High Level design models for IaaS Cloud Architectures

Ales Komarek, Jakub Pavlik, and Vladimir Sobeslav

Faculty of Informatics and Management, University of Hradec Kralove,
Rokitanskeho 62, 50003 Hradec Kralove, Czech Republic
{ales.komarek,jakub.pavlik.7,vladimir.sobeslav}@uhk.cz
http://cepsos.cz

Abstract. This paper explains how ontology can be used to model various OpenStack architectures. OpenStack is the largest open source cloud computing IaaS platform. It has been gaining wide spread popularity among users as well as software and hardware vendors over past few years. It's a very flexible system that can support a wide range of virtualization scenarios at scale.
In our work we propose a formalization of OpenStack architectural model that can be automatically validated and serve suitable meta-data to configuration management tools. The OWL-DL based ontology defines service components and their relations and provides foundation for further reasoning. Provided models can support simple all-in-one architecture as as well as large architectures with service components in High Availability setup.

Keywords: OpenStack, SOA, MDA, etc.

1 Introduction

Nowadays IT infrastructure is the key component for almost every organization across different domains, but it must be maximally effective with the lowest investment and operating costs. For this reason, cloud Infrastructure as a Service (IaaS) is gradually being accepted as the right solution regardless hosting as private, public or hybrid form. Lots of key vendors had tried to develop own solutions for IaaS clouds during several years ago, but infrastructure is being too complex and heterogeneous. Different vendors means different technology, which caused vendor lock-in and limitations in migrations for future growth. In addition, every organization has different requirements for hardware, software and its use.

Based on the idea of openness, scalability and standardization of IaaS cloud platform NASA together with RackSpace founded in 2010 project called OpenStack, which is a free and open source cloud operating system that controls large pools of compute, storage, and networking resources across the datacenter. It is the largest open-source cloud computing platform today [1]. Community is driven by industry vendors as IBM, Hewlett-Packard, Intel, Cisco, Juniper,

Red Hat, VMWare, EMC, Mirantis, Cannonical, etc. In terms of numbers the OpenStack community contains about 2,292 companies, 8,066 individual members and 89,156 code contributors from 130 different countries [2]. These figures confirm that OpenStack belongs to the largest solution for IaaS cloud.

OpenStack is a modular, scalable system, which can run on a single personal computer or on the hundreds of thousands servers as e.g. CERN [3] or PayPal [4].

Lots of vendors and wide community mean lots of ways how OpenStack can be deployed. Each vendor tries to extend core functions and write new service backends to fit their business goals. The actual system consists of many modules and components designed with plugin architecture that allows custom implementations for various service backends. These components can be combined and configured to match available software and hardware resources and real use-case needs.

Each implementation has its own component combination and use some form of configuration management tool to enforce the service states on designated servers and possibly other network components. These tools require data that covers configuration of all components. However, there is not best practise or recommendations how to build suitable OpenStack cloud for different use cases. Detecting component inconsistencies manually is painful and time consuming process. Companies need standardization and validation process for their specific infrastructure requirements, which can help them automate whole implementation, operating and future upgrades of controlling software or physical hardware.

Our project goals are to find a solution to the following issues:

1. Propose high level architecture model definition (Logical model)
2. Implement service that transforms architecture model to solution level model for configuration management tools (physical realization)
3. Provide way how to define and validate architecture based on available hardware resources and target use case.
4. Automate the whole process from high level modelling to actual enforcement on targetted resources.

This paper is focused on designing and creation of high level architecture model, where we propose a formalization of OpenStack service architecture model, based on the approaches developed in classic knowledge representation domain, especially Service-Oriented Architecture by OpenGroup. Component definition is encoded in an ontology using the standard OWL-DL language, which enables sharing of knowledge about configurations across various systems. Reasoning can be used on the specification to automate validation of configuration changes.

When dealing with hundreds of components with thousands of properties and relations, keeping track of changes throughout its life cycle is very challenging. Current approaches are ad hoc, even OpenStack Fuel (Mirantis OpenStack deployment tool [5]) has severe limitations, there exists no standard for specifying common OpenStack architectural model. The question how to convert the

proposed OWL-DL schema to metadata format that configuration management tools can process is discussed. We are working on external node classification service that uses graph database to serialize the OWL ontology with REST API that configuration management tools can use as metadata provider. This can streamline the process of adopting new services and service backends in predictable way.

2 Service Architecture Models

Infrastructure as a Service platforms at it's core controls various virtualization interfaces and services and allows to launch a new virtual server from given disk image at chosen host server, connect it to provided physical or virtual network and add block storage device to it. The OpenStack platform provides services that address needed to provide the very basic compute, network and storage services. The identity provider and image store provide further services needed to provide the basic sevices.

Nebula was the predecessor of OpenStack and was superseded because it did not scale well. The services withing OpenStack communicate through 3 various communication channels

TCP/SQL - Services store it's state in SQL datastore

AMQP - Service internally communicate over asychronous communication bus which .... Compute service calls network service

HTTP - All services expose REST APIs that allow higher level integration, control ...

2.1 Architectural Level

OpenStack is complete Infrastructure as a Service platform. It allows to create virtual servers on virtual networks using virtual block devices.

Further versions of OpenStack introduce more sophisticated services that use basic services to Data processsing

These core services are followed by growing number of services covering for example telemetry, orchestration or data processing. All services within OpenStack architecture have pluggable backends. This allows vendors to develop plugin for their resources, that can be accessed and managed by the OpenStack API.

Following Figure shows the basic configuration of OpenStack in Icehouse version.

Klasicky logicky model openstack architektury

2) OpenStack architecture moduls

Database

Message queue

Time service

Identity - Keystone

Image - Glance

Compute - Nova

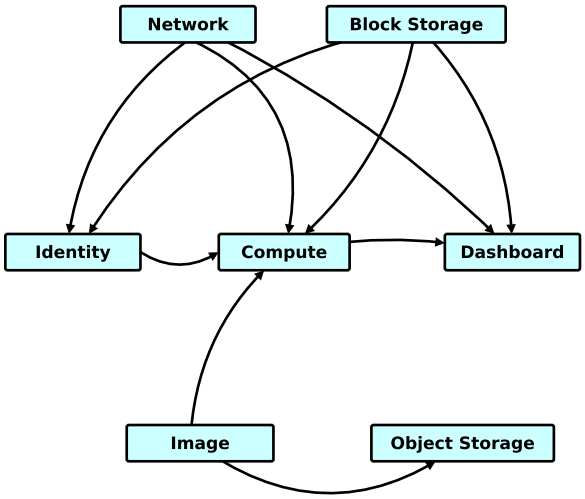

Fig. 1. Logical Model of Havana OpenStack

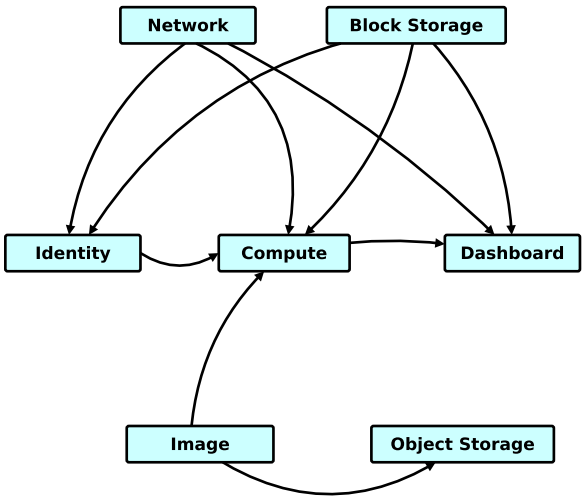

Fig. 2. Logical Model of Icehouse OpenStack service achitecture

Network - Neutron
Volume - Cinder

2.2 IaaS Controller Support Services

This section covers the
Real implementations of Architectural models

High Availability Services Cluster software - corosync/pacemaker - keepalived

Communication Services RPC
- rabbitmq
- qpid
- 0mq

Database Services Database
- mysql/galera
- postgressql/xtradb

Time Synchronization Services time
- ntp

2.3 Iaas Controller Core Services

APIs
Pluggable backends

Identity Service Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family.
- sql
- ldap

Image Service OpenStack Image service (code-named Glance) provides services for virtual disk images. Compute service uses image service to get the starting image of the virtual server.

OpenStack Image service handles variety of disk image formats, including Raw, Machine (kernel/ramdisk outside of image, also known as AMI), VHD (Hyper-V), VDI (VirtualBox), and qcow2 (Qemu/KVM).
- dir
- swift
- s3

Compute Service OpenStack Compute service (code-name Nova) is designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud-computing platform. It provides the software required to orchestrate compute instances through using libvirt or or other virtualization client libraries. OpenStack Compute service is both hardware and hypervisor agnostic, currently supporting a variety of standard hardware configurations and major hypervisors.
- kvm

- qemu
- docker
- hyper-v
- vsphere

Network Service Neutron is an OpenStack networking project focused on delivering networking as a service. Neutron has replaced the original networking application program interface (API) in OpenStack. Neutron is designed to address deficiencies in baked-in networking technology found in cloud environments, as well as the lack of tenant control (in multi-tenant environments) over the network topology and addressing, which makes it hard to deploy advanced networking services.

- flat networking
- ovs-gre/vxlan
- sdn
- opencontail
- nsx

Volume Service Cinder

- lvms
- sans

Rzn zpsoby nasazen ukzky reln architektury - promapovat ve 4 na ontologii

2.4 Use Cases

Why we choose different openstack setups

Locality 1 At CEPSOS laboratory have deployed

- 20 hypervisors, kvm, ovs-gre, local hdd

Locality 2 At Cloudlab in datacenter in Pisek we tested

- 4 hypervisoers, kvm, sdn-contrail, san

Locality 3 At Cloudlab in datacenter in Pisek we tested

- 3 hypervisors, ... l2 networking

3 OpenStack Deployment Options

There are many ways how to deploy OpenStack infrastructure which are more or less automated. Some of them require to fill in answer files, some configuration files. Some tools have graphiceal user interface and allow to provision entire hardware infrastructure as some just configure the services on the provisioned servers.

Model je popsanej dokumentem a nen to iteln, automatizace. Nen validita modelu. Chyby se debuguj na rovni reality.

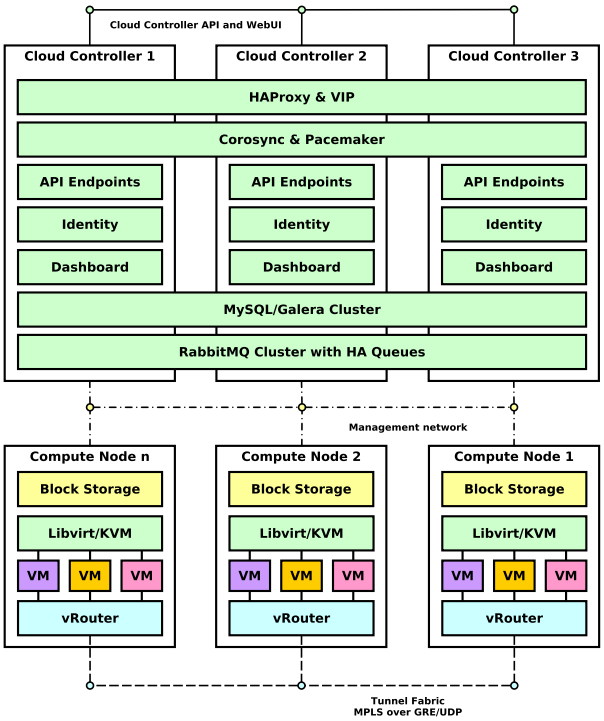

Fig. 3. Locality 1 Architecture

3.1 Development Environment Installers

For testing and developing OpenStack ...

PackStack Packstack is a utility that uses Puppet modules to deploy various parts of OpenStack on multiple pre-installed servers over SSH automatically. Currently only Fedora, Red Hat Enterprise Linux (RHEL) and compatible derivatives of both are supported.

Devstack DevStack has evolved to support a large number of configuration options and alternative platforms and support services. That evolution has grown well beyond what was originally intended and the majority of configuration combinations are rarely, if ever, tested.

3.2 Production Environment Managers

For production installations of OpenStack ...

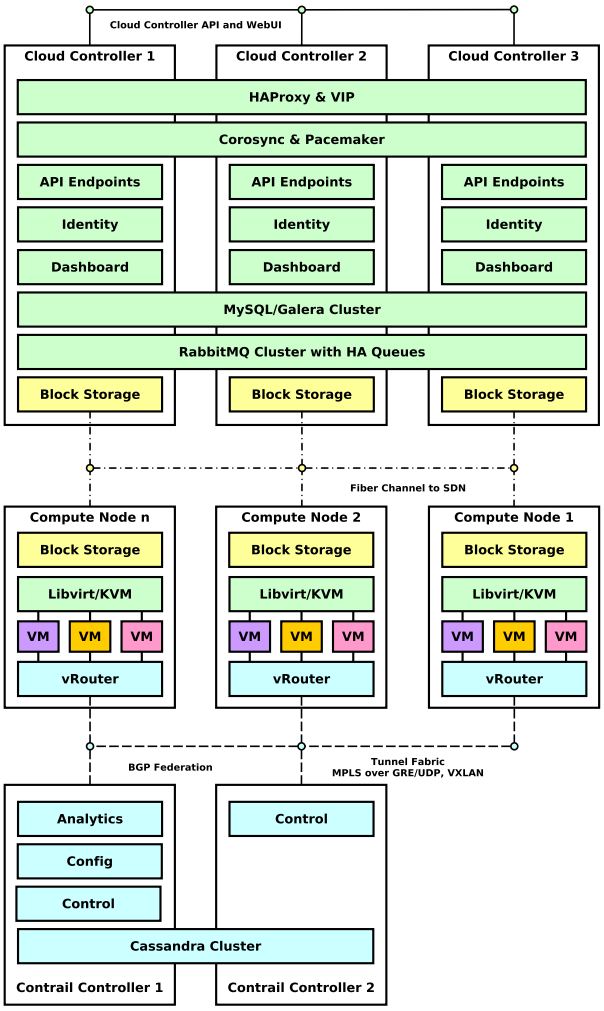

Fig. 4. Locality 2 Architecture

Fuel Fuel is an open source deployment and management tool for OpenStack. Developed as an OpenStack community effort, it provides an intuitive, GUI-driven experience for deployment and management of OpenStack, related community projects and plug-ins.

Foreman You can setup Foreman to deploy RDO. The metadata is provided in Host Groups.

3.3 Configuration Management Tools

You can install OpenStack by configuration management tool

Puppet It's already used by Fuel an Foreman

Salt Salt is another approach to install OpenStack.

4 IaaS Service Ontology

What is an otlology? An ontology is a specification of a conceptualization. According to Gruber [8] an ontology defines a set of representational primitives with which to model a domain of knowledge. These primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.

There formal definition of cloud comptuting architecture

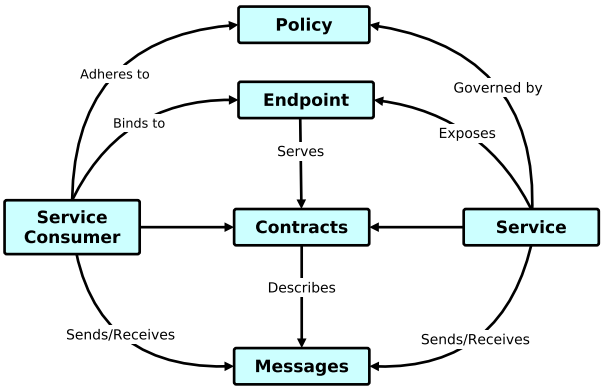

Fig. 5. SOA Sbject Property Service Interface

1. Servers (physical and virtual)
2. Core Infrastructure Services (DNS, DHCP, NTP, image management)
3. Storage (NAS and SAN)
4. Network (Routers, Switches, Firewalls, Load Balancers)
5. Facilities (Power, Cooling, Space)

4.1 Ontological Standards

The ontologies define the relations between terms, but does not prescribe exactly how they should be applied.

OWL has three increasingly expressive sub-languages: OWL-Lite, OWL-DL, and OWL-Full [OWL]. The sub-language OWL-DL provides the greatest expressiveness possible while retaining computational completeness and decidability.

Service-Oriented Architecture The SOA ontology specification was developed in order to aid understanding, and potentially be a basis for model-driven implementation of software systems

The ontology is represented in the Web Ontology Language (OWL) defined by the World-Wide Web Consortium (W3C).

The ontology contains classes and properties corresponding to the core concepts of SOA. The formal OWL definitions are supplemented by natural language descriptions of the concepts, with graphic illustrations of the relations between them, and with examples of their use. For purposes of exposition, the ontology also include.

OSLC Configuration Management

Dublin Core Metadata Initiative

Basic Formal Ontology The Basic Formal Ontology (BFO) is a formal ontological framework developed by Barry Smith and his associates that consists in a series of sub-ontologies at different levels of granularity. The ontologies are divided into two varieties: Continuant (or snapshot) ontologies, comprehending continuant entities such as three-dimensional enduring objects, and occurrent ontologies, comprehending processes conceived as extended through (or as spanning) time.

4.2 Ontology Serialization Formats

There are many ways how to serialize ontology.

In it's core ontology representation is set of graph edges connecting subject and object vertices.

The

XML Documents RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a triple). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes.

This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

Graph databases Graph databases can map OWL based ontologies very well as they have format very similar to RDF format which is standard format of any XML based graph database, just very different implementation. Graph database uses graph structures with nodes, edges, and properties to represent and store data. A graph database is any storage system that provides index-free adjacency.This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary.

je to servica, tzn overhead oproti xml filu, ale zas ma api atd ...

4.3 Plain Meta-data Serialization Formats

It's tree structure

Hierarchical Databases Subject (id) or property driven

reclass allows you to define your nodes through class inheritance, while always able to override details further up the tree (i.e. in more specific nodes). Think of classes as feature sets, as commonalities between nodes, or as tags. Add to that the ability to nest classes (multiple inheritance is allowed, well-defined, and encouraged), and you can assemble your infrastructure from smaller bits, eliminating duplication and exposing all important parameters to a single location, logically organised. And if that isnt enough, reclass lets you reference other parameters in the very hierarchy you are currently assembling.

4.4 Ontology structure

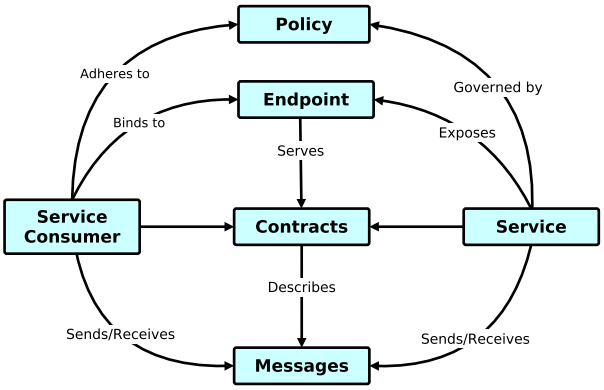

Fig. 6. SOA Sbject Property Service Interface

5 Ontology Usage

The journey to mapping high level models in ontologies was long. It was a process of describing everchanging architectures of OpenStack cloud installations over past 2 years.

At the beginning we started mapping meta-data of these models in separate files containing complete meta-data set for each server or device. The format of these files was YAML. This approach was manageable at earlier versions of OpenStack up to Grizzly version and just the basic set of OpenStack services existed.

```
service_name:
  service_role:
    data_parameter1: data_value1
    data_parameter2: data_value2
    data_parameter3:
    - data_value3a
    - data_value3b
    object_parameter:
     object1:
       data_parameter1a:
```

The next step was storing the data in hierarchical databases. The meta-data was split into components. All services can be very well described by ontology as they communicate over common message bus, serialize their state and expose services through interfaces in a standard way. Nevertheless the optimal installations consist of at least following services.

All nodes providing contain basic set of

linux.system, linux.network, linux.storage - Basic configuration of network interfaces, users, volume, mounts, etc.

ntp.client - Time synchronisation is important when using common message bus

salt.minion/pupper.agent - Configuration management client provide

collectd.client, sensu.client - Monitoring services that provide common checks and meters for other services hosted on the same servers.

Controller node

mysql.cluster, rabbitmq.cluster, haproxy.proxy, corosync.cluster -

keystone.server, glance.server, nova.server, neutron.server, cinder.server, horizon.server -

neutron.bridge -

Compute node

nova.compute, neutron.switch

```
service_name:
  service_role:
    data_parameter1: data_value1
    data_parameter2: {service_name:service_role:data_paramer1}
```

```
object_parameter:
 {otherobject:object1}
```

The Ontology can support many individual implementations at the time

5.1 Implementation details

Initial work on creating our Ontology was done in Protege, open-source ontology editor and framework for building intelligent systems.

The ontology is transformed into graph database using our python-bases service named django-ENC that can read and write ontology from OWL-DL XML files created by Protege and communicates with neo4j graph database through REST API. The graph databases are part of family of NoSQL databases and offer much better performance at any volume of data.

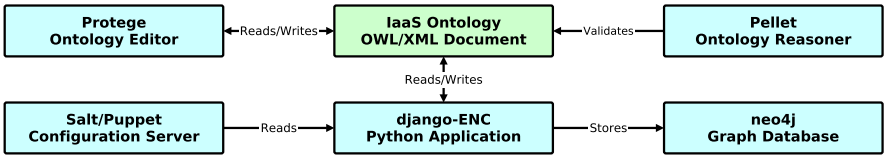

Fig. 7. Ontology Service Architecture

The django-ENC service use web framework Django to deliver web services and asynchronous task queue Celery to perform time consuming tasks like ontology assertions and synchronizations between XML and graph database. Service expose it's owns HTTP REST API that can be consumed by configuration management tools like Salt or Puppet through their External Node Classification option.

The metada passed to CM tools is valid for 1st level of Cloud computing ontology [cite].

We have successfully tested service status enforcemet of several complete OpenStack installations by SaltStack configuration management tool with metadata acquired from Ontology ENC API.

The deployment process is not yet fully automated as there is need of setting up network and storage resources manually, but the progress in both configuration management tools and network and storage will allow better automation of these components by in-place agents or access protocols like SSH in the future.

5.2 Samples of Ontology

Given use case scenario Lab1 we have 3 virtual servers providing OpenStack and other core services in high-availability mode. These servers are virtualised in common. 20 physical servers

Service components of controller1 Services located on controller server

```
<owl:Class rdf:about="#Glance">
  <owl:disjointWith>
    <owl:Class rdf:about="#"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Composition"/>
  </rdfs:subClassOf>
</owl:Class>
```

Detail of service glance.image On of the services defined on the controller node is image service Glance. Following definitiong

```
<owl:Class rdf:about="#Glance">
  <owl:disjointWith>
    <owl:Class rdf:about="#"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Composition"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Composition"/>
  </rdfs:subClassOf>
</owl:Class>
```

Detail of data property type The resource can have data property and are derived mostly from Dublin Core metadata terms.

```
<owl:Class rdf:about="#Glance">
  <owl:disjointWith>
    <owl:Class rdf:about="#"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceInterface"/>
  </owl:disjointWith>
```

```
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Composition"/>
    </rdfs:subClassOf>
  </owl:Class>
```

Detail of object property database Resources can have object property types that describe more complex relations.

```
  <owl:Class rdf:about="#Glance">
    <owl:disjointWith>
      <owl:Class rdf:about="#"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#ServiceInterface"/>
    </owl:disjointWith>
  </owl:Class>
```

6 Conclusion

We have managed to do the first step to formalize and use IaaS Service Ontology to alleviate system operators tasks. The ontology can be used to create and validate meta-data for individual OpenStack installations. The generated meta-data is used to set-up variety of services from core, maintenance to actual OpenStack services.

The ontology is

Ontologick reprezentace prosted, kter je vhodn pro agentov prosted, aby bylo mon provdt autonomn rozhodnut.

6.1 Future work

In the future we plan expand mapping ontology to network and storage resources in addition to servers with configuration tools. Ontology model is suitable for software agent processing and their rational decisions. It is possible to define processes that will maintain the configuration of real services in accordance to model systems.

References

1. *OpenStack.org, (2014). OpenStack Open Source Cloud Computing Software. [online] Available at: http://openstack.org [Accessed 12 Oct. 2014].*
2. *Stackalytics.com, (2014). Stackalytics — OpenStack community contribution in Kilo release. [online] Available at: http://stackalytics.com [Accessed 12 Oct. 2014].*
3. *Information-technology.web.cern.ch, (2014). OpenStack Information. [online] Available at: http://information-technology.web.cern.ch/book/cern-private-cloud-user-guide/openstack-information [Accessed 12 Oct. 2014].*

4. *Openstack.org, (2014). PayPal OpenStack Open Source Cloud Computing Software. [online] Available at: http://www.openstack.org/user-stories/paypal/ [Accessed 12 Oct. 2014].*
5. *Wiki.openstack.org, (2014). Fuel - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Fuel [Accessed 13 Oct. 2014].*
6. *NIST. The NIST Definition of Cloud Computing* http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
7. Ivan Ivanov, Marten van Sinderen and Boris Shishkov, editors. *Cloud Computing and Services Science* Springer Science, 978-1461423256, New York, USA, 2012
8. *Encyclopedia of Database Systems* Ling Liu and M. Tamer zsu (Eds.), Springer-Verlag, 2009.
9. *Web Ontology Language (OWL)* http://www.w3.org/2004/OWL
10. *Service-Oriented Architecture Ontology, Version 2.0* ISBN: 1-937218-50-8, The Open Group, 2014
11. *Configuration Management Resource Definitions* http://open-services.net/wiki/configuration-management/Configuration-Management-Resource-Definitions/
12. *DCMI Metadata Terms* http://dublincore.org/documents/dcmi-terms/
13. *The Open Biological and Biomedical Ontologies* http://www.obofoundry.org/
14. *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering* http://www.w3.org/2001/sw/BestPractices/SE/ODA/
15. *Pellet: OWL 2 Reasoner for Java* http://clarkparsia.com/pellet/
16. *OpenStack. Fuel Wiki* https://wiki.openstack.org/wiki/Fuel
17. *Fedora Project. OpenStack devstack* http://fedoraproject.org/wiki/OpenStack_devstack
18. *CFEngine. FCEngine 3.5 Documentation* https://cfengine.com/docs/3.5/index.html
19. *The Foreman. The Manual: Compute Resources* http://theforeman.org/manuals/1.4/index.html
20. *reclass. Recursive external node classification* http://reclass.pantsfullofunix.net/
21. *Puppet Labs. Creating Hierarchies* http://docs.puppetlabs.com/hiera/1/hierarchy.html