CoCoOn: Cloud Computing Ontology for IaaS Price and Performance Comparison

Qian Zhang$^{1[0000-0001-7206-4722]}$, Armin Haller$^{1[0000-0003-3425-0780]}$, and Qing Wang^{1}

Australian National University, Canberra, ACT 2601, Australia
{miranda.zhang,armin.haller,qing.wang}@anu.edu.au
https://cecs.anu.edu.au/

Abstract. In this paper, we present an OWL-based ontology, the Cloud Computing Ontology (CoCoOn), that defines concepts, features, attributes and relations to describe Cloud infrastructure services. We also present datasets that are built using CoCoOn and scripts (i.e. SPARQL template queries and web applications) that demonstrate the real-world applicability of the ontology. We also describe the design of the ontology and the architecture of related services developed with it.

Keywords: ontology · cloud-computing · semantic-web.

1 Introduction

Consumers of Cloud services often face the challenge of selecting the right services for a given use case from a large set of heterogeneous offers. For example, a 2013 survey from Burstorm[1] shows that there are over 426 Compute and Storage service providers with deployments in over 11,072 locations. This problem is further aggregated by the non-standardized naming conventions on heterogeneous types of services (CPU, Storage, Network etc.) and features (Virtualisation technology, SLA model, billing model, Cloud location, cost, etc.)

A unified model is needed as the foundation for data collection, reasoning and analytics to fulfil the goal of a smart Cloud service recommendation To this end, this paper presents our work on the Cloud Computing Ontology (CoCoOn) version 1.0.1, which consolidates Cloud computing concepts: https://w3id.org/cocoon/v1.0.1. The relevant code, data and ontology are made available online as a Github project. Figure 1 depicts the IaaS related parts of CoCoOn v1.0.1. The major additions of CoCoOn v1.0.1 compared to its previous version [18, 19] are the Cloud service pricing and QoS modelling features. The datasets presented in this paper are completely new, along with all the tools and code we used to produce the data. When CoCoOn was first developed, there were little existing domain ontologies to reuse, e.g. CoCoOn predated the development of PROV-O[2], schema.org, Unit of Measure Ontology (QUDT)[3], SSN [6] and Wikidata

[1] https://www.burstorm.com/platform/
[2] https://www.w3.org/TR/prov-o/
[3] http://qudt.org/

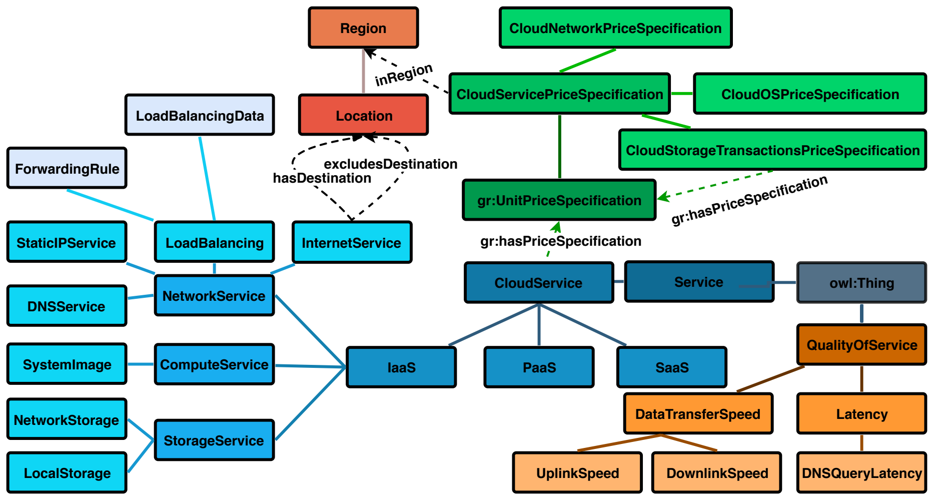

Fig. 1. CoCoOn v1.0.1: IaaS related parts

[16]. The new CoCoOn makes use of those now popular existing ontologies. Consequently, we also removed parts from the old CoCoOn ontology, which are now covered by those standardised/popular domain ontologies. Also, to improve the reusability, we added more rdfs:comments, metadata, documentation, and use cases. Because our old site on purl.org is hard to maintain and update, we moved the ontology and the documentation to GitHub. Also, we are using w3id.org as the permanent URL service instead, which should lead to better sustainability. More specifically, our model aims to facilitate the publication, discovery and comparison of IaaS, by: i) Providing a schema for constructing and executing complex queries; ii) Defining frequently referenced data as named individuals; iii) Providing a unified machine-readable specification, as opposed to provider-specific APIs and documentation. In addition to these, we demonstrate the capabilities of our model by providing real-life usage datasets. Those datasets include services from the Google Cloud and the Microsoft Azure Cloud, which is detailed in Section 4.

2 Related Work

There are existing ontologies and models that focus on Web Services [12,14] and their architectures [5] in general. Unlike these works, our model focuses on Cloud computing Infrastructure as a Service (IaaS), i.e. its features and price models.

Previously, Parra-Royon and J.M. Benítez[4] have developed two small Cloud ontologies. The set of concepts and features they cover are limited and, as a result, their examples are limited to some simple cases. For instance, the examples

[4] http://cookingbigdata.com/linkeddata/dmservices

presented in Section 3.4.1 cannot be modelled with their ontologies. Furthermore, the main reason why we did not use their “ccpricing”, “ccinstances”, and “ccregions” ontologies is because they used global scope constraints (i.e. rdfs:domain and rdfs:range) on most (if not all) of the classes and object properties, which we believe are too restrictive and can cause unintended inferences.

K. Boukadi et al. have developed a Cloud Service Description Ontology (CSO) [1], primarily for the modelling of Cloud service brokerage. Their price model is rather simple and cannot model real-world scenarios. Their model and data are also not available online anymore to be evaluated further or to be reused in other contexts.

In the mOSAIC project [13], researchers proposed an OWL ontology for Cloud services negotiation (i.e. between customers and providers) and composition (i.e. by an administrator). Their ontology is different in scope to ours.

K. Joshi et al. have developed an OWL Ontology for the Lifecycle of IT Services in the Cloud [9]. This ontology provides models for the steps involved in the phases of discovery, negotiation, composition, and consumption of Cloud services. The modelling of Cloud service features is minimal, and their link to an example of a storage service[5] is no longer accessible.

In the area of Quality of Service (QoS) modelling, some papers have proposed QoS ontologies (i.e. QoSOnt [2] and OWL-QoS [20]). However, they did not publish the actual specifications, and only figures/graphs were given. In this paper, we provide formal modelling of QoS parameters and make it readily available for general use (see Section 3.5).

Overall, all the models above have a different scope compared to our ontology. Our focus is on modelling concepts, features, attributes and relations of Cloud infrastructure services. We do not consider models for orchestration [9,13] nor brokerage processes [1] in this paper. Nonetheless, our ontology could be extended in this regard using the models proposed in these works mentioned above. Furthermore, we have also developed tools for automatically adding semantics to information from providers' APIs. We have used existing ontologies whenever fits, such as QUDT for defining price with currency values. For the full list of ontologies we have referenced, see the online documentation.[6]

3 Concepts and Design of CoCoOn v1.0.1

3.1 New Features

In our previous work [17, 18], we proposed a simpler model describing concepts of Cloud infrastructure services (IaaS).[7] In this paper we have significantly extended the capabilities of our initial model, i.e. changes have been made to

[5] https://www.csee.umbc.edu/~kjoshi1/storage_ontology.owl

[6] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/vocabularies.md

[7] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/revision_history.md

classes, properties, relationships and axioms, with a strong focus on flexibility and extensibility.

In Section 3.3, we describe the syntax, semantics, design and formalisation of CoCoOn v1.0.1, and the rationale behind such design, and some usages. In Section 4, we illustrate tools for mapping CoCoOn v1.0.1 to Google Cloud and the Microsoft Azure Cloud services. These tools demonstrate the usability and strength of the ontology we developed.

3.2 Design Rationale

The classes and properties are arranged according to subsumption hierarchies, which represent the skeleton of the model and establish the basic relationships between the components. Following the principle of minimal commitment [3], we use guarded restrictions (i.e. owl:someValuesFrom) instead of domain range restrictions (rdfs:domain, rdfs:range). As such, the domain and ranges are more permissive, keeping the model more flexible and extensible. We also use qualified cardinality restrictions (e.g., exactly, owl:qualifiedCardinality; max, owl:maxQualifiedCardinality) when there is a known cardinality restriction.

Most building blocks of IaaS services naturally correspond to OWL2 classes (e.g., cocoon:CloudService, cocoon:ComputeService, and cocoon:StorageService), object properties (e.g., cocoon:hasMemory, cocoon:hasStorage, and cocoon:inRegion) and data properties (e.g., cocoon:numberOfCores). The more challenging part is to capture constraints posed by the possible combination of services in IaaS in the models using ontological axioms. We next describe how this can be accomplished using a combination of OWL 2 axioms and integrity constraints.

We use Turtle syntax throughout our examples, and use Manchester OWL Syntax when explaining the ontology specifications.

3.3 Cloud Service

The class cocoon:CloudService is the main class hosting our Cloud feature vocabularies. We define a top level class cocoon:Service to be its parent, and make it the union of schema:Service and sosa:FeatureOfInterest. So our Cloud service definitions are compatible with the schema.org vocabulary [4] and the SOSA ontology [8] from which we reuse terms.

Cloud services are usually classified into three categories: cocoon:IaaS, cocoon:PaaS and cocoon:SaaS. Some examples of cocoon:SaaS are *database as a service*, *machine learning as a service*, Google Cloud Composer, etc. Some examples of cocoon:PaaS are the Google App Engine, Heroku, etc.

We use gr:UnitPriceSpecification and its associated object property gr:hasPriceSpecification to model price (see Section 3.4.1 for more details about price specification). Existential quantifiers (i.e., some, owl:someValuesFrom) are used on gr:hasPriceSpecification.

Note that, although some is the same as min 1, it is not the same as database integrity constraints. We can still define valid Cloud services individuals without a price specification. Under the open world assumption, missing information is just not known but may exist, whereas, in databases (closed world assumption), absence of information often assumes that information does not exist. This open world assumption serves us well because we cannot guarantee that every service will have a price specification. There are services available upon requests, but the price is negotiated later. For example, we may want to specify that secure data centres for governmental use are available, but detailed price information is probably not disclosed publicly.

We assume each service can belong to exactly one provider. A qualified cardinality restriction exactly (owl:qualifiedCardinality) is used to define this type of assumption. We reuse gr:BusinessEntity to define a provider (see Section 3.7 for more details).

Infrastructure as a Service can be classified into 3 categories: cocoon:ComputeService (see Section 3.3.1), cocoon:StorageService (see Section 3.3.2), and cocoon:NetworkService (see Section 3.3.3).

3.3.1 Compute Service The number of cores available on a virtual machine (VM) is defined by the data property cocoon:numberOfCores. Because it can have a non-integer value, we define its datatype as xsd:decimal. For Google Cloud, cores and vCPU refer to the same thing. The performance power of the CPU can be described by cocoon:hasCPUcapacity. The memory size of a VM is specified by cocoon:hasMemory.

The cocoon:LocalStorage available on a VM can be specified with cocoon:hasStorage. We use an existential quantification (i.e. some) on this property, so that it is possible to define more cocoon:NetworkStorage later. Google has a limit for the maximum number of disks that can be attached to a VM, which we model with the object property cocoon:hasMaxNumberOfDisks. Additionally, Google also has a limit for the maximum total disk size that can be attached to a VM, which is modelled with cocoon:hasMaxStorageSize.

We use schema:TypeAndQuantityNode to describe the quantity of things. So value, unit, and type of an object can all be captured (see Section 3.7 for more details).

Note that cocoon:ComputeService also inherits properties from its super classes, e.g. the following property is inherited from cocoon:CloudService:

gr:hasPriceSpecification some gr:UnitPriceSpecification

There are data access fees on local disks of the Azure VM.[8] To model this we use gr:hasPriceSpecification max 1 cocoon:StorageTransactionsPriceSpecification. For a short example of cocoon:ComputeService, see Listing 1.1.

Listing 1.1. Virtual Machine

```
@prefix schema: <https://schema.org/> .
@prefix unit: <http://qudt.org/vocab/unit#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

[8] https://www.rhipe.com/azure-storage-transactions/

```
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix cocoon: <https://w3id.org/cocoon/v1.0.1#> .
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<2019-02-12/ComputeService/Gcloud/CP-COMPUTEENGINE-VMIMAGE-N1-HIGHCPU-96-PREEMPTIBLE>
       a cocoon:ComputeService ;
       rdfs:label "CP-COMPUTEENGINE-VMIMAGE-N1-HIGHCPU-96-PREEMPTIBLE" ;
       gr:hasPriceSpecification [ a cocoon:CloudServicePriceSpecification ;
                                 gr:hasCurrency "USD" ;
                                 gr:hasCurrencyValue 0.72 ;
                                 gr:hasUnitOfMeasurement unit:Hour ;
                                 cocoon:inRegion <Region/Gcloud/us-east1>
                               ] ;
       cocoon:hasMemory [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood 86.4 ;
                                 schema:unitCode cocoon:GB
                               ] ;
       cocoon:hasProvider cocoon:Gcloud ;
       cocoon:numberOfCores "96"^^xsd:decimal ;
       schema:dateModified "2019-02-12"^^xsd:date .
```

3.3.2 Storage Service Two subclasses for cocoon:StorageService have been defined: cocoon:LocalStorage and cocoon:NetworkStorage.

On the Azure Cloud, snapshot options are available for storage, which is modelled with the object property cocoon:canHaveSnapshot. This information is manually interpreted from the documentation.[9] There are also caps on input/output operations per sec (IOPS) and throughput, which are modeled with cocoon:hasStorageIOMax and cocoon:hasStorageThroughputMax. We have also defined corresponding units, which is explained in Section 3.7.

In Listing 1.2, we show a cocoon:NetworkStorage service from cocoon:Azure, which is a Cloud provider we have pre-defined as a named instance. More details on its corresponding storage transaction prices can be found in Section 3.4.3.

Next, an example of the Azure provisional Ultra SSD storage service is presented. It has configurable IOPS and throughput. Prices are based on provisioned storage size, IOPS and throughput. There is also a reservation charge imposed if you enable Ultra SSD capability on the VM without connecting an Ultra SSD disk, whose rate is provisioned at per vcpu/hour.

Listing 1.2. Storage

```
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<2019-03-07/NetworkStorage/Azure/premiumssd-p30>
       a cocoon:NetworkStorage ;
       rdfs:label "premiumssd-p30" ;
       gr:hasPriceSpecification <CloudStorageTransactionsPriceSpecification/Azure/
            ↪ managed_disk/transactions-ssd> ;
       gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                      gr:hasCurrency "USD" ;
                                      gr:hasCurrencyValue 0.13200195133686066 ;
                                      gr:hasUnitOfMeasurement cocoon:GBPerMonth ;
                                      cocoon:inRegion <Region/Azure/australia-east>
                                    ] ;
       cocoon:canHaveSnapshot <NetworkStorage/Azure/standardssd-snapshot> , <NetworkStorage/
            ↪ Azure/standardhdd-snapshot-zrs> , <NetworkStorage/Azure/premiumssd-snapshot>
            ↪ , </NetworkStorage/Azure/standardhdd-snapshot-lrs> ;
       cocoon:hasProvider cocoon:Azure ;
```

[9] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/azure/storage.md#disk-snapshots

```
        cocoon:hasStorageIOMax [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood "5000"^^xsd:nonNegativeInteger ;
                                 schema:unitCode cocoon:IOPs
                                   ] ;
        cocoon:hasStorageSize [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood "1024"^^xsd:nonNegativeInteger ;
                                 schema:unitCode cocoon:GB
                                   ] ;
        cocoon:hasStorageThroughputMax [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood "200"^^xsd:nonNegativeInteger ;
                                 schema:unitCode unit:MegabitsPerSecond
                                   ].

<2019-03-07/NetworkStorage/Azure/ultrassd>
        a cocoon:NetworkStorage ;
        rdfs:label "ultrassd" ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                rdfs:label "vcpu" ;
                                gr:hasCurrency "USD" ;
                                gr:hasCurrencyValue 0.003 ;
                                gr:hasUnitOfMeasurement cocoon:VcpuPerHour ;
                                cocoon:inRegion <Region/Azure/us-east-2>
                              ] ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                rdfs:label "throughput" ;
                                gr:hasCurrency "USD" ;
                                gr:hasCurrencyValue 0.000685 ;
                                gr:hasUnitOfMeasurement cocoon:MegabitsPerSecondPerHour ;
                                cocoon:inRegion <Region/Azure/us-east-2>
                              ] ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                rdfs:label "stored" ;
                                gr:hasCurrency "USD" ;
                                gr:hasCurrencyValue 0.000082 ;
                                gr:hasUnitOfMeasurement cocoon:GBPerHour ;
                                cocoon:inRegion <Region/Azure/us-east-2>
                              ] ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                rdfs:label "iops" ;
                                gr:hasCurrency "USD" ;
                                gr:hasCurrencyValue 0.000034 ;
                                gr:hasUnitOfMeasurement cocoon:IOPsPerHour ;
                                cocoon:inRegion <Region/Azure/us-east-2>
                              ] .
```

3.3.3 Network Service We classify network services into the following categories: cocoon:InternetService, cocoon:LoadBalancing, cocoon:StaticIPService and cocoon:DNSService.

Internet Service There is generally no charge to ingress cocoon:InternetService, unless there is a load balancer used. We use the cocoon:hasDirection object property to indicate the direction of traffic. A class cocoon:TrafficDirection is also defined with two disjoint subclasses, cocoon:Egress and cocoon:Ingress. Those can be used to indicate the direction of traffic.

Internet egress rates are based on usage and destination. For example, Google Cloud has three destination categories[10]: Australia, China (excluding Hong Kong) and Worldwide (excluding China and Australia, but including

[10] Effective until end of June 2019, when this paper has been submitted, after that new pricing takes effect based on not only the destination but also the sources.

Hong Kong). In this case, the object properties cocoon:hasDestination and cocoon:excludesDestination can be used to specify destination ranges. Because traffic destinations are not constrained by Cloud service regions, cocoon:Location is used, which has more explanations in Section 3.6.

The internet egress traffic rates can be modelled by cocoon:CloudNetworkPriceSpecification. For more details, see Section 3.4.4.[11]

Load Balancing Both hardware and software-based load balancing solutions exist. Here we consider load balancing as a hardware feature unless it is known otherwise. We create a class cocoon:LoadBalancing to represent such a service. It is further broken down into two subclasses: cocoon:LoadBalancingData and cocoon:ForwardingRule.

Ingress data processed by a load balancer is charged (per GB) based on its region. Listing 1.3 models such cases with cocoon:LoadBalancingData.

Listing 1.3. Load Balancing Data Price Specification

```
@base <https://w3id.org/cocoon/data/v1.0.1/2019-02-12/> .
<LoadBalancingData/Gcloud>
       a cocoon:LoadBalancingData ;
       gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                 gr:hasCurrency "USD" ;
                                 gr:hasCurrencyValue 0.008 ;
                                 gr:hasUnitOfMeasurement cocoon:GB ;
                                 cocoon:inRegion <Region/Gcloud/us>
                               ] ;
       cocoon:hasDirection cocoon:Ingress ;
       cocoon:hasProvider cocoon:Gcloud ;
       schema:dateModified "2019-02-12"^^xsd:date .
```

Forwarding rules that are created for load balancing are also charged on an hourly base, regardless of how many forwards. This can be modelled by cocoon:ForwardingRule and cocoon:CloudNetworkPriceSpecification.[12]

Static IP Address The IP address of a VM instance usually is not guaranteed to stay the same between reboots/resets. So you may want to reserve a static external IP address for your customers or users to have reliable access. It can be modelled with cocoon:StaticIPService and cocoon:CloudServicePriceSpecification.

3.4 Cloud Service Price

For price modelling, we extend the GoodRelations vocabulary [7]. GoodRelations is a Web Ontology Language-compliant ontology for Semantic Web online data, dealing with business-related goods and services. In November 2012, it was integrated into the Schema.org ontology.

[11] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#internet-service

[12] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#forwarding-rule

3.4.1 Cloud Service Price Specification We define cocoon:CloudServicePriceSpecification as a subclass of gr:UnitPriceSpecification. As one service can be offered in multiple regions, we extend our specialized class with: cocoon:inRegion some cocoon:Region. For more details on region, see Section 3.6.

In GoodRelations, there is a gr:hasCurrencyValue property taking a xsd:float as range. However, floats can introduce cumulative rounding errors. So we extend the existing class to allow xsd:decimal, which can represent exact monetary values: cocoon:hasCurrencyValue exactly 1 xsd:decimal.[13] For more usages, see Section 3.3.1.

We also define specialized subclasses to handle the following scenarios: price of a VM image (cocoon:CloudOSPriceSpecification), price of storage transactions (cocoon:CloudStorageTransactionsPriceSpecification), and price of network services (cocoon:CloudNetworkPriceSpecification). These sub-classes are owl:disjointWith each other. Because each case has very different requirements, it is clearer to model them with different subclasses rather than define all properties in the base class cocoon:CloudServicePriceSpecification.

3.4.2 Price of Virtual Machine Images Under the class cocoon:CloudOSPriceSpecification, the data property cocoon:chargedPerCore specifies if the price is charged per core. For instance, Windows Server images on some machine types from Google Cloud are charged based on the number of CPUs available, i.e., n1-standard-4, n1-highcpu-4, and n1-highmem-4 are machine-types with four vCPUs, and are charged at \$0.16 USD/hour (4 × \$0.04 USD/hour).

The data property cocoon:forCoresMoreThan is used to describe a price for machines with more than the specified number of cores. Similarly, cocoon:forCoresLessEqual is used to describe a price for machines with less than or equal to the specified number of cores. They can be used together to quantify a range. Listing 1.4 presents an example for OS Price Specification.

Listing 1.4. OS Price Specification

```
@base <https://w3id.org/cocoon/data/v1.0.1/2019-02-12/> .
<SystemImage/Gcloud/suse-sap>
       a cocoon:SystemImage ;
       rdfs:label "suse-sap" ;
       gr:hasPriceSpecification [ a cocoon:CloudOSPriceSpecification ;
                                 gr:hasCurrency "USD" ;
                                 gr:hasCurrencyValue 0.41 ;
                                 cocoon:chargedPerCore false ;
                                 cocoon:forCoresMoreThan "4"^^xsd:decimal
                               ] ;
       gr:hasPriceSpecification [ a cocoon:CloudOSPriceSpecification ;
                                 gr:hasCurrency "USD" ;
                                 gr:hasCurrencyValue 0.34 ;
                                 cocoon:chargedPerCore false ;
                                 cocoon:forCoresLessEqual "4"^^xsd:decimal ;
                                 cocoon:forCoresMoreThan "2"^^xsd:decimal
```

[13] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#cloud-service-price-specification

```
                                ] ;
    gr:hasPriceSpecification [ a cocoon:CloudOSPriceSpecification ;
                                gr:hasCurrency "USD" ;
                                gr:hasCurrencyValue 0.17 ;
                                cocoon:chargedPerCore false ;
                                cocoon:forCoresLessEqual "2"^^xsd:decimal
                                ] .
```

3.4.3 Price of Storage Transactions

For storage transactions, we use the class cocoon:CloudStorageTransactionsPriceSpecification to define the price. There are different prices in different regions, but there is a common transaction price specification for a group of cloud storage offers.[14]

3.4.4 Price of Network Services

cocoon:CloudNetworkPriceSpecification can be used to model network services prices, including internet egress traffic and load balancing forwarding rules.

For instance, there are three (monthly) usage tiers for Google Internet egress traffic price: 0-1 TB, 1-10 TB and 10+ TB. Properties cocoon:forUsageLessEqual and cocoon:forUsageMoreThan can be used to specify the upper/lower usage limits. We combine this with schema:TypeAndQuantityNode to define the values with their units.

There are also some special rates, e.g., for Google Cloud Internet Traffic: Egress between zones in the same region (per GB) is 0.01; egress between regions within the US (per GB) is 0.01; egress to Google products (such as YouTube, Maps, and Drive), whether from a VM in GCP with an external or internal IP address is no charge. The property cocoon:specialRateType can be used to model those situations. See an online example for price of Google internet egress between zones in the same region.[15]

3.5 Cloud Service Performance

We use terms from a number of ontologies when modeling QoS, such as SSN [6] and SOSA [8]. The Semantic Sensor Network (SSN) ontology is an ontology for describing sensors and their observations, involved procedures, studied features of interest, samples, and observed properties, as well as actuators. SSN includes a lightweight but self-contained core ontology called SOSA (Sensor, Observation, Sample, and Actuator) for its elementary classes and properties. "SSN System" contains the terms defined for system capabilities, operating ranges, and survival ranges.

[14] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#cloud-storage-transactions-price-specification

[15] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#cloud-network-price-specification

3.5.1 Quality Of Service Property QoS parameters are grouped under cocoon:QualityOfService. We define cocoon:QualityOfService to be an equivalent class of ssn-system:SystemProperty. Then we extend it with the subclass cocoon:DataTransferSpeed.

Data Transfer Speed cocoon:DataTransferSpeed is measured multiple times with different file sizes, for both the uplink and downlink, which are represented by subclasses cocoon:DownlinkSpeed and cocoon:UplinkSpeed. See an example online.[16]

Latency There is an existing ssn-system:Latency class, which we can use. We extend this class with a specialized subclass cocoon:DNSQueryLatency, which is the latency for completing the DNS query. The term latency is most commonly referred to as the round-trip delay time, which is the one-way latency for the request to travel from a source to a destination plus the one-way latency for the response to travel back.

3.5.2 Measurement QoS measurements are modeled with cocoon:Measurement, which is an equivalent class to sosa:Observation. The cocoon:Measurement can use sosa:hasFeatureOfInterest to specify which feature it measures. Since cocoon:Service is equivalent to sosa:FeatureOfInterest, all its subclasses can be used to describe features, and we have some examples can be viewed online.[17]

3.5.3 Device We extend sosa:Sensor with a subclass cocoon:Device to describe computers used to measure QoS. Listing 1.5 shows an example for device.

Listing 1.5. Device

```
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<Device/150.203.213.249/lat=-35.271475/long=149.121434>
       a cocoon:Device ;
       rdfs:comment "The computer used to conduct the tests, belongs to Australian National
            ↪ University College of Engineering & Computer Science."@en ;
       rdfs:label "CECS-030929"@en ;
       cocoon:inPhysicalLocation [ a schema:Place ;
                                 schema:geo [ a schema:GeoCoordinates ;
                                             schema:address "Hanna Neumann Building #145,
                                                  ↪ Science Road, Canberra ACT 2601" ;
                                             schema:latitude -35.271475 ;
                                             schema:longitude 149.121434
                                            ]
                               ] ;
       cocoon:ipv4 "150.203.213.249" .
```

[16] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#downlink-speed

[17] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#measurement

3.6 Location and Region

cocoon:Location is a permissible class that can be used to represent any kind of location, i.e. Worldwide, Australia and Hong Kong. In comparison, cocoon:Region, the subclass of cocoon:Location, is more specialized to represent known/predefined cloud service regions. We link regions from each Cloud provider to GeoNames data[18], and at the same time make it compatible with Schema.org. So we define it as the union of gn:Feature and schema:Place. If a specific location or address is known, a physical location can be set with cocoon:inPhysicalLocation. Otherwise, we only describe the approximate location with cocoon:inJurisdiction. Some regions can be in multiple jurisdictions, i.e. `nam-eur-asia1` belongs to North America, Europe, and Asia. Usually, a region cannot be in more than one physical location. Each region can also specify which cocoon:continent it is in, which provider it belongs to (with cocoon:hasProvider), and a human readable name with rdfs:label. Currently, there is a simple script written for matching a region to a gn:Feature, but it can be further optimised in future work. We have also obtained some geographic coordinates from the QoS measurements, and modelled such information with schema:geo and schema:GeoCoordinates. Some examples for Location and Region are available online.[19]

3.7 Named Individuals

We define several useful named individuals to be included in this ontology.

Unit: We define cocoon:UnitOfMeasure as an owl:equivalentClass of qudt:Unit, and then use the instances from the unit vocabulary, i.e. unit:Hour and unit:MegabitsPerSecond. We also define a number of custom units with reference to qudt:InformationEntropyUnit and qudt:DataRateUnit, i.e., cocoon:GB, cocoon:GBPerHour, cocoon:GBPerMonth, cocoon:GCEU (which is the Google Compute Engine Unit), cocoon:IOPs, cocoon:IOPsPerHour, cocoon:MegabitsPerSecondPerHour, cocoon:TB, and cocoon:VcpuPerHour.

Provider: We define providers as a gr:BusinessEntity, i.e. cocoon:Gcloud and cocoon:Azure.

Quantity and type: We define some frequently used quantities as named individuals, using schema:TypeAndQuantityNode, i.e. cocoon:1TB. This will save us from redefining each value every time it is used. Since schema:unitCode can take schema:URL, it means we can pass in any external defined units, i.e. cocoon:UnitOfMeasure.

[18] https://www.geonames.org/
[19] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#location-and-region

4 Usage Cases

CoCoOn's intended usage is illustrated in Figure 2. A possible visualisation of Azure's Compute service offers and regions is shown in Figure 3, with offers in green and regions in purple. Regions with more offers have a bigger size.

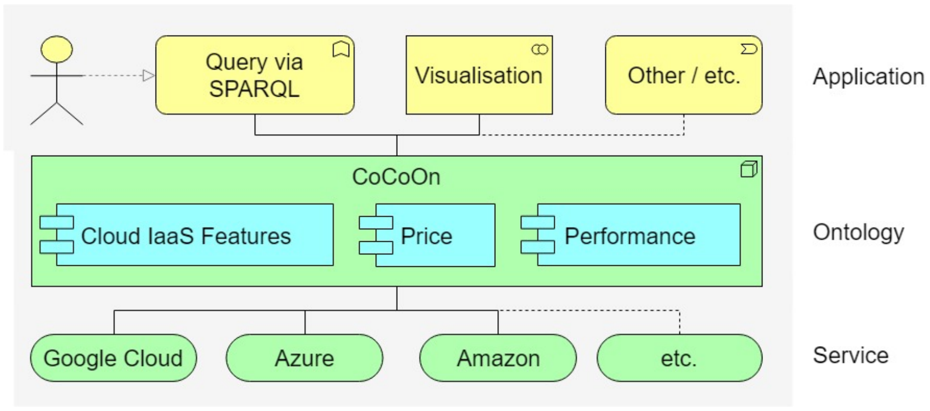

Fig. 2. CoCoOn Data Integration Workflow

4.1 Mapping Service Info to Ontology

Data can be obtained from a provider's API, either in JSON or JS format. We first clean up/transform such data with jq. Next, we map the cleaned data to our ontology.[20] Additional information is added both in `jq` and SPARQL-Generate scripts. Listing 1.6 shows a jq script example which transforms json data from Google API. In this script, we add the number of cores obtained from the vendor's documentation.

Listing 1.6. Script in jq that transforms data from Google API

```
.gcp_price_list | . |=with_entries(select(.key| contains("VMIMAGE") )) |
[ to_entries[] |
    {
        "name": .key,
        "cores":(
            if (.key|contains("F1-MICRO")) then
                0.2
            elif (.key|contains("G1-SMALL")) then
                0.5
            else .value.cores end
        ),
        "memory": .value.memory,
        "gceu": (
            if .value.gceu == "Shared CPU, not guaranteed" then
                null
```

[20] The complete process with input and output for each step is documented online

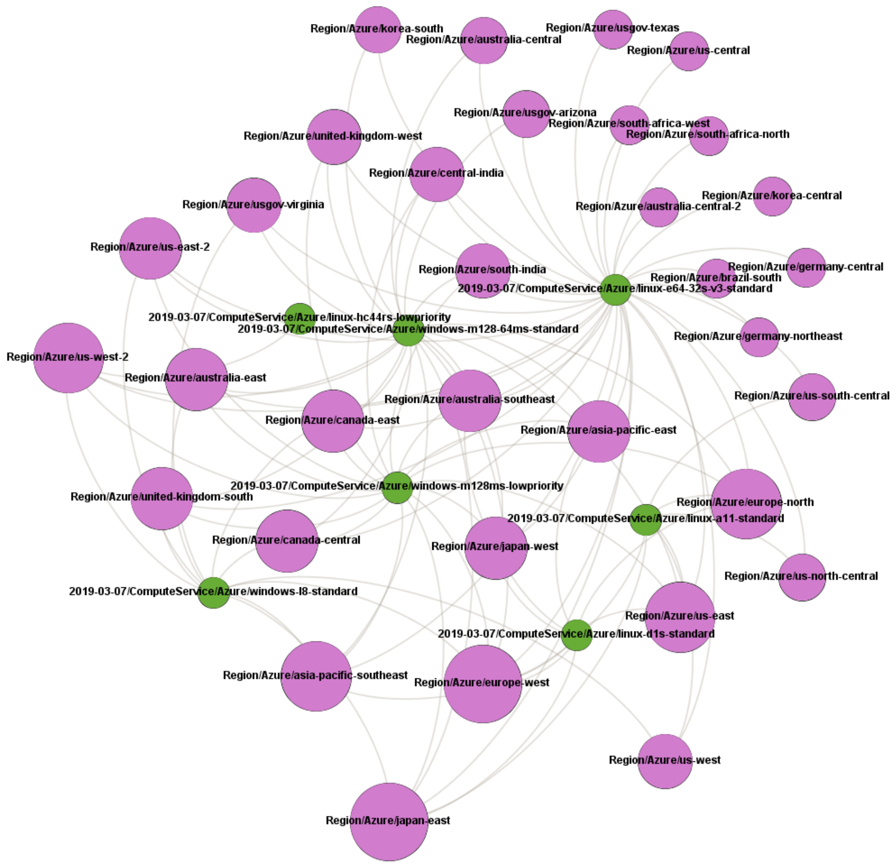
Region/Azure/korea-south
Region/Azure/australia-central
Region/Azure/usgov-texas
Region/Azure/us-central
Region/Azure/usgov-arizona
Region/Azure/south-africa-west
Region/Azure/south-africa-north
Region/Azure/united-kingdom-west
Region/Azure/central-india
Region/Azure/korea-central
Region/Azure/australia-central-2
Region/Azure/usgov-virginia
Region/Azure/us-east-2
Region/Azure/germany-central
Region/Azure/south-india
Region/Azure/brazil-south
2019-03-07/ComputeService/Azure/linux-e64-32s-v3-standard
2019-03-07/ComputeService/Azure/linux-hc44rs-lowpriority
2019-03-07/ComputeService/Azure/windows-m128-64ms-standard
Region/Azure/germany-northeast
Region/Azure/us-west-2
Region/Azure/australia-east
Region/Azure/us-south-central
Region/Azure/australia-southeast
Region/Azure/canada-east
Region/Azure/asia-pacific-east
2019-03-07/ComputeService/Azure/windows-m128ms-lowpriority
Region/Azure/united-kingdom-south
Region/Azure/europe-north
2019-03-07/ComputeService/Azure/linux-a11-standard
Region/Azure/canada-central
Region/Azure/japan-west
Region/Azure/us-north-central
2019-03-07/ComputeService/Azure/windows-l8-standard
Region/Azure/us-east
2019-03-07/ComputeService/Azure/linux-d1s-standard
Region/Azure/asia-pacific-southeast
Region/Azure/europe-west
Region/Azure/us-west
Region/Azure/japan-east

Fig. 3. Azure Services Regions

```
            else .value.gceu end
        ),
        "maxNumberOfPd": .value.maxNumberOfPd,
        "maxPdSize": .value.maxPdSize,
        "price":
         [
            .value | del(
                .cores, .memory, .gceu,
                .fixed, .maxNumberOfPd, .maxPdSize, .ssd)
            | to_entries[] | { "region": .key, "price": .value }
         ]
    }
]
```

For converting data from various sources to semantic data, we used SPARQL-Generate [11, 15] for defining the mappings. We developed many SPARQL-

Generate scripts for this process. For example, a script can map json data from Azure API to CoCoOn v1.0.1, and produce annotated RDF data.

4.2 Gathering QoS Stats

We provided live demos of QoS tests, e.g. Downlink Speed and Latency tests for Google Cloud Services. Uplink tests scripts are written in Python as selenium is required. Additional details on using cloudharmony for measuring QoS are documented online.[21]

4.3 Result Datasets

We have also made the complete datasets (132,282 triples) available at https://w3id.org/cocoon/data. It is hosted with a Linked Data Fragments Server on the Google Cloud. This server can be slow to access as we only used an always free micro instance. It is recommended to download the data and investigate with a triplestore. For example, you can run a query as shown in Listing 1.7, and the results are shown in Table 1.

Table 1. Instance counts of the classes

Class	Count
cocoon:NetworkStorage	45
cocoon:ComputeService	1021
cocoon:Region	55
cocoon:StorageService	161
cocoon:Location	5
cocoon:InternetService	6
cocoon:SystemImage	10

Listing 1.7. A SPARQL query

```
PREFIX cocoon: <https://w3id.org/cocoon
    ↪ /v1.0.1#>
PREFIX gr: <http://purl.org/
    ↪ goodrelations/v1#>
SELECT ?cls (COUNT(?s) AS ?count)
{
    VALUES ?cls {cocoon:ComputeService
        ↪ cocoon:SystemImage cocoon:
        ↪ StorageService cocoon:
        ↪ NetworkStorage cocoon:
        ↪ NetworkService cocoon:
        ↪ InternetService cocoon:
        ↪ Region cocoon:Location gr:
        ↪ BusinessEntity
    } ?s a ?cls
} GROUP BY ?cls
```

5 Conclusion

This work presents CoCoOn v1.0.1, which captures Cloud service characteristics, including the price and QoS of public cloud service offers. We also presented several semantic datasets developed using this ontology and a range of solutions for different use-case scenarios of our ontology and datasets.

For future work, several possible extensions can be made: More providers should be included to verify the completeness of our model further; Units can be improved with Custom Datatypes (`cdt:ucum` [10]), so composite units do not need to be defined specifically, i.e. instead of cocoon:MegabitsPerSecondPerHour, something like "MB/s/h" could be used; Improve mapping regions to Geonames dataset; and modelling various discounts.

[21] https://github.com/miranda-zhang/cloud-computing-schema/tree/master/example/cloudharmony

References

1. Boukadi, K., Rekik, M., Ben-Abdallah, H., Gaaloul, W.: Cloud service description ontology : Construction, evaluation and interrogation. In: Cloud Service Description Ontology : Construction , Evaluation and Interrogation (2016)
2. Dobson, G., Lock, R., Sommerville, I.: QoSOnt: a QoS Ontology for Service-Centric Systems. In: SEAA. pp. 80–87 (2005)
3. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. Int Journal of Human-Computer Studies **43**(5-6), 907–928 (1995)
4. Guha, R.V., Brickley, D., MacBeth, S.: Schema.org: Evolution of structured data on the web. Queue **13**(9), 10:10–10:37 (Nov 2015)
5. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A semantic service-oriented architecture. In: ICWS. pp. 321–328 (2005)
6. Haller, A., Janowicz, K., Cox, S.J.D., Lefrançois, M., Taylor, K., Phuoc, D.L., Lieberman, J., García-Castro, R., Atkinson, R., Stadler, C.: The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. Semantic Web **10**(1), 9–32 (2019)
7. Hepp, M.: GoodRelations: An ontology for describing products and services offers on the web. In: EKAW. pp. 329–346 (2008)
8. Janowicz, K., Haller, A., Cox, S.J., Phuoc, D.L., Lefrançois, M.: SOSA: a lightweight ontology for sensors, observations, samples, and actuators. Journal of Web Semantics **56**, 1 – 10 (2019)
9. Joshi, K.P., Yesha, Y., Finin, T.W.: Automating cloud services life cycle through semantic technologies. IEEE Transactions on Services Computing **7**, 109–122 (2014)
10. Lefrançois, M., Zimmermann, A.: The unified code for units of measure in RDF: cdt:ucum and other UCUM datatypes. In: ESWC (2018)
11. Lefrançois, M., Zimmermann, A., Bakerally, N.: A SPARQL extension for generating RDF from heterogeneous formats. In: ESWC (2017)
12. Martin, D.L., Burstein, M.H., McDermott, D.V., McIlraith, S.A., Paolucci, M., Sycara, K.P., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing semantics to web services with OWL-S. World Wide Web **10**(3), 243–277 (2007)
13. Moscato, F., Aversa, R., Martino, B.D., Fortis, T.F., Munteanu, V.I.: An analysis of mOSAIC ontology for Cloud resources annotation. FedCSIS pp. 973–980 (2011)
14. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web service modeling ontology. Applied Ontology **1**(1), 77–106 (2005)
15. SPARQL-generate, http://w3id.org/sparql-generate/
16. Wikidata, https://www.wikidata.org/wiki/Wikidata:WikiProject_Ontology
17. Zhang, M., Ranjan, R., Georgakopoulos, D., Strazdins, P., Khan, S.U., Haller, A.: Investigating Techniques for Automating the Selection of Cloud Infrastructure Services. International Journal of Next-Generation Computing **4**(3) (2013)
18. Zhang, M., Ranjan, R., Haller, A., Georgakopoulos, Dimitrios Menzel, M., Nepal, S.: An ontology-based system for Cloud infrastructure services' discovery. In: CollaborateCom (2012)
19. Zhang, M., Ranjan, R., Nepal, S., Menzel, M., Haller, A.: A declarative recommender system for cloud infrastructure services selection. In: Economics of Grids, Clouds, Systems, and Services. pp. 102–113 (2012)
20. Zhang, Y., Teng, J., He, H., Wang, Z.: On P2P-based semantic service discovery with QoS measurements for pervasive services in the universal network. Journal of Computers **9** (2014)