to Unified Cloud Service Access, Manipulation and Dynamic Orchestration via Semantic Cloud Service Operation Specification Framework

Daren Fang[1*], Xiaodong Liu[1], Imed Romdhani[1], Claus Pahl[2]

*Correspondence:
d.fang@napier.ac.uk
[1]School of Computing, Edinburgh Napier University, 10 Colinton Road, Edinburgh, EH10 5DT, UK
Full list of author information is available at the end of the article

Abstract

Cloud computing offers various computational resources via convenient on-demand service provision. Currently, heterogeneous services and cloud resources are usually utilized and managed through diverse service portals. This significantly limits the effectiveness and efficiency for tasks implementation. Fundamentally, it is due to the lack of adequate specifications for service concepts, operations and interfaces from diverse cloud service models and types. This paper proposes a service management operation semantic description framework for comprehensive cloud service operation specification. Relying on ontological modelling techniques, cloud service operations are specified via entity classification, attribute assertion, relationship assertion and annotation assertion. Further, the proposed framework benefits from operation reasoning application. It enables intelligent assistance for multiple operation preparation and remote execution tasks. Based on the approach, a cloud service operation ontology and a unified service access and manipulation system prototype are implemented. Extensive experiments are conducted over different cloud service providers and for distinct service models. Obtained results demonstrate that the approach outperforms existing practices by facilitating reliable and effective service access, manipulation and interaction tasks.

Keyword: Cloud Computing, service orchestration, API, semantic modelling, OWL

Introduction

In the era of cloud computing, Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) providers offer on-demand services and resources for various compute/platform/software needs [1][2]. While many cloud service providers (CSPs) provide unique management portals for their own services and resources, the interfaces, functionalities and service operation environments are mostly diverse. Indeed, this is due to the fact that different CSPs tend to act distinctly while addressing service quality of service (QoS), features, customizability, requirements, etc. [3][4]. As a result, while trying to manage multiple cloud services and resources, users often have to use a variety of cloud portals for different CSPs. This significantly limits the effectiveness and efficiency for tasks deployment and implementation.

To deal with the above issues, a number of approaches to cloud service and resource interoperability and portability are proposed. These solutions include but are not limited to: open cloud API (Application Programming Interface) development such as jclouds [5], libcloud [6], fog [7]; service specifications such as TOSCA [8], mOSAIC [9]; generic cloud management protocols/drivers such as OCCI [10]. Despite their capabilities of handling certain specific service and resource categories, it is difficult to find any that allows adequate management for diverse CSPs' services and resources via a common interface. This is mainly due to the lack of a unified service specification framework that can interpret cloud service and resource entities and deal with the interoperability among CSPs [11].

This paper proposes a new service operation semantic specification approach, called Service Access and Manipulation Operation Specification (SAMOS) framework. The framework is underpinned by ontological modelling techniques. It is capable of modelling comprehensive specifications for cloud service operations regardless of the service/operation/provider types. The variety of service operations are specified into two categories: service information requests and service manipulation requests. For each category, the proposed framework describes the detailed operation elements involved, including the parameters, requirement, outcome, condition changes, etc.

Based on SAMOS, a cloud service operation ontology and a unified service remote management prototype tool are dev eloped. In this paper, "unified" refers to the term that an integrated and versatile (specification and management) solution is provided for heterogeneous services and resources in multiple cloud environments. To such extent, the proposed model can interpret and preserve the complexity which lies behind cloud service operation executions, in a formal systematic way. By utilizing a cloud service API mapping mechanism, the tool is com-

patible with real IaaS, PaaS and SaaS services from multiple provider clouds. Accordingly, the ontology and the tool enable users to effectively view, create and manipulate a wide range of cloud service and resource data via a unified structured interface. Additionally, featured with ontology reasoning techniques, the proposed approach can provide a series of operation assistances. This means that appropriate service operations can be dynamically prepared and/or composed into groups and then executed intelligently according to the real-time status of the target cloud services and resources, even if they are originated from multiple clouds. Consequently, the proposed unified cloud service operation specification and reasoning approach is capable of providing effective semantic support to deal with cloud (service) interoperability issues.

The rest of the paper is organized as follows: Section "Background and related work" discusses the background and related work regarding open cloud API, service semantic specification and generic cloud service management tool. Section "SAMOS service operation modelling framework" describes the core and detailed elements of SAMOS framework. Section "Service operation requirement verification and dynamic assistance reasoning" outlines cloud service operation requirement verification and reasoning assistance mechanisms. Section "Prototype implementation" provides the design of the prototype system and components, including the API (and specification) mapping implementation. Section "Case studies" demonstrates case studies and experiments on real cloud services. Section "Evaluation and discussion" evaluates the proposed approach based on findings and the obtained results. Finally, Section "Conclusions and future work" concludes the paper with summaries and future work.

Background and related work

Considerable efforts have been made on enhancing the interoperability and portability of cloud services. The practices are widely applied in open cloud API developments, comprehensive service interface specifications, versatile cloud management protocols or drivers, etc.

Open cloud service specification framework

The Open Cloud Computing Interface (OCCI [10]) is one of the earliest attempts to eliminate cloud resource heterogeneity. Originally, it was developed only to deal with IaaS remote management tasks such as resource deployment, monitoring and automated scheduling. Later, the evolved Rendering and Extension specification frameworks enable wider application for PaaS and SaaS services, which consequently make it a generic management driver for a number of cloud resources.

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA [8]) is a recently established standard for clouds. With the aim to enhance cloud service portability, it enables specifications for diverse cloud service resources, their relationships and operational behaviors. With several templates (e.g. service and policy templates) and types (e.g. node, relationship, requirement and capability types) specifications, the topology framework can provide semantic support for many cloud service management and orchestration tasks.

Other than the above well-established practices, a series of research projects are also implemented towards the aim. mOSAIC [4], for instance, advocates application/provider/language-independent cloud service semantic specifications. The mOSAIC ontology model enables separation of application-logic and cloud layers and consequently enhances cloud portability. Likewise, the RASIC framework [12] attempts to enhance service interoperability through modelling three horizontal layers (i.e. service frontend, SOA, virtualization/execution) and two vertical layers (i.e. semantic and governance). Similarly, the Intercloud [13] architecture comprises multilayer cloud service models and a series of management, federation and operations frameworks. They serve as cloud middleware to support the service integration. However, these approaches are developed mainly for infrastructure services and resources, and cannot be effectively applied to PaaS and SaaS models.

Open cloud service API

Deltacloud [14] provides a REST [15]-based cloud abstraction API that enables service management functions for a number of IaaS resources. The wide range of CSP support makes it feasible to manage heterogeneous resources across diverse clouds. Fundamentally, it runs a series of cloud drivers, which serve as individual service adapters for each CSP specifically. Deltacloud API along with the management interface enables long-term stability for cloud resource utilization.

On the other hand, a number of (stand-alone) language-dependent cloud APIs are also found. Libcloud [6], for instance, is a Python library that offers wide support for many popular CSPs. The library provides interfacing functions mainly for compute, storage, load balancer, etc. services. Fog [7] is an API library for Ruby developers. It also has flexible support for several services from mainstream CSPs. Jclouds [5] and Dasein [16], are examples of java API library that supports a wide range of CSPs. Likewise, they can be applied to manage various CSPs' IaaS compute, platform, database, storage, etc. services.

Some open cloud service API research is also found in the field. Bastião Silva et al. [17] propose a common API and SDCP (Service Delivery Cloud Platform) for delivering services over multi-vendor cloud resources. The platform is capable of describing diverse cloud concepts (e.g. agent, domain, and provider) and managing service data and abstraction conventions. Similarly, Petcu et al. [18] propose the mOSAIC java API as an example of open interface for service deployment and portability. Nonetheless, a drawback is that it cannot effectively handle the unique features offered among distinct cloud services (i.e. lack of support for many provider-specific service features).

Service and resource management tools for heterogeneous clouds

Bernabe et al. [19] demonstrate an access control system for multi-vendor cloud resource management. The proposed ontology specifications can describe various entities (e.g. cloud, system, software, etc.), whereas the authorization model can deal with user authentication and authorization tasks. Despite its advanced hierarchical role-based access control, the application is currently limited to IaaS resources and a single CSP (AWS EC2 [20]).

Cloud Data Imager (CDI) [21] is seen as a complete system to provide comprehensive functionalities for accessing and managing storage resources across diverse clouds (i.e. Dropbox, Google Drive, and Microsoft SkyDrive). The proposed CDI library is able to handle a variety of functions including user authentication, folder listing, file downloading, etc. Another work addressing resource utilization monitoring issues over heterogeneous multi-tenant clouds is found in DARGOS [22]. The proposed architecture can provide highly reliable monitoring functions. These approaches would only work for their own limited cloud service models/types.

A model-based cloud service integration platform [23] is advocated to drive service orchestration for business application integration. The proposed framework addresses three levels of modelling: cross-organizational business processes, service operation/orchestration, and dynamic member services binding. Nonetheless, the approach focuses mainly on enhancing cloud service and resource integration for certain specific business processes. The platform is not an ideal tool for a diversity of cloud service and resource management tasks.

In summary, there are well-established cloud specification standards and considerable native/third-party open cloud service API libraries for application over various cloud resources and many CSPs. Meanwhile, many approaches are proposed as a means towards generic cloud service management. They bring some alternatives for avoiding vendor lock-in plus flexible management of services and resources. Nevertheless, due to the gaps among existing studies, currently no solution is available for a unified and effective management of diverse cloud service models/types regardless of CSPs. Consequently, this significantly limits the effectiveness and efficiency for cloud service management and composition tasks.

SAMOS cloud service operation modelling framework

SAMOS models cloud service operation via three components: cloud service entity and operation classification, cloud service entity datatype specification, and cloud service entity operational relationship specification.

Cloud service entity and operation classification specifications

In fact, certain membership or association relationships can often be found among the various service and operation entities for every individual CSP. These facts can be well modelled with ontology classification techniques (see Figure 1).

Specifically, cloud services and CSPs are asserted as ontology classes. According to their membership functions (e.g. a service belongs to a certain CSP), the service class is seen as a subclass for the CSP class. Then, cloud service instances (CSIs) created within a cloud service are specified as individuals of the service class.

The classification also applies to additional cloud entities, such as service operation parameter entities, service configuration entities, and service accountability and user authorization data (e.g. service regions, instance attributes, user account data, etc.). Here, considering that almost all of such entities tend to be CSP-specific (i.e. the data formats, names, descriptions, etc. of the entities would all be unique from one CSP to another), they are specified as "provider-specific service aspect" (PSSA). Nonetheless, some PSSAs, i.e. common service/software aspects, may fundamentally indicate the same entity, despite their distinct PSSA names (e.g. public IP addresses, open VM images and SQL database data entries). These associated entities are declared with equivalence (via "equivalent class" or "same individual" axioms). Figure 1

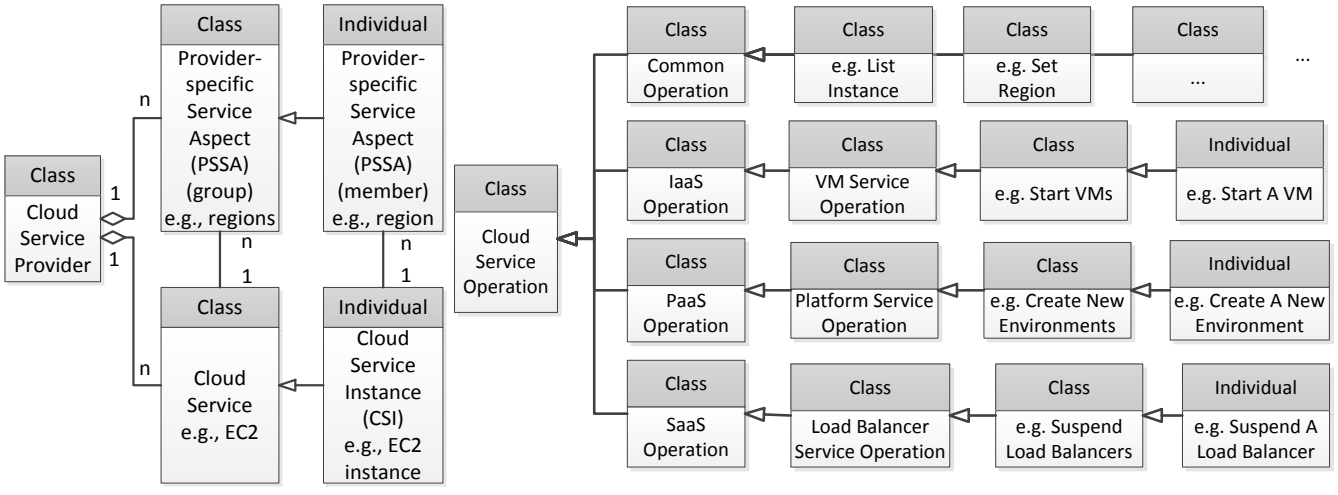

Figure 1 Cloud service entity and operation classification specifications

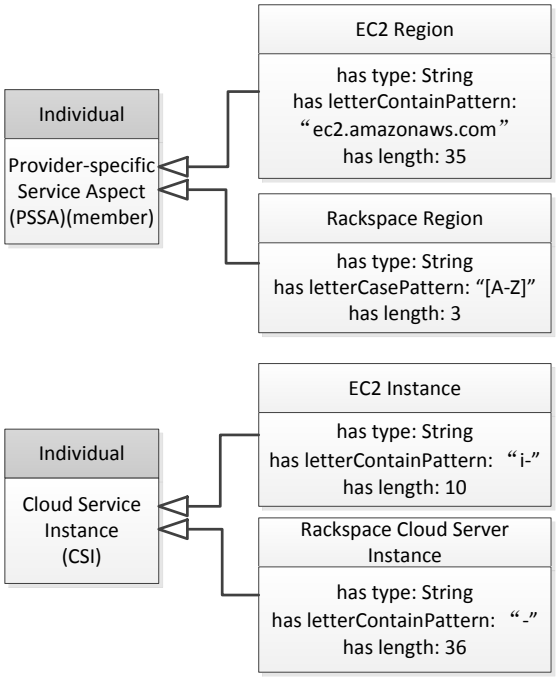

Figure 2 Cloud service entity datatype specifications

summarizes the association relationships among cloud services, CSIs and PSSAs.

Further, the diversity of cloud service operations are asserted as ontology classes and individuals according to their associations. As illustrated in Figure 1, cloud service operation class comprises four subclasses: IaaS, PaaS, SaaS and Common Operations. They each own a set of relevant operations. Obviously, for IaaS, PaaS and SaaS, the majority of the categorized operations would be different from each other. Then, the common cloud service operations, which are available widely for all service models (e.g. "list instance", "set region"), are gathered in the Common Operation class.

Cloud service entity datatype specifications

As cloud concepts and entities are established in appropriate class hierarchy, their datatype specifications are to be described in detail. Generally speaking, cloud services appear the same (entity) to all users, and hence own no typical data-relevant properties. In contrast, CSIs, which are created and owned by certain users, are unique to the owners; they own specific data formats, i.e. IDs, creation times, names, etc. These details are attached to the entities via ontology datatype property assertions. With such, a CSI can be easily recognized with its unique ID, whereas other relevant datatype information can be effectively addressed when required. Similarly, datatype specification also applies to all PSSAs, e.g. strings, integers, dates, URLs, or some unique provider-specific service data formats (see Figure 2).

Indeed, these data properties enable precise datatype format demonstration, differentiation and validation for cloud entities and operations. With the extracted data presentation patterns and respected pattern examination mechanism, validations can be effectively implemented for cloud service operation preparation and execution (e.g. validations of authorization, input, output, condition, etc.).

Table 1 Classification of cloud service operations

Operation Type	Description	Examples
Service Information Request (SIR)	Operation requested to retrieve service entities and entity information	List owned service instances, get instance ID, get available platforms
Service Manipulation Request (SMR)	Operation requested to make changes to cloud services, CSIs or PSSAs	Create new instance, terminate instance, modify instance name

Cloud service entity operational relationship specifications

Cloud service operations can be seen as reflections of the operational relationships among relevant cloud service and operation entities. For instance, I) "Create instance" and "List instance" can describe the creation and inclusion relationships from a cloud service to its instance(s). II) "Get instance ID" and "Modify instance name" can clarify the retrievable and modifiable relationships from a service instance to its property and condition. This is how SAMOS tackles cloud service operation specification by modelling the diversity of service entity operational relationships.

Classification of cloud service operations

Shown in Table 1, based on the different nature and intentions of cloud service operations, we first divide them into two categories: service information request (SIR) and service manipulation request (SMR).

SIR

At the cloud service level, SIR often relies on collecting all available cloud service's settings and instances (e.g. get available regions and list instances). At the CSI level, it is usually to retrieve the diverse real-time information of a certain service instance (e.g. get instance status). At the PSSA level, it tends to acquire the real-time information of the specific CSP entities (e.g. get VM image ID). Generally, SIR operations would not alter any cloud service, CSI or PSSA after execution.

SMR

At the cloud service level, SMR is usually to handle the general service settings and overall instances management tasks (e.g. set region and delete all instances). At the CSI level, it is mainly regarding some specific instance control and modification functions (e.g. reboot VM instance). At the PSSA level, it is for the manipulations implemented on those unique CSP entities (e.g. delete VM image). On successful execution, SMR should alter the target cloud service/CSI/PSSA in an intended way.

Cloud service operation object property specifications

Based on the proposed operation classifications, the diverse cloud service operations can then be described, shown in the form of various operational relationships among relevant cloud services, CSIs and PSSAs. These relationships can be adequately described using ontology object property assertions. Figure 3 illustrates the representation of cloud service operations using object proper-

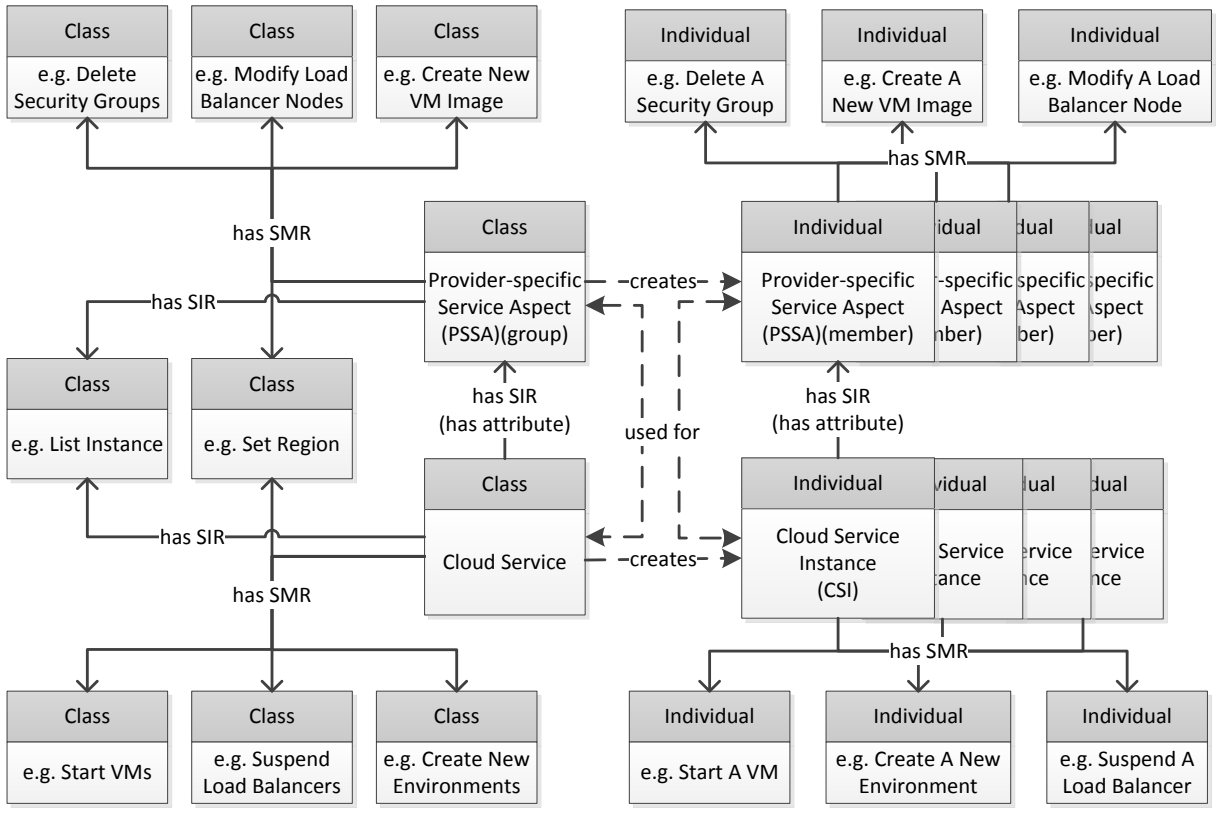

Figure 3 Cloud service entity object property specifications

ty specifications. Basically, "hasSIR" and "hasSMR" are asserted to describe the types of operations available, between cloud service/CSI/PSSA and relevant operation concepts. For instance, the "create" and "list" operations between cloud services and CSIs can be represented with "hasSMR create instance" and "hasSIR list instance", respectively; the "get attribute" and "modify" operations between CSIs and PSSAs can be represented with "hasSIR get attribute" and "hasSMR modify…", to demonstrate their operational relationships.

Additionally, the details of cloud service operations, i.e. operation conditions, parameters, outcomes and changes are comprehensively described in SAMOS. Here, we utilize a series of systematic operation specification elements to specify the aspects that may be involved (before, during and after operations). These details are stored in the form of ontology annotations for the respective object property relation assertions.

Table 2 SRParameter symbol notations

SRParameter Attribute	Denotation (in object property annotation)	Examples Service Operations and SRParameters
Mandatory	"[]"	
Optional	""	
Single	"()"	
Multiple	"<>"	
Mandatory single	"[()]"	Rename: [("new_name")]
Mandatory multiple	"[<>]"	Reboot: [<"vm1,2…">]
Optional single	"()"	Set authorization: ("basic")
Optional multiple	"<>"	Deny access: <"user1,2...">

Specification of cloud service information/manipulation request elements

Request Subject (SRSubject)

SRSubject is recognized as the target of a cloud service operation. As a user selects a cloud service for operation, the service becomes the target. Similarly, SRSubject can apply to all CSIs and PSSAs if selected.

Request Parameters (SRParameter)

Although some cloud service operations can execute with only SRSubjects, many others do need certain parameter inputs, e.g. relevant restrictions, options, customized data, etc. These are required by CSPs to enable accurate and successful service operations. SRParameters specify such details for applicable service operations. Generally, PSSAs make up the majority of SRParameters; CSIs can also be involved as SRParameters; it is unlikely for any cloud service to be used as SRParameter.

Due to the complexity of cloud service operation parameter requirement, a SRParameter attribute notation system is developed. The denotations and examples of the SRParameter attributes are shown in Table 2. Basically, according to the operation requirements, each parameter is specified with mandatory/optional differentiations. Then, depending on whether an operation accepts single/multiple parameters of the same entity type, the parameters are also differentiated. Consequently, this ought to enable a precise specification and interpretation for diverse cloud service operation (parameter) requirements.

Request Outcome (SROutcome)

As a cloud service operation is executed, certain data response would be returned from its CSP, informing the

execution result, e.g. execution status, obtained service information, newly altered service entities, etc. This operation element is represented as SROutcome. Typically, for the majority of operation requests, SROutcome reveals the expected service entities to be returned from the respected CSP. For some simple SMR operations, SROutcome would only be the (expected) success response.

Request Pre Condition (SRPreCondition)

According to CSP restrictions, there may be various operation request condition requirements. Some operations can be initiated without any condition constraints whilst others do need specific condition fulfilments. This is defined as SRPreConditions of cloud service operations. For SRPreCondition specification, the condition text can be either positive ("VM == off") or negative (e.g. "service != updating"), where specific data formats and symbols can be involved (e.g. "instance count restriction <= 100"). SRPreCondition applies only to those operations which genuinely require so; others would have an empty entry ("unconditional") for the element.

Request Post Condition (SRPostCondition)

Using the same specification pattern as SRPreCondition, SRPostConditions describes the (expected) post execution condition of a cloud service operation. Specifically, the condition is accounted whenever a "new" condition is created. This is to drive as many as operation compositions (by paring SRPreCondition and SRPostCondition) towards possible cloud service orchestrations. Hence, a SRPostCondition does not necessarily need to contradict the respected SRPreCondition: some operations may only have SRPostCondition and no SRPreCondition; for some operations, the entities involved in SRPostCondition may be different from those in the respected SRPreCondition. For instance, a VM creation operation can have SRPreCondition of "account restriction == OK" and SRPostCondition of "VM == running".

```
INPUT: Operation op, SRParameterType srpt1, srpt2, ..., srptn;
SRParameterData srpd1, srpd2, ..., srpdn,

 1 INIT SRParameterRequirement srpr1, srpr2, …, srprn to
   CALL getMandatoryParameter with op;
   ParameterSatisfied to FALSE;
 2 IF ParameterCount = 0 THEN
 3    SET ParameterSatisfied to TRUE
 4 END IF
 5 ELSE THEN
 6    INIT matchCount to 0;
 7    FOR each srprn in SRParaterRequirement
 8       IF srprn = srptn THEN /*parameter type matches*/
 9          INCREMENT matchCount
10       END IF
11    END FOR
12    IF matchCount >= ParameterCount THEN
      /*all mandatory parameters satisfied*/
13       SET ParameterSatisfied to TRUE
14       CALL fillParameter with op, srpd1 to srpdn
         /*pass parameter data*/
15    END IF
16 END ELSE

OUTPUT: ParameterSatisfied
```

Figure 4 Cloud service operation parameter verification algorithm

Service operation requirement verification and dynamic assistance reasoning

While cloud service entities, their attributes and relationships, and operation elements are comprehensively specified, relevant service operation assistance reasoning can be introduced. In fact, this enables effective cloud service operation execution management, even for cloud service orchestration tasks.

Basic service operation preparation and execution

A typical use of the cloud service operation specifications is verification. Given SRSubject, relevant SRParameters and fulfilled SRPreCondition, an operation can be remotely executed through appropriate programming interface request. If successfully executed, this would result in certain service entity (data) which matches the respected SROutcome and/or SRPostCondition.

Service Operation Parameter Verification

```
INPUT: Operation op

 1 INIT SRPreconditionRequirement srprec1, srprec2, …,
   Srprecn to CALL getPrecondition with op;
   ConditionSatisfied to FALSE;
 2 IF ConditionCount = 0 THEN
 3    SET ConditionSatisfied to TRUE
 4 END IF
 5 ELSE THEN
 6    INIT SatisfyCount to 0;
 7    FOR each srprecn in SRPreconditionRequirement
 8       INIT condition to CALL
         getCurrentServiceEntityCondition with op, srprecn
 9       IF srprecn HAS "==" THEN
10          IF srprecn = condition THEN /* SRPreCondition
            fulfils certain positive condition requirement */
11             INCREMENT SatisfyCount
12          END IF
13       END IF
14       ELSE IF srprecn HAS "!=" THEN
15          IF srprecn NOT EQUAL condition THEN /*
            SRPreCondition fulfils certain negative condition
            requirement */
16             INCREMENT SatisfyCount
17          END IF
18       END ELSE IF
19       ELSE THEN
20          IF  condition COMPLY with srprecn THEN /*
            SRPreCondition fulfils certain numerical (>=/<=)
            condition requirement */
21             INCREMENT SatisfyCount
22          END IF
23       END ELSE
24    END FOR
25    IF SatisfyCount >= ConditionCount THEN /* all
      preconditions satisfied
26       SET ConditionSatisfied to TRUE
27    END IF
28 END ELSE

OUTPUT: ConditionSatisfied
```

Figure 5 Cloud service operation precondition verification algorithm

Specifically, the verification algorithm of cloud service operation parameter requirements in Figure 4 demonstrates the first control prior to operation execution. Basically, the service entities, either selected or manually entered data, are processed in two sets of key-value pairs. One holds the entity name and type information whilst the other holds the actual entity data and format information. Then, according to the retrieved SRParameter specification, the parameters (names, types and format) are verified against the relevant mandatory requirements. As all of the mandatory parameters are satisfied, it would indicate that the parameter verification process is complete. Subsequently, the respected parameter data can be sent to appropriate operation handler for further action.

Service Operation Precondition Verification

The service operation precondition verification is implemented depending on the nature and format of the condition specification, shown in Figure 5. Specifically, if the candidate requires no SRPreCondition (unconditional), the verification will be complete instantly. Otherwise, the positive, negative or numerical condition is verified based on a dynamically initiated real-time service entity information check. Once all mandatory verifications are complete, the operation can then execute as dispatched.

While the above verifications are mainly for use of simple individual cloud service operations, SAMOS also offers advanced usages, known as operation assistances. Indeed, the assistances can be widely enabled, such as to automatically prepare preconditions and gather parameters, to program the execution schedules for multiple relevant operations, or to assess the applicability for potential service interactions and compositions. They are provided based on the reasoning analysis of cloud service entity operational relationships and the operation specification elements involved. For instance, operation grouping applicability can be analyzed by seeking operations with similar types/requirements in both their SRParameters and SROutcomes; Operation chaining applicability can be determined when the operations own SROutcome and SRParameter match, or SRPostCondition and SRPreCondition match (equivalence). The description summary of the proposed cloud service operation assistances can be found in Table 3.

Basic Assisted Service Operation Request (BASR) assistance reasoning

BASR serves to help users understand the required SRPreConditions and SRParameters for service operations. It also assists in relevant operation preparations during the operation process. Basically, as a user selects certain cloud service/CSI/PSSA and SRParameter(s), BASR can actively examine all available operations for unsatisfied operation conditions and parameters. Then, relevant information regarding how to fulfil such conditions or/and obtain the mandatory parameter(s) is produced to assist the user for further actions.

BASR is implemented on a per cloud service/service instance and per service request basis. This means that the assistance algorithm does not consider the potential subsequent impact resulted from one operation to another (or from service entity to another).

Concurrent Combined Service Operation Request (CCSR) assistance reasoning

As some cloud services own the same model/type/function, their service operations are very similar.

Table 3 Cloud service operation reasoning assistance type

Reasoning Assistance Name	Assistance Description	Reasoning Scale	Operation Scheduling	Precondition & Parameter Preparation	Reasoning Steps
BASR	To assist in preparation of precondition and parameters for unsatisfied service operations	Single cloud (CSP)	None	Guided manual input	1. For the unsatisfied SRParameters and SRPreConditions, list possible options based on current selected SRSubject and SRParameters, plus the real-time status of them;
CCSR	To assist in multiple concurrent service operations of similar types	Multiple clouds (CSPs)	None	Manual input	1. Get SRSubjects' operations which have satisfied SRParameters; 2. Filter the operations based on whether their preconditions fulfil the real-time SRSubject statuses; 3. Produce the operation lists for the applicable SRSubjects;
SCSR	To assist in automatic scheduled executions of a series of operations in a logical sequence	Single cloud (CSP)	Yes	Manual input	1. Get SRSubjects' operations which have satisfied SRParameters; 2. For the operations, seek for those which have precondition SRPostCondition matches; 3. Compose these operations into sequenced chains by filtering Them from their factorial combinations, according to the two-two sequenced connections; 4. Filter the operation chains based on whether the first operation's preconditions fulfil the real-time SRSubject status; Produce the operation lists for the applicable SRSubjects
IOSR	To assist in seeking possible service interactions by linking appropriate operations in a scheduled sequence	Multiple clouds (CSPs)	Yes	Automatic preparation	1. For all SRSubjects' operations, seek for those which have SROutcomes SRParameters (equivalence) matches; 2. For all SRSubjects' operations, seek for those which have SRPreCondition SRPostCondition matches; 3. Compose these operations into sequenced chains as long as their SRPostConditions and SRPreConditions are not contradictory, according to the two-two sequenced connections; 4. Filter the operation chains based on whether the first operation's preconditions fulfil the real-time SRSubject status; 5. Produce the operation lists for the applicable SRSubjects

Indeed, the operation specification patterns for such operations often coincide. Due to this nature, these operations can be composed and then executed concurrently towards greater efficiency.

CCSR reasoning algorithm enables a convenient means of executing multiple similar service operations for eligible cloud services or instances, even across multiple clouds. Specifically, as a user selects multiple SRSubjects and a series of required SRParameters, CCSR collects the eligible operations into simultaneously executable groups, and offers the options to the user. For the reason that CCSR assistance is provided at a per request basis, although it acts to control multiple cloud service entities, the algorithm does not consider the potential impact resulted from one request to another. Hence, the assistance still does not adopt any scheduling controls.

Sequenced Chained Service Operation Request (SCSR) assistance reasoning

For some service operations, the SRPostCondition of an operation may happen to match (or be equivalent to) another's SRPreCondition. In other words, the successful execution of some operations would enable certain subsequent operations dynamically by satisfying their precondition requirements. As a series of such operations are linked one another with precondition and postcondition matches, a sequenced operation chain can be composed in real-time. SCSR serves to seek such operation chains and provide assistances in scheduling their sequenced executions. Although this may happen over multiple clouds, we consider that it would be more reasonable for implementation in a single cloud. Hence, the scale of SCSR reasoning is restricted to that.

SCSR reasoning seeks for operation combinations which satisfy the following three requirements: Firstly, within the available operations of the chosen SRSubjects, there are coherent/equivalence matches between their SRPreConditions and SRPostConditions from one to another (empty condition, like "unconditional", is not accounted here). Secondly, all of the SRParameters (if any) of such operations are presented in prior simultaneously. Thirdly, the real-time service (entity) condition meets the SRPreCondition of the first operation (in the chain). Fourthly, any duplicate/repeated operation is to be eliminated from the chain. Finally, these dynamically composed operation chains are arranged, where users can select and execute them depending on their intension.

Interactive Orchestrated Service Operation Request (IOSR) assistance reasoning

With a successfully executed service operation, the SROutcome retrieved in real-time can be used as SRParameter for further service operations. This provides another means towards dynamic service (operation) orchestrations. IOSR is designed to provide operation orchestration assistances that require minimum user effort. It can intelligently select and compose all necessary operations, and automatically prepare the SRPreConditions and SRParameters for relevant service interaction tasks. Hence, IOSR assisted operation tasks only ask users for SRSubjects entry and require no further user interventions.

In order to provide the assistance with proactive condition and parameter preparation, IOSR reasoning algorithm does not simply consider the exact match between operations' SROutcome and SRParameter or between SRPreCondition and SRPostCondition. Instead, it checks the equivalence of SROutcome and SRParameter entities. As long as there is no direct conflict between the SRPreCondition and SRPostCondition, a link can be formed (for certain valid interaction intentions). Indeed, whenever there are commonly recognizable service and resource entities, they can result in "orchestration points" between the interactive pair of clouds, regardless of their CSPs or service types.

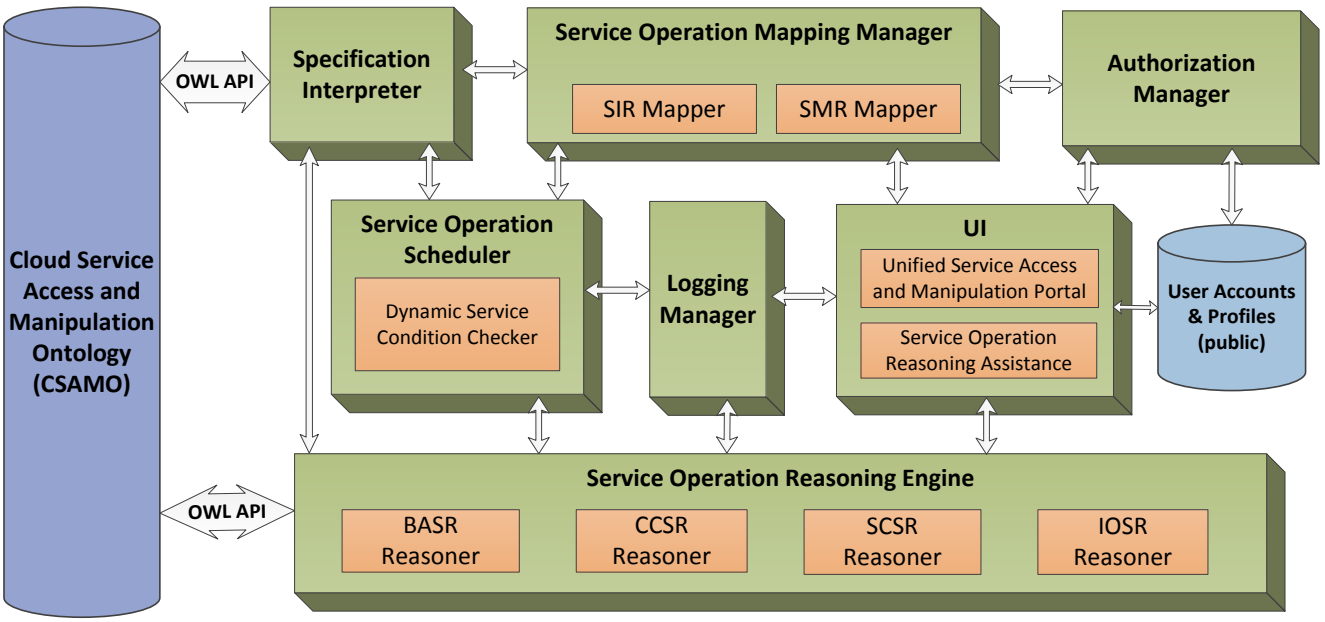

Figure 6 USAMS system architecture

Prototype implementation

This section outlines the implementation in terms of the deployment of cloud service access and manipulation ontology (CSAMO) and system prototype tool (USAMS). Moreover, for real cloud service management enablement, details of cloud service operation API mapping are presented, as a key component of the prototype.

System architecture

USAMS is implemented in java. As Figure 6 shows, it consists of six main components: Specification Interpreter, Service Entity & Operation Mapping Manager, Service Operation Scheduler, Service Operation Reasoning Engine, Authorization Manager and UI.

CSAMO is deployed based on SAMOS framework. It encapsulates various real cloud service operation specification data, written in OWL 2 [24]. While CSAMO is utilized as the main knowledge source for cloud service operations, the User Accounts & Profiles database used here stores users' cloud service account data (including CSP API credentials). This is mandatory for most service operations over different clouds.

Specification Interpreter is responsible for retrieving and translating granular service operation and entity specifications and the respected API request and response data. This includes interpreting all sorts of cloud service entities in SIR and SMR operations, plus gathering the entire operation element details (i.e. SRPreCondition, SRParameter, SROutcome, etc.).

Service Entity & Operation Mapping Manager manages the operation API mapping entries so that users' service operation requests can be implemented properly. It has two separate mappers inside for use of SIR and SMR operations respectively. Indeed, due to the many different characteristics between the two operation categories, the operation handling processes are treated separately. This prevents potential issues as attempting to schedule a group of mixed operation tasks.

USAMS UI deals with a wide range of service operation execution tasks by providing a unified portal for real-world cloud service access and manipulation. It has two sub components: Unified Service Access and Manipulation Portal and Service Operation Reasoning Assistance. While the former allows users view and execute various service operations, the latter actively assists users for advanced operation execution tasks. Together, these enable a generic and interactive portal for service access, manipulation and interaction regardless of service models/types/CSPs.

Service Operation Reasoning Engine incorporates four individual reasoners, each works for a certain operation assistance scenario. BASR Reasoner assists in preparation of the required operation data so as to guide users throughout operation process. The scale of its reasoning is restricted to operations in a single cloud. CCSR Reasoner assists in grouping similar operations for users so as to enable simultaneous executions, even if such are implemented across distinct clouds. SCSR Reasoner assists in scheduling chained tasks when a series of operation are found with certain execution dependency relations one another. Finally, IOSR Reasoner assists in implementing service orchestration tasks by analyzing the possibilities for potential operation interactions for selected services.

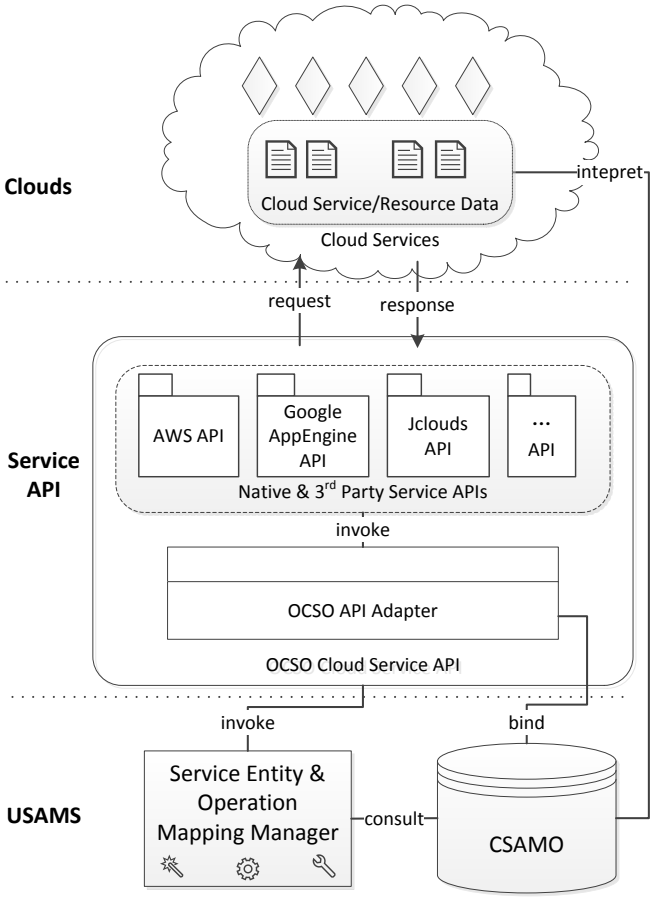

Figure 7 Cloud service operation specification and API call mapping

Service Operation Scheduler acts to control the schedule and execution of operation tasks. Whenever a user initiates an operation, the component would first verify the user's API credentials (for the target cloud). Then, the necessary data format verifications (for SRParameter and SRPreCondition) are performed. If no error occurs, the mapped operation task will be executed execution. Further, for advanced service operation tasks, the component works closely with Service Operation Reasoning Engine, enabling the tasks reasoned by the reasoners. With its internal Dynamic Service Condition Checker, it provides various automatic dynamic scheduling controls for grouped SIR and SMR tasks. For every SMR operations executed, the logs are forwarded to Logging Controller.

Logging Controller documents critical system and operation logs so that users can examine them as required. This enables event tracking, diagnostics and performance evaluation tasks via the platform.

Mapping service operation ontology specifications to service API calls

Nowadays, most cloud service providers also release native service API libraries and/or complete SDKs as customized service and resource control interfaces. Meanwhile, a series third-party service APIs are also available as an alternative programmable service and resource entrance. In fact, these service API call/respond operations

often enable more effective service access and enhanced service function manipulation, since they allow users to control services and relevant resources from a much lower level [25].

While each of the cloud service API requests can be addressed with certain service operation specification data, a mapping can be implemented between the API and respected operation specifications. To enable flexible and generic service access and manipulation via a single point of interface, we adopt OCSO API [25]. It comprises OCSO API Adapter which can flexibly invoke cloud service API libraries as (user) requested. Seen in Figure 7, the mapping is controlled via Service Entity and Operation Mapping Manager. Here, depending on the designated API library and the operation requirement, each service's operation (object property assertion) is mapped to at least one API request (for each library). Moreover, the additional operation specifications (e.g. SRParameter, SRPreCondition and SROutcome) must be consistent with the relevant information against the API request.

As the mapping between service operation specifications and API calls is established, the modelled cloud service operations can be prepared and executed (with relevant cloud API user account credentials) as retrieved.

Case studies

To demonstrate the practice use and to evaluate the proposed approach, we present some case studies on service operation specifications and service access, manipulation and orchestration tasks, using a series of services from multiple popular clouds. In addition to the contents presented below, more specifications for other cloud services and providers are provided in Appendix A, B and C.

EC2 service operation specifications

Table 4 shows the specifications of some AWS EC2 [20] operations (retrieved from CSAMO). Within SAMOS, the operations are divided into two categories, and seen in three entity levels. For instance, at EC2 service level, there are SIRs for overall instance information retrieval (e.g. "List VM Instance") and SMRs for (overall) instance management (e.g. "Create VM Instances"). At EC2 CSI level, there are SIRs for individual data retrieval (e.g. "Get VM Architecture") and SMRs for individual control (e.g. "Terminate VM Instance"). At EC2 PSSA level, take EC2 AMI as an example, it has SIRs and SMRs for specific entity (data) retrieval and control (e.g. "Get Image Platform", "Delete Image").

Further, the detailed operation elements can be well presented with SAMOS. For instance, basic SIRs such as "List VM Instance" and "Get VM Architecture", they do not require any SRPreCondition and only require a single SRParameter (each); they would not trigger any SRPostCondition change after execution. In contrast, SMRs act differently: they would alter certain service/CSI/PSSA conditions on successful execution, and may require several SRParameters (e.g. "Create VM Instance(s)"). Additionally, even for the same operation, the specifications can differ for distinct SRSubjects: the two "Create VM Instance(s)" executable for EC2 and EC2 AMI appear to be very similar, except that one requires fewer SRParameters than the other.

Unified cloud service access and manipulation

A practice use of the cloud service operation specifications is seen as the enablement of unified service access and manipulation through USAMS. Figure 8 demon-

Table 4 EC2 service/CSI/PSSA operation specifications (from CSAMO)

Cloud Service Level Operations	***AWS EC2***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List VM Instance	SIR	Unconditional	EC2 Region(M)	EC2 InstanceIDs	Unconditional
Create VM Instance(s)	SMR	< account allowance, i.e. 20 instances per region	EC2 RequestCount(O), EC2 InstanceType(M), EC2 AMIID(M), EC2 KeyName (M), EC2 SecurityGroup(O), EC2 Region(M), EC2 Monitor(O), EC2 AvailabilityZone (O), etc.	EC2 InstanceID(s)	Instance(s) is in "running" state
Resize VM Instances	SMR	Instances are in "stop" state	EC2 InstanceIDs(M), EC2 InstanceTypes(M)	Operation Succeeded	Instances are in "stop" state
CSI Level Operations	***AWS EC2 Instance***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get VM Architecture	SIR	Unconditional	EC2 InstanceID(M)	EC2 Instance Architecture	Unconditional
Create VM Image	SMR	Unconditional	EC2 InstanceID(M)	EC2 AMIID	AMI is in "available" state
Terminate VM Instance	SMR	Instance is NOT in "terminated" state	EC2 InstanceID(M)	Operation Succeeded	Instance is in "terminated" state
PSSA Level Operations	***AWS EC2 AMI (VM image)***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Image Platform	SIR	Unconditional	EC2 AMIID(M)	EC2 Instance Platform	Unconditional
Create VM Instance(s)	SMR	< account allowance, i.e. 20 instances per region	EC2 InstanceID(M) EC2 RequestCount(M), EC2 InstanceType(M), EC2 KeyName (O), EC2 SecurityGroup(O), EC2 Monitor(O), etc.	EC2 InstanceID(s)	Instance(s) is in "available" state
Delete Image	SMR	Image is in "available" state	EC2 AMIID(M)	Operation Succeeded	Unconditional

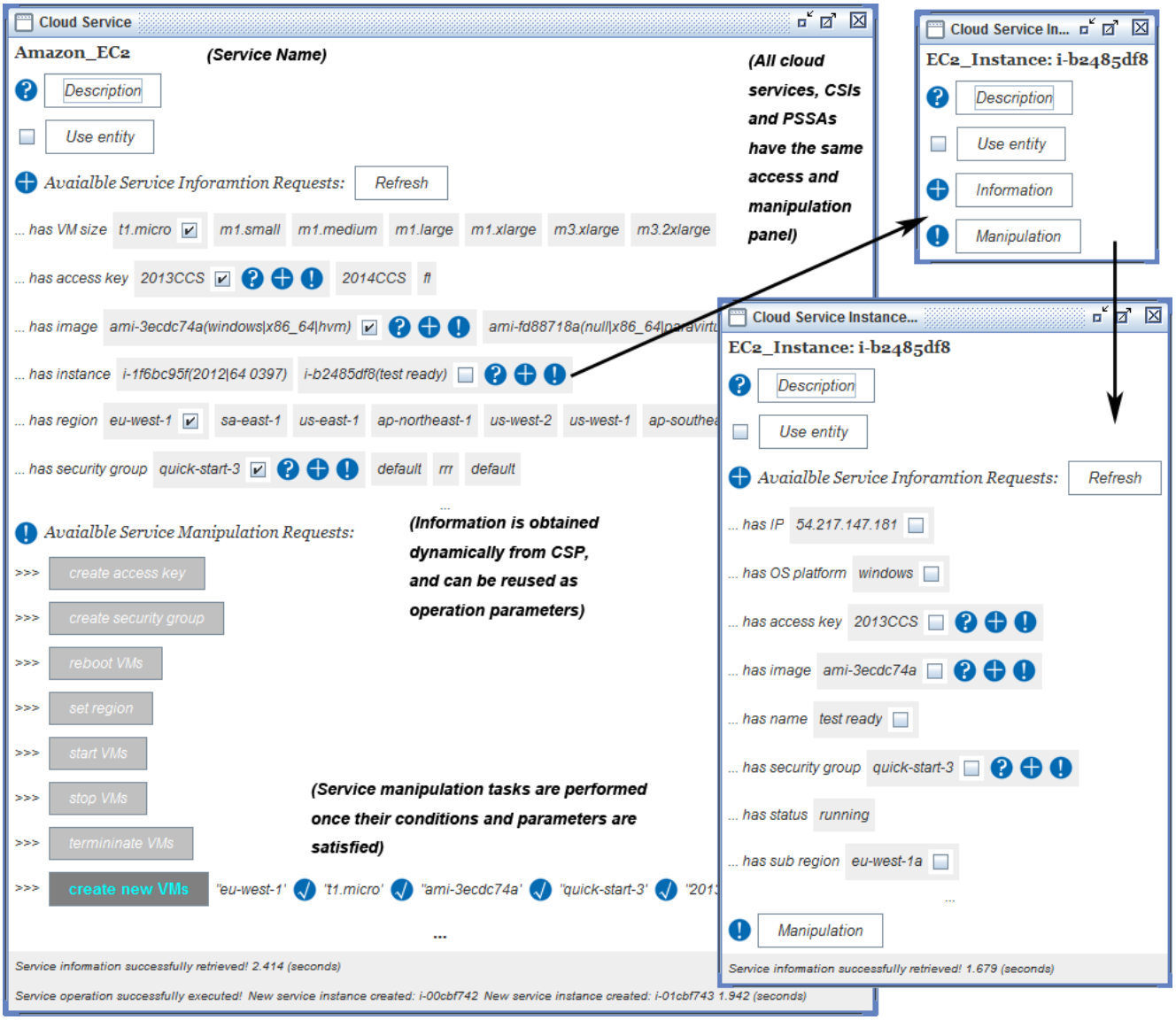

Figure 8 Unified cloud service access and manipulation

strates the appearance of USAMS unified service operation interface (with EC2).

Every SRSubject is initially displayed in a small panel. There are four buttons in the panel: "Description", "Use Entity", "Information" and "Manipulation". By clicking "Description", users can view its annotation description through external knowledge sources [26]. "Information" and "Manipulation" buttons lead to the respected SIR and SMR operation retrieval (from CSAMO). Then, if a user's API account authorization permits, one can execute operations via USAMS.

Due to the simple nature, SIR operations are executed concurrently as the user clicks the "Refresh" button aside. SMR operations are to be executed individually; they would only execute when the all of the requirements (i.e. SRPreCondition and SRParameter(s)) are satisfied, which are checked dynamically by relevant USAMS components. To enhance operation parameter input experience, all dynamically acquired service entity (information) can be reused as parameters.

In addition, for any CSI or PSSA acquired through SIR operations, their operation panels can be called from a click (see Figure 8). Afterwards, the remaining options follow the same steps as the above. In this way, USAMS achieves a unified service access and manipulation through a common presentation and execution interface, regardless of the CSP, service or operation types.

Cloud service operation reasoning assistance: IOSR

As previously discussed, an additional benefit of SAMOS framework is its operation reasoning assistance. Here, we present an IOSR assistance case on orchestrating EC2 and Rackspace Cloud Load Balancer (RSLB) [27].

As a user selects the two services to seek orchestration feasibility, the reasoning engine analyzes the possible entity relationships throughout CSAMO. Here, the common nature of public IP address becomes the key link for the orchestration: EC2 instances own public IPs; RSLB instance needs public IPs for node entries; so EC2 instances can be inserted as nodes for RSLB instances. As a result, a series of relevant operations can then be selected

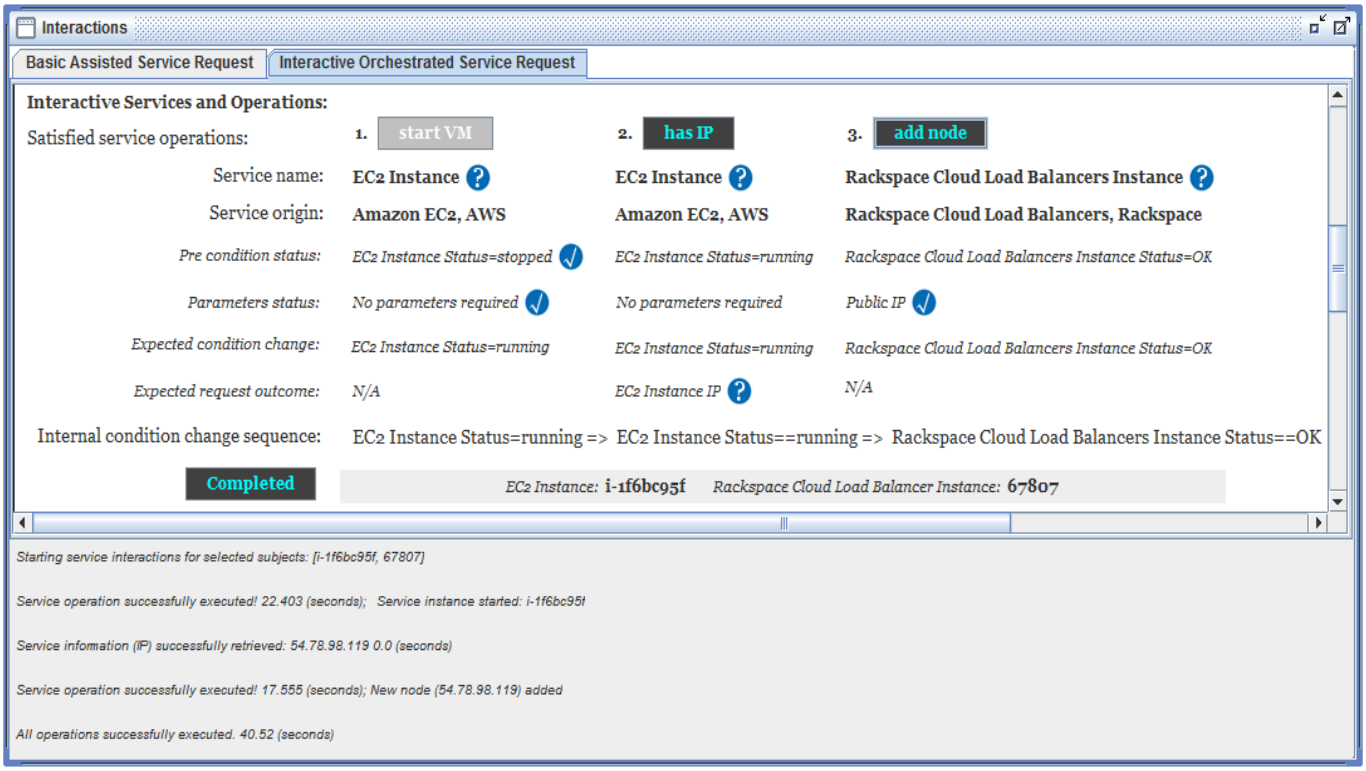

Figure 9 Service orchestration with IOSR operation reasoning assistance

and composed into chains. Seen in Figure 9, based on the real-time service conditions, USAMS reasoning engine outputs an operation chain with two EC2 operations and one RSLB operation. The two EC2 operations are selected to obtain an IP address whilst the rest is to complete the orchestration by using the address. According to the relevant specifications displayed below each operation, "start VM" would turn a VM on; "has IP" would acquire its IP address; "add node" would add the VM into a RSLB instance by using its IP.

The case validates that IOSR execution process runs intelligently on operation condition and parameter preparation: for SRPreCondition fulfillment, it adds the "start VM" operation at the beginning of the chain due to the "has IP" needing such condition requirement; for SRParameter fulfilment, the IP address is not manually entered but a dynamic real-time service entity acquired form CSP.

Service operation remote execution performance

The cloud service operation execution performance study involves a variety of services and operations from multiple clouds. We provide operation execution time comparisons between USAMS and provider native command line interfaces (CLI). To deal with the test data deviations (e.g. due to unexpected/slight QoS fluctuation), the results are regulated: the operation tests are conducted on two separate days; the results are obtained from several sample

Table 4 Single SIR access time comparison (via native CLI/USAMS)

Service provider	Typical SIR / Method	List cloud VM instances (IaaS)	List cloud database instances (PaaS)	List cloud files (SaaS)	List cloud load balancers (SaaS)	Success rate (based on 200 tests)
AWS	Via native CLI	0.757 sec	0.519 sec	0.666 sec	0.550 sec	100%
	Via USAMS	1.185 sec	1.032sec	1.143 sec	1.263 sec	100%
Rackspace	Via native CLI	3.242 sec	3.280 sec	4.009 sec	3.202 sec	100%
	Via USAMS	5.534 sec	5.281 sec	5.129 sec	5.483 sec	100%

Table 5 Single SMR execution time comparison (via native CLI/USAMS)

Service provider	Typical SMR / Method	Create cloud VM instance (IaaS)	Terminate cloud VM instance (IaaS)	Create cloud load balancer (SaaS)	Update cloud load balancer (SaaS)	Success rate (based on 200 tests)
AWS	Via native CLI	2.450 sec	2.086 sec	1.038 sec	0.588 sec	100%
	Via USAMS	2.732 sec	1.943 sec	0.758 sec	0.682 sec	100%
Rackspace	Via native CLI	3.237 sec	3.742 sec	3.235 sec	3.249 sec	100%
	Via USAMS	3.383 sec	4.147 sec	3.539 sec	3.761 sec	100%

tests, where any excessive values are eliminated.

The services selected for the operation experiment are EC2, Relational Database Service (RDS) [28], Elastic Load Balancer [29], Cloud Servers [30], Cloud Databases [31] and Cloud Load Balancers [27]. They belong to AWS and Rackspace two CSPs respectively. Accordingly, the two CLIs involved in the experiment are AWS CLI [32] and Rackspace CLI (rumm [33]). For SIR, a series of service instance data retrieval operations are tested to justify the typical performances for IaaS, PaaS and SaaS operations individually. For SMR, various service instance manipulation operations are tested, including instance creation, deletion, updating, etc.

SIR(s) remote execution performance

Table 4 demonstrates the experiment results of accessing AWS and Rackspace IaaS/PaaS/SaaS service instance data. While the two interfaces both show consistent success rates of 100%, the response time varies. Generally speaking, the CLIs offer faster response than USAMS for all the SIR operations. Specifically, for AWS SIRs, all the operations are handled within 1 second via AWS CLI, whereas USAMS takes some milliseconds extra. Meanwhile, Rackspace CLI offers slower accesses of more than 3 seconds for the SIRs, and USAMS requires approximately additional 2 seconds.

The multiple service operation remote execution experiment is implemented in AWS EC2 only, due to its reliable success rates and with reasonable elapsed time. For the SIR experiment on retrieving multiple EC2 instances data, it reveals similar access and response patterns between the CLI and USAMS (see Figure 10). Faster completion is found via the CLI, regardless of the total number of operations. However, the overall performance difference between the two interfaces is minimal, i.e. generally within 0.3 seconds.

SMR(s) remote execution performance

The SMR operations selected for the experiment are IaaS and SaaS service instance creation, modification and termination tasks. Specifically, the IaaS VM creation operations are deployed with plain Linux Red Hat 7.0 image on m3.large (2vCPU/7.5GB RAM) for EC2 and 4GB standard instance (2vCPU/4GB RAM) for Rackspace Cloud Servers. Then, the instances created are used for the termination tests. Meanwhile, the SaaS cloud load balancer creation and update operations are performed by creating an http load balancer, followed by node adding modifications.

Seen from the experiment data in Table 5, overall, the

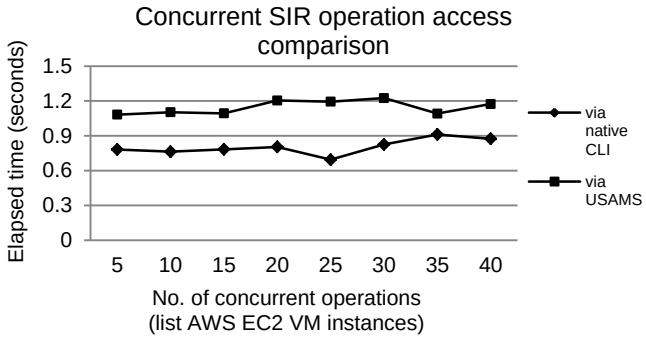

Figure 10 Multiple SIR operations execution comparison

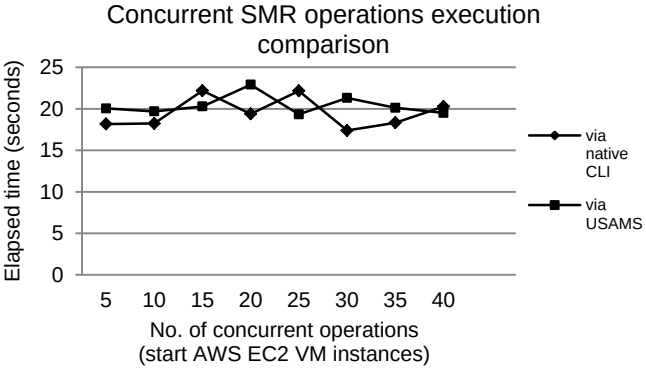

Figure 11 Multiple SMR operations execution comparison

success rates of all the operations remain at 100%. Yet, each operation execution respond appears to be different. For AWS IaaS operations, the VM creation tasks are completed a little faster through CLI, but termination operations respond quicker via USAMS. For AWS SaaS operations, the load balancer creation is handled slightly sooner via USAMS, although CLI manages to update the load balancers with a bit less time. On the other hand, with regard to Rackspace SMR tasks, all the VM and load balancer instance manipulation tasks tend to consume slightly more time via USAMS.

Further, considering the SMR experiment on starting multiple VM instances, the execution time data is illustrated in Figure 11. For the 5 to 40 tasks deployed, both interfaces require some 20 seconds for completion. No obvious increase is found despite more operations being involved. Regarding the performance differences between the two interfaces, there is not any clear distinction.

Evaluation and discussion

Table 6 Cloud service specification framework comparison

Approach	Syntax/ Semantics	Model Core/Base Concepts	Management Interface	Service Orchestration
OCCI	OCCI Grammar	Category, Kind, Mixin, Resource Instantiation, Collections, Discovery /Entity, Resource, Link, Action) [35]	Testing tool, doyouspeak OCCI,OCCI API	OCCI client
TOSCA	YAML	Topology Templates, Plans /Service, Node, Relationship, Requirement, Capability, Artifact, Policy, Cloud Service Archive [36]	OpenTOSCA, jclouds and PyTosca API	Pre-defined Plans
mOSAIC	OWL	Environment, Infrastructure, Resource, Runtime Component, Stateful Component, Stateless Component, etc. [37]	mOSAIC API	mOSAIC Cloud Agency
SAMOS	OWL	Entity and operation classifications, Entity datatype specifications, Entity operational relationship Specifications, etc.	USAMS prototype tool, flexible choice of API libraries via OCSO API	Lightweight automatic reasoning

As illustrated in the EC2 case study, SAMOS framework can adequately model a wide range of operations. Its classifications of cloud service entities and operations enable structured specification presentation layout. The relevant operation element specifications reveal sufficient details for operation executions. Additionally, as shown in Appendix A, B and C, the approach can be flexibly applied to other IaaS providers (e.g. Rackspace Cloud Servers), and so as other PaaS and SaaS services (e.g. AWS Elastic Beanstalk [32], Rackspace Cloud Load Balancers). Consequently, this would enhance cloud service interoperability and composition. Further, to evaluate SAMOS against other well-established cloud (service) specification frameworks/models, we provide the data comparison with OCCI, TOSCA and mOSAIC. Shown in Table 6, the four approaches involve dissimilar core/base model concepts with different specification semantics. They adopt distinct management tools/APIs as cloud service interfaces and enable service orchestration with own solutions. In contrast, SAMOS achieves a distinguished outcome for service management and orchestration tasks due to the flexible choices of API libraries and the lightweight operation reasoning assistances.

Meanwhile, the performance evaluation with USAMS covers a wide range of typical service operations. Obtained experiment results illustrate some performance differences between the proposed approach and provider-native CLIs. For SIR operation handling performance, the prototype demonstrates the same solid success rates regardless of the type/nature of operations, although there are some minor processing overheads. Considering SMR operation tasks, USAMS demonstrates competitive performance in comparison with the native CLIs. There is no obvious delay or distinction for many service management tasks involved. In fact, the slight overheads are caused by two main factors: the API libraries (AWS Java SDK version 1.8.3 and jclouds Rackspace API libraries version 1.7.0) used (by OCSO) decide the main processing time; USAMS components also consume minor extra time while processing the obtained data, preparing for the operations and other additional tasks. In overall, USAMS enables reliable cloud service remote management with acceptable performance. The proposed approach offers a more flexible and intelligent remote management solution than individual portals of vast cloud service providers.

Conclusions and future work

This paper proposes a cloud service operation specification approach which can be applied to diverse cloud service models and resource types, namely SAMOS. The framework can reveal comprehensive information with regard to the involved service entities, their attributes and relationships, plus a series of operational elements including parameters, conditions and outcomes. The ontological modelling approach also enables a range of operation reasoning which can provide assistances for advanced tasks such as simultaneous, chained and service orchestration operations.

To evaluate the proposed approach, CSAMO semantic model and USAMS prototype are implemented. Together, they enable a unified cloud service access and manipulation via a structured management interface. This is validated through considerable experiments that are conducted over Amazon and Rackspace IaaS, PaaS and SaaS clouds. The test results suggest that the proposed approach can provide competitive service operation reliability and effectiveness, especially while handling groups of operation tasks.

In future work, we plan to extend the approach by introducing service recommendation engine and service interaction agent. The recommendation module should enable more user friendly service selection and operation experiences. The service interaction agent would drive more effective service compositions with enhanced operation reasoning applications.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

DF from Edinburgh Napier University developed the approach, algorithm, model and prototype. XL from Edinburgh Napier University supervised, tested the developments. IM from Edinburgh Napier University and CP from Dublin City University provided domain specific expertise in the experiment and evaluation of the approach. All Authors read and approved the final manuscript.

Acknowledgment

We wish to thank British Royal Society of Edinburgh (RSE-Napier E4161) and Lawrence Ho Research Fund (LH-Napier2012) for supporting the work presented. We acknowledge the support from a joint grant by the British Royal Society and the Royal Irish Academy on a Cloud Migration Framework 2014-2016, IE131105, and Science Foundation Ireland grant 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie).

Author details

[1] School of Computing, Edinburgh Napier University, 10 Colinton Road, Edinburgh, EH10 5DT, UK
[2] Lero, School of Computing, Dublin City University, Dublin, Dublin 9, Ireland

References

[1] Marinescu DC (2013) Cloud computing: Theory and practice. Elsevier. Waltham. USA:2-17
[2] Buyya R, Vecchiola C, Thamarai S (2013) Master cloud computing: Foundations and applications programming. Elsevier. Waltham. USA:3-27
[3] Orozco JMS (2012) Applied ontology engineering in cloud services, networks, and management systems. Springer Science+Business Media:23-52

[4] Moreno-Vozmediano R, Montero RS, Llorente IM (2011) Key Challenges in Cloud Computing: Enabling the Future Internet of Services. IEEE Internet Computing 17(4):18-25
[5] Apache Jclouds. http://jclouds.apache.org/. Accessed 30 Jul 2014
[6] Apache Libcloud. https://libcloud.apache.org/. Accessed 12 Aug 2014
[7] Fog. http://fog.io/. Accessed 02 Aug 2014
[8] TOSCA Overview. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca#overview. Accessed 27 May 2014
[9] Moscato F, Aversa R, Martino BD, Venticinque S (2011) An ontology for the cloud in mOSAIC. Cloud Computing: Methodology, System, and Applications. CRC Press:467-485
[10] OCCI. http://occi-wg.org/. Accessed 31 Jul 2014
[11] Toosi AN, Calheiros RN, Buyya R (2014) Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. ACM Computeing Surveys 47(1):7
[12] Loutas N, Peristeras V, Bouras T, Kamateri E, Zeginis D, Tarabanis K (2010) Towards a Reference Architecture for Semantically Interoperable Clouds. IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)., pp 143-150
[13] Demchenko Y, Ngo C, de Laat C, Rodriguez J, Contreras LM, Garcia-Espin JA, Figuerola S, Landi G, Ciulli N (2013) Intercloud Architecture Framework for Heterogeneous Cloud based Infrastructure Services Provisioning On-Demand. IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA)., pp 777-784
[14] Deltacloud. https://deltacloud.apache.org/. Accessed 22 May 2014
[15] Fielding R (2000) Architectural Styles and the Design of Network-based Software Architectures. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Accessed 16 April 2015
[16] The Dasein Cloud API. http://dasein-cloud.sourceforge.net/. Accessed 26 Jun 2014
[17] Bastião Silva LA, Costa C, Oliveira JL (2013) A common API for delivering services over multi-vendor cloud resources. J Systems and Software 86(9):2309-2317
[18] Petcu D, Craciun C, Neagul M, Lazcanotegui I, Rak M (2011) Building an interoperability API for Sky computing. International Conference on High Performance Computing and Simulation (HPCS)., pp 405-411
[19] Bernabe JB, Marin Perez JM, Alcaraz Calero JM, Garcia Clemente FJ, Perez GM, Gomez Skarmeta AF (2014) Semantic-aware multi-tenancy authorization system for cloud architectures. Future Generation Computer Systems 32:154-167
[20] Amazon EC2 User Guide. http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-ug.pdf. Accessed 17 May 2014
[21] Federici C (2014) Cloud Data Imager: A unified answer to remote acquisition of cloud storage areas. Digital Investigation 11(1):30-42
[22] Povedano-Molina J, Lopez-Vega JM, Lopez-Soler JM, Corradi A, Foschini L (2013) DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. Future Generation Computer Systems 29(8):2041-2056
[23] Li Q, Wang Z, Li W, Cao Z, Du R, Luo H (2013) Model-based services convergence and multi-clouds integration. Computer in Industry 64(7): 813-832
[24] OWL 2 Web Ontology Language Document Overview. http://www.w3.org/TR/owl2-overview/. Accessed 23 Dec 2013
[25] Fang D, Liu X, Liu L, Yang H (2014) OCSO: Off-the-cloud service optimization for green efficient service resource utilization. J Cloud Computing 3(9). http://www.journalofcloudcomputing.com/content/3/1/9. Accessed 17 Aug 2014
[26] Fang D, Liu X, Romdhani I, Pahl C, Jamshidi P (2015) An agility-oriented and fuzziness-embedded semantic model for collaborative cloud service search, retrieval and recommendation. Future Generation Computer Systems (unpublished).
[27] Rackspace Cloud Load Balancers Developer Guide. http://docs.rackspace.com/loadbalancers/api/v1.0/clb-devguide/clb-devguide-20140630.pdf. Accessed 13 Apr 2014
[28] Amazon Relational Database Service User Guide. http://awsdocs.s3.amazonaws.com/RDS/latest/rds-ug.pdf. Accessed 02 May 2014
[29] Amazon Elastic Load Balancing Developer Guide. http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-dg.pdf. Accessed 16 Sep 2014
[30] Rackspace Next Generation Cloud Servers Developer Guide. http://docs.rackspace.com/servers/api/v2/cs-devguide/cs-devguide-20140311.pdf. Accessed 27 Mar 2014
[31] Rackspace Cloud Database Developer Guide. http://docs.rackspace.com/cdb/api/v1.0/cdb-devguide/cdb-devguide-latest.pdf. Accessed 06 May 2014
[32] AWS Command Line Interface User Guide. http://docs.aws.amazon.com/cli/latest/userguide/aws-cli.pdf/. Accessed 03 Jun 2015
[33] Rackspace rumm. http://rackspace.github.io/rumm/. Accessed 03 Jun 2015
[34] AWS Elastic Beanstalk API Reference. http://s3.amazonaws.com/awsdocs/ElasticBeanstalk/latest/awseb-api.pdf. Accessed 16 April 2015
[35] Open Cloud Computing Interface – Core. https://www.ogf.org/documents/GFD.183.pdf. Accessed 21 Dec 2014
[36] Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. Accessed 22 Nov 2014
[37] Moscato F, Fortis F, Munteanu V (2014) Cloud ontology and Cloud resources representations. mOSAIC public deliverables D 1.2. http://www.mosaic-cloud.eu/index.php?option=com_chronocontact&Itemid=186. Accessed 29 Oct 2014

Appendix A: IaaS Service/CSI/PSSA operation specifications for Rackspace Cloud Servers

Cloud Service Level Operations	***Rackspace Cloud Servers***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List VM Instances	SIR	Unconditional	Rackspace Region(M) Rackspace FlavorID(M)	Rackspace CloudServerIDs	Unconditional
Create VM Instance	SMR	< Rackspace CloudServersAbsolute Limits, i.e. 100	Rackspace Server name(M), Rackspace ImageRef(M), Rackspace OSDiskConfig (O), Rackspace Metadata(O), Rackspace KeyPair(O), etc.	Rackspace CloudServer InstanceID	Instance is in "ACTIVE" state
Reboot VM Instances	SMR	Unconditional	Rackspace CloudServerID(M), Rackspace RebootType(M), e.g. SOFT, HARD	Operation Succeeded	Instances are in "ACTIVE" state
Resize VM Instances	SMR	Instances are Rackspace Standard Flavor	Rackspace CloudServerID, Rackspace FlavorID(M)	Operation Succeeded	Unconditional
CSI Level Operations	***Rackspace Cloud Servers Instance***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get VM Flavor	SIR	Unconditional	Rackspace CloudServer InstanceID(M)	Rackspace FlavorID	Unconditional
Get VM Image	SIR	Unconditional	Rackspace CloudServer InstanceID(M)	Rackspace ImageRef	Unconditional
Create VM Image	SMR	Unconditional	Rackspace CloudServer InstanceID(M)	Rackspace ImageRef	Image is in "ACTIVE" state
Terminate VM Instance	SMR	Instance is NOT in "DELETED" state	Rackspace CloudServer InstanceID(M)	Operation Succeeded	Instance is in "DELETED" state
PSSA Level Operations	***Rackspace Cloud Servers Image***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Image Architecture	SIR	Unconditional	Rackspace ImageRef (M)	Rackspace ImageArch	Unconditional
Get Image OS Type	SIR	Unconditional	Rackspace ImageRef (M)	Rackspace ImageOSType	Unconditional
Create VM Instance(s)	SMR	< Rackspace CloudServersAbsolute Limits, i.e. 100	Rackspace Server name(M), Rackspace ImageRef(M), Rackspace OSDiskConfig (O), Rackspace Metadata(O), Rackspace KeyPair(O), etc.	Rackspace CloudServer InstanceID	Instance is in "ACTIVE" state
Delete Image	SMR	Image is NOT in "DELETED" state	EC2 AMIID(M)	Operation Succeeded	Unconditional

Appendix B: PaaS Service/CSI/PSSA operation specifications for AWS Elastic Beanstalk

Cloud Service Level Operations	***Elastic Beanstalk***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List Applications	SIR	Unconditional	ElasticBeanstalk Region(M)	ElasticBeanstalk Application-Name(s)	Unconditional
List Application Environment	SIR	Unconditional	ElasticBeanstalk Region(M)	ElasticBeanstalk EnvironmentID(s)	Unconditional
Delete Application	SMR	Unconditional	ElasticBeanstalk ApplicationName(M)	Operation Succeeded	Unconditional
Delete Application Environment	SMR	Unconditional	ElasticBeanstalk EnvironmentID(M)	Operation Succeeded	Unconditional
CSI Level Operations	***Elastic Beanstalk Application Instance***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Application Environment	SIR	Unconditional	ElasticBeanstalk ApplicationName(M)	ElasticBeanstalk EnvironmentID	Unconditional
Get Application Versions	SIR	Unconditional	ElasticBeanstalk ApplicationName(M)	ElasticBeanstalk ApplicationVer-sionDescrptions	Unconditional
Create Application	SMR	Unconditional	Elastic Beanstalk ApplicationName(M), Elastic Beanstalk ApplicationDescription(O)	ElasticBeanstalk ApplicationName	Elastic Beanstalk EnvironmentStatus is in “Ready” state
Update Application	SMR	Elastic Beanstalk EnvironmentStatus is in “Ready” state	Elastic Beanstalk ApplicationName(M), Elastic Beanstalk ApplicationDescription(O)	ElasticBeanstalk ApplicationName	Elastic Beanstalk EnvironmentStatus is in “Ready” state
PSSA Level Operations	***Elastic Beanstalk Application Environment***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Application Environment VMs	SIR	Unconditional	ElasticBeanstalk EnvironmentID (M)	EC2 InstanceIDs	Unconditional
Get Application Environment LoadBalancers	SIR	Unconditional	ElasticBeanstalk EnvironmentID (M)	Elastic Load-BalancerID	Unconditional
Create Application Environment	SMR	Unconditional	ElasticBeanstalk ApplicationName(M), Elastic-Beanstalk EnvironmentDescription(O), Elastic-Beanstalk EnvironmentName(M), Elastic Bean-stalk ConfigurationOptionSettings<…>(O), etc.	ElasticBeanstalk EnvironmentID	Unconditional
Update Environment Configuration	SMR	Elastic Beanstalk EnvironmentStatus is in “Ready” state	Elastic Beanstalk ConfigurationOptionSet-tings<…>(M)	ElasticBeanstalk EnvironmentID	Elastic Beanstalk EnvironmentStatus is in “Ready” state

Appendix C: SaaS Service/CSI/PSSA operation specifications for Rackspace Cloud Load Balancers

Cloud Service Level Operations	***Rackspace Cloud Load Balancers***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List Load Balancer Instance Names	SIR	Unconditional	Rackspace Region(M)	Rackspace Cloud-LoadBalancer In-stanceNames	Unconditional
List Load Balancer Instance Addresses	SIR	Unconditional	Rackspace Region(M)	Rackspace Cloud-LoadBalancer Ad-dresses	Unconditional
Create Load Balancer Instance	SMR	< Rackspace LoadBalancer Absolute Limits, i.e. 25	Rackspace Region(M), LoadBalancer-Name(M), LoadBalancerPort(M), Rack-space CloudServer(O),Rackspace CloudLoadBalancer ExternalNode(O), Rackspace VirtualIP(M) , etc.	Rackspace Cloud-LoadBalancer In-stanceID	Load Balancer In-stance is in "ACTIVE" state
Delete Load Balancer	SMR	Load Balancer is NOT in "UPDAT-ING" state	Rackspace CloudLoadBalancer In-stanceID(M)	Operation Succeeded	Load Balancer In-stance is in "ACTIVE" state
CSI Level Operations	***Rackspace Cloud Load Balancer Instance***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Load Balancing Algorithm	SIR	Load Balancer is in "ACTIVE" state	Rackspace Cloud LoadBalancer InstanceID(M)	Rackspace Load-BalancingAlgorithm	Unconditional
List Load Balancer Instance Nodes	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalance InstanceID(M)	Rackspace Cloud-LoadBalancer InstanceNodeID(s)	Unconditional
Edit Load Balancer In-stance Health Moni-tor	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceID(M), Rackspace CloudLoad-Balancer HealthMonitor(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
Add Load Balancer Instance Nodes	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudServer(O), Rackspace CloudLoadBalancer Exter-nalNode(O), Rackspace CloudLoad-Balancer InstanceNodePort(O), etc.	Operation Succeeded	Load Balancer is in "ACTIVE" state
PSSA Level Operations	***Rackspace Cloud Load Balancer Instance Node***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get LoadBalancer Instance Node IP	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Rackspace Cloud LoadBalancer InstanceNodeIP	Unconditional
Get LoadBalancer Instance Node Port	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Rackspace Cloud LoadBalancer InstanceNodePort	Unconditional
Edit LoadBalancer Instance Node Weight	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M), Rackspace Cloud-LoadBalancer InstanceNodeWeight(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
Delete Load Balancer Instance Node	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state