

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diploma/Master/Student Thesis No. 0838-006

Persistence and Discovery of Reusable Cloud Application Topologies

Hao Ding



Course of Study: INFOTECH

Examiner: Prof. Dr. Dr. h. c. Frank Leymann

Supervisor: Santiago Gómez Sáez

Commenced: 16th July, 2015

Completed: 15th January, 2016

CR-Classification: C.2.4, D.2.11, D 2.8

Abstract

Due to the benefits introduced by the Cloud computing paradigm and the increase of available Cloud services (VM- and non VM-oriented), in the last years the number of application developers strongly supporting a partial or complete migration of application component to Cloud environments has significantly increased. For example, it is possible to host the application's database off-premise (e.g. in a DBaaS solution) while keeping the remaining components (presentation or business logic components) on-premise. However, the previous application deployment is only one possible distribution alternative, and the existence of further alternatives allows the generation of a wide variety of distribution combinations. In addition, the challenges for application developers to efficiently select optimal strategy of application's deployment by considering evolving application performance with fluctuating workload has increased rapidly. How to select, configure and deploy an application optimally to satisfy functional and non-functional requirements of business and operation has been a research area in both academic and industry domains.

In this Master thesis, basing on the approaches proposed in previous work, we first conduct a research on existing approaches and technologies about how to persist, retrieve and build typed graph-based Cloud application topologies leveraging the benefits introduced and developed in graph databases and graph database technologies, respectively. Consequently, we develop the core algorithms for persisting and discovering application topologies focusing on their similar characteristics. Such conceptual models relate to the required structural aspects representing the relationship between the application topologies, their performance aspects, and their evolving workload. As a result of this thesis, a prototypical implementation of a RESTful-based framework to support discovering and building reusable viable topologies of Cloud application w.r.t. evolving functional and non-functional aspects is provided, e.g. taking into account its performance, its corresponding profile and its corresponding evolving workload.

Contents

1	Introduction	1
1.1	Problem Statement	1
1.2	Motivating Scenario	3
1.3	Definitions and Conventions	4
1.4	Outline	5
2	Fundamentals	7
2.1	Graph-based Database	7
2.2	Neo4j Graph Database System	8
2.3	Graph Transformation Theory	9
2.4	Cloud Computing	10
2.5	Cloud Application Topology	12
2.5.1	Optimal Distribution of Cloud Applications	12
2.6	TOSCA	13
2.7	OpenTOSCA	13
2.8	OpenTOSCA Winery Topology Modeling Environment	13
2.9	REST	14
3	Related Works	17
4	Concept and Specification	21
4.1	Data Modeling	21
4.1.1	Graph Database Modeling Notations	21
4.1.2	Type Graph with Inheritance Modeling	22
4.1.3	Modeling Example	24
4.2	System Requirements	29
4.2.1	Topology requirements	30
4.2.2	Workload Requirements	32
4.2.3	Performance(KPI) Requirements	32
4.3	Use Case	32
4.4	System Overview	50
5	Design	51
5.1	System Architecture	51
5.2	REST Interface Design	53
5.2.1	Resource Identification	53
5.2.2	Resource Representation	54
5.2.3	Endpoint and Action Representation	71

5.3	Core Algorithm	82
5.3.1	Viable Topology Discovery	82
5.3.2	Similar Topology Matching	83
6	Implementation	85
6.1	Implementation Environment	85
6.2	Implementation Details	86
6.2.1	RESTFul Interface	87
6.2.2	Interpreter	89
6.2.3	Data Access	92
6.2.4	Business Logic	93
7	Validation	97
7.1	Methodology	97
7.2	Basic Elements	100
7.2.1	NameSpace	100
7.2.2	Relationship Type	102
7.3	Alpha Topology	104
7.3.1	Alpha Topology	104
7.3.2	Similar Topology	106
7.4	Gamma Topology	108
7.4.1	Abstract Sub-Topology	108
7.4.2	Concrete Node	110
7.4.3	Instance Node	113
7.5	Topology Enrichments	116
7.5.1	Workload	116
7.5.2	Performance	119
7.6	Viable Distribution Topologies Discovery	123
8	Outcome and Future Work	129
	Bibliography	131

List of Figures

1.1	Number of Visitors During a Day of Site PIWIK Forum	2
1.2	Page Load Time of Facebook Over last year	2
1.3	Topology Model of MmediaWiki	3
2.1	Neo4j Architecture	9
2.2	Cloud Computing Layers	11
4.1	Graph database notation	22
4.2	Node Definition	22
4.3	Node type tree	24
4.4	γ -topology modeling	25
4.5	α -topology modeling	26
4.6	Performance Modeling	27
4.7	Workload,KPI performing and viable history	29
4.8	Use Case Diagram	33
4.9	System Overview	50
5.1	System Architecture	51
5.2	Work flow within Service Layer	52
5.3	Resource Modeling	53
6.1	Maven Module Dependency	86
6.2	Class Diagram for the Viable Topology Discovery Component	94
7.1	Validation Scenario Overview	98
7.2	Validation Sample Application	99
7.3	Request for Persisting a NameSpace	100
7.4	Response of the Request for Persisting a NameSpace	100
7.5	Retrieve one NameSpace By ID	101
7.6	Retrieving all NameSpaces	101
7.7	NameSpace in Database	102
7.8	Request for Persisting a Relationship Type	102
7.9	Response of the Request for Persisting a RelationshipType	103
7.10	Retrieve one RelationshipType By ID	103
7.11	RelationshipType in Database	103
7.12	Request and Response for Persisting an Alpha Topology	104
7.13	Retrieve one an Alpha Topology By ID	105
7.14	Retrieve all Alpha Topologies	105
7.15	Alpha Topology in Database	106

7.16 Find Similar Alpha Topology-1	106
7.17 Find Similar Alpha Topology-2	107
7.18 Request and Response for Persisting an Abstract Sub-Topology	108
7.19 Retrieve one an Abstract Sub-Topology By ID	109
7.20 Abstract Sub-Topology in Database	109
7.21 Request and Response for Persisting a Concrete Node	110
7.22 Retrieve a Concrete Node By ID	111
7.23 Concrete Nodes for Alpha Topology	111
7.24 Retrieving all Concrete Nodes	112
7.25 Concrete Nodes in Database	112
7.26 Request and Response for Persisting an Instance Node	113
7.27 Retrieve an Instance Node By ID	113
7.28 Retrieve all Instance Nodes	114
7.29 Instance Nodes in Database	115
7.30 Instance Nodes linked to a Concrete Node	115
7.31 Request and Response for Persisting a Workload	116
7.32 Retrieve a Workload By ID	117
7.33 Retrieve all Workloads	117
7.34 An Alpha Topology performs a workload	118
7.35 An alpha topology retrieves all its Workloads	118
7.36 An Alpha Topology queries its Workloads	119
7.37 Request and Response for Persisting a Performance	120
7.38 Retrieve a Performance By ID	120
7.39 Retrieve all Performances	121
7.40 An Alpha Topology performs a Performance	121
7.41 An alpha topology retrieves all its Performances	122
7.42 An Alpha Topology queries its Performances	122
7.43 Discover Viable Topologies	123
7.44 Verify the generated Topology in Winery-1	124
7.45 Verify the generated Topology in Winery-2	124
7.46 Persist a Viable Topology	125
7.47 Get all Viable Topologies of an Alpha Topology	126
7.48 Query Viable Topologies of an Alpha Topology	126
7.49 An Alpha Topology With its Viable Topologies, workloads and performances	127

List of Tables

4.1	Use Case Description: Persist an abstract sub-topology	34
4.2	Use Case Description: Retrieve abstract sub-topology	34
4.3	Use Case Description: Persist a concrete node	35
4.4	Use Case Description: Delete a concrete node	35
4.5	Use Case Description: Retrieve concrete nodes	36
4.6	Use Case Description: Retrieve instance nodes refers to a concrete node	36
4.7	Use Case Description: Persist an instance node	36
4.8	Use Case Description: Delete an instance node	37
4.9	Use Case Description: Retrieve instance nodes	37
4.10	Use Case Description: Persist Relationship Type	38
4.11	Use Case Description: Retrieve Relationship types	38
4.12	Use Case Description: Delete one Relationship type	38
4.13	Use Case Description: Find similar alpha topologies	39
4.14	Use Case Description: Discover All Viable Topologies for a given alpha topology	39
4.15	Use Case Description: Persist one viable topology for an application	40
4.16	Use Case Description: Persist one workload	40
4.17	Use Case Description: Retrieve one workload	40
4.18	Use Case Description: Retrieve all workloads in database	41
4.19	Use Case Description: Delete one workload	41
4.20	Use Case Description: Persist one performance	42
4.21	Use Case Description: Retrieve one performance	42
4.22	Use Case Description: Retrieve all performances in database	42
4.23	Use Case Description: Delete one performance	43
4.24	Use Case Description: Persist an alpha topology	43
4.25	Use Case Description: Retrieve an alpha topology	44
4.26	Use Case Description: Retrieve all alpha topologies	44
4.27	Use Case Description: Delete one alpha topology	44
4.28	Use Case Description: An alpha topology performs a performance	45
4.29	Use Case Description: Retrieve a performance of an alpha topology	45
4.30	Use Case Description: Retrieve all performances of an alpha topology	46
4.31	Use Case Description: Retrieve performances performing history of an alpha topology	46
4.32	Use Case Description: An alpha topology performs a workload	47
4.33	Use Case Description: Retrieve a workload of an alpha topology	47
4.34	Use Case Description: Retrieve all workloads of an alpha topology	48

4.35	Use Case Description: Retrieve workloads performing history of an alpha topology	48
4.36	Use Case Description: Retrieve viable topology of an alpha topology	49
4.37	Use Case Description: Retrieve all viable topologies of an alpha topology	49
5.1	Resources for Topology Persistence and Discovery System	54
5.2	Allowed operations for abstract sub-topology resource	72
5.3	Allowed operations for concrete node resource	73
5.4	Allowed operations for instance node resource	74
5.5	Allowed operations for workload resource	75
5.6	Allowed operations for performance(KPI) resource	76
5.7	Allowed operations for viable topology resource	77
5.8	Allowed operations for alpha topology resource	78
5.9	Allowed operations for alpha topology resource cont.	79
5.10	Allowed operations for relationship type resource	80
5.11	Allowed operations for namespace resource	81
6.1	Development Tools List	85

List of Listings

5.1	XML schema for persisting an Alpha Topology	54
5.2	XML schema for retrieving Alpha Topologies	55
5.3	XML schema for abstract sub-Topology	56
5.4	XML schema for retrieving viable topologies	58
5.5	XML schema for discovering similar alpha topologies	59
5.6	XML schema for performance	60
5.7	XML presentation for performing performance(KPI)	67
5.8	XML presentation for persisting workload	68
5.9	XML presentation for performing workload	68
5.10	XML schema for retrieving concrete nodes	69
5.11	XML schema for retrieving instance nodes	70
5.12	XML schema for retrieving relationship type	70
5.13	XML schema for NameSpace	71
6.1	Persist an Alpha Topology	87
6.2	Delete an Alpha Topology	88
6.3	Query Performed Workload History of Alpha Topology)	89
6.4	Create Static Unmarshaller Instance	90
6.5	Create Static Marshaller Instance	90
6.6	Well Annotated Workload Class	91
6.7	Use Annotated Workload Class as Response	91
6.8	Get All labels of a Node	92
6.9	Get All Instance Nodes of a Concrete node	93
6.10	Combination Generation Algorithm	94

1 Introduction

Cloud computing has significantly changed the IT industry over the past few decades. It makes the computing resource '*pay per use*' just like other normal utilities in daily life such as water, gas and electricity [Rou16]. Cloud computing dramatically decreases the cost of purchasing and maintaining IT hardware for both enterprises and individuals. In recent years providers have provided various Cloud service basing on different pricing and capacity model across different Cloud models. How to discover all potential application topologies when the application is distributedly deployed and select the optimal one by considering various criteria from different dimensions is a current research topic . In this section, we introduce the problems that this thesis is targeting.

1.1 Problem Statement

IT industry has been shaped by the rapid development of the cloud computing paradigm and services. Many Cloud service providers offer customizable services with respect to QoS, price, access speed, storage capacity, etc across different Cloud computing models, so both enterprises and individuals can select comparable service accordingly. For example, when users want to select an optimal Cloud offering by considering OPEX(Operating expense) especially, then the pricing model of service is sensitive to users. However, when users require a relative bigger capacity or faster accessing speed of computing resource, they have to pay more.

A number of approaches [VAL13, BBKL14] have provided decision support for users to select from different Cloud offerings when deploying application in Cloud. However these approaches do not consider the application topology. For example, an application topology can be divided into two categories: application specific and application non-specific. With the help of application topology description languages like TOSCA, each component of the application topology – application specific or application non-specific – can be well described. So instead of deploying the application as a whole stack on only one service provider, it becomes possible for the application developer to explore the strategies of application's deployment – which Cloud offering to use to host which parts of the application stack. Then it makes the distributed deployment of application and reusing the non-specific component become possible.

When selecting and configuring application topology optimally, the price is not the only key factor. For example, application performance is another important factor to be considered as it determines the users' experience. It can be easily imagined that a Web application with hundreds of concurrent accessing or with millions of concurrent accessing under same physical

environment behaves totally different. Another scenario is that when multiple applications running on a same physical environment of the provider, the impact to each other can not be predicted. Additionally the workload and performance of an application can change from time to time. For example, the number of accessing of a Web application is significantly distinct during working hours and late night: figure 1.1 is provided by PIWIK¹, which shows the number of visitors who access site PIWIK forum during a day. In another example, figure 1.2 analyzes *page load time* of Facebook² in year 2015. This analytics is made by GTmetrix³, which provides analytics of site performance. It shows that the performance of website varies from time to time with respect to various factors: users' behaviors, special event and hardware changing. So considering evolving workload and performance demands when selecting and configuring the application distribution has an important meaning.

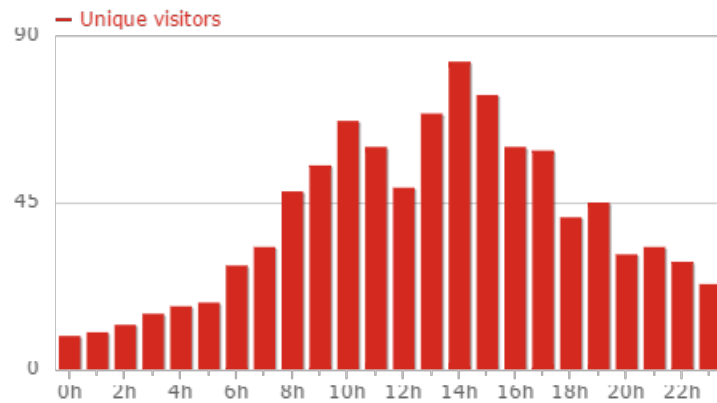


Figure 1.1: Number of Visitors During a Day of Site PIWIK Forum

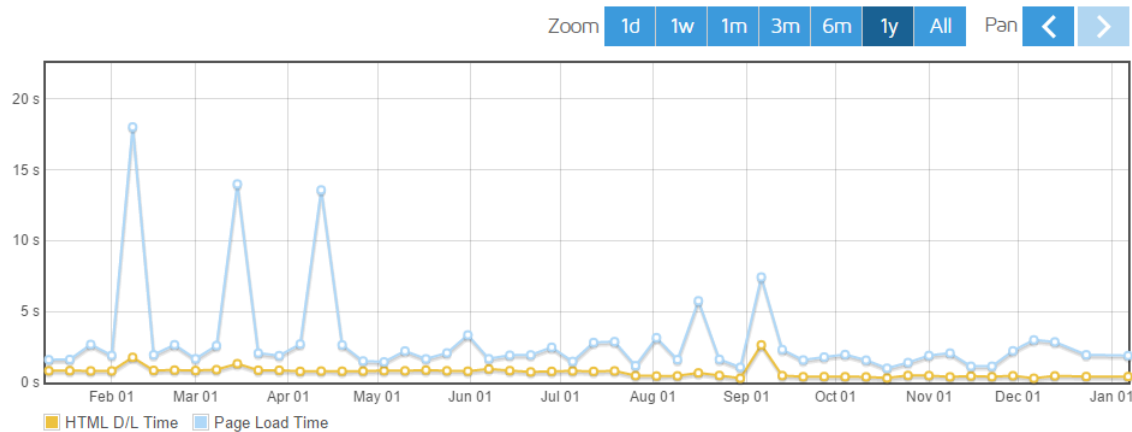


Figure 1.2: Page Load Time of Facebook Over last year

¹<http://piwik.org/>

²<https://www.facebook.com/>

³<https://gtmetrix.com/>

1.2 Motivating Scenario

Figure 1.3 depicts the topology representation of an Web application *MediaWiki*. The application is divided into two parts. The nodes in gray on the top is application specific part which is a two tiers Web application. Other nodes of the application are application non-specific. As showing in figure 1.3, the dash line indicates the possibilities of alternative deployment. For example, the component *Apache_HTTP_Server* of the application can be deployed on different service offerings: either on Azure⁴ or AWS⁵. In this case, the components of one application can be distributedly deployed.

When selecting optimal topology from all discovered topologies, different criteria should be taken into account such as security, QoS and storage capacity. Furthermore, as mentioned previously the performance of the application is a key factor to be considered. The performance of the application highly depends on the characteristics of workload behavioral of its components. For example, as showing in Figure 1.3, the resource demand, expected performance and workload behavior for each component of the application, like front-end, persistence components of application specific and underlying infrastructure like *Web_Server* of application non-specific should be considered together to fulfill the overall requirements when selecting and configuring the application topology.

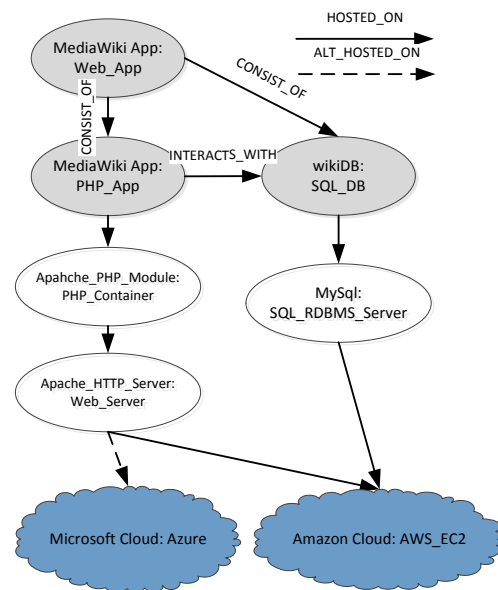


Figure 1.3: Topology Model of MmediaWiki

To resolve the problem discussed above, [ASLW14] has proposed a framework which can model, verify and automatically generate alternative scenarios for the distribution of an application. Then an optimal topology among them can be selected with respect to various dimensions using different criteria. Basing on that, in [SAGF15] the concept of evolving performance and workload is established and implemented to enrich an application topology.

To provide a comprehensive realization for above theory in the area of topology discovering and enrichment with respect to optimal topology selection, a system is needed to persist the relevant elements of topology mentioned above. This thesis focuses on how to model and design a persistence framework using latest graph database for topology elements and its enrichment. Graph database has native advantages when handling application topology as

⁴<https://azure.microsoft.com/en-us/>

⁵<https://aws.amazon.com/>

topology itself is directed graph. Basing on the data model over persistence layer, besides the algorithms which are developed in the business logic layer for the implementation of optimal topology and similar topologies discovery, this thesis also researches on how to deal with topology enrichments and its retrieve, its querying and the evolutionary aspect of discovered topology and enrichments.

1.3 Definitions and Conventions

The following list contains abbreviations which are used in this document.

API Application Programming Interface

Capex Capital Expenditure

CRUD Create, Read, Update and Delete

DBMS Database Management System

DOM Document Object Model

HTTP Hypertext Transfer Protocol

IaaS Infrastructure-as-a-Service

JAXB Java Architecture for XML Binding

JAX-RS Java API for RESTful Web Services

JSON JavaScript Object Notation

JVM Java Virtual Machine

NIST National Institute of Standards and Technology

NoSQL Not only Structured Query Language

OPEX Operating Expense

PaaS Platform-as-a-Service

POJO Plain Old Java Object

RDBMS Relational Database Management System

RJE Remote Job Entry

SaaS Software-as-a-Service

SOA Service-Oriented Architecture

SQL Structured Query Language

TOSCA OASIS Topology and Orchestration Specification for Cloud Applications

URI Uniform Resource Identifier

VPN Virtual Private Network

XML eXtensible Markup Language

QoS Quality of Service

1.4 Outline

The remaining of this thesis structures as follows:

- **Fundamentals, Chapter 2:** introduces and provides the necessary background, technologies, and products used in this thesis.
- **Related Works, Chapter 3:** reviews the development and discusses state of art of the studying area of this thesis so to locates the position of our work.
- **Concept and Specification, Chapter 4:** discusses the concepts established in this thesis for the usage of design, implementation and validation, requirements of system are analyzed here and a system overview is provided.
- **Design, Chapter 5:** provides a general introduction on the system architecture, proposes algorithm for the implementation.
- **Implementation, Chapter 6:** provides implementation details basing on the design principles in the form of coding.
- **Validation, Chapter 7:** design a real scenario basing on a sample application to test the prototype of system.
- **Outcome and Future Work, Chapter 7:** provides a conclusion of the work done in this thesis and analyze the merit and demerit of the work so that an extension can be done basing on this thesis in the future.

2 Fundamentals

2.1 Graph-based Database

Graph, is an object which consists of two sets called its vertex set and its edge set. The elements of the vertex are called vertices and the elements of the edge set are called edges. Vertex set is a finite nonempty set, and the element of edges is two-elements subsets of vertex.[Tru13] [Gar85]

Graph are nowadays uses as the basis for the description of information. The most intuitive examples of graph are social network like Facebook and Twitter. More than that, Gartner ¹ has provided five graphs in the world of business - social, intent, consumption, interest and mobile. In fact, most of the real world data and relationships among them can be modeled by graph model. That is how Graph Database comes.

Graph database management system(henceforth graph database),is an online database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model. Graph databases are generally built for use with transactional (OLTP) systems.[RWE15] The most popular form of graph model is labeled property graph,a labeled property graph has the following characteristics:[RWE15]

1. It contains nodes and relationships.
2. Nodes contain properties (key-value pairs).
3. Nodes can be labeled with one or more labels.
4. Relationships are named and directed, and always have a start and end node.
5. Relationships can also contain properties.

The power of Graph Database is obvious. First, the query speed of graph database is faster than relational database. [VMZ⁺10] has done some researches for comparing relational database and graph database. It uses Neo4j and MySQL for structure type queries and full-text character searches. The result shows that graph database did better at the structural type queries than the relational database. In full-text character searches, the graph databases performed significantly better than the relational database: the speed of graph database is five times faster than relational database. Second, the overhead of graph database is smaller. Compared to the overhead of relational database when it is struggling with highly connected domains, such as Join Table, Foreign Key and very costly Reciprocal Queries, graph database can decrease the execution time when querying and remain relatively constant when dataset gets bigger. Besides, Graph database is naturally additive, meaning it can add new kinds of

¹Five Graphs Deliver a Sustainable Advantage: <https://www.gartner.com/doc/2081316>

relationship, nodes, labels without disturbing existing queries. This characteristic reduces the maintenance and risk of database.

There are already some mature and well known graph database systems in industry. For example, FlockDB² is created and used by Twitter, it is much simpler than other graph databases as it only focus on fewer problems such as traversal. OrientDB³ is an open source NoSQL database management system written in Java. It provides multi-models: Graph, Document, Key/Value, and Object models. So OrientDB can be as a replacement for a product in any of these categories. [Ori] ArangoDB is a NoSQL database developed by triAGENS GmbH.⁴ It supports multi-model as well and graph data is stored together and queried with a common language. The most popular graph database system in industry is Neo4j, which is an open-source graph database implemented in Java and accessible from software written in other languages. Cypher is a declaration language which is used to query data of database. In this thesis, We mainly focus on Neo4j graph database system. All chapters in the following like concept, design and implementations are based on Neo4j.

2.2 Neo4j Graph Database System

Neo4j is developed by Neo Technology, Inc. Neo4j is an open-source graph database implemented in JAVA, it is a transactional, ACID-compliant database.

Neo4j support scalability, high availability and fault-tolerance requirements in order to deal with OLTP workload.[VB14] The query language of Neo4j is Cypher. Cypher⁵ is a declarative database query language, it tells database what data is asked by declaring the pattern.

There are three editions of Neo4j: Community, Enterprise, and Government. The Community edition is free but it can only run on only one node due to this edition does not support clustering.

Neo4j has two deployment solutions: embedded mode and server mode. Embedded mode means that the database is inside the application and in the same JVM as the application. So the access to database is fast by Java API and Cypher query language. The limitation is that in this way the database is locked by the JVM process, other application can not access the database. With server mode, accessing database is available only by REST API provided by Neo4j, Java API is off limits. It means many applications can access Neo4j by different programming language but with a slower access speed comparing to embedded mode. [Fro14]

Up to November 2015, Neo4j ranks number one and is the most popular graph database.[de15]

²<https://en.wikipedia.org/wiki/FlockDB>

³<http://orientdb.com/>

⁴<http://de.triagens.com/>

⁵<http://neo4j.com/developer/cypher-query-language/>

2.3 Graph Transformation Theory

Figure 2.1 is the architecture of Neo4j graph database system. There are four main kinds of primitives in Neo4j: nodes, relationships, relationship type and properties. At the bottom of the architecture, primitives of Neo4j are stored in disk as records. To optimize the writing and reading speed, Neo4j uses caches. As showing in figure, one type of caches is the file buffer cache. This layer caches the Neo4j data in the same format as it is represented on the durable storage media. At the top of the architecture is database API and another type of cache: object cache. The object cache caches individual nodes and relationships and their properties in a form that is optimized for fast traversal of the graph. There are two categories of object caches. One is reference cache. It holds as much of JVM heap memory to as it can to hold primitives. Another object cache is high-performance cache which is used to provide fast speed query. [Neo16]

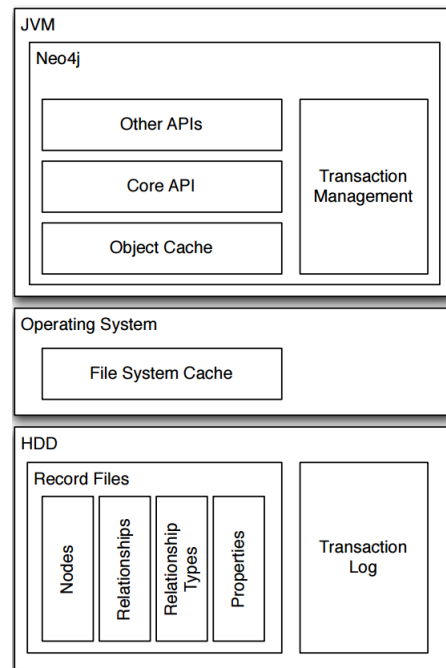


Figure 2.1: Neo4j Architecture

2.3 Graph Transformation Theory

Graph transformation, is a well-known mechanism to generate new graph from an original graph algorithmically. It is originally evolved to deal with non-linear structures and was first time proposed in late sixties for the usage of image recognition, translation of diagram languages. [PR69]

Graph is playing a very important role in dealing with complicated problem, particularly in computer science. A variety of visual notations have been proposed in the area of visualization for software engineering, for example, data and control flow diagram, function block diagram, UML description language and cloud application topology. These diagrams can be treated as graphs directly. The procedure of graph generating, interpreting and evolving makes graph transformation theory involved.

As described in [Hec06], the basic approach of graph transformation is to represent the reality in real world by modeling and extract the concrete object, rule and behavior by generalization. The PacMan example in [Hec06] make a good explanation on instance graph and type graph, here quoting it directly: 'A fixed type graph TG represents the type (concept) level and its instance graphs the individual snapshots. This distinction is a recurring pattern, like in class and objects, data base schema and states, XML schema and documents, etc.'

Basing on above definitions, [BEDL⁺03] proposes the definition of Type Graph with Inheritance (TG_I).

Definition 1 (*Type Graph with Inheritance*) A type graph with inheritance is a triple (TG, I, A) consisting of a type graph $TG=(N, E, s, t)$ (with a set N of nodes, a set E of edges, a source and a target function $s, t: E \rightarrow N$), an inheritance graph I sharing the same set of nodes N , and a set $A \subseteq N$, called abstract nodes. For each node n in I the inheritance clan is defined by $clan_I(n) = \{n \in N \mid \exists \text{path } n' \rightarrow^* n \text{ in } I\}$ where path of length 0 is included, i.e. $n \in clan_I(n)$.

TG_I extends node of type graph to concrete node and abstract node, abstract node has only inheritance relations with other nodes. TG_I can be transformed to ordinary type graph by graph transformation theory, so it can be seen as a convenient notation for ordinary type graphs.

Graph morphism, which is another important definition in graph theory. A graph morphism is a mapping between two graphs that respects their structure. More concretely it maps adjacent vertices to adjacent vertices. [BEDL⁺03] propose the definition of clan morphism:

Definition 2 (*Clan Morphism*) Given a type graph with inheritance (TG, I, A) , $type': G \rightarrow TG$ is a clan-morphism, if for all $e \in G_E$ holds:

$$type'_N \circ s_G(e) \in clan_I(s_{TG} \circ type'_E(e)) \text{ and}$$

$$type'_N \circ t_G(e) \in clan_I(t_{TG} \circ type'_E(e)).$$

Cloud application topology uses above definition and special notation to represent the structure of application, so graph transformation theory can be performed on topology as well.

2.4 Cloud Computing

Cloud computing, also known as 'on-demand computing' has significantly changed the IT service over the past few decades years. Cloud computing means storing and accessing data and application over Internet instead of users own PCs, laptops or servers at house. It makes companies or persons to consume computing resources as a utility, just like water, gas and electricity. It reduces CAPEX and OPEX for computing resource users: the cost for purchasing hardware and the cost to run and maintain them.

The history of cloud computing can be traced back to the seventies. In the mid-1970s, IBM developed and released its VM Operating System to provide time-sharing system know as RJE. In the 1990s, some important telecommunication company offered VPN service to enterprise user. VPN can provide comparable good quality of service with a low cost. It is the prototype of cloud computing. After that scientist and industry starting focusing on the theory development of cloud computing. Since 2000, cloud computing has come into existence. In early 2008, NASA released the first open source software OpenNebula for deploying private and hybrid clouds. In Feb 2010 Microsoft released its cloud computing platform 'Windows Azure'. In July 2010, Rackspace Hosting and NASA jointly launched an

2.4 Cloud Computing

open-source cloud-software initiative known as OpenStack. In 2012, Oracle announced the Oracle Cloud. [wik16]

The definition of cloud computing model is defined by NIST in [MG11]. This cloud model is composed of five essential characteristics (On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, Measured Service); three service models (Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS), Cloud Infrastructure as a Service (IaaS)).

The five essential characteristics of cloud computing model are:

- **On-demand self-service:** Users can customize and provision computing resource such as server time, server capacity without human interaction with service provider.
- **Broad network access:** Services can be accessed through standard mechanisms by user with different client. (PC, Mobile Phone...)
- **Resource pooling:** The resource of providers are in a pool to serve multiple consumers using a multi-tenant model. Users has no knowledge about the resource location.
- **Rapid elasticity:** Computing resource can be provisioned and released, in some cases automatically.
- **Measured service:** Computing resource is automatically controlled and optimized. Transparency of usage of computing resource is provided for both the provider and users.

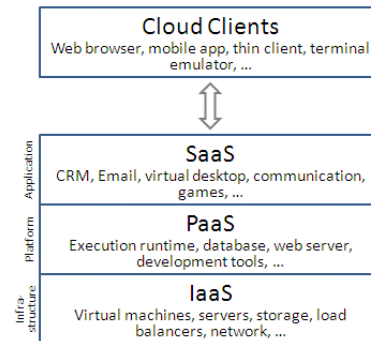


Figure 2.2: Cloud Computing Layers

Figure 2.2 is the layers of cloud computing service model defined by NIST.

- **Software as a Service (SaaS):** Users can access the application running on cloud infrastructures from various clients, for example, a Web Browser. The users do not concern about underlying cloud infrastructure including network, servers, operating systems or storage.
- **Platform as a Service (PaaS):** PaaS provides users computing platforms so users can deploy its own created application or required application. PaaS typically includes operating system, programming language execution environment, database, web server etc.
- **Infrastructure as a Service (IaaS):** IaaS is a self-service models for accessing, monitoring, and managing remote data-center infrastructures, such as compute (virtualized or bare metal), storage, networking, and networking services (e.g. firewalls).

2.5 Cloud Application Topology

Cloud computing is starting a revolution in how applications are design and realized. It is not simply outsourcing the computing resource to external provider any more. Today's applications are totally different, the application itself is becoming more and more complicated, the number of users is unknown, so it is a hard to predict the load. From cloud service customers' perspective, operational expenditure(OPEX) is the very important factor to be considered. By comparing the service quality factor(price,QoS,etc), the requirement to deploy application components to different cloud service providers and making application portable is increased. From service providers' perspective, management of offered service is one of the biggest cost today, how to make the service automatically,dynamically and self-maintained is the key point.

In [ASLW14], the definition of application topology is given:

Definition 3 (*Application Topology*) An application topology is a labeled graph $G = (N^L, E^L, s, t)$ where N is a set of nodes, E is a set of edges, L a set of labels, and s, t the source and target functions $s, t: E^L \rightarrow N^L$. The topology graph is called typed, if the label set L contains only elements $\langle \text{name:type} \rangle$ (for nodes) and $\langle \text{type} \rangle$ (for edges), in which case the graph is denoted by T .

2.5.1 Optimal Distribution of Cloud Applications

To deal with issues discussed above, lots researches focus on providing a unified topology description language like TOSCA,Blueprint,etc. On the other hand, how to dynamically and optimally discover possible application topologies is another hot topic. [ASLW14] proposes definition of viable topology and α, γ and μ -topology for the usage of discovering cloud topology.

Definition 4 (*Viable Topology*) A typed topology T is viable w.r.t a type graph with inheritance TG_I , iff all elements of T are labeled(typed) over the elements of TG , i.e. there exists a graph morphism $m: TG_I \rightarrow T$ which uses the inheritance clan relation.

Definition 5 (α, γ and μ -topology) The type graph with inheritance TG_I for a viable application topology T is called its μ -topology. We denote by α -topology the application-specific sub-graph of a μ -topology, and by γ -topology the non application-specific (and therefore reusable) sub-graph of a μ -topology.

By separating application topology into α, γ -topology, the topology itself is divided into two parts logically: one part is application specific which can not be reused, the other part is application non-specific which can be reused. To find all possible viable topology of an application is to discover all application non-specific topology.

2.6 TOSCA

Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS standard language to describe a topology of cloud based web service. TOSCA is used to substantially enhance the portability of cloud applications and the IT services that comprise them running on complex software and hardware infrastructure. [OAS15a]

TOSCA can define the structure of cloud service, the structure is defined by a topology — a graph of typed nodes and directed typed edges. TOSCA uses Extensible Markup Language (XML) to describe component of topology, properties (ability, policy, requirement, plan, constraint) of component and relationships among them. It also provides the portability for an application deploying on any TOSCA complied cloud, migrating of existing application to cloud and dynamically choosing of multi-cloud provider.

The participants of TOSCA includes most of famous IT companies like Cisco, SAP, IBM, INTEL. TOSCA is widely used and has been accepted by many cloud service provider as cloud service description language.

2.7 OpenTOSCA

OpenTOSCA is an open source ecosystem. The key task of Open TOSCA are to operate management operations, run plans, and manage state. [BBH⁺13]

The OpenTOSCA ecosystem consists of three components [Uni15]:

1. OpenTOSCA Container, a TOSCA runtime environment. It consists of three components which are Implementation Artifact Engine (IAE), Plan Engine (PE) and Controller. As TOSCA has defined an Artifact that is executed on the target node, so IAE is used to deploy all ImplementationArtifacts contained in CSAR files fully automated and provides them this way for management plans. PE is used to implement management plans using different work-flow languages such as BPEL or BPMN. Plans invoke the management operations provided by ImplementationArtifacts. Controller is used to control all other components.
2. Winery, a graphical modeling TOSCA tool which is presented in the next section.
3. Vinothek, a Web-based Self-Service Portal that hides the technical details of TOSCA Runtimes and provides end users a simple graphical interface to provision Cloud applications on demand.

2.8 OpenTOSCA Winery Topology Modeling Environment

Winery [KBBL13a] as mentioned above, is a web based graphical modeling tool supporting the modeling and creation of TOSCA-based applications. It provides HTML5 based web GUI for topology developer to model the topology of cloud application. The Topology Elements

Manager, Topology and Plan Modeler, BPMN4TOSCA plan modeler and Repositories are the major components of Winery (see 5.1).

TOSCA defines 45 [KBBL13a] elements for cloud application topology. Winery separates them into two categories: one is related to visualization like relationship template, node template, which is used by topology modeler; the other one is related to define TOSCA reusable artifacts and configuration like node type and relationship type. Topology elements manager provides management for the second part of elements. Right now Winery uses local file system as the repository. This repository provides REST interface to winery element manager and topology modeler. The repository can save reusable TOSCA elements and export existing TOSCA element to CSAR format or import CSAR file into winery. A GUI is implemented with JAVA and HTML5 for user to interact with winery.

TOSCA is extended and enhanced gradually. [SAGF15] extends winery to PERFinery by providing workload and performance repository which can enrich topology.

2.9 REST

REST stands for Representational State Transfer, which is a lightweight web service architecture proposed by Dr. Roy Fielding in his PhD thesis in 2000 [Fie00]. REST has following principles: [Bur13]

1. Addressable resources: Every entity in real world which is abstracted and represented as data in REST is a resource, which should be addressable through a URI (Uniform Resource Identifier).
2. A uniform, constrained interface: when talking about REST, although it is non-protocol specific, usually means REST over HTTP. Previous technologies like SOAP only take HTTP as a transport protocol. REST uses well predefined HTTP native methods like *PUT, GET, DELETE, POST* to manipulate resources.
3. Representation-oriented: it means using different representations to interact with services. A resource referenced by URI can have different formats. For example, HTML for browser, JSON for JavaScript and XML for JAVA.
4. Communicate statelessly: it means server is stateless. Server does not save any client session data, instead it only stores the state of resource.
5. Hypermedia As The Engine Of Application State (HATEOAS): As REST is stateless, REST uses embedding hyperlinks to other services in the response of server as the engine of application state. For example, when buying a book on Amazon, when customer decides to buy a book, before the credit card is charged, customer is asked to fill additional forms. The server guides customer what to do next by each link it provides to the browser. This is how HATEOAS works: server sends back new actions for each request from client, it tells what the client can do and where to go, it makes the transition of application state.

2.9 REST

Now REST has become very popular for web service development. Major frameworks for different programming language with REST have started appear. For example, Java API for RESTful Web Services(JAX-RS) which is defined in JSR 311, is the standard for REST with JAVA. Currently there are three implementations: Jersey, RESTlet and RestEasy.

REST also has been accepted by industry. Mainstream Web 2.0 service providers like Google,Yahoo,Facebook,Twitter have used REST architecture to develop resource-oriented web service. It can be predicted that REST will be the trend of web service architecture in the future.

3 Related Works

Cloud model is defined in NIST definition of cloud computing [MG11]. It is composed of three service models: SaaS, PaaS and IaaS. There are lots of studies focus on these three areas for providing cloud-based application development.

Some original studies focus on low-level(IaaS) cloud issues such as scalability and load balance. For example, in [RMVG⁺10], an abstract layer called 'Claudia' is proposed to provide a friendly interface which is not too close to the infrastructure layer, so that SPs can reduce their administration burden during the whole service life-cycle. Then it is found there is limitation of each providers with respect of capabilities at their delivery level. Besides that, customization and extension ability is another issue for the user of cloud service. So there is a need to involve every possible SaaS, PaaS and IaaS providers for one comprehensive service at a higher level like application level. In this case, it is necessary to go across the three cloud service models to create a Service-based Application(SBA). For example, [LK09] proposes a systematic way to develop high-quality cloud SaaS. Design criteria of SaaS is defined and commonality is taking into account for reusability. In [Mie10], Cafe(composite application framework) applications is proposed. It provides a whole life-cycle for user to select application provider(can be PaaS, IaaS) and customize the application, finally it is automatically deployed across different providers. In [TSB10], a Service Oriented Cloud Computing Architecture(SOCCA) is proposed. It aims at the issue that different cloud service provider has its own interpretation of cloud computing. With SOCCA different clouds can inter-operate with other ones. This architecture tries to provide a cloud service across different cloud service models and support cloud provider information publishing, dynamic SLA Negotiation and multi-tenancy architecture by a layer named "Cloud Borker Layer". But as mentioned in the paper, the limitation is that at that time there is no powerful modeling language to support developments for multiple platforms so that a service package can be re-deployed on a different cloud.

To track the issues, cloud modeling language is needed so that cloud application could be well described and interpreted by different cloud service provider. In this case, cloud application migrating and distributed deploying will not be an armchair strategist.

Amazon cloud service [Ama] provides a template based service description language for user to customize the needed cloud service. But the limitation is obvious: user is tied to only one cloud service provider. In [ARB12], a service specification language based on *Unified Service Description Language* is proposed to describe both technical and business aspects which includes the capabilities and non-functional characteristics of services. By extending USDL to USDL-SLA [LM12], it can enable attaching guaranteed service states and actions. [BPM12] also propose a cloud modeling language named CloudML, it is provided as Domain-Specific Language (DSL). CloudML focus on the area of software deployment of a cloud service, propose a component based approach to model software deployment across different clouds service

provider. Some other specification modeling language like Open Virtual Format (OVF) [BC] which defines the standard for packaging and distribution information in IaaS layer. Microsoft Azure provides an ad-hoc XML format language on PaaS level [Wil12]. The limitations of above languages focus on a specified layer and does not provide automatic deployment. Topology and Orchestration Specification for Cloud Applications (TOSCA) is defined in [TOS], which is an OASIS standard to describe the components, relationships of cloud service and manage them. TOSCA stays on the level of SaaS, it can provide the ability to automatically deploy application by the definition of capability and requirements pairs. [NLPVDH12] proposes uniform specification language called 'BluePrint' for cloud services description. It aiming at providing cloud service developer to publish, query and compose cloud service. In [ARSL14], The Generalized Topology Language (GENTL) is proposed aiming to identify the optimal distribution of an application in the cloud, potentially across offerings and providers. It also support mapping to other topology language like TOSCA and Blueprints.

With above modeling language supported, the process of application distribution can be automatically executed. For example, in [ARB12], a 5-staged process is defined to dynamically distribute topology orchestration. It transforms application topology into multiple service deployment requests and request dependencies based on an existing and valid specification in the first stage named 'Request Handling and Scheduling'. In the second stage 'Infrastructure Preparation', it configures the infrastructure with details like IP address. This approach is implemented in GEYSERS project [EPN⁺11].

Furthermore, with the help of topology modeling language, cloud-based service application developer can explore which cloud offering to use to host which parts of the application stack. However, when developing application topologies, developers always facing 'reinventing the wheel' problems. A similar solution may be created for different application by different developers for many times. So how to reuse existed application topologies or components of the topologies become a more crucial problem.

In [BS14] and [BS13], the author proposed TOSCA based method to find the matching node types then to decide whether it can be reused in another service template. This approach resolve it by substituting a node type by a service template, and is the first one to clearly categories the matching level of a node type. Based on that, [SBB⁺15], a method named 'TOSCA-MART' is proposed to enable deriving and reusing existing TOSCA solutions. The developers can also specify and customize components from a repository in their own applications.

When cloud application topology or part of it can be reused, another topic how to discover it automatically. There are may several possible topology components can be reused, how to generate all possible topology and how to find the one which meet some functional and non-functional requirement.

[BFL⁺12] proposes Enterprise Topology Graph (ETG), which is a graph-based model for enterprise topologies capturing all entities of enterprise IT and their logical, functional, and physical relationships. ETG is influenced by TOSCA, it generalize TOSCA concepts to extend application models towards the representation of enterprise topology instances. An ETG composes of Node types, Edge types, entities and properties which is key-value-pair to

represent properties of entity. To discover topologies the essential algorithm is searching. Based on ETG, VF2 algorithm [CFSV04] is adopted and optimized. VF2 algorithm is a famous sub graph isomorphism algorithm for matching large graphs. Basing on [BFL⁺12], in [BBKL13], the author proposed a plug-in method for iteratively retrieving application topology. This approach basing on Enterprise Topology Graph (ETG) repository, which represents the level of abstraction required for the desired field of application. Another approach is proposed in [ASLW14], in this paper it provides a formal definition for cloud application topology. By separating an application topology into two parts: application specific part(α -topology) and non-specific part(γ -topology), a theory to discover all possible application topologies is established. The author proposed a method to find a optimal distribution as well. To track the issue of lack of insight into application non-functional requirement, [SAGF15] proposes a method to enrich topology by evolving workload and KPI.

This Master thesis bases on the approach proposed in [ASLW14] and [SAGF15]. We design and implement a framework to persist application topology and its enrichment. Business logic is implemented over the persistence layer for potential application topologies discovery and relevant operation of topology enrichments.

4 Concept and Specification

In this chapter, we establish the concept and specification to describe database model of application topology and its enrichments. The concept and specification are followed during design, implementation and validation phases. As described in previous sections, topology is a directed graph. So Graph database has inherent ability and advantage to present graph. Modeling Cloud application topology based on a graph database is the key step for further work. Once the data model is established, requirements of system are analyzed and relevant use cases are provided. By following use cases, we provide an overview of Topology Persistence and Discovery system and its components. This system can store, retrieve and query α, γ and viable topologies of an application, discover potential viable topologies for an α topology as described in [ASLW14] and find similar topologies for a given α topology. In addition, the system supports operations of evolving workload and performance defined in [SAGF15], which includes storing, retrieving and querying and performing them to enrich a selected topology.

In the first part of this chapter we model each entity which is persisted in database and provide the example. In the second part, we specify the functional and non-functional requirements the system must fulfill. In the third part, a list of use cases are provided. Finally basing on previous models and definitions, an system overview is presented.

4.1 Data Modeling

Database model determines the logical structure of a database and fundamentally determines in which manner application topology and its enrichments can be stored, organized, and manipulated. As graph database is used in the system, so first a brief introduction of modeling method of graph database is presented. Currently there exists no standard way to present graph database model, mostly it is presented by a real example. So following this convention, we design, establish and present the data models of each elements of application topology and relationships among them.

4.1.1 Graph Database Modeling Notations

There are four building blocks that will be used in the remaining of this section to present data model.

1. Nodes: nodes are used to represent entities. Every node can contain multiple properties to describe entity.

2. Relationships: relationship defines the relationship among nodes. It has a name and a direction, which can contain properties as well.
3. Properties: properties are named values where the name is a string. Property is used to describe the characteristics of nodes or relationships.
4. Labels: labels assign roles or types to nodes. Every node can have zero or more labels attached, which are used to group nodes.

Figure 4.1 shows the notations which are used in this paper to represent node, node with property, node with property and label, relationship respectively.

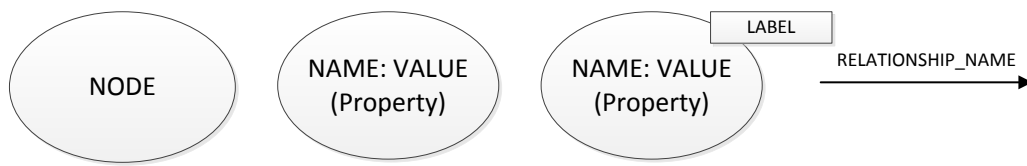


Figure 4.1: Graph database notation

4.1.2 Type Graph with Inheritance Modeling

Type Graph with Inheritance(TGI) is explored in [BEDL⁺03]. Basing on that, to verify and automatically generate alternative scenarios for the distribution of an application across Cloud offerings, a new concept *viable topology* in paper [ASLW14] is proposed as discussed in previous section.

For the usage of modeling α, γ and viable topologies in graph database, first *node* is defined as following:

Node Definition for Data Modeling

In Figure 4.2, a node which represents a typed node used in TGI modeling is defined. It consists of three labels and a set of properties:

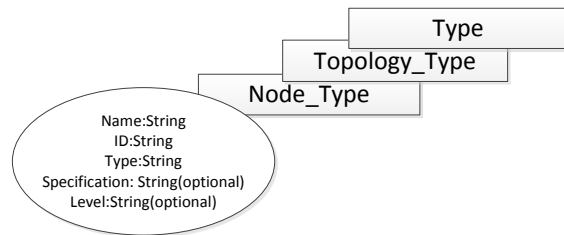


Figure 4.2: Node Definition

Label definition

Topology_Type is used to denote which topology does this node belong to: α , γ or μ .

Type is used to denote what type this node is. For example, in Figure 4.3, the node type on the very top is *WEB_SERVER*, which indicates this node is an entity typed of *WEB_SERVER*.

Node_Type is used to denote what node type this node is. The value of node type can be one of the following:

1. *abstractNode*: indicate this node is an abstract node. Abstract node represents one category of service with common general characteristics, from which concrete node (the node which is labeled *concreteNode*) with more specific properties, requirements and capabilities can be extended.
2. *concreteNode*: indicate this node is a concrete node. Concrete node is a reusable entity that defines the type of one or more instance node.
3. *instanceNode*: indicate this node is an instance node. Instance node is an instance of a concrete node (like Object to Class), it specifies the occurrence of a concrete node as a component of a service and it can have specific requirements and capabilities.
4. *abstractSubTopologyIndex*: indicate this node is an index of an abstract sub-Topology. It contains the specification of the abstract sub-topology and from this node all other abstract nodes can be retrieved and accessed.
5. *alphaTopologyIndex*: indicate this node is an index of an alpha topology. It contains the specification of the alpha topology and from this node all other instance nodes belong to the alpha topology can be retrieved and accessed.
6. μ -topology: represent a viable topology which contains information such as which application it belongs to, created time, whether it is obsolete, etc.
7. *workload*: indicate this node is a workload node. It contains the evolving workload information.
8. *performance*: indicate this node is a performance node. It contains the evolving performance information.
9. *requirement*: indicate this node is a requirement node. It contains the requirements of service of the node.
10. *capability*: indicate this node is a capability node. It contains the capabilities of service of the node.
11. *RelationshipType*: save the type of relationship, whose instance is used to connect nodes.

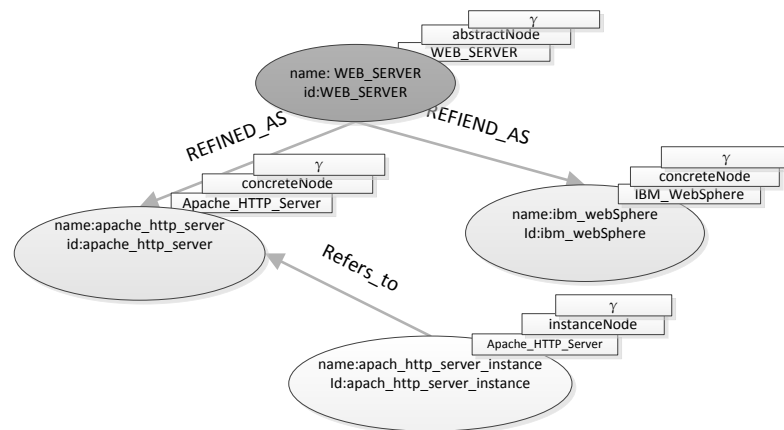


Figure 4.3: Node type tree

Property definition

1. name: This attribute stores name of the entity.
2. id: This attribute stores identification of the entity .
3. type: This attribute stores type of the entity.
4. specification: This attribute stores topology description described by topology description language. For example, if TOSCA is used to describe topology, then this attribute should save TOSCA definition in a *STRING* format.
5. level: This attribute stores the level information of the node. For example, if the node is the root node of the topology, the value of this attribute will be *root*.

4.1.3 Modeling Example**Node type tree**

Figure 4.3 depicts the modeling for one Cloud service: Web Server. In this example, two concrete nodes *Apache_HTTP_Server* and *IBM_WebSphere* refine abstract node *WEB_SERVER*. Node *Apache_HTTP_Server_instance* instantiates concrete node *Apache_HTTP_Server*. The abstract nodes, their refined concrete nodes and their instance nodes compose node type tree. The relationships defined in the following table connect abstract, concrete and instance nodes. Node type tree is used for modeling γ -topology in the following.

Relationship Type	Description
REFINED_AS	connect abstract node with concrete node.
REFERS_TO	connect one instance node to a concrete node.

γ -topology modeling

Figure 4.4 models a γ -topology. A γ -topology composes of different node type trees and relationships which connect them.

Taking abstract node with property *Name:ApachePHPModule* for example, this node is refined by one concrete node with property *Name:ApachePHPModule*, which has a capability node and is referred by an instance node. Then this abstract node connect to another abstract node *Name:ApacheWebserver* with relationship *HOSTED_ON*.

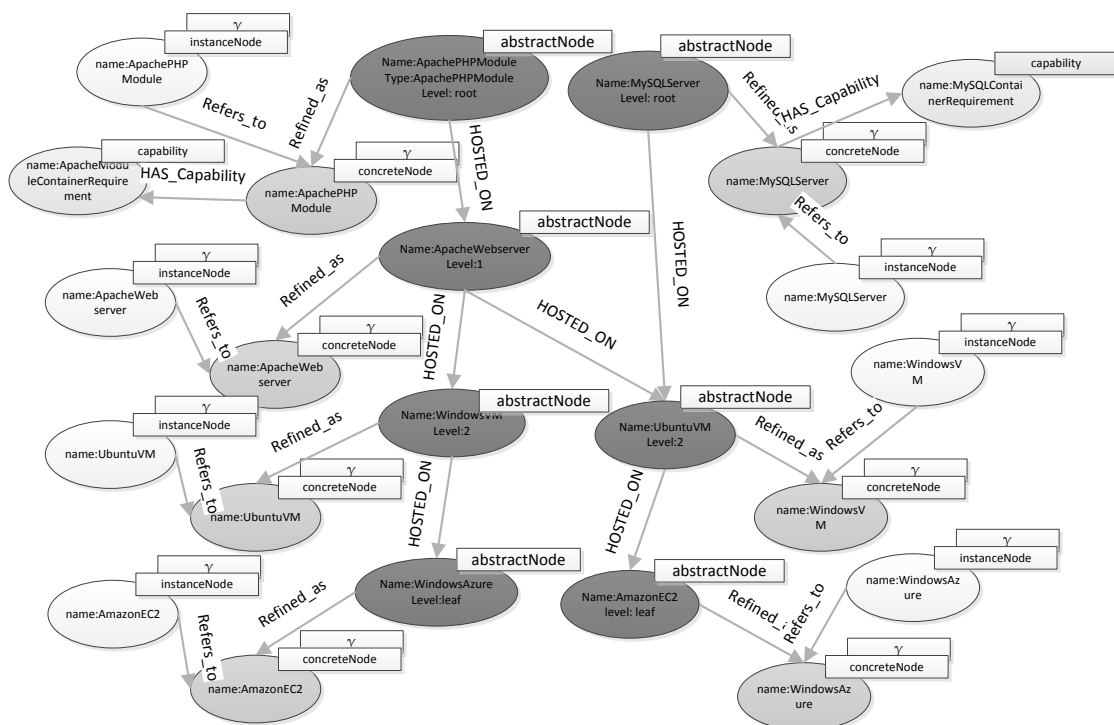


Figure 4.4: γ -topology modeling

α -topology modeling

Figure 4.5 models an α -topology which comes from MediaWiki application in Figure 1 of [SAGF15]. An α -topology model composes of instance nodes and relationships which connect instance nodes together. An index node labeled by *alphaTopologyindex* connects all α -topology nodes with relationship *INCLUDES*, the index node is used to retrieve all nodes of α -topology and save characteristics of an α -topology.

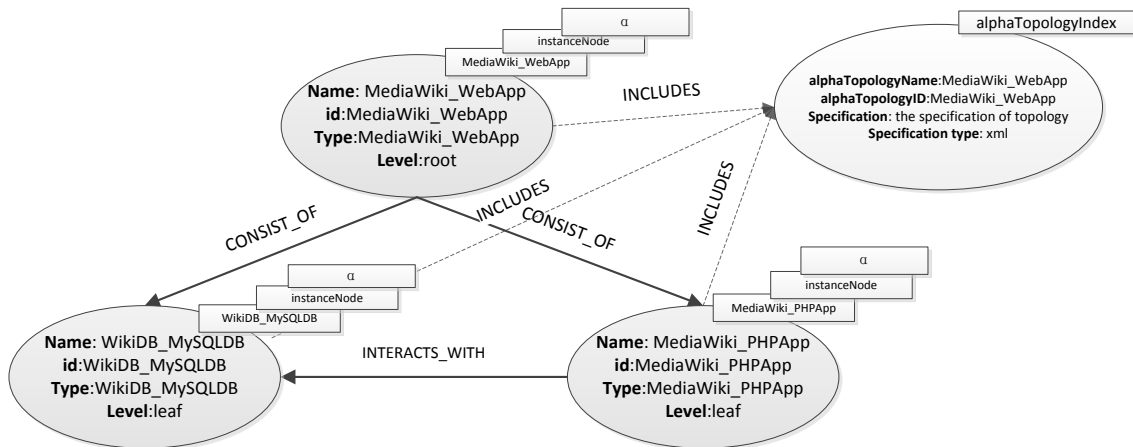


Figure 4.5: α -topology modeling

Topology Enrichment - Performance Indicators Modeling

In [SAGF15], an application performance is partitioned in two correlated groups: Operational Requirements and Business Requirements. These two groups can be defined and estimated by the usage of Metrics.

Figure 4.6 models performance demand specification. The node in the center labeled by *performance* is the performance index node from which all performance attributes can be retrieved.

4.1 Data Modeling

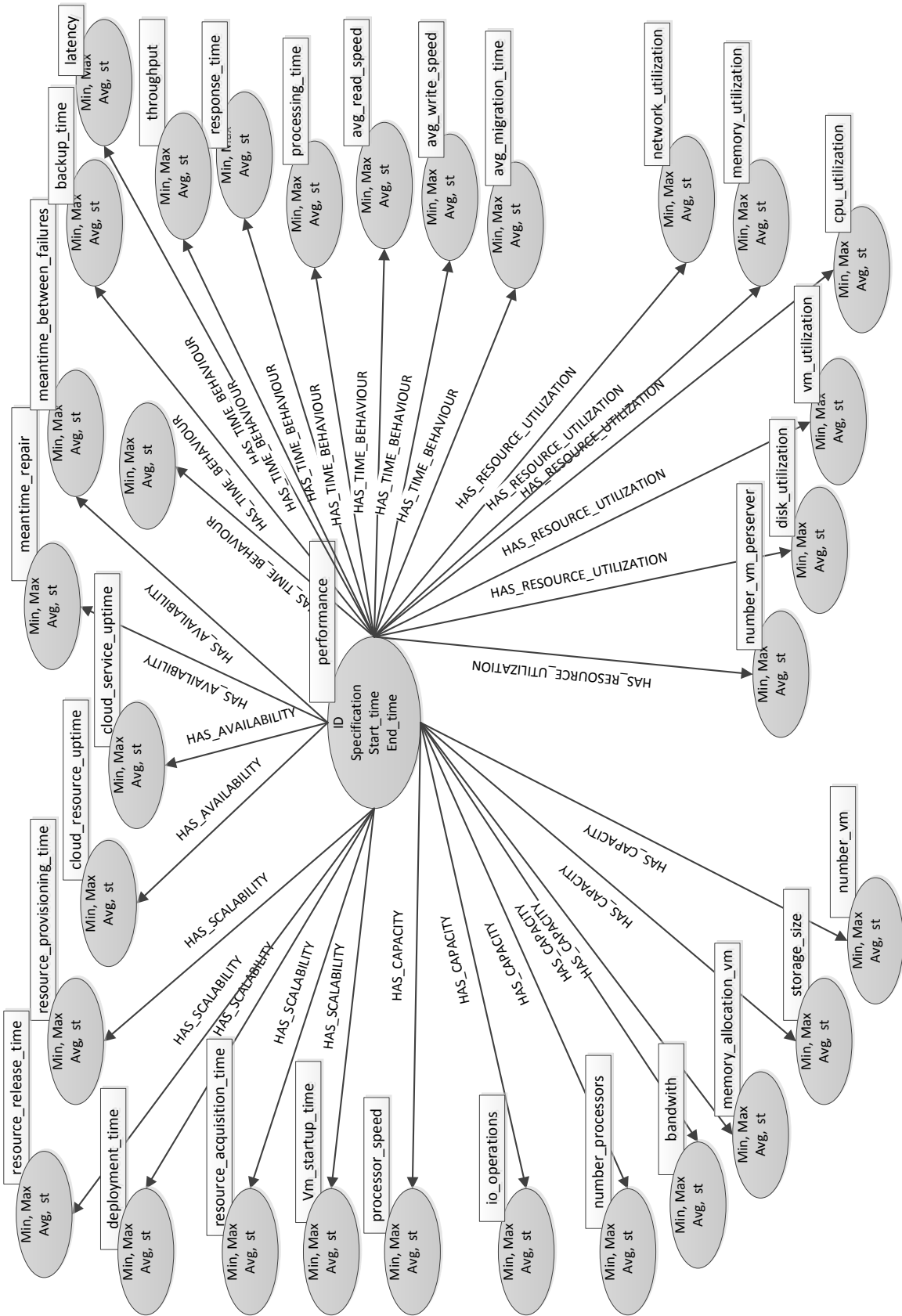


Figure 4.6: Performance Modeling

Topology Enrichment - Workload Behavior Modeling

In [SAGF15], an application workload is proposed. It plays a role in provisioning of new or re-configuration of existing Cloud resources for Cloud applications, estimation of the necessary resources and analysis of the application behaviors.

In Figure 4.7, the node labeled by *workload* models the workload.

Topology enriched by performance and Workload modeling

Topology can be enriched by evolving performance and workload. Figure 4.7 models how an α topology *performr* workload and performance. When a workload profile or a performance demand needs to be performed by an α topology, a relationship is established between the topology index node and workload or performance node with relationship *PERFORM_WORKLOAD* or *PERFORM_PERFORMANCE*. The attributes *start_time* and *end_time* indicate the valid period of workload or performance.

As showing in Figure 4.7, one alpha topology index node can perform multiple workloads and performances.

Dynamically Viable (μ) Topology Retrieving

To discovery all viable topologies, first we give the definition of abstract sub-topology. As showing in figure 4.4, an abstract sub-topology (node with color of darkest gray) is the topology which consists of all abstract nodes and relationships which connect them. An abstract sub-topology describes basic non-specific structure of an application. Each abstract node can have concrete nodes which refine the abstract node with more specific properties, requirement and capability. Finally each concrete node can have instance nodes which represent the real Cloud service with much more details.

Compared to the number of concrete node and instance node, the number of abstract node is relatively small and the structure is stable once abstract sub-topology is established. As instance nodes represent one kind of service from Cloud services provider, it may vary more often (e.g: capability, price, configuration, etc.). So it can be added or deleted by maintaining the relationship to its parent node without affecting the application structure. In this way, the data model can be easily maintained and extended.

For a given α -topology, to explore all possible viable topologies, there are several steps to process:

1. For a given α -topology, find all leaf nodes.
2. For each leaf node of the α -topology, find all requirements.
3. For each γ -topology, check if the capability of the root nodes of the γ -topology fulfill the requirements.

4.2 System Requirements

4. For a qualified γ -topology, traversing the abstract node of the abstract sub-topology. For each abstract sub-topology, retrieve all its leaf nodes(instance nodes) by traversing the node type tree.
5. Then the combinations of all instance nodes basing on the abstract sub-topology structure can establish all possible viable topologies.

Topology History Retrieving

Once one viable topology is selected, it should be stored for the usage of topology evolution observation. To achieve this, a new node labeled by *mu-topology* in the database is created and the new generated viable topology specification will be saved in this node. So this node will record the whole viable topology and other information like created date, whether it is obsolete as showing in Figure 4.7.

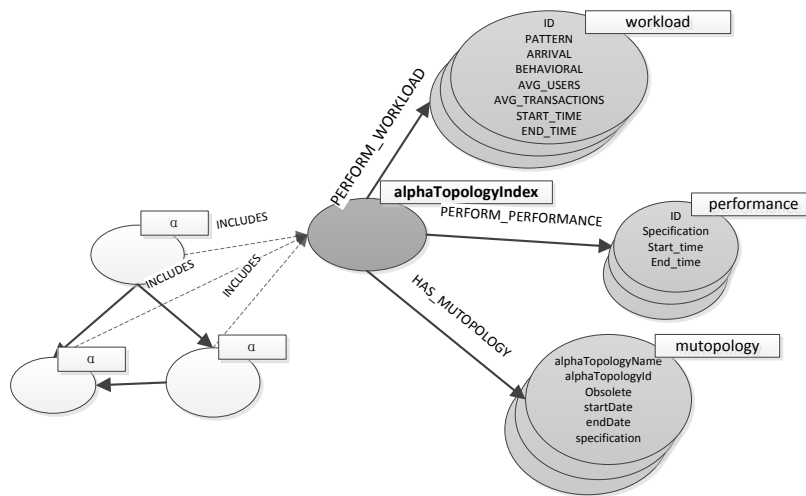


Figure 4.7: Workload, KPI performing and viable history

4.2 System Requirements

In this section, basing on previous data model, detail requirements of the system from the perspective of functional and non-functional are provided. The requirements are divided into two categories: topology requirements and requirement of topology enrichments. They are presented in details in the following sections.

4.2.1 Topology requirements

In [ASLW14], the definition of viable topology is proposed for reusing application topology. The topologies of an application are categorized as α -topology, γ -topology and μ -topology from the perspective of topology functionality and re-usability.

Following are the requirements which topology persistence and discovery system should fulfill.

General Requirements

1. The persistence of application topologies should not be coupled to a concrete specification. It should be designed and provided in a generic manner. This means system can persist topology which is described by any kind of topology description language(e.g. TOSCA,GENTL).
2. The Cloud application specification which describes a component(e.g. concretenode, instancenode) should be persisted as a string with component together. For example, in previous section exists a property *specification* in concrete node, then this attribute should save *NodeType* definition if TOSCA is used as the Cloud application description language.
3. Some parameters should be extracted from specification for particular usage, e.g. service matching. For example, if TOSCA is selected, then requirements of α -topology and capabilities of γ -topology should be extracted and persisted respectively.

Following are the requirements for each kind of topology:

α -topology requirements

α -topology is application specific topology. It describes the application's characteristic so it can not be re-used. For an α -topology, following requirements should be fulfilled:

1. An α -topology can be persisted in database no matter by which topology specification it is described. When it is persisted, the requirements of the component of the topology should be extracted and persisted separately.
2. An α -topology can be deleted. Once it is deleted, its index node and requirement nodes should be deleted as well.
3. An α -topology can be retrieved in the format of its specification, e.g. if the α -topology is described by TOSCA, when this α -topology is retrieved, it should be in the format of TOSCA.
4. An α -topology can be enriched with a workload performed for a concrete time interval of the application production's phase.
5. At one time an α -topology can perform multiple workloads.

4.2 System Requirements

6. Workloads of one α -topology has performed can be queried and retrieved by time interval or ID.
7. An α -topology can be enriched with a performance(KPI) performed for a concrete time interval of the application production's phase.
8. At one time an α -topology can perform multiple performances(KPIs).
9. Performance of one α -topology can be queried and retrieved by time interval or ID.

γ -topology requirements

γ -topology is application non-specific. It contains multiple Cloud service from different Cloud service providers. γ -topology is dynamic as it can change from time to time. For an γ -topology, following requirements should be fulfilled:

1. γ -topology can be persisted as modeled in 4.4 . The creating order of a γ -topology is: abstract sub-topology \rightarrow concrete nodes \rightarrow instance nodes. Once it is persisted, the capabilities of the root node of γ -topology should be extracted and persisted separately.
2. An γ -topology can be queried and retrieved in different levels and manners. For example, an abstract sub-topology can be retrieved as a whole entity. For each concrete node of the abstract sub-topology, its linked instance nodes can be retrieved separately .
3. γ -topology can be extended and modified on the level of concrete node and instance node. As concrete node represents one kind of Cloud service, when adding a new concrete node to an existing abstract sub-topology and linking instance nodes to the concrete node, the γ -topology is extended and modified.
4. When a concrete node of abstract sub-topology is deleted, its linked instance nodes should be deleted as well.
5. When an instance node of concrete node is deleted, there should be no impact on concrete node.

viable-topology(μ -topology) requirements

Viable topology(μ -topology) can be explored by α and γ topology, so the operation of μ -topology is basing on the existence of α and γ topology.

1. For a given α -topology, all possible viable topologies can be discovered as candidates according to γ -topology stored in database.
2. User can select one of the candidate of viable topologies and persist it for further processing.
3. At one time one application(α -topology) can have only one viable topology.
4. The viable topology of one application can be re-discovered and re-selected, persisted.

5. The discovering and selecting history of viable topologies of one application can be queried and retrieved.

4.2.2 Workload Requirements

Workload plays a role in provisioning of new or re-configuration of existing Cloud resources for Cloud applications, estimation of the necessary resources and analysis of the application behaviors. That means, workload defines the application behavior. The requirements of workloads are listed below:

1. Workload can be persisted, deleted and queried. A workload should be persisted with a concrete time interval to indicate the valid period of this workload. When a workload is deleted, if it is performed by one or more α -topologies, the *perform* relationships are deleted as well. A workload can be queried and retrieved by its ID or by α -topology which performs it.
2. A workload can be performed as the enrichments by different α -topologies.

4.2.3 Performance(KPI) Requirements

KPI evaluates and analyzes the performance of the application. The requirements of KPIs are listed as below:

1. Performance can be persisted, deleted and queried. A performance should be persisted with a concrete time interval to indicate the valid period of this workload. When a performance is deleted, if it is performed by one or more α -topologies, the *perform* relationships are deleted as well. The performance can be queried and retrieved by its ID or by α -topology which performs it.
2. A performance can be performed by different α -topologies.

4.3 Use Case

In this section, use cases of the operations performed on topology, performance and workload are presented. The user of system is topology developer who performs CRUD operations on the level of node, relationship, topology, performance and workload. An overview of the set of use cases for the developers is presented in Figure 4.8. Following the figure the set of use cases are described in details.

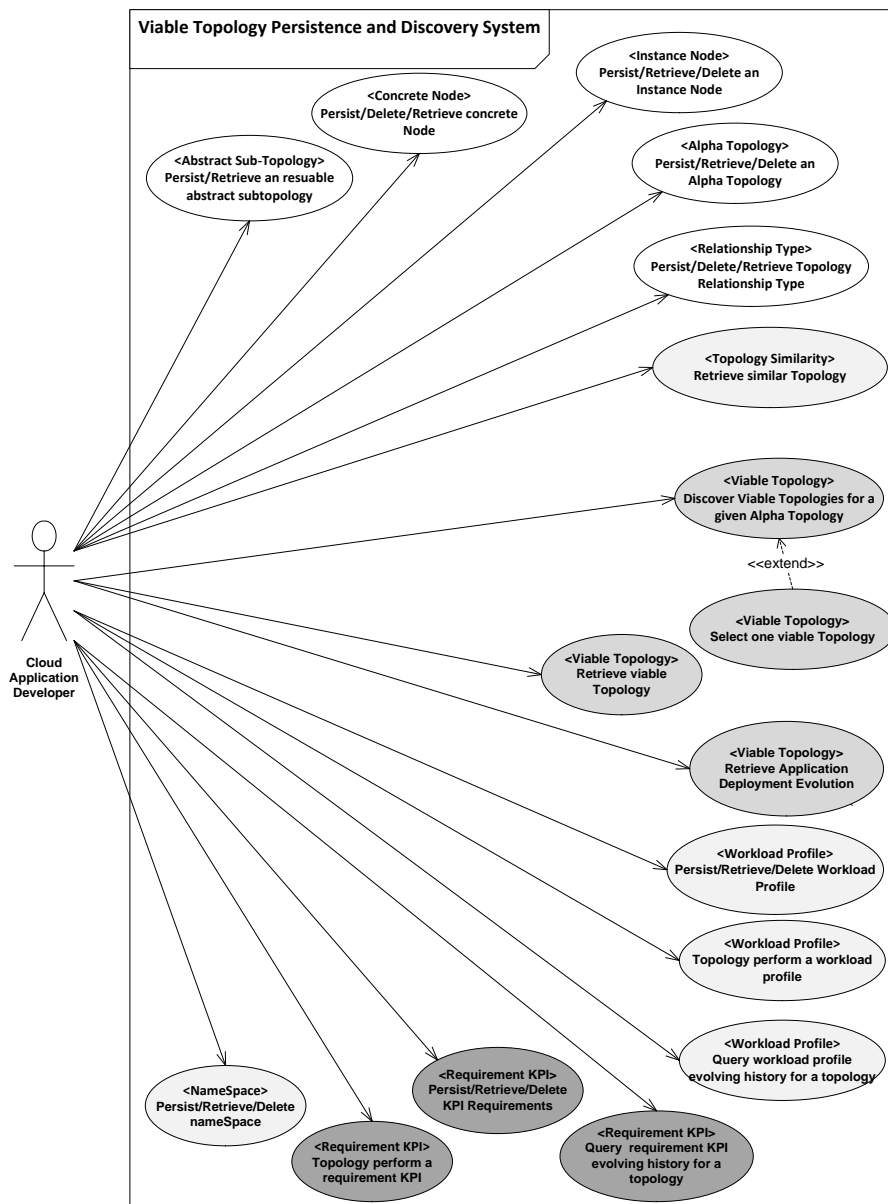


Figure 4.8: Use Case Diagram

Name	Persist an abstract sub-topology
Goal	The developer wants to persist an abstract sub-topology
Actor	Developer
Pre-Condition	-
Post-Condition	The abstract sub-topology is stored in database successfully
Post-Condition in Special Case	The abstract sub-topology is not stored successfully
Normal Case	1. The developer persist an abstract sub-topology.
Special Cases	1. abstract sub-topology with same ID already existed. a) The system shows an error message.

Table 4.1: Use Case Description: *Persist an abstract sub-topology*.

Name	Retrieve abstract sub-topology
Goal	The developer wants to retrieve abstract sub-topologies
Actor	Developer
Pre-Condition	The abstract sub-topology already exists
Post-Condition	The abstract sub-topologies are retrieved successfully
Post-Condition in Special Case	The abstract sub-topologies are not retrieved
Normal Case	1. The developer retrieves abstract sub-topologies.
Special Cases	1. The abstract sub-topology does not exist. a) The system shows an message.

Table 4.2: Use Case Description: *Retrieve abstract sub-topology*.

Name	Persist a concrete node
Goal	The developer wants to persist a concrete node with a specific type of abstract node
Actor	Developer
Pre-Condition	abstract node with the type indicated by concrete node is existed
Post-Condition	The node is persisted successfully and linked to an abstract node, ability and requirement nodes are created if concrete node has

4.3 Use Case

Post-Condition in Special Case	The node is not persisted successfully
Normal Case	1. The developer persists a concrete node with specific type.
Special Cases	1. The node is not persisted. a) The system shows a message.

Table 4.3: Use Case Description: *Persist a concrete node.*

Name	Delete a concrete node
Goal	The developer wants to delete a concrete node
Actor	Developer
Pre-Condition	The node already exists
Post-Condition	The node is deleted successfully,all its relationship,linked instance nodes,capability nodes and requirement noes are deleted as well
Post-Condition in Special Case	The node is not deleted successfully
Normal Case	1. The developer delete a concrete node by its database ID.
Special Cases	1. The node does not exist. a) The system shows an error message.

Table 4.4: Use Case Description: *Delete a concrete node.*

Name	Retrieve concrete nodes
Goal	The developer wants to retrieve concrete nodes
Actor	Developer
Pre-Condition	The concrete nodes already exist
Post-Condition	The concrete nodes are retrieved successfully
Post-Condition in Special Case	The concrete nodes are not retrieved successfully
Normal Case	1. The developer retrieve concrete nodes.
Special Cases	1. The node does not exist. a) The system shows an error message.

Table 4.5: Use Case Description: *Retrieve concrete nodes.*

Name	Retrieve instance nodes refers to a concrete node
Goal	The developer wants to retrieve instance nodes refer to a concrete node
Actor	Developer
Pre-Condition	The instance nodes already exists
Post-Condition	The instance nodes are retrieved successfully
Post-Condition in Special Case	The instance nodes are not retrieved successfully
Normal Case	1. The developer retrieve instance nodes refers to a concrete node.
Special Cases	1. The concrete node does not exist. a) The system shows an error message.

Table 4.6: Use Case Description: *Retrieve instance nodes refers to a concrete node.*

Name	Persist an instance node
Goal	The developer wants to persist an instance node
Actor	Developer
Pre-Condition	Concrete node with the type indicated by instance node is existed
Post-Condition	The node is persisted successfully and linked to a concrete node
Post-Condition in Special Case	The node is not persisted successfully
Normal Case	1. The developer persists an instance node with specific type.
Special Cases	1. The node is not persisted. a) The system shows a message.

Table 4.7: Use Case Description: *Persist an instance node.*

Name	Delete an instance node
Goal	The developer wants to delete an instance node
Actor	Developer
Pre-Condition	The node already exists
Post-Condition	The node is deleted successfully,all its relationships are deleted as well

4.3 Use Case

Post-Condition in Special Case	The node is not deleted successfully
--------------------------------	--------------------------------------

Normal Case	1. The developer delete an instance node by its ID.
-------------	---

Special Cases	1. The node does not exist. a) The system shows an error message.
---------------	--

Table 4.8: Use Case Description: *Delete an instance node.*

Name	Retrieve instance nodes
------	--------------------------------

Goal	The developer wants to retrieve instance nodes
------	--

Actor	Developer
-------	-----------

Pre-Condition	The instance nodes already exist
---------------	----------------------------------

Post-Condition	The instance nodes are retrieved successfully
----------------	---

Post-Condition in Special Case	The instance nodes are not retrieved successfully
--------------------------------	---

Normal Case	1. The developer retrieve instance nodes.
-------------	---

Special Cases	1. The instance nodes does not exist. a) The system shows an error message.
---------------	--

Table 4.9: Use Case Description: *Retrieve instance nodes.*

Name	Persist Relationship Type
------	----------------------------------

Goal	The developer wants to persist a relationship type
------	--

Actor	Developer
-------	-----------

Pre-Condition	There is no same relationship type in database existing in database already
---------------	---

Post-Condition	The relationship type is persisted successfully
----------------	---

Post-Condition in Special Case	The relationship type is not persisted successfully
--------------------------------	---

Normal Case	1. The developer persist a relationship type .
-------------	--

Special Cases	1. A same relationship type exists already. a) The system shows an error message.
---------------	--

Table 4.10: Use Case Description: *Persist Relationship Type*.

Name	Retrieve Relationship types
Goal	The developer wants to retrieve relationship types
Actor	Developer
Pre-Condition	The relationship types exists already
Post-Condition	The relationship types are retrieved successfully
Post-Condition in Special Case	The relationship types are not retrieved successfully
Normal Case	1. The developer retrieves relationship types.
Special Cases	1. The relationship type does not exist. a) The system shows a message.

Table 4.11: Use Case Description: *Retrieve Relationship types*.

Name	Delete one Relationship type
Goal	The developer wants to delete a relationship type
Actor	Developer
Pre-Condition	The relationship type exists already
Post-Condition	The relationship type is deleted successfully
Post-Condition in Special Case	The relationship type is not deleted successfully
Normal Case	1. The developer deletes a relationship type.
Special Cases	1. The relationship type does not exist. a) The system shows an error message.

Table 4.12: Use Case Description: *Delete one Relationship type*.

Name	Find similar alpha topologies
Goal	The developer wants to find all similar alpha topologies for a given alpha topology
Actor	Developer
Pre-Condition	Alpha topologies exist in database already

4.3 Use Case

Post-Condition	Similar alpha topologies are founded successfully
Post-Condition in Special Case	No similar alpha topologies are founded
Normal Case	1. The developer found similar alpha topologies .
Special Cases	1. There are no similar alpha topologies in the database. a) The system shows a message.

Table 4.13: Use Case Description: *Find similar alpha topologies.*

Name	Discover All Viable Topologies for a given alpha topology
Goal	The developer wants to discover all viable topologies for a given alpha topology
Actor	Developer
Pre-Condition	The Gamma Topology is established with correct abstract sub-topology, concrete nodes,instance nodes and relationship type in database already
Post-Condition	Viable Topologies are discovered successfully
Post-Condition in Special Case	Viable Topology is not discovered
Normal Case	1. The developer discovers all viable Topologies for a given alpha topology.
Special Cases	1. Viable Topology is not discovered. a) The system shows a message.

Table 4.14: Use Case Description: *Discover All Viable Topologies for a given alpha topology .*

Name	Persist one viable topology for an application
Goal	The developer wants to persist one viable topology for an application
Actor	Developer
Pre-Condition	At least one Viable topology is discovered in the previous step
Post-Condition	Viable Topology is persisted successfully
Post-Condition in Special Case	Viable Topology is not persisted
Normal Case	1. The developer persist one viable topology for an application.

Special Cases	<ol style="list-style-type: none"> 1. Viable Topology is not persisted. <ol style="list-style-type: none"> a) The system shows one message.
---------------	--

Table 4.15: Use Case Description: *Persist one viable topology for an application.*

Name	Persist one workload
Goal	The developer wants to persist one workload
Actor	Developer
Pre-Condition	There is one workload with same ID existed in database already
Post-Condition	workload is persisted successfully
Post-Condition in Special Case	workload is not persisted successfully
Normal Case	<ol style="list-style-type: none"> 1. The developer persists one workload.
Special Cases	<ol style="list-style-type: none"> 1. One same workload already exists. <ol style="list-style-type: none"> a) The system shows a message.

Table 4.16: Use Case Description: *Persist one workload.*

Name	Retrieve one workload
Goal	The developer wants to retrieve one workload
Actor	Developer
Pre-Condition	The workload already exists
Post-Condition	workload is retrieved successfully
Post-Condition in Special Case	workload is not retrieved successfully
Normal Case	<ol style="list-style-type: none"> 1. The developer retrieves one workload.
Special Cases	<ol style="list-style-type: none"> 1. The workload to be retrieved does not exist. <ol style="list-style-type: none"> a) The system shows a message.

Table 4.17: Use Case Description: *Retrieve one workload.*

Name	Retrieve all workloads in database
Goal	The developer wants to retrieve all workloads in database

4.3 Use Case

Actor	Developer
Pre-Condition	Workload already exists
Post-Condition	All workloads are retrieved successfully
Post-Condition in Special Case	workload is not retrieved successfully
Normal Case	1. The developer retrieves all workloads.
Special Cases	1. No workload is retrieved. a) The system shows a message.

Table 4.18: Use Case Description: *Retrieve all workloads in database.*

Name	Delete one workload
Goal	The developer wants to delete a workload
Actor	Developer
Pre-Condition	The workload exists already
Post-Condition	The workload is deleted successfully
Post-Condition in Special Case	The workload is not deleted successfully
Normal Case	1. The developer deletes a workload.
Special Cases	1. The workload does not exist. a) The system shows an error message.

Table 4.19: Use Case Description: *Delete one workload.*

Name	Persist one performance
Goal	The developer wants to persist one performance
Actor	Developer
Pre-Condition	There is one performance with same ID existed in database already
Post-Condition	performance is persisted successfully
Post-Condition in Special Case	performance is not persisted successfully
Normal Case	1. The developer persists one performance.

Special Cases	<ol style="list-style-type: none"> 1. One same performance already exists. <ol style="list-style-type: none"> a) The system shows a message.
---------------	---

Table 4.20: Use Case Description: *Persist one performance.*

Name	Retrieve one performance
Goal	The developer wants to retrieve one performance
Actor	Developer
Pre-Condition	The performance already exists
Post-Condition	Performance is retrieved successfully
Post-Condition in Special Case	Performance is not retrieved successfully
Normal Case	<ol style="list-style-type: none"> 1. The developer retrieves one performance.
Special Cases	<ol style="list-style-type: none"> 1. The performance to be retrieved does not exist. <ol style="list-style-type: none"> a) The system shows a message.

Table 4.21: Use Case Description: *Retrieve one performance.*

Name	Retrieve all performances in database
Goal	The developer wants to retrieve all performances in database
Actor	Developer
Pre-Condition	Performance already exists
Post-Condition	All performances are retrieved successfully
Post-Condition in Special Case	Performance is not retrieved successfully
Normal Case	<ol style="list-style-type: none"> 1. The developer retrieves all performances.
Special Cases	<ol style="list-style-type: none"> 1. No performance is retrieved. <ol style="list-style-type: none"> a) The system shows a message.

Table 4.22: Use Case Description: *Retrieve all performances in database.*

Name	Delete one performance
Goal	The developer wants to delete a performance

4.3 Use Case

Actor	Developer
Pre-Condition	The performance exists already
Post-Condition	The performance is deleted successfully
Post-Condition in Special Case	The performance is not deleted successfully
Normal Case	1. The developer deletes a performance.
Special Cases	1. The performance does not exist. a) The system shows an error message.

Table 4.23: Use Case Description: *Delete one performance.*

Alpha Topology Use Cases

Name	Persist an alpha topology
Goal	The developer wants to persist an alpha topology
Actor	Developer
Pre-Condition	-
Post-Condition	The alpha topology is stored in database successfully
Post-Condition in Special Case	The alpha topology is not stored successfully
Normal Case	1. The developer persist an alpha topology.
Special Cases	1. Alpha topology with same ID already existed. a) The system shows an error message.

Table 4.24: Use Case Description: *Persist an alpha topology.*

Name	Retrieve an alpha topology
Goal	The developer wants to retrieve an alpha topology by ID
Actor	Developer
Pre-Condition	The alpha topology already exists
Post-Condition	The alpha topology is retrieved successfully
Post-Condition in Special Case	The alpha topology is not retrieved

Normal Case	1. The developer retrieves an alpha topology.
Special Cases	1. The alpha topology does not exist. a) The system shows an message.

Table 4.25: Use Case Description: *Retrieve an alpha topology.*

Name	Retrieve all alpha topologies
Goal	The developer wants to retrieve all alpha topologies
Actor	Developer
Pre-Condition	Alpha topology already exists
Post-Condition	All alpha topologies are retrieved successfully
Post-Condition in Special Case	No alpha topology is retrieved
Normal Case	1. The developer retrieves all alpha topologies.
Special Cases	1. No alpha topology exists. a) The system shows a message.

Table 4.26: Use Case Description: *Retrieve all alpha topologies.*

Name	Delete one alpha topology
Goal	The developer wants to delete an alpha topology
Actor	Developer
Pre-Condition	The alpha topology exists already
Post-Condition	The alpha topology is deleted successfully
Post-Condition in Special Case	The alpha topology is not deleted successfully
Normal Case	1. The developer deletes an alpha topology.
Special Cases	1. The alpha topology does not exist. a) The system shows an error message.

Table 4.27: Use Case Description: *Delete one alpha topology.*

4.3 Use Case

Name	An alpha topology performs a performance
Goal	The developer wants to perform a performance for an alpha topology
Actor	Developer
Pre-Condition	The performance and alpha topology already exists
Post-Condition	The performance is performed successfully
Post-Condition in Special Case	The performance is not performed successfully
Normal Case	1. The developer makes an alpha topology performing a performance .
Special Cases	<ol style="list-style-type: none"> 1. When performing performance , the performance does not exist. <ol style="list-style-type: none"> a) The system shows an error message. 2. When performing performance ,the alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.28: Use Case Description: *An alpha topology performs a performance.*

Name	Retrieve a performance of an alpha topology
Goal	The developer wants to retrieve a performance of an alpha topology by ID
Actor	Developer
Pre-Condition	The alpha topology already exists, the performance has been performed
Post-Condition	The performance is retrieved successfully
Post-Condition in Special Case	The performance is not retrieved
Normal Case	1. The developer retrieves a performance of an alpha topology.
Special Cases	<ol style="list-style-type: none"> 1. The performance to be retrieved does not exist. <ol style="list-style-type: none"> a) The system shows an error message. 2. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.29: Use Case Description: *Retrieve a performance of an alpha topology.*

Name	Retrieve all performances of an alpha topology
Goal	The developer wants to retrieve all performances of an alpha topology
Actor	Developer

Pre-Condition	The alpha topology already exists
Post-Condition	The performances are retrieved successfully
Post-Condition in Special Case	The performances are not retrieved
Normal Case	1. The developer retrieves all performances of an alpha topology.
Special Cases	<ol style="list-style-type: none"> 1. No performance has been performed by the alpha topology. <ol style="list-style-type: none"> a) The system shows a message. 2. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.30: Use Case Description: *Retrieve all performances of an alpha topology.*

Name	Retrieve performances performing history of an alpha topology
Goal	The developer wants to retrieve all performances of an alpha topology over a period of time
Actor	Developer
Pre-Condition	The alpha topology already exists
Post-Condition	The performances are retrieved successfully
Post-Condition in Special Case	The performances are not retrieved
Normal Case	1. The developer retrieves all performances of an alpha topology over a period of time.
Special Cases	<ol style="list-style-type: none"> 1. No performance has been performed by the alpha topology. <ol style="list-style-type: none"> a) The system shows a message. 2. No performance has been performed by the alpha topology for a given period of time. <ol style="list-style-type: none"> a) The system shows a error message. 3. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.31: Use Case Description: *Retrieve performances performing history of an alpha topology.*

Name	An alpha topology performs a workload
Goal	The developer wants to perform a workload for an alpha topology

4.3 Use Case

Actor	Developer
Pre-Condition	The workload and alpha topology already exists
Post-Condition	The workload is performed successfully
Post-Condition in Special Case	The workload is not performed successfully
Normal Case	1. The developer makes an alpha topology performing a workload .
Special Cases	<ol style="list-style-type: none"> 1. When performing workload , the workload does not exist. <ol style="list-style-type: none"> a) The system shows an error message. 2. When performing workload ,the alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.32: Use Case Description: *An alpha topology performs a workload.*

Name	Retrieve a workload of an alpha topology
Goal	The developer wants to retrieve a workload of an alpha topology by ID
Actor	Developer
Pre-Condition	The alpha topology already exists, the workload has been performed
Post-Condition	The workload is retrieved successfully
Post-Condition in Special Case	The workload is not retrieved
Normal Case	1. The developer retrieves a workload of an alpha topology.
Special Cases	<ol style="list-style-type: none"> 1. The workload to be retrieved does not exist. <ol style="list-style-type: none"> a) The system shows an error message. 2. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.33: Use Case Description: *Retrieve a workload of an alpha topology.*

Name	Retrieve all workloads of an alpha topology
Goal	The developer wants to retrieve all workloads of an alpha topology
Actor	Developer
Pre-Condition	The alpha topology already exists
Post-Condition	The workloads are retrieved successfully

Post-Condition in Special Case	The workloads are not retrieved
Normal Case	1. The developer retrieves all workloads of an alpha topology.
Special Cases	<ol style="list-style-type: none"> 1. No workload has been performed by the alpha topology. <ol style="list-style-type: none"> a) The system shows a message. 2. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.34: Use Case Description: *Retrieve all workloads of an alpha topology.*

Name	Retrieve workloads performing history of an alpha topology
Goal	The developer wants to retrieve all workloads of an alpha topology over a period of time
Actor	Developer
Pre-Condition	The alpha topology already exists
Post-Condition	The workloads are retrieved successfully
Post-Condition in Special Case	The workloads are not retrieved
Normal Case	1. The developer retrieves all workloads of an alpha topology over a period of time.
Special Cases	<ol style="list-style-type: none"> 1. No workload has been performed by the alpha topology. <ol style="list-style-type: none"> a) The system shows a message. 2. No workload has been performed by the alpha topology for a given period of time. <ol style="list-style-type: none"> a) The system shows a error message. 3. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.35: Use Case Description: *Retrieve workloads performing history of an alpha topology.*

Name	Retrieve viable topology of an alpha topology
Goal	The developer wants to retrieve valid viable topology of an alpha topology over a period of time
Actor	Developer

4.3 Use Case

Pre-Condition	The alpha topology already exists
Post-Condition	The viable topology is retrieved successfully
Post-Condition in Special Case	The viable topology is not retrieved
Normal Case	1. The developer retrieves the valid viable topology of an alpha topology over a period of time.
Special Cases	<ol style="list-style-type: none"> 1. No viable topology was discovered of the alpha topology. <ol style="list-style-type: none"> a) The system shows a message. 2. No viable topology is existed for a given period of time. <ol style="list-style-type: none"> a) The system shows a error message. 3. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.36: Use Case Description: *Retrieve viable topology of an alpha topology.*

Name	Retrieve all viable topologies of an alpha topology
Goal	The developer wants to retrieve all viable topology(valid and obsolete) of an alpha topology
Actor	Developer
Pre-Condition	The alpha topology already exists
Post-Condition	The viable topology is retrieved successfully
Post-Condition in Special Case	The viable topology is not retrieved
Normal Case	1. The developer retrieves all viable topology(valid and obsolete) of an alpha topology.
Special Cases	<ol style="list-style-type: none"> 1. No viable topology was discovered of the alpha topology. <ol style="list-style-type: none"> a) The system shows a message. 2. The alpha topology does not exist. <ol style="list-style-type: none"> a) The system shows an error message.

Table 4.37: Use Case Description: *Retrieve all viable topologies of an alpha topology.*

4.4 System Overview

In Figure 4.9, topology persistence and discovery system with two main parts is presented. From bottom to top, the first part is the storage for topologies, workloads and performances. As topology itself is a graph, more than that, topology can be enriched by evolving workloads and KPIs, so a graph database is used to speed up the accessing.

The second part of the system is topology persistence and discovery framework which provides interfaces for external users to perform CRUD and other particular operations of topologies, workloads and performances. Topology developer can use existing topology modeling framework to access topology persistence and discovery system for persisting, retrieving and discovering topology. Then topology is provided to topology provisioning system for further processing.

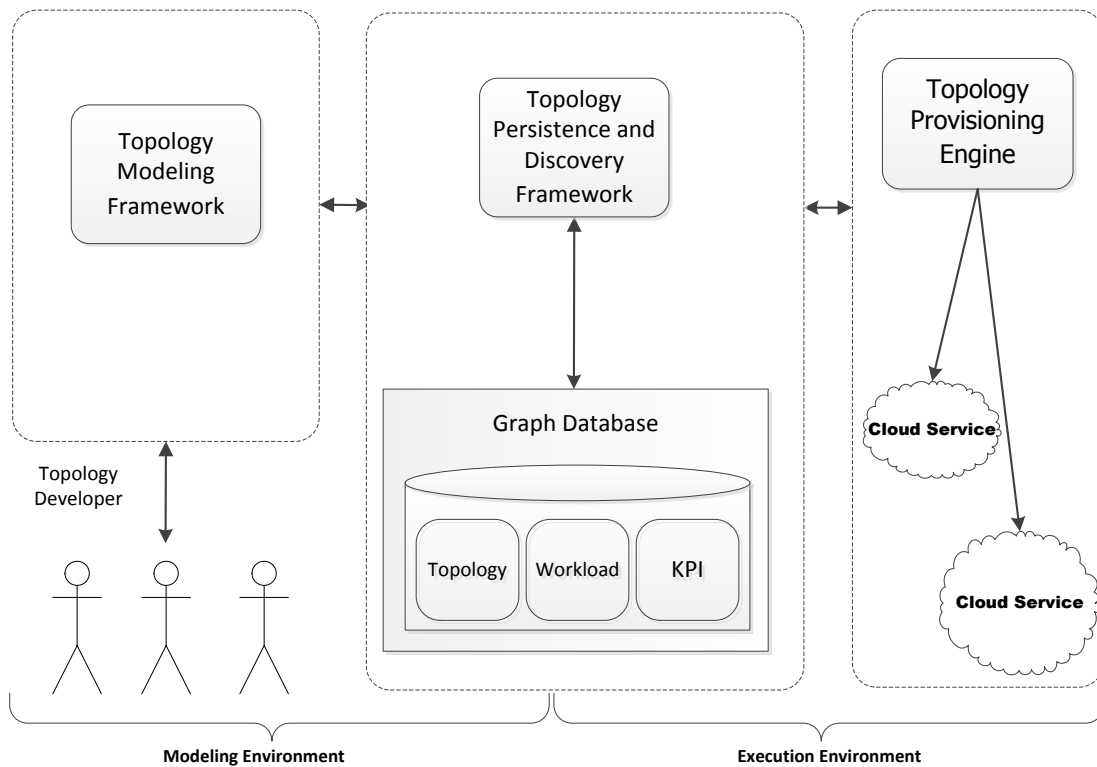


Figure 4.9: System Overview

5 Design

5.1 System Architecture

In this chapter the topology persistence and discovery system is designed. This system provides a unified persistence for cloud application topology complied to different specifications and offers unified interfaces to perform operations on topologies, performance(KPI) and workloads.

Figure 5.1 presents the architecture of topology persistence and discovery system. This is a three-layered system which consists of REST Interface layer, business logical layer and data storage layer. In the following the three layers are described in details from the perspective of designing, respectively:

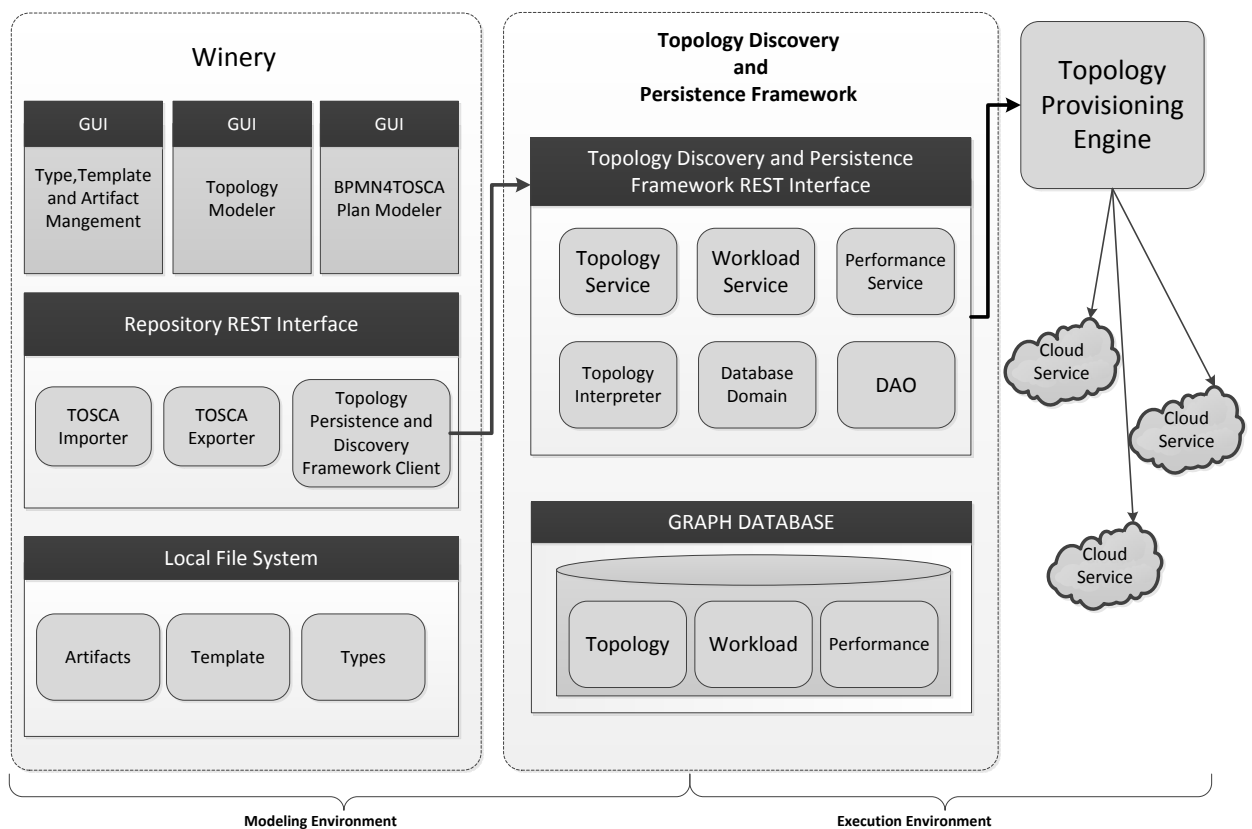


Figure 5.1: System Architecture

1. *Graph Database Layer*: This layer is persistence layer. Topology and its enrichments are persisted in graph database as modeled in Chapter 4.
2. *Service Layer*: There are six sub-modules in service layer which can be divided into four categories described as following. Figure 5.2 shows the work flow within service layer.
 - a) *Interpreter*: This module is used to parse representation of workload, performance and topologies comes from topology modeler. The Interpreter extracts useful information which is interested by service logic module and passes this information to it; on the other hand, this module translates the data from database when performing operations on it, and sends it back to Topology Modeler with per-defined XML presentation.
 - b) *Business Logic*(Topology, Workload, Performance): The real business logic module. There are three sub-modules: topology logic, workload logic and performance(KPI) logic. Each sub-module accepts the data comes from Interpreter and process it according to the requirement described in previous chapter.
 - c) *Database Domain*: After data is processed by service module, data is transformed to database domain as the data models designed in Chapter 4. The data format of domain is the one which graph database can easily persist, modify and retrieve.
 - d) *DAO*: Data Access Object module, is a standard module which is used in many application design. DAO plays a role to access database. All operations of database are implemented in this module. By separating business logic and physical operation of database, DAO module can be reused in further for application extending.
3. *REST Interface Layer*: REST interface layer provides Restful API to outside world. Details of this layer are presented in following section *REST Interface Design*.

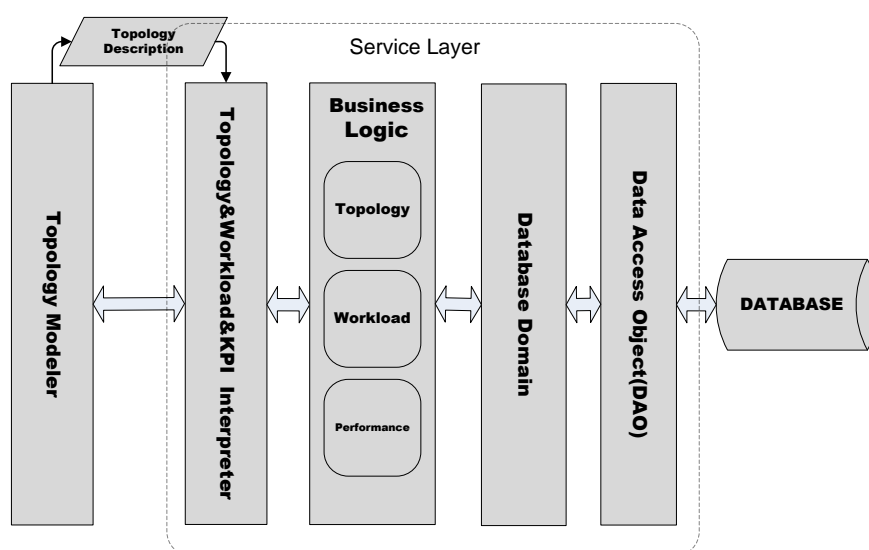


Figure 5.2: Work flow within Service Layer

5.2 REST Interface Design

In this section, we design the REST API. The three steps of designing RESTful API described in [VB15] is followed, which are: resource identification, resource representation, endpoint identification and action identification.

5.2.1 Resource Identification

By analyzing requirements listed in section 4.2, here we first identify the resource.

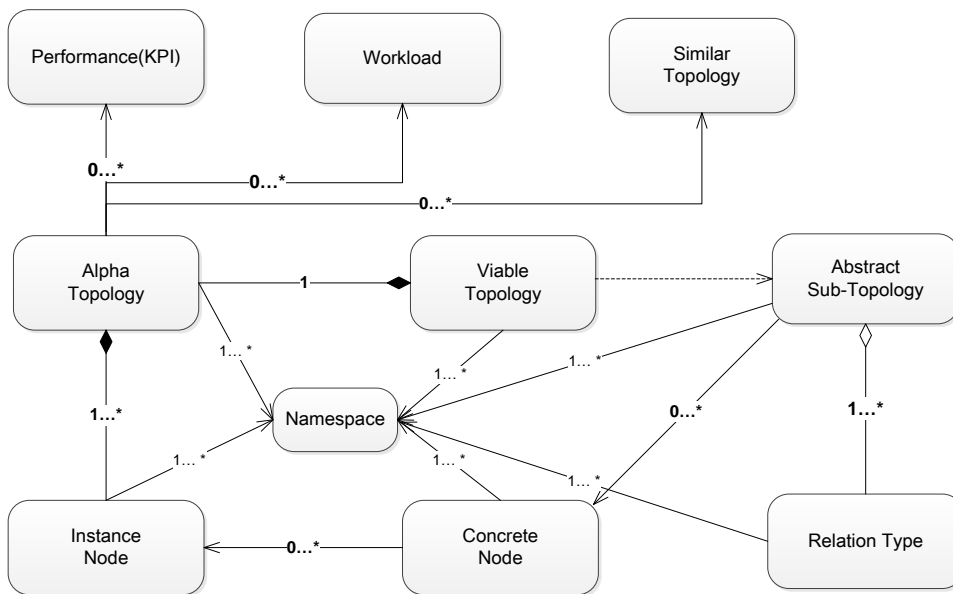


Figure 5.3: Resource Modeling

Figure 5.3 models the resource entity used in the REST API and relationship to other resources.

1. An alpha topology is composed of instance nodes and can perform multiple KPIs or Workloads.
2. An abstract sub-topology is composed of abstract nodes¹ and relation type which define the relations among abstract nodes. Each abstract node can be refined by concrete node and each concrete node can be referred by instance node as described in 4.1.2
3. A viable topology is discovered by giving alpha topology and depends on corresponding abstract sub-topology with its linked concrete nodes and instance nodes.
4. By giving an alpha topology, its similar topologies can be retrieved.

¹abstract sub-topology is not fine grained to the level of abstract nodes for REST resource, but it does for database modeling, refer to 4.1.3

5. Namespace provides the mapping between URL and prefix which are used by other entities.

As discussed above, table 5.1 lists all resource used in REST API.

Resource	Description
AlphaTopology	alpha topology resource
AbstractSubTopology	abstract sub-topology resource
ViableTopology	viable topology resource
SimilarTopology	similar topology resource
Performance(KPI)	performance(KPI) resource
Workload	workload resource
ConcreteNode	concrete node resource
InstanceNode	instance node resource
RelationshipType	relationship type resource
Namespace	namespace resource

Table 5.1: Resources for Topology Persistence and Discovery System

5.2.2 Resource Representation

The next step in the REST API design process is to define resource representations. REST APIs typically support multiple formats such as HTML, JSON, and XML. As TOSCA is the topology description language we are using in implementation part, so here we choose XML as preferred format for resource representation.

Alpha Topology Resource Representation

As showing in list 5.1, this is a XML schema when persisting an alpha topology modeled in 4.1.3. Within root element *AlphaTopologyTemplate* there are three elements:

1. `specificationType`: indicate the format of the specification. For instance, if TOSCA is chose, then the `specificationType` is *xml*.
2. `specification`: the real topology description specification is wrapped here. Within it the sub-element *Definitions* extends TOSCA *tDefinitions*. Here *ServiceTemplate* defined in TOSCA should be used for an alpha topology resource representation.
3. `nodelevel`: indicate the level of each node.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
   oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
   targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0"
   ">
2 <xs:element name="documentation" type="tDocumentation"/>
3 <xs:element name="AlphaTopologyTemplate">
```

```

4 <xs:complexType>
5 <xs:sequence>
6   <xs:element name="specificationType" type="xs:string" />
7   <xs:element name="specification">
8     <xs:complexType>
9       <xs:sequence>
10        <xs:element name="tns:Definitions">
11          <xs:complexType>
12            <xs:complexContent>
13              <xs:extension base="tDefinitions"/>
14            </xs:complexContent>
15          </xs:complexType>
16        </xs:element>
17      </xs:sequence>
18    </xs:complexType>
19  </xs:element>
20  <xs:element name="nodelevel">
21    <xs:complexType>
22      <xs:sequence>
23        <xs:element name="node" maxOccurs="unbounded" minOccurs="0">
24          <xs:complexType>
25            <xs:sequence>
26              <xs:element type="xs:string" name="level"/>
27            </xs:sequence>
28            <xs:attribute type="xs:string" name="id" use="required"/>
29          </xs:complexType>
30        </xs:element>
31      </xs:sequence>
32    </xs:complexType>
33  </xs:element>
34 </xs:sequence>
35 </xs:complexType>
36 </xs:element>
37 </xs:schema>

```

Listing 5.1: XML schema for persisting an Alpha Topology

List 5.2 presents XML schema when retrieving alpha topologies. The element *Definitions* is defined in TOSCA specification, the schema of which can be found at [Oas15b]. The attribute *DatabaseId* is the system generated ID of alpha topology.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
   oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
   targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0

```

```

    ">
2  <xs:element name="AlphaTopologList">
3    <xs:complexType>
4      <xs:sequence>
5        <xs:element maxOccurs="unbounded" minOccurs="0" name="specification"
          nillable="true">
6          <xs:complexType>
7            <xs:sequence>
8              <xs:element ref="tns:Definitions"/>
9            </xs:sequence>
10           <xs:attribute name="DatabaseId" type="xs:long" use="required"/>
11         </xs:complexType>
12       </xs:element>
13     </xs:sequence>
14   </xs:complexType>
15 </xs:element>

```

Listing 5.2: XML schema for retrieving Alpha Topologies

Abstract Sub-Topology Resource Representation

As showing in list 5.3, this is a XML schema when persisting an abstract sub-topology as modeled in 4.1.3. Within the root element *AbstractSubTopology*, there are two sub-elements:

1. AbstractNode: define the abstract nodes of abstract sub-topology.
2. RelationshipOfAbstractNode: define the relationships among abstract nodes.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
  oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
  targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0"
  ">
2 <xs:element name="AbstractSubTopology">
3   <xs:complexType>
4     <xs:sequence>
5       <xs:element name="AbstractNode" maxOccurs="unbounded" minOccurs="0">
6         <xs:complexType>
7           <xs:sequence>
8             <xs:element type="xs:byte" name="level"/>
9           </xs:sequence>
10          <xs:attribute type="xs:string" name="name" use="optional"/>
11          <xs:attribute type="xs:string" name="id" use="optional"/>
12          <xs:attribute type="xs:string" name="type" use="optional"/>
13        </xs:complexType>

```

```
14     </xs:element>
15     <xs:element name="RelationshipOfAbstractNode" maxOccurs="unbounded"
16         minOccurs="0">
17         <xs:complexType>
18             <xs:sequence>
19                 <xs:element name="SourceElement">
20                     <xs:complexType>
21                         <xs:simpleContent>
22                             <xs:extension base="xs:string">
23                                 <xs:attribute type="xs:string" name="ref" use="optional"/>
24                             </xs:extension>
25                         </xs:simpleContent>
26                     </xs:complexType>
27                 </xs:element>
28                 <xs:element name="TargetElement">
29                     <xs:complexType>
30                         <xs:simpleContent>
31                             <xs:extension base="xs:string">
32                                 <xs:attribute type="xs:string" name="ref" use="optional"/>
33                             </xs:extension>
34                         </xs:simpleContent>
35                     </xs:complexType>
36                 </xs:element>
37             </xs:sequence>
38             <xs:attribute type="xs:string" name="type" use="optional"/>
39         </xs:complexType>
40     </xs:element>
41     <xs:attribute type="xs:string" name="name"/>
42     <xs:attribute type="xs:string" name="id"/>
43 </xs:complexType>
44 </xs:element>
45 </xs:schema>
```

Listing 5.3: XML schema for abstract sub-Topology

Viable Topology Resource Representation

When persisting a viable topology, *tServiceTemplae* defined in TOSCA specification is used directly here, the schema locates at [Oas15b].

List 5.4 presents XML schema when retrieving viable topologies:

1. alphaTopologyId: indicate alpha topology ID of this viable topology.

2. alphaTopologyName: indicate alpha topology name of this viable topology.
3. alphaTopologyNameSpace: indicate alpha topology namespace of this viable topology.
4. definitions: real topology description specification is wrapped here. Here *ServiceTemplate* defined in TOSCA is directly used for a viable topology resource representation.
5. obsolete: indicate whether this viable topology is currently used.
6. createDate: indicate the create data of this viable topology.
7. endDate: if this viable topology is obsolete, this attribute saves the ended date.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
  oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
  targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0
  ">
2 <xs:element name="ViableTopologyList">
3   <xs:complexType>
4     <xs:sequence>
5       <xs:element maxOccurs="unbounded" minOccurs="0" name="
        viableTopologyWithDatabaseID" nillable="true">
6         <xs:complexType>
7           <xs:sequence>
8             <xs:element form="unqualified" name="viableTopology">
9               <xs:complexType>
10                <xs:sequence>
11                  <xs:element name="alphaTopologyId" type="xs:string"/>
12                  <xs:element name="alphaTopologyName" type="xs:string"/>
13                  <xs:element name="alphaTopologyNameSpace" type="xs:string"/
14                  >
15                  <xs:element ref="tns:Definitions"/>
16                  <xs:element name="obsolete" type="xs:string"/>
17                  <xs:element name="createDate" type="xs:string"/>
18                  <xs:element name="endDate" type="xs:string"/>
19                </xs:sequence>
20              </xs:complexType>
21            </xs:element>
22          </xs:sequence>
23          <xs:attribute name="databaseId" type="xs:long" use="required"/>
24        </xs:complexType>
25      </xs:element>
26    </xs:sequence>
27  </xs:complexType>
28 </xs:element>
</xs:schema>

```

Listing 5.4: XML schema for retrieving viable topologies

Similar Topology Resource Representation

As showing in list 5.5, this is a XML schema when querying all similar alpha topologies for a given alpha topology. Within root element *SmilarAlphaTopologyList* , exist the list of matching similar alpha topology founded by system.

1. specification: specification wraps TOSCA definition for one alpha topology, the attribute *alphaTopologyId* indicate the alpha topology id in database .

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified" targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0">
2   <xs:element name="SmilarAlphaTopologyList">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element maxOccurs="unbounded" minOccurs="0" name="specification" nillable="true">
6           <xs:complexType>
7             <xs:sequence>
8               <xs:element ref="tns:Definitions"/>
9             </xs:sequence>
10            <xs:attribute name="alphaTopologyId" type="xs:long" use="required"/>
11          </xs:complexType>
12        </xs:element>
13      </xs:sequence>
14    </xs:complexType>
15  </xs:element>
16 </xs:schema>
```

Listing 5.5: XML schema for discovering similar alpha topologies

Performance(KPI) Resource Representation

As showing in list 5.6, this is the XML schema of performance proposed in [Nie16], which represents performance(KPI) modeled in 4.6. The root element is *Performance*. The attributes *endTime* and *startTime* indicate the valid time period of this performance. Like element *response_time*, for each element there are four sub-elements inside, they are *min,max,avg,st*.

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
2   <xs:element name="Performance">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="time_behaviour">
6           <xs:complexType>
7             <xs:sequence>
8               <xs:element name="response_time">
9                 <xs:complexType>
10                  <xs:sequence>
11                    <xs:element name="min" type="xs:short"/>
12                    <xs:element name="max" type="xs:short"/>
13                    <xs:element name="avg" type="xs:short"/>
14                    <xs:element name="st" type="xs:short"/>
15                  </xs:sequence>
16                </xs:complexType>
17              </xs:element>
18              <xs:element name="throughput">
19                <xs:complexType>
20                  <xs:sequence>
21                    <xs:element name="min" type="xs:short"/>
22                    <xs:element name="max" type="xs:short"/>
23                    <xs:element name="avg" type="xs:short"/>
24                    <xs:element name="st" type="xs:short"/>
25                  </xs:sequence>
26                </xs:complexType>
27              </xs:element>
28              <xs:element name="processing_time">
29                <xs:complexType>
30                  <xs:sequence>
31                    <xs:element name="min" type="xs:short"/>
32                    <xs:element name="max" type="xs:short"/>
33                    <xs:element name="avg" type="xs:short"/>
34                    <xs:element name="st" type="xs:short"/>
35                  </xs:sequence>
36                </xs:complexType>
37              </xs:element>
38              <xs:element name="avg_read_speed">
39                <xs:complexType>
40                  <xs:sequence>
41                    <xs:element name="min" type="xs:short"/>
42                    <xs:element name="max" type="xs:short"/>
43                    <xs:element name="avg" type="xs:short"/>
44                    <xs:element name="st" type="xs:short"/>
```



```
45         </xs:sequence>
46     </xs:complexType>
47 </xs:element>
48 <xs:element name="avg_write_speed">
49     <xs:complexType>
50         <xs:sequence>
51             <xs:element name="min" type="xs:short"/>
52             <xs:element name="max" type="xs:byte"/>
53             <xs:element name="avg" type="xs:byte"/>
54             <xs:element name="st" type="xs:byte"/>
55         </xs:sequence>
56     </xs:complexType>
57 </xs:element>
58 <xs:element name="avg_migration_time">
59     <xs:complexType>
60         <xs:sequence>
61             <xs:element name="min" type="xs:byte"/>
62             <xs:element name="max" type="xs:byte"/>
63             <xs:element name="avg" type="xs:byte"/>
64             <xs:element name="st" type="xs:byte"/>
65         </xs:sequence>
66     </xs:complexType>
67 </xs:element>
68 <xs:element name="latency">
69     <xs:complexType>
70         <xs:sequence>
71             <xs:element name="min" type="xs:byte"/>
72             <xs:element name="max" type="xs:byte"/>
73             <xs:element name="avg" type="xs:byte"/>
74             <xs:element name="st" type="xs:byte"/>
75         </xs:sequence>
76     </xs:complexType>
77 </xs:element>
78 <xs:element name="backup_time">
79     <xs:complexType>
80         <xs:sequence>
81             <xs:element name="min" type="xs:byte"/>
82             <xs:element name="max" type="xs:byte"/>
83             <xs:element name="avg" type="xs:byte"/>
84             <xs:element name="st" type="xs:byte"/>
85         </xs:sequence>
86     </xs:complexType>
87 </xs:element>
88 </xs:sequence>
```

```
89     </xs:complexType>
90 </xs:element>
91 <xs:element name="capacity">
92   <xs:complexType>
93     <xs:sequence>
94       <xs:element name="bandwith">
95         <xs:complexType>
96           <xs:sequence>
97             <xs:element name="min" type="xs:byte"/>
98             <xs:element name="max" type="xs:byte"/>
99             <xs:element name="avg" type="xs:byte"/>
100            <xs:element name="st" type="xs:byte"/>
101          </xs:sequence>
102        </xs:complexType>
103      </xs:element>
104      <xs:element name="processor_speed">
105        <xs:complexType>
106          <xs:sequence>
107            <xs:element name="min" type="xs:byte"/>
108            <xs:element name="max" type="xs:byte"/>
109            <xs:element name="avg" type="xs:byte"/>
110            <xs:element name="st" type="xs:byte"/>
111          </xs:sequence>
112        </xs:complexType>
113      </xs:element>
114      <xs:element name="storage_size">
115        <xs:complexType>
116          <xs:sequence>
117            <xs:element name="min" type="xs:byte"/>
118            <xs:element name="max" type="xs:byte"/>
119            <xs:element name="avg" type="xs:byte"/>
120            <xs:element name="st" type="xs:byte"/>
121          </xs:sequence>
122        </xs:complexType>
123      </xs:element>
124      <xs:element name="memory_allocation_vm">
125        <xs:complexType>
126          <xs:sequence>
127            <xs:element name="min" type="xs:byte"/>
128            <xs:element name="max" type="xs:byte"/>
129            <xs:element name="avg" type="xs:byte"/>
130            <xs:element name="st" type="xs:byte"/>
131          </xs:sequence>
132        </xs:complexType>
```

```
133     </xs:element>
134     <xs:element name="number_vm">
135         <xs:complexType>
136             <xs:sequence>
137                 <xs:element name="min" type="xs:byte"/>
138                 <xs:element name="max" type="xs:byte"/>
139                 <xs:element name="avg" type="xs:byte"/>
140                 <xs:element name="st" type="xs:byte"/>
141             </xs:sequence>
142         </xs:complexType>
143     </xs:element>
144     <xs:element name="number_processors">
145         <xs:complexType>
146             <xs:sequence>
147                 <xs:element name="min" type="xs:byte"/>
148                 <xs:element name="max" type="xs:byte"/>
149                 <xs:element name="avg" type="xs:byte"/>
150                 <xs:element name="st" type="xs:byte"/>
151             </xs:sequence>
152         </xs:complexType>
153     </xs:element>
154     <xs:element name="io_operations">
155         <xs:complexType>
156             <xs:sequence>
157                 <xs:element name="min" type="xs:byte"/>
158                 <xs:element name="max" type="xs:byte"/>
159                 <xs:element name="avg" type="xs:byte"/>
160                 <xs:element name="st" type="xs:byte"/>
161             </xs:sequence>
162         </xs:complexType>
163     </xs:element>
164 </xs:sequence>
165 </xs:complexType>
166 </xs:element>
167 <xs:element name="resource_utilization">
168     <xs:complexType>
169         <xs:sequence>
170             <xs:element name="network_utilization">
171                 <xs:complexType>
172                     <xs:sequence>
173                         <xs:element name="min" type="xs:byte"/>
174                         <xs:element name="max" type="xs:byte"/>
175                         <xs:element name="avg" type="xs:byte"/>
176                         <xs:element name="st" type="xs:byte"/>
```

```
177         </xs:sequence>
178     </xs:complexType>
179 </xs:element>
180 <xs:element name="memory_utilization">
181     <xs:complexType>
182         <xs:sequence>
183             <xs:element name="min" type="xs:byte"/>
184             <xs:element name="max" type="xs:byte"/>
185             <xs:element name="avg" type="xs:byte"/>
186             <xs:element name="st" type="xs:byte"/>
187         </xs:sequence>
188     </xs:complexType>
189 </xs:element>
190 <xs:element name="disk_utilization">
191     <xs:complexType>
192         <xs:sequence>
193             <xs:element name="min" type="xs:byte"/>
194             <xs:element name="max" type="xs:byte"/>
195             <xs:element name="avg" type="xs:byte"/>
196             <xs:element name="st" type="xs:byte"/>
197         </xs:sequence>
198     </xs:complexType>
199 </xs:element>
200 <xs:element name="cpu_utilization">
201     <xs:complexType>
202         <xs:sequence>
203             <xs:element name="min" type="xs:byte"/>
204             <xs:element name="max" type="xs:byte"/>
205             <xs:element name="avg" type="xs:byte"/>
206             <xs:element name="st" type="xs:byte"/>
207         </xs:sequence>
208     </xs:complexType>
209 </xs:element>
210 <xs:element name="vm_utilization">
211     <xs:complexType>
212         <xs:sequence>
213             <xs:element name="min" type="xs:byte"/>
214             <xs:element name="max" type="xs:byte"/>
215             <xs:element name="avg" type="xs:byte"/>
216             <xs:element name="st" type="xs:byte"/>
217         </xs:sequence>
218     </xs:complexType>
219 </xs:element>
220 <xs:element name="number_vm_perserver">
```

```
221         <xs:complexType>
222             <xs:sequence>
223                 <xs:element name="min" type="xs:byte"/>
224                 <xs:element name="max" type="xs:byte"/>
225                 <xs:element name="avg" type="xs:byte"/>
226                 <xs:element name="st" type="xs:byte"/>
227             </xs:sequence>
228         </xs:complexType>
229     </xs:element>
230 </xs:sequence>
231 </xs:complexType>
232 </xs:element>
233 <xs:element name="scalability">
234     <xs:complexType>
235         <xs:sequence>
236             <xs:element name="resource_acquisition_time">
237                 <xs:complexType>
238                     <xs:sequence>
239                         <xs:element name="min" type="xs:byte"/>
240                         <xs:element name="max" type="xs:byte"/>
241                         <xs:element name="avg" type="xs:byte"/>
242                         <xs:element name="st" type="xs:byte"/>
243                     </xs:sequence>
244                 </xs:complexType>
245             </xs:element>
246             <xs:element name="resource_provisioning_time">
247                 <xs:complexType>
248                     <xs:sequence>
249                         <xs:element name="min" type="xs:byte"/>
250                         <xs:element name="max" type="xs:byte"/>
251                         <xs:element name="avg" type="xs:byte"/>
252                         <xs:element name="st" type="xs:byte"/>
253                     </xs:sequence>
254                 </xs:complexType>
255             </xs:element>
256             <xs:element name="deployment_time">
257                 <xs:complexType>
258                     <xs:sequence>
259                         <xs:element name="min" type="xs:byte"/>
260                         <xs:element name="max" type="xs:byte"/>
261                         <xs:element name="avg" type="xs:byte"/>
262                         <xs:element name="st" type="xs:byte"/>
263                     </xs:sequence>
264                 </xs:complexType>
```

```
265     </xs:element>
266     <xs:element name="resource_release_time">
267         <xs:complexType>
268             <xs:sequence>
269                 <xs:element name="min" type="xs:byte"/>
270                 <xs:element name="max" type="xs:byte"/>
271                 <xs:element name="avg" type="xs:byte"/>
272                 <xs:element name="st" type="xs:byte"/>
273             </xs:sequence>
274         </xs:complexType>
275     </xs:element>
276     <xs:element name="vm_startup_time">
277         <xs:complexType>
278             <xs:sequence>
279                 <xs:element name="min" type="xs:byte"/>
280                 <xs:element name="max" type="xs:byte"/>
281                 <xs:element name="avg" type="xs:byte"/>
282                 <xs:element name="st" type="xs:byte"/>
283             </xs:sequence>
284         </xs:complexType>
285     </xs:element>
286 </xs:sequence>
287 </xs:complexType>
288 </xs:element>
289 <xs:element name="availability">
290     <xs:complexType>
291         <xs:sequence>
292             <xs:element name="cloud_service_uptime">
293                 <xs:complexType>
294                     <xs:sequence>
295                         <xs:element name="min" type="xs:byte"/>
296                         <xs:element name="max" type="xs:byte"/>
297                         <xs:element name="avg" type="xs:byte"/>
298                         <xs:element name="st" type="xs:byte"/>
299                     </xs:sequence>
300                 </xs:complexType>
301             </xs:element>
302             <xs:element name="cloud_resource_uptime">
303                 <xs:complexType>
304                     <xs:sequence>
305                         <xs:element name="min" type="xs:byte"/>
306                         <xs:element name="max" type="xs:byte"/>
307                         <xs:element name="avg" type="xs:byte"/>
308                         <xs:element name="st" type="xs:byte"/>
```

```
309         </xs:sequence>
310     </xs:complexType>
311 </xs:element>
312 <xs:element name="meantime_between_failures">
313     <xs:complexType>
314         <xs:sequence>
315             <xs:element name="min" type="xs:byte"/>
316             <xs:element name="max" type="xs:byte"/>
317             <xs:element name="avg" type="xs:byte"/>
318             <xs:element name="st" type="xs:byte"/>
319         </xs:sequence>
320     </xs:complexType>
321 </xs:element>
322 <xs:element name="meantime_repair">
323     <xs:complexType>
324         <xs:sequence>
325             <xs:element name="min" type="xs:byte"/>
326             <xs:element name="max" type="xs:byte"/>
327             <xs:element name="avg" type="xs:byte"/>
328             <xs:element name="st" type="xs:byte"/>
329         </xs:sequence>
330     </xs:complexType>
331 </xs:element>
332 </xs:sequence>
333 </xs:complexType>
334 </xs:element>
335 </xs:sequence>
336 <xs:attribute name="id" type="xs:string"/>
337 <xs:attribute name="startTime" type="xs:string"/>
338 <xs:attribute name="endTime" type="xs:string"/>
339 </xs:complexType>
340 </xs:element>
341
342 </xs:schema>
```

Listing 5.6: XML schema for performance

An alpha topology can be enriched by performing a performance(KPI), following XML schema showing in list 5.7 is used. The element *id* is database ID of performance(KPI) persisted in database. With the URL in table 5.8, an alpha topology can perform KPI.

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
2   <xs:element name="PerformPerformanceID">
3     <xs:complexType>
```

```

4     <xs:sequence>
5         <xs:element name="id" type="xs:long"/>
6     </xs:sequence>
7 </xs:complexType>
8 </xs:element>
9 </xs:schema>

```

Listing 5.7: XML presentation for performing performance(KPI)

Workload Resource Representation

As showing in list 5.8, this is the XML schema of workload modeled in figure4.7, which is proposed in [Nie16]. The root element is *Workload*, the attributes *endTime* and *startTime* indicate the valid time period of this workload.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
2   <xs:element name="Workload">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="pattern" type="xs:string"/>
6         <xs:element name="arrival" type="xs:string"/>
7         <xs:element name="behavioral" type="xs:string"/>
8         <xs:element name="avg_users" type="xs:short"/>
9         <xs:element name="avg_transactions" type="xs:short"/>
10      </xs:sequence>
11      <xs:attribute name="id" type="xs:string"/>
12      <xs:attribute name="startTime" type="xs:string"/>
13      <xs:attribute name="endTime" type="xs:string"/>
14    </xs:complexType>
15  </xs:element>
16 </xs:schema>

```

Listing 5.8: XML presentation for persisting workload

An alpha topology can be enriched by performing a workload, following XML representation showing in list 5.9 is used. The element *id* is database ID of workload persisted in database. With the URL in table 5.8, an alpha topology can perform workload.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
2   <xs:element name="PerformWorkloadID">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="id" type="xs:long"/>

```



```
6     </xs:sequence>
7     </xs:complexType>
8 </xs:element>
9 </xs:schema>
```

Listing 5.9: XML presentation for performing workload

Concrete Node Resource Representation

When persisting a concrete node, *tNodeType* defined in TOSCA specification is used directly here. The schema locates at [Oas15b].

List 5.10 presents XML schema when retrieving concrete nodes. Concrete nodes are wrapped within element *ConcreteNodeList*. Attribute *DatabaseId* indicates the concrete node ID generated by database.

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
   oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
   targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0
   ">
2   <xs:element name="ConcreteNodeList">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element maxOccurs="unbounded" minOccurs="0" name="specification"
           nillable="true">
6           <xs:complexType>
7             <xs:sequence>
8               <xs:element ref="tns:Definitions"/>
9             </xs:sequence>
10            <xs:attribute name="DatabaseId" type="xs:long" use="required"/>
11          </xs:complexType>
12        </xs:element>
13      </xs:sequence>
14    </xs:complexType>
15  </xs:element>
16 </xs:schema>
```

Listing 5.10: XML schema for retrieving concrete nodes

Instance Node Resource Representation

When persisting an instance node, *tNodeTemplate* defined in TOSCA specification is used directly here. The schema locates at [Oas15b].

List 5.11 presents XML schema when retrieving instance nodes. Instance nodes are wrapped within element *InstanceNodeList*. Attribute *DatabaseId* indicates the instance node ID generated by database.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
  oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
  targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0
  ">
2 <xs:element name="InstanceNodeList">
3   <xs:complexType>
4     <xs:sequence>
5       <xs:element maxOccurs="unbounded" minOccurs="0" name="specification"
6         nillable="true">
7         <xs:complexType>
8           <xs:sequence>
9             <xs:element ref="tns:Definitions"/>
10            </xs:sequence>
11            <xs:attribute name="DatabaseId" type="xs:long" use="required"/>
12          </xs:complexType>
13        </xs:element>
14      </xs:sequence>
15    </xs:complexType>
16  </xs:element>
</xs:schema>

```

Listing 5.11: XML schema for retrieving instance nodes

RelationshipType Resource Representation

When persisting a relationshipType, *tRelationshipType* defined in TOSCA specification is used directly here. The schema locates at [Oas15b].

List 5.12 presents XML schema when retrieving relationshipTypes. RelationshipTypes are wrapped within element *RelationshipTypeList*. Attribute *DatabaseId* indicates the ID of relationshiptype generated by database.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://docs.
  oasis-open.org/tosca/ns/2011/12" elementFormDefault="qualified"
  targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12" version="1.0
  ">
2 <xs:element name="RelationshipTypeList">
3   <xs:complexType>
4     <xs:sequence>

```

```
5     <xs:element maxOccurs="unbounded" minOccurs="0" name="specification"
6         nillable="true">
7         <xs:complexType>
8             <xs:sequence>
9                 <xs:element ref="tns:Definitions"/>
10            </xs:sequence>
11            <xs:attribute name="DatabaseId" type="xs:long" use="required"/>
12        </xs:complexType>
13    </xs:element>
14 </xs:sequence>
15 </xs:complexType>
16 </xs:element>
17 </xs:schema>
```

Listing 5.12: XML schema for retrieving relationship type

Namespace Resource Representation

As showing in list 5.13, this is XML schema of namespace resource. Namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined in a W3C recommendation.

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
2   <xs:element name="NameSpace">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="prefix" type="xs:string"/>
6         <xs:element name="namespaceurl" type="xs:anyURI"/>
7       </xs:sequence>
8       <xs:attribute name="id" type="xs:string"/>
9     </xs:complexType>
10  </xs:element>
11 </xs:schema>
```

Listing 5.13: XML schema for NameSpace

5.2.3 Endpoint and Action Representation

A REST endpoint provides way to map a URI and HTTP method for accessing a resource. In this section, totally 45 endpoints for each resource is presented.

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/topology/abstractsubtopology	Body:abstract sub-topology as described in 5.3	Status:201 Body:empty	Status:500	Persists new abstract sub-topology
GET	/topology/abstractsubtopology/{id}	Body:Empty	Status:200 Body:alpha topology	Status:404	Retrieve an abstract sub-topology by database ID
GET	/topology/abstractsubtopology	Body:Empty	Status:200 Body:alpha topology list	Status:404	Retrieve all abstract sub-topologies

Table 5.2: Allowed operations for abstract sub-topology resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/concretenode/{type}	Body:concrete node as described in 5.2.2	Status:201 Body:empty	Status:500	Creates new concrete node
GET	/concretenode/{id}	Body:Empty	Status:200 Body:concrete node	Status:404	Retrieve one concrete node by database ID
GET	/concretenode	Body:Empty	Status:200 Body:concrete node List	Status:404	Retrieve all concrete nodes
GET	/concretenode/{concreteNodeId} /instancenode/{instanceNodeId}	Body:Empty	Status:200 Body:instance node	Status:404	Retrieve instance node which refers to this concrete node
GET	/concretenode/{concreteNodeId} /instancenode	Body:Empty	Status:200 Body:instance node list	Status:404	Retrieve all instance nodes which refer to this concrete node
DELETE	/concretenode/{id}	N/A	Status:202	Status:501	Delete a concrete node by database ID

Table 5.3: Allowed operations for concrete node resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/instancenode	Body:instance node as described in 5.2.2	Status:201 Body:empty	Status:500	Creates new instance node
GET	/instancenode/{ <i>id</i> }	Body:Empty	Status:200 Body:instance node	Status:404	Retrieve one instance node by database ID
GET	/instancenode	Body:Empty	Status:200 Body:instance node List	Status:404	Retrieve all instance nodes
DELETE	/instancenode/{ <i>id</i> }	N/A	Status:202	Status:501	Delete a instance node by database ID

Table 5.4: Allowed operations for instance node resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/workload	Body:workload as described in 5.8	Status:201 Body:empty	Status:500	Creates new workload
GET	/workload/{id}	Body:Empty	Status:200 Body:workload	Status:404	Retrieve one workload by database ID
GET	/workload	Body:Empty	Status:200 Body:workload List	Status:404	Retrieve all workloads
DELETE	/workload/{id}	N/A	Status:202	Status:501	Delete workload by database ID

Table 5.5: Allowed operations for workload resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/performance	Body:performance as described in 5.6	Status:201 Body:empty	Status:500	Creates new performance
GET	/performance/{id}	Body:Empty	Status:200 Body:performance	Status:404	Retrieve one performance by database ID
GET	/performance	Body:Empty	Status:200 Body:performance List	Status:404	Retrieve all performances
DELETE	/performance/{id}	N/A	Status:202	Status:501	Delete a performance by database ID

Table 5.6: Allowed operations for performance(KPI) resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
GET	/discoverytopology/{ <i>alphaTopologyID</i> }	Body:Empty	Status:200 Body:viable topology list	Status:404	discover all viable topologies by an alpha topology database ID
POST	/viabletopology	Body:viable topology as described in ??	Status:201 Body:empty	Status:500	Persist a discovered viable topology, the viable topology will be automatically linked to corresponding alpha topology

Table 5.7: Allowed operations for viable topology resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/topology/alphatopology	Body:alpha topology as described in 4.5	Status:200 Body:empty	Status:501	Persist an alpha topology
GET	/topology/alphatopology/{id}	Body:Empty	Status:200 Body:alpha topology	Status:404	Retrieve one alpha topology by database ID
GET	/topology/alphatopology	Body:Empty	Status:200 Body:alpha topology List	Status:404	Retrieve all alpha topologies
DELETE	/topology/alphatopology/{id}	N/A	Status:202	Status:501	Delete an alpha topology by database ID
GET	/topology/alphatopology/{alphaTopologyID}/viabletopology/{viableTopologyID}	Body:Empty	Status:200 Body:viable topology	Status:404	Retrieve one viable topology generated according to an alpha topology
GET	/topology/alphatopology/{alphaTopologyID}/viabletopology	Body:Empty	Status:200 Body:viable topology list	Status:404	Retrieve all viable topologies generated according to an alpha topology
GET	/topology/alphatopology/{alphaTopologyID}/viabletopology/?from={timeStamp}&to={timeStamp}	Body:Empty	Status:200 Body:viable topology list	Status:404	query viable topologies generated according to an alpha topology for a given time period

Table 5.8: Allowed operations for alpha topology resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/topology/alphatopology/ { <i>alphaTopologyID</i> }/performance	Body:performance id as described in 5.7	Status:201 Body:empty	Status:501	An alpha topology performs a performance query and retrieve all performance of an alpha topology performed during a given time period
GET	/topology/alphatopology/ { <i>alphaTopologyID</i> }/performance ?from={ <i>timeStamp</i> }& to={ <i>timeStamp</i> }	Body:Empty	Status:200 Body:performance list	Status:404	Retrieve all performances of an alpha topology performed
POST	/topology/alphatopology/ { <i>alphaTopologyID</i> }/workload	Body:workload id as described in 5.9	Status:201 Body:empty	Status:501	An alpha topology performs a workload query and retrieve all workloads of an alpha topology performed during a given time period
GET	/topology/alphatopology/ { <i>alphaTopologyID</i> }/workload ?from={ <i>timeStamp</i> }& to={ <i>timeStamp</i> }	Body:Empty	Status:200 Body:workload list	Status:404	Retrieve all workloads of an alpha topology performed
GET	/topology/alphatopology/ { <i>alphaTopologyID</i> }/workload	Body:Empty	Status:200 Body:workload list	Status:404	Retrieve all workloads of an alpha topology performed

Table 5.9: Allowed operations for alpha topology resource cont.

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/relationshiptype	Body:relationship type as described in 5.2.2	Status:201 Body:empty	Status:500	persist a new relationship type
GET	/relationshiptype/{id}	Body:Empty	Status:200 Body:workload	Status:404	Retrieve one relationship type by database ID
GET	/relationshiptype	Body:Empty	Status:200 Body:relationship type List	Status:404	Retrieve all relationship types
DELETE	/relationshiptype/{id}	N/A	Status:202	Status:501	Delete a relationship type by database ID

Table 5.10: Allowed operations for relationship type resource

HTTP Method	Resource URI	Input	Success Response	Error Response	Description
POST	/namespace	Body:namespace as described in 5.13	Status:201 Body:empty	Status:500	persist a new namespace
GET	/namespace/{id}	Body:Empty	Status:200 Body:namespace	Status:404	Retrieve one namespace by database ID
GET	/namespace	Body:Empty	Status:200 Body:namespace type List	Status:404	Retrieve all namespaces
DELETE	/namespacetype/{id}	N/A	Status:202	Status:501	Delete a namespace type by database ID

Table 5.11: Allowed operations for namespace resource

5.3 Core Algorithm

In this section, two algorithms: discovering potential viable topologies based on abstract sub-topology and finding similar topology are proposed.

5.3.1 Viable Topology Discovery

Algorithm 1 is composed of two parts: the first part as showing in algorithm 1. In this algorithm, a box is a container which contains elements of type T (like $List<T>$ in Java). T represents generic type. By using recursion, it finds all combinations of elements with type T. Each time one element is selected from one box, neither the number of elements in one box nor the number of boxes is unknown. For instance, If there are two boxes which contain elements of type T, and T is String, box1: "A","B" and box2: "C","D". Then all possible combinations will be AC, AD, BC and BD.

Algorithm 1 Generate all combination

```

function GENERATECOMBINATION(boxes, oneCombination)
  oneBox = boxes[0]                                ▷ the first box in boxes
  for element : all elements in oneBox do
    newBoxes = boxes
    newBoxes.remove(oneBox)
    if boxes.size()>1 then
      GENERATECOMBINATION(newBoxes, oneCombination.add(element))
    else
      oneCombination.add(element)
      allCombinations.add(oneCombination)          ▷ the final result
    end if
  end for
end function

```

As discussed in previous chapter 4.1.3, γ -topology is a directed graph and the abstract sub-topology we are using to model γ -topology is directed graph as well.

Let $G = (V, E)$ and $v \in V$. The in-degree of v is denoted $deg^-(v)$ and its out-degree is denoted $deg^+(v)$. A vertex node with $deg^-(v) = 0$ is called a root, as it is the origin of each of its incident arrows. Similarly, a vertex with $deg^+(v) = 0$ is called a leaf.

First we get all possible paths from one root node to each leaf node, then we get all combinations of the path of each root node, so each combination is a possible abstract sub-topology. For example, as the model example in Figure 4.4, for root node ApachePHPModule, there are two paths to reach leaf nodes WindowsAzure and AmazonEC2:

1. ApachePHPModule->ApacheWebserver->WindowsVM->WindowsAzure
2. ApachePHPModule->ApacheWebserver->UbuntuVm->AmazonEC2

5.3 Core Algorithm

for root node MySQLServer, there are one path to reach leaf node AmazonEC2:

1. MySQLServer->UbuntuVm->AmazonEC2

So there are two possible abstract sub-topologies:

1. ApachePHPModule->ApacheWebserver->WindowsVM->WindowsAzure and MySQLServer->UbuntuVm->AmazonEC2
2. ApachePHPModule->ApacheWebserver->UbuntuVm->AmazonEC2 and MySQLServer->UbuntuVm->AmazonEC2

Algorithm 2 finds all possible abstract sub-topologies by using combination discovery in algorithm 1:

Algorithm 2 Generate All abstract Topologies

```
1: allRootNodesOfTopology = getAllRootNodesOfTopology();
2: allLeafNodesOfTopology = getAllLeafNodesOfTopology();
3: for oneRootNode : all Nodes in allRootNodesOfTopology do
4:   for oneLeafNode : all Nodes in allLeafNodesOfTopology do
5:     path = one path from oneRootNode to oneLeafNode
6:     allPathsForOneRootNode.add(path)
7:   end for
8:   PathBoxesOfAllRootNodes.add(allPathsForOneRootNode)
9: end for
10: GENERATECOMBINATION(PathBoxesOfAllRootNodes, oneCombination)
```

Once we get the abstract sub-topologies, for each abstract node we get all combinations of its concrete node by iterating using algorithm 1, then we do the same thing to retrieve all combinations of instance node. Finally all viable topologies can be discovered.

5.3.2 Similar Topology Matching

In current version we only provide algorithm to check if two alpha topologies are *equal*. *Equal* means that two alpha topologies are isomorphic:

Definition 6 (*Isomorphic Graphs*) Two graphs which contain the same number of graph vertices connected in the same way are said to be isomorphic. Formally, two graphs G and H with graph vertices $V_n = \{1, 2, \dots, n\}$ are said to be isomorphic if there is a permutation p of V_n such that u, v is in the set of graph edges $E(G)$ iff $\{p(u), p(v)\}$ is in the set of graph edges $E(H)$.²:

Basing on the definition, we design an algorithm to judge if two topologies are equal. First we get all paths from each root node to each leaf node. Then we can check if each path of topology.1 has exactly same paths in topology.2 and vice versa as defined in definition.

²<http://mathworld.wolfram.com/IsomorphicGraphs.html>

Algorithm 3 Compare two topologies if same

```
1: function IFTWOTOPOLOGIESSAME(topology1,topology2)
2:   allPathFromEachRootNodeToEachLeafNodeOfTopology1 = getAllPathesOfTopology();
3:   allPathFromEachRootNodeToEachLeafNodeOfTopology2 = getAllPathesOfTopology();
4:   for onepath : all paths in allPathFromEachRootNodeToEachLeafNodeOfTopology1 do
5:     if Exists one same path as onepath in
6:       allPathFromEachRootNodeToEachLeafNodeOfTopology2 then
7:         move the same path out of allPathFromEachRootNodeToEachLeafNodeOfTopology2
8:         Break;
9:       else
10:        return false
11:      end if
12:    end for
13:    return true
14: end function
```

6 Implementation

In this chapter, basing on the concepts established and REST API designed in previous chapters, the details of implementation of topology persistence and discovery system prototype are presented here. In the first section, an overview of tools used for implementation is presented; in the second section, some code snippets are listed for better explanation of the implementation details.

6.1 Implementation Environment

As described previously, topology is a directed graph, and a graph database has instinctive ability to handle a graph structure, so graph database is the best option to use for implementation.

Neo4j ranks number one in the area of graph database and has become more popular in both scientific and industry area. Furthermore, it provides native Java API and traversing framework. With above reason Neo4j graph database is the database we use for the persistence of topology and related entities. As discussed in 2.2, there are two deployments solutions of Neo4j: embedded database and remote server. Embedded mode means that the database is inside the application and in the same JVM as the application. Considering current usage scenario, topology modeler(winery) is the only user of our framework; furthermore, for better using native Java API and speed up the accessing, embedded mode is our choice.

Currently there are many frameworks support REST web service development over Java, like RestEasy, Restlet and Jersey, which three frameworks are the implementation of Java API for RESTful Web Services (JAX-RS). We choose Jersey as framework for our REST web service development .

We use Eclipse, a mature and popular integrated development environment(IDE) as the development environment. To better manage project, we use Maven to manage project and plug-in. Following table lists the main tools and version are used for implementation.

Tools	Version	Description
Maven	3.3.3	project comprehension and management.
Neo4j	2.7.1	graph database.
Jersey	2.22.1	REST Web service Java framework.
Eclipse	4.4.2	integrated development environment.
JDK	1.7	Java development kit.

Table 6.1: Development Tools List

For logically and functionally differentiating services model and the possibility of reusing code in the future, We use Maven to divide our framework into five sub-modules as showing in Figure 6.1. The dependency relationship means a module dependency on another module or a plug-in dependency on another module in the build process.

1. web_resource: provides web service using Jersey.
2. service: provides business logic.
3. interpreter: parses representation of workload,KPI and topologies to database domain and vice versa.
4. domain: format of entity persist in database.
5. dao: Data Access Object, handle the interaction with database.

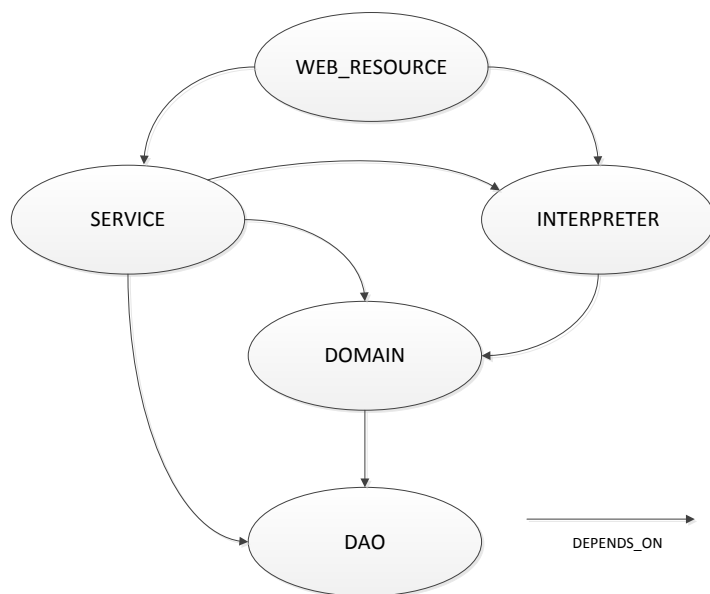


Figure 6.1: Maven Module Dependency

6.2 Implementation Details

In this section the details of implementation are presented. We follow previous division of modules and explain the implementation of them one by one. Typical and important code snippets are directly listed here as there is no more straightforward way than code itself can do the explanation better.

6.2.1 RESTful Interface

Since Java Platform Standard Edition(Java SE) 5, JAX-RS is introduced to simplify the development of web service clients and endpoints according to REST architectural pattern. Since Java Platform Enterprise Edition(Java EE) 6, JAX-RS has become an official part of Java.

As one implementation of JAX-RS, the latest major version of Jersey is 2.0 which was released in May 2013. Jersey Mainly uses Java annotation ¹ to map a Java Object as a web service, it contains following basic annotations[wik15c] which are used in our implementations:

1. @Path specifies the relative path for a resource class or method.
2. @GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP request type of a resource.
3. @Produces specifies the response Internet media types (used for content negotiation).
4. @Consumes specifies the accepted request Internet media types.
5. @PathParam binds the method parameter to a path segment.
6. @QueryParam binds the method parameter to the value of an HTTP query parameter.

Here we take endpoint and URL designed in table 5.8 for example to present the detail implementation for alpha topology web resource.

Simple URI

Code snippet 6.1 shows how to persist an alpha topology with the URI pattern '/topology/alphatopology/'. Annotation @Path defines the root path of this web resource, @POST and @Consumes accept XML format media whose schema defined in listing 5.1 and pass the input stream to Service layer of back-end; after the alpha topology is persisted successfully, the newly created alpha topology ID is built in the response and sent back.

```
1  @Path("topology/alphatopology")
2  public class AlphaTopologyResource {
3      @POST
4      @Consumes("application/xml")
5      public Response createAlphaTopology(InputStream is) throws IOException {
6          AlphaTopology alphaTopology = null;
7          try {
8              AlphaTopologyTransformer transformer = new AlphaTopologyTransformer(is);
9              alphaTopology = transformer.getDomainType();
10         } catch (InputWrongType e) {
11             e.printStackTrace();
```

¹An annotation, in the Java computer programming language, is a form of syntactic metadata that can be added to Java source code.²[wik15a]

```

12     }
13     AlphaTopologyService service = new AlphaTopologyService();
14     Node alphaTopologyIndex = service.AddAlphaTopology(alphaTopology);
15     return Response.created(URI.create("/alphanetworking/"+alphaTopologyIndex.
16         getId())).build();
17     }
18     ...
19 }

```

Listing 6.1: Persist an Alpha Topology

URI with Parameters

Code snippet 6.2 shows how to delete an alpha topology with URI pattern: `'/topology/alphanetworking/{alphaTopologyID}'`. Annotation `@PathParam` accepts the parameter `'alphaTopologyID'` in URI and passes it to service layer.

```

1  @Path("topology/alphanetworking")
2  public class AlphaTopologyResource {
3      @DELETE
4      @Consumes("application/xml")
5      @Path("{alphaTopologyId}")
6      public String deleteAlphaTopologyById(InputStream is, @PathParam("
7          alphaTopologyId") long alphaTopologyId) throws JAXBException{
8          AlphaTopologyService service = new AlphaTopologyService();
9          if(service.deleteAlphaTopologyById(alphaTopologyId)){
10             return "delete alphaTopology ID:"+alphaTopologyId+" "+"from database
11                 Successfully!";
12         }
13         else{
14             throw new WebApplicationException(Response.Status.NOT_IMPLEMENTED);
15         }
16     }
17     ...
18 }

```

Listing 6.2: Delete an Alpha Topology

URI for Query

Code snippet 6.3 shows how to query workload history of an alpha topology with pattern: `'/topology/alphanetworking/{alphaTopologyID}/workload?from={timeStamp}&to={timeStamp}'`.

6.2 Implementation Details

Annotation @GET and @Produces generate XML format media which contains workload list.

```
1 @Path("topology/alphatopology")
2 public class AlphaTopologyResource {
3     @GET
4     @Produces("application/xml")
5     @Path("{alphaTopologyId}/workload")
6     public WorkloadList getWorkloadsHistory( @PathParam("alphaTopologyId") long
7         alphaTopologyId,@QueryParam("from") String from,@QueryParam("to") String
8         to) throws JAXBException{
9         AlphaTopologyService service = new AlphaTopologyService();
10        WorkloadList workloadHistoryList =service.queryWorkloadHistory(
11            alphaTopologyId, from, to);
12        return workloadHistoryList;
13    }
14    ...
15 }
```

Listing 6.3: Query Performed Workload History of Alpha Topology)

6.2.2 Interpreter

In this section, the implementation of Interpreter module is presented. Interpreter module is used to transform XML presentation of workload,KPI and topologies designed in section 5.2.2 to domain and vice versa.

There are several approaches for parsing XML in Java. We choose two approaches in our implementation due to particular requirements.

JAXB Approach

Java Architecture for XML Binding (JAXB), is an annotation framework that maps Java classes to XML and XML schema.JAXB is not part of JAX-RS,but it provides a very convenient way for Java developers to play with XML. As mentioned before, there are two opposite process, one is unmarshalling which deserializes XML data into newly created Java Class, the other is marshalling which serializes Java Class back into XML data.

In fact, Jersey has implemented a built-in JAXB support which can directly handle marshaling and unmarshalling without importing extra JAXB library. The reason of creating a separate module for handling XML parsing particularly is that the whole input stream should be saved as an attribute 'specification' sometimes.As mentioned before, topology persistence

and discovery framework is not specification specific, it requires the system extracting useful information from the specification but without losing others. So the whole specification should be saved. Alpha Topology, concrete node, instance node, relation type and viable topology are defined and presented by TOSCA specification, these entities need save the whole specification as a string value.

As JAXB consumes lots of resource during initializing phase, so the Marshaller and Unmarshaller instance should be static, following code snippet shows how to create them. Once they are created, they can be used by interpreter to parse XML.

```
1 public static Unmarshaller createUnmarshaller() {
2     try {
3         return JAXBSupport.context.createUnmarshaller();
4     } catch (JAXBException e) {
5         throw new IllegalStateException(e);
6     }
7 }
```

Listing 6.4: Create Static Unmarshaller Instance

```
1 public static Marshaller createMarshaller(boolean
2     includeProcessingInstruction) {
3     Marshaller m;
4     try {
5         m = JAXBSupport.context.createMarshaller();
6         m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
7         m.setProperty("com.sun.xml.bind.namespacePrefixMapper", JAXBSupport.
8             prefixMapper);
9         if (!includeProcessingInstruction) {
10            m.setProperty(Marshaller.JAXB_FRAGMENT, Boolean.TRUE);
11        }
12    } catch (JAXBException e) {
13        throw new IllegalStateException(e);
14    }
15    return m;
16 }
```

Listing 6.5: Create Static Marshaller Instance

Jersey Built-in Approach

JAX-RS specification requires implementations to automatically support the marshalling and unmarshalling of classes which are annotated with JAXB XML annotation like @XmlRootElement

6.2 Implementation Details

ment, `@XmlType` and `@XmlElement`. By default, the creation of JAXB Context instances Unmarshaller and Marshaller in previous approach, is also managed by Jersey. So once the Java class is well annotated, without additional effort the built-in JAXB can be used.

Workload, performance, abstract sub-topology and namespace, these entities are defined and used by framework only, so they are using Jersey built-in JAXB to parse XML.

Code Snippet 6.6 is Workload class which is well annotated with JAXB XML annotations. It can be used directly as the response of web service as listed in code Snippet 6.7.

```
1 @XmlAccessorType(XmlAccessType.FIELD)
2 @XmlType(name = "", propOrder = {
3     "pattern",
4     "arrival",
5     "behavioral",
6     "avgUsers",
7     "avgTransactions"
8 })
9 @XmlRootElement(name = "Workload")
10 public class Workload {
11
12     @XmlElement(required = true)
13     protected String pattern;
14     @XmlElement(required = true)
15     protected String arrival;
16     @XmlElement(required = true)
17     protected String behavioral;
18     @XmlElement(name = "avg_users")
19     protected short avgUsers;
20     @XmlElement(name = "avg_transactions")
21     protected short avgTransactions;
22     @XmlAttribute(name = "id")
23     protected String id;
24     @XmlAttribute(name = "startTime")
25     protected String startTime;
26     @XmlAttribute(name = "endTime")
27     protected String endTime;
28     ...
29 }
```

Listing 6.6: Well Annotated Workload Class

```
1 @GET
2 @Produces("application/xml")
3 @Path("/{id}")
```

```

4 public Workload getWorkload(@PathParam("id") String id) throws IOException {
5     Workload workload = null;
6     WorkloadService service = new WorkloadService();
7     workload = service.get(id);
8     if(workload==null) {
9         throw new WebApplicationException(Response.Status.NOT_FOUND);
10    }
11    return workload;
12    }

```

Listing 6.7: Use Annotated Workload Class as Response

6.2.3 Data Access

Data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, DAO provide some specific data operations without exposing details of the database.[wik15b]

Neo4j Graph database provides two approaches to access database in embedded mode. One approach is using Java native API when the data operations is relatively simple. When things become complicated, it is suggested using Cypher directly.

For example, Node is the basic element in graph database. With native Java API, a Node can get its label, property, in-degree, out-degree and so on. For example, code snippet 6.8 shows how to get all labels of a Node by using native Java API.

```

1 public Iterator<Label> getNodeLabelById(long id){
2     Iterator<Label> labelIterator = null;
3     try ( Transaction tx = db.beginTx();){
4         Node node = getNodeById(id);
5         labelIterator = node.getLabels().iterator();
6     }
7     return labelIterator;
8 }

```

Listing 6.8: Get All labels of a Node

For more complicated data operations, Cypher is to be considered. Cypher is Neo4j query language, it is defined in the official document of Neo4j as follows: 'Cypher is a declarative, SQL-inspired language for describing patterns in graphs. It allows us to describe what we want to select, insert, update or delete from a graph database without requiring us to describe exactly how to do it.'

6.2 Implementation Details

For example, code snippet 6.9 shows how to get all instance nodes which refers to a concrete node. The variable 'query' stores Cypher as a String. The 'MATCH' clause is used to specify the patterns which Neo4j will search in the database.

```
1 public List<Node> getInstanceNodes(Node concreteNode) {
2     List<Node> instanceNodes = new ArrayList<Node>();
3     String query = "MATCH (a)-[r:REFERS_TO]->(b) WHERE id(b)="+ concreteNode.
4         getId()+" "+"RETURN a";
5     try ( Transaction tx = db.beginTx();Result result = db.execute(query);)
6     {
7         while ( result.hasNext() )
8         {
9             Map<String,Object> row = result.next();
10            for ( String key : result.columns() )
11            {
12                Node instanceNode= (Node) row.get( key );
13                instanceNodes.add(instanceNode);
14            }
15            tx.success();
16        }
17    return instanceNodes;
18 }
```

Listing 6.9: Get All Instance Nodes of a Concrete node

6.2.4 Business Logic

The real business logic stays at service layer. It accepts the domain object from interpreter layer and calls the DAO layer to handle data. From functionality perspective, service layer contains three sub-service module: topology service, workload service and performance service. Topology service provides functionality of viable topology and similar topology discovery . In this section, we focus on the implementation details of topology discovery.

There are several main classes which work together to discover viable topologies in database. Figure 6.2 is a simple class diagram which describes the relationship of these classes.

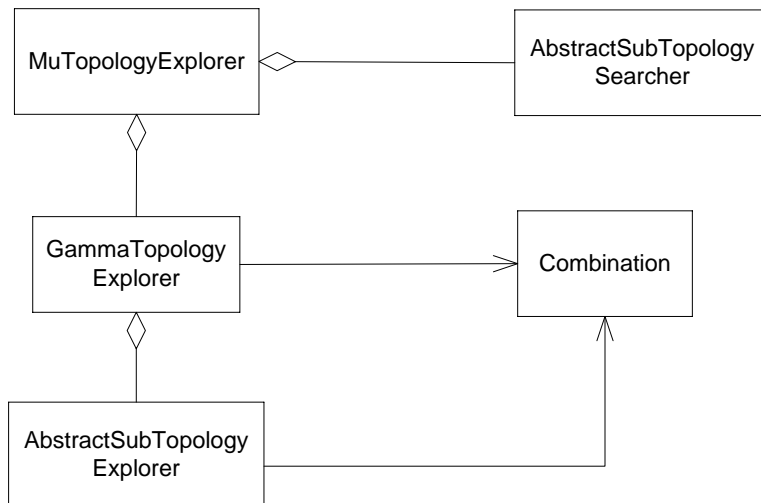


Figure 6.2: Class Diagram for the Viable Topology Discovery Component

In section 5.3, the core algorithms is proposed for generating combination and discovering abstract sub-topology. Code snippet 6.10 presents the details implementation of the algorithm 1 in 5.3, which is in the class Combination.

```

1
2 private void generateCombinations(List<List<T>> Boxes, List<T>
   oneCombinationResult) {
3     List<T> ElementlistOfOneBox = Boxes.get(0);
4     List<T> tempResultForStack = new ArrayList<T>(oneCombinationResult);
5     combinationResultStack.push(tempResultForStack);
6     for(T oneElement : ElementlistOfOneBox) {
7         List<List<T>> newBoxes = new ArrayList<List<T>>(Boxes);
8         newBoxes.remove(ElementlistOfOneBox);
9         if(Boxes.size() > 1) {
10            oneCombinationResult.add(oneElement);
11            generateCombinations(newBoxes, oneCombinationResult);
12            oneCombinationResult.clear();
13            List<T> restultFromStack = (List<T>) combinationResultStack.pop()
                ;
14            for(T oneData: restultFromStack){
15                oneCombinationResult.add(oneData);
  
```

```
16     }
17     oneCombinationResult.remove((oneCombinationResult.size()-1));
18     } else {
19     oneCombinationResult.add(oneElement);
20     List<T> tempResult = new ArrayList<T>(oneCombinationResult);
21     combinationsResults.add(tempResult);
22     T lastElement = ElementlistOfOneBox.get(ElementlistOfOneBox.size
23     (-1));
24     if(oneElement.equals(lastElement)){
25     }
26     else{
27     oneCombinationResult.remove(oneElement);
28     }
29     }
30 }
```

Listing 6.10: Combination Generation Algorithm

Then this algorithm can be used by other object. AbstractSubTopology can use it iteratively finding all abstract sub-topologies. GammaTopology can use it iteratively finding all linked concrete nodes and instance nodes. Finally MuTopologyExplorer will connect the gamma topology with alpha topology to create a viable topology.

7 Validation

In this chapter, we validate the implementation to check if topology persistence and discovery framework fulfill functional and non-functional requirements as previously described. We start from the scratch to fill a blank graph database with necessary data, step by step to perform operations for topologies and its enrichments, verify REST API and check the corresponding response. The sample of Neo4j database in this section can be retrieved in Bitbucket.¹

7.1 Methodology

Figure 7.1 simulate the scenario of validation. TOSCA topology elements like *Service Template*, *Node Type*, *Node Template*, *Relationship Type* and corresponding XML Namespace used by topology modeler(Winery) are persisted in database with correct order as indicated by white arrow. Solid arrow shows the necessary data which is used by database entities.

The validation process is divided into five steps:

1. **Basic Elements:** In this step *RelationType* and *NameSpace* from Winery are persisted in database. These two elements are the necessary components of other entities, so they should be persisted and verified first.
2. **Alpha Topology:** In TOSCA, a Topology is defined by *ServiceTemplate*. So in this step, *ServiceTemplate* of an Alpha Topology from Winery is used. Once there are more than one alpha topologies existed in database, discovering of similar alpha topologies can be verified.
3. **Gamma Topology:** To establish a gamma topology, abstract sub-topology, concrete node and instance node should be persisted one by one in this step.
4. **Topology Enrichment:** Alpha Topology is application specific, so somehow it represents an application. An application can be enriched by evolving workloads and performance. In this step, the operations for the enrichments are validated.
5. **Viable Topology:** Based on previous steps, viable topology can be discovered in this step.

¹<https://shmily1140@bitbucket.org/shmily1140/pertos-sample.git>

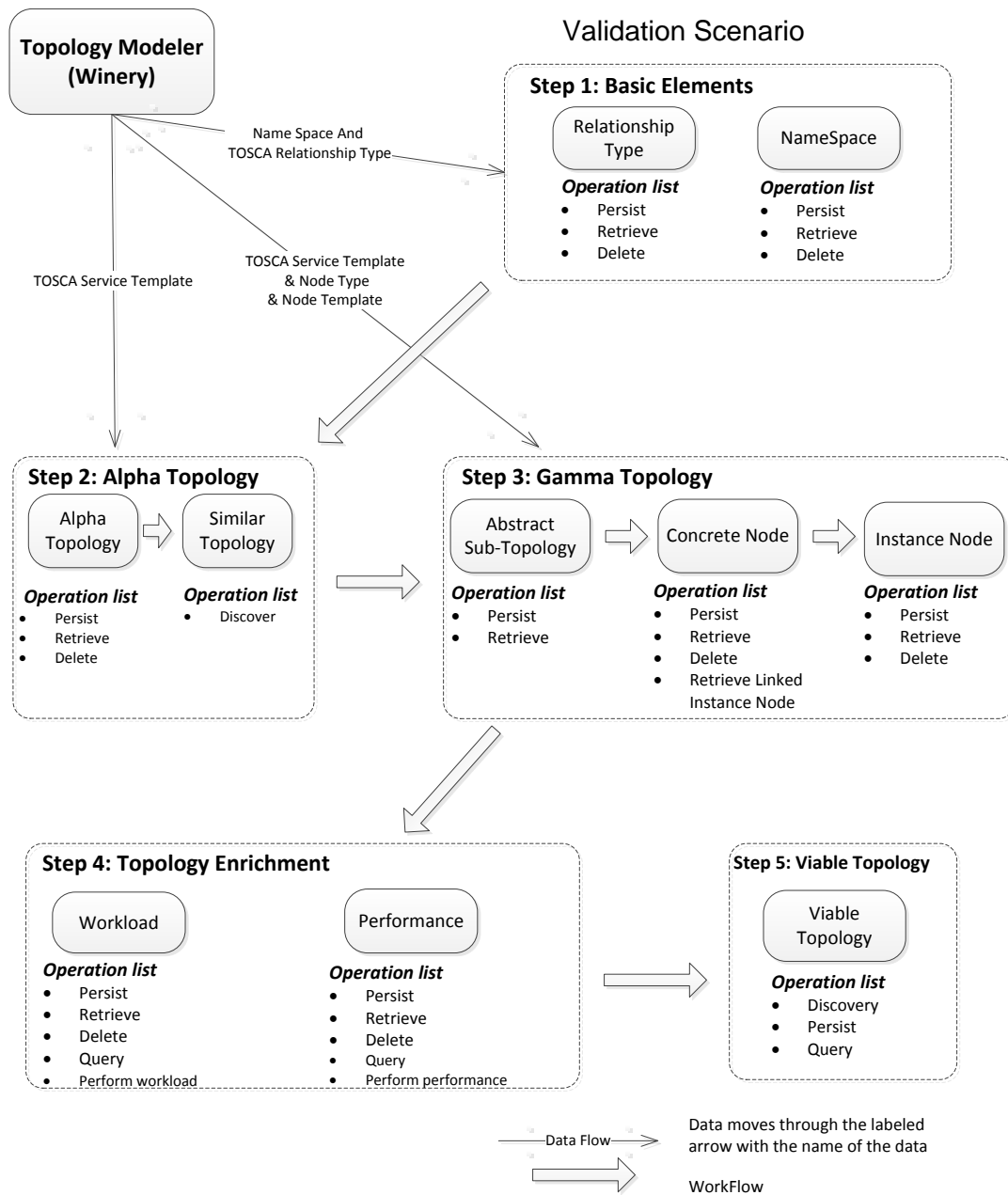


Figure 7.1: Validation Scenario Overview

7.1 Methodology

We use following three tools to do the validation:

1. Postman²: a Restful API testing tool which can create and send HTTP request using powerful GUI, write test cases to validate response data, response times and response messages.
2. Neo4j Browser³: the default Neo4j server which has a powerful, customizable data visualization tool based on the built-in D3.js library. It looks like a lightweight IDE through which user can write Cypher to query database directly.
3. Winery⁴: topology modeler which can visualize TOSCA based topology service template.

The results in the form of screenshot for each step are checked by Postman first to verify if the REST API, HTTP request and HTTP response are the ones as expected. Then Neo4j Browser is used to verify if the data in Graph database is correct with respect to corresponding REST API. For viable Topology, Winery is used to verify generated TOSCA Service Template.

Figure 7.2 is the topology of sample application *MediaWikiApp* which we are going to validate in the following. The nodes of alpha topology are marked in gray, which is application specific. The bottom half is gamma topology, which is application non-specific. From the figure we can know that for this application exists two potential topologies: the node *Web_Server* can either *hosted_on* a virtual windows OS or a virtual Ubuntu OS. Moreover, the application is enriched by one performance and one workload notated with dark gray circle in figure. It should be clear that actually the performance and workload are not performed directly on the node *Web_App*. Instead, it should be performed on the *alphaTopologyIndex* node as modeled previously. In the following sections we follow the steps designed above to validate our system from the scratch.

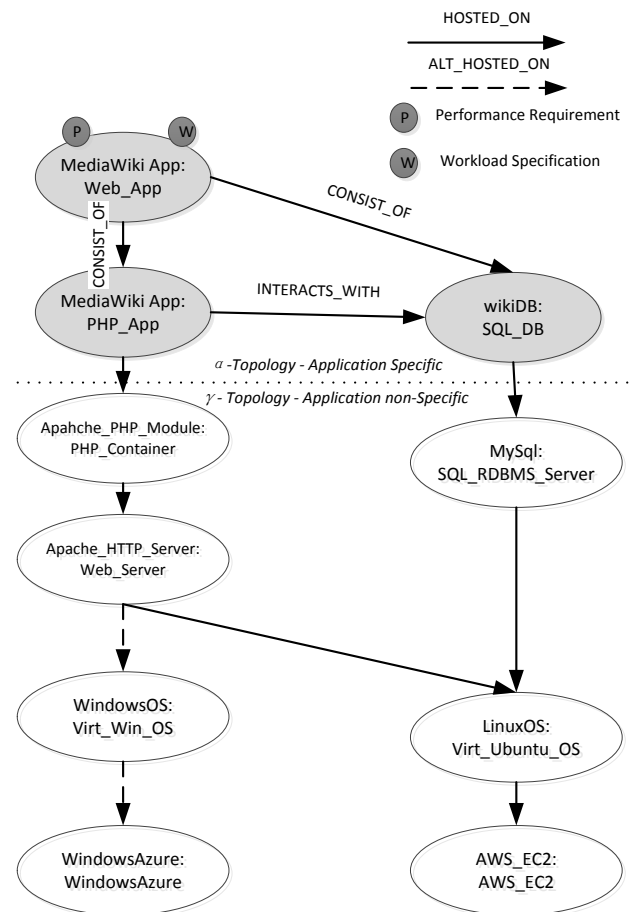


Figure 7.2: Validation Sample Application

²<https://www.getpostman.com/>

³<http://neo4j.com/developer/guide-neo4j-browser/>

⁴<http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php>

7.2 Basic Elements

In this section, the validation of basic elements are presented.

7.2.1 NameSpace

As TOSCA is represented by XML which uses namespace to uniquely identify named elements and attributes. Winery has a component to specially save namespaces, so to be consistent with Winery, the namespace used by Winery should be saved in system as well.

Figure 7.3 shows persisting a namespace element.

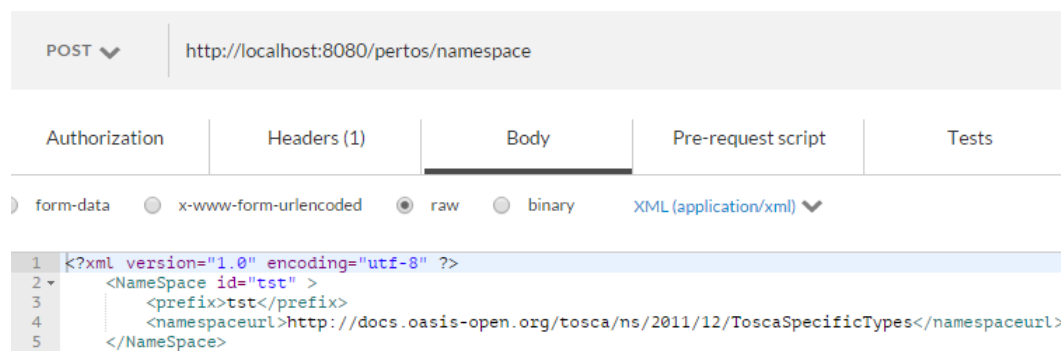


Figure 7.3: Request for Persisting a NameSpace

System accepted the request and created namespace. In the headers of response, location field save the newly created resource with id 2 as showing in Figure 7.4.

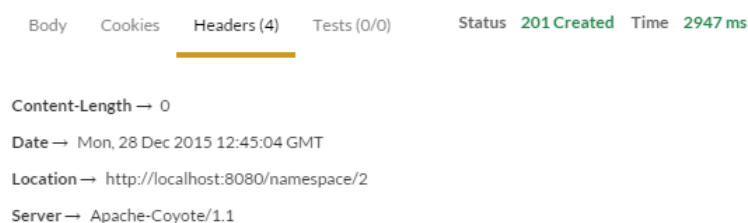


Figure 7.4: Response of the Request for Persisting a NameSpace

To verify if the namespace is persisted successfully, Figure 7.5 shows getting the namespace by ID which equals 2. The result shows the retrieved namespace is the one we just persisted.

7.2 Basic Elements

GET ▼ | http://localhost:8080/pertos/namespace/2

Authorization | Headers (1) | Body | Pre-request script | Tests

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status 200 OK | Time 13147 ms

Pretty | Raw | Preview | XML ▼ | ☰

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Namespace id="tst">
3   <prefix>tst</prefix>
4   <namespaceurl>http://docs.oasis-open.org/tosca/ns/2011/12/ToscaSpecificTypes</namespaceurl>
5 </Namespace>
```

Figure 7.5: Retrieve one Namespace By ID

More necessary namespaces are persisted as well which can be validated by retrieve all namespace as showing in Figure 7.6.

GET ▼ | http://localhost:8080/pertos/namespace

Authorization | Headers (1) | Body | Pre-request script | Tests

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status 200 OK | Time 146 ms

Pretty | Raw | Preview | XML ▼ | ☰

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <NamespaceList>
3   <namespaceWithDatabaseID databaseId="2">
4     <nameSpace id="tst">
5       <prefix>tst</prefix>
6       <namespaceurl>http://docs.oasis-open.org/tosca/ns/2011/12/ToscaSpecificTypes</namespaceurl>
7     </nameSpace>
8   </namespaceWithDatabaseID>
9   <namespaceWithDatabaseID databaseId="3">
10    <nameSpace id="tbt">
11      <prefix>tbt</prefix>
12      <namespaceurl>http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes</namespaceurl>
13    </nameSpace>
14  </namespaceWithDatabaseID>
15  <namespaceWithDatabaseID databaseId="4">
16    <nameSpace id="exc">
17      <prefix>exc</prefix>
18      <namespaceurl>http://www.example.com</namespaceurl>
19    </nameSpace>
20  </namespaceWithDatabaseID>
21  <namespaceWithDatabaseID databaseId="5">
22    <nameSpace id="exnt">
23      <prefix>exnt</prefix>
24      <namespaceurl>http://example.com/NodeTypes</namespaceurl>
25    </nameSpace>
26  </namespaceWithDatabaseID>
27 </NamespaceList>
```

Figure 7.6: Retrieving all NameSpaces

Finally we check the namespaces persisted in database to see if it is consistent as showing in Figure 7.7.

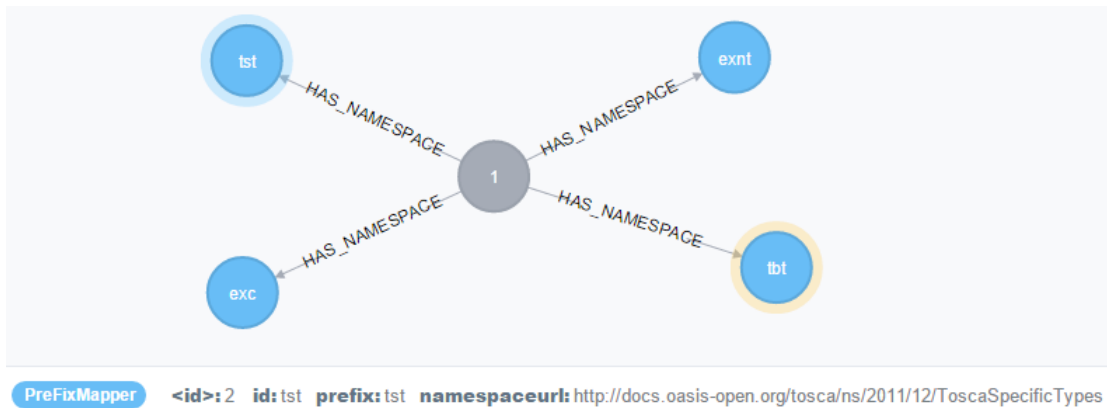


Figure 7.7: NameSpace in Database

7.2.2 Relationship Type

In TOSCA, a Relationship Type is a reusable entity that defines the type of one or more Relationship Templates between Node Templates. When building gamma topology and viable topology, Relationship Type are used, so it should be persisted first as basic elements as well.

First, we persist relationship type by HTTP method *POST* as showing in Figure 7.8

POST http://localhost:8080/pertos/relationshiptype

Authorization	Headers (1)	Body	Pre-request script	Tests
<input type="radio"/> form-data	<input type="radio"/> x-www-form-urlencoded	<input checked="" type="radio"/> raw	<input type="radio"/> binary	XML (application/xml)
<pre> 1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?> 2 <tosca:Definitions id="winery-defs-for_ns4-HostedOn" targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes" locati 3 <tosca:Import namespace="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes" locati 4 <tosca:RelationshipType name="HostedOn" targetNamespace="http://docs.oasis-open.org/tosca/n 5 <tosca:documentation>Hosted on</tosca:documentation> 6 <tosca:DerivedFrom typeRef="tbt:RootRelationshipType" xmlns:tbt="http://docs.oasis-open 7 <tosca:ValidSource typeRef="tbt:ContainerRequirement" xmlns:tbt="http://docs.oasis-open 8 <tosca:ValidTarget typeRef="tbt:ContainerCapability" xmlns:tbt="http://docs.oasis-open 9 </tosca:RelationshipType> 10 </tosca:Definitions> 11</pre>				

Figure 7.8: Request for Persisting a Relationship Type

HTTP location headers of the response contains the newly created relationship type with ID 29 as showing in figure 7.9.

7.2 Basic Elements

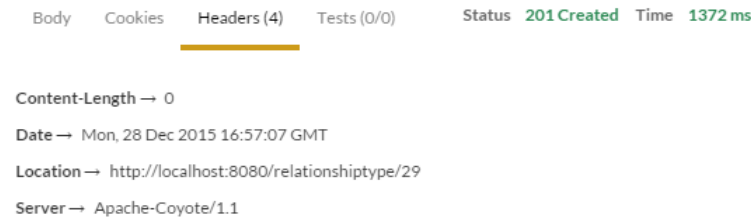


Figure 7.9: Response of the Request for Persisting a RelationshipType

To verify if Relationship Type is truly persisted, we can retrieve the relationship type with ID 29 as showing in Figure 7.10. The body of response contains the same one as we just persisted.

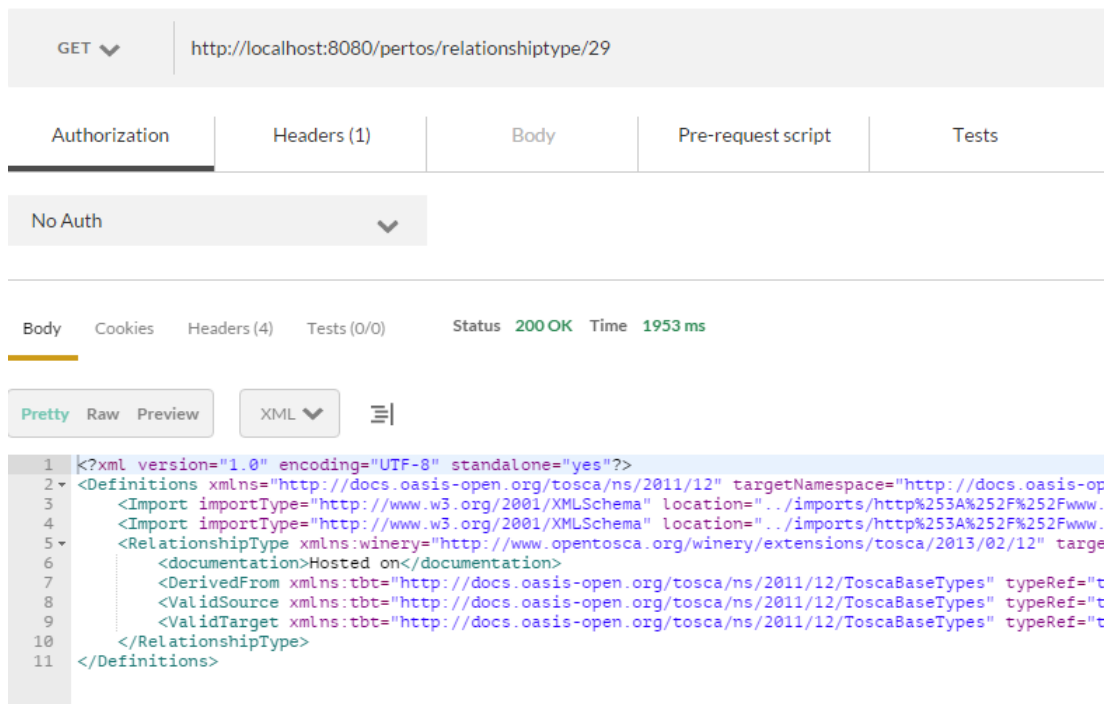


Figure 7.10: Retrieve one RelationshipType By ID

Finally we take a look at the relationship type in database, as we only use Relationship Type *CONSIST_OF* in our validation, so only one relationship type is persisted.

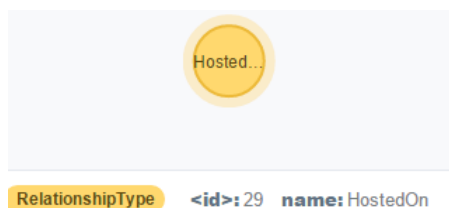


Figure 7.11: RelationshipType in Database

7.3 Alpha Topology

Alpha Topology is the application specific part. As showing in Figure 7.12 first we persist one alpha topology. In the location headers of response contains the newly created alpha topology with ID 50.

7.3.1 Alpha Topology

POST http://localhost:8080/pertos/topology/alphatopology

Authorization Headers (1) Body Pre-request scrip

form-data x-www-form-urlencoded raw binary XML (application/xml)

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <AlphaTopologyTemplate >
3 <specificationType>xml</specificationType>
4 <specification>
5 <tosca:Definitions id="winery-defs-for_ex1-MediaWiki" targetNamespace="htt
6 <tosca:ServiceTemplate id="MediaWiki_2" name="MediaWiki" targetNamespa
7 <winery:Properties xmlns:ns1="http://www.eclipse.org/winery/model/
8 </winery:Properties>
9 <tosca:TopologyTemplate>
10 <tosca:NodeTemplate name="MediaWiki_WebApp" id="MediaWiki_WebA
11 <tosca:Policies>
12 <tosca:Policy name="policy" policyType="ns15:Performan
13 </tosca:Policies>
14 </tosca:NodeTemplate>
15 <tosca:NodeTemplate name="WikiDB_MySQLDB" id="WikiDB_MySQLDB"
16 <tosca:Policies>
17 <tosca:Policy name="2" policyType="ns15:PerformancePol
18 </tosca:Policies>
19 </tosca:NodeTemplate>
20 <tosca:NodeTemplate name="MediaWiki_PHPApp" id="MediaWiki_PHPA
21 <tosca:RelationshipTemplate name="con_337" id="con_337" type="
22 <tosca:SourceElement ref="MediaWiki_WebApp"/>
23 <tosca:TargetElement ref="WikiDB_MySQLDB"/>
24 </tosca:RelationshipTemplate>
25 <tosca:RelationshipTemplate name="con_346" id="con_346" type="
26

```

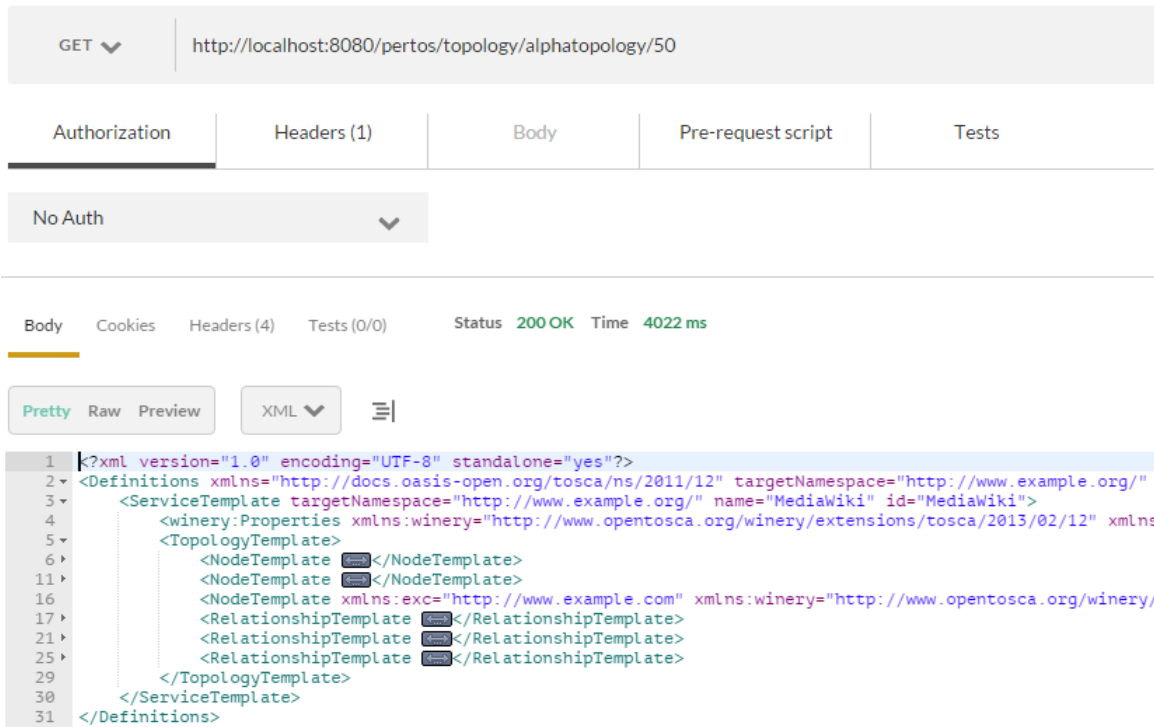
Body Cookies Headers (4) Tests (0/0) Status 201 Created Time 9505 ms

Content-Length → 0
Date → Mon, 28 Dec 2015 19:30:45 GMT
Location → http://localhost:8080/alphatopology/50
Server → Apache-Coyote/1.1

Figure 7.12: Request and Response for Persisting an Alpha Topology

Then we retrieve this alpha topology using ID 50 to check if this alpha topology is persisted successfully as showing in Figure 7.13

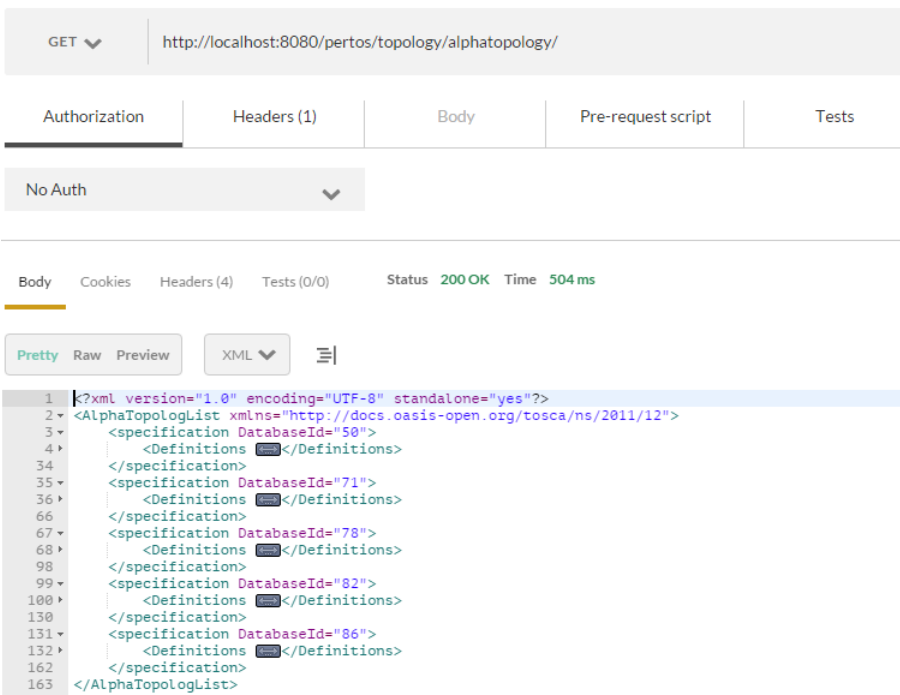
7.3 Alpha Topology



```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Definitions xmlns="http://docs.oasis-open.org/tosca/ns/2011/12" targetNamespace="http://www.example.org/"
3   <ServiceTemplate targetNamespace="http://www.example.org/" name="Mediawiki" id="Mediawiki">
4     <winery:Properties xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/02/12" xmlns
5       <TopologyTemplate>
6         <NodeTemplate />
11        <NodeTemplate />
16        <NodeTemplate xmlns:exc="http://www.example.com" xmlns:winery="http://www.opentosca.org/winery/"
17          <RelationshipTemplate />
21          <RelationshipTemplate />
25          <RelationshipTemplate />
29        </TopologyTemplate>
30      </ServiceTemplate>
31 </Definitions>
```

Figure 7.13: Retrieve one an Alpha Topology By ID

More alpha topologies are persisted as well. As showing in Figure 7.14, all alpha topologies in database are retrieved. The TOSCA definition is wrapped in element *specification*.



```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <AlphaTopologList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <specification DatabaseId="50">
4     <Definitions />
34   </specification>
35   <specification DatabaseId="71">
36     <Definitions />
66   </specification>
67   <specification DatabaseId="78">
68     <Definitions />
98   </specification>
99   <specification DatabaseId="82">
100    <Definitions />
130   </specification>
131   <specification DatabaseId="86">
132    <Definitions />
162   </specification>
163 </AlphaTopologList>
```

Figure 7.14: Retrieve all Alpha Topologies

Figure 7.15 is the screenshot of all alpha topologies in database. The nodes in red color is

node template and the one in yellow is alpha topology index.

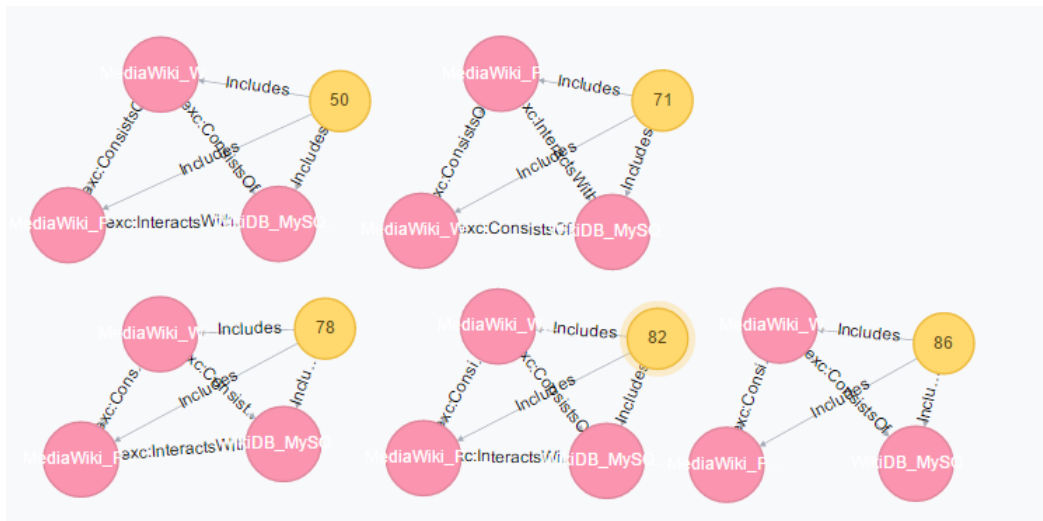


Figure 7.15: Alpha Topology in Database

7.3.2 Similar Topology

As showing in Figure 7.15, alpha topology with ID 50 is same as the one with ID 71. The remaining three are different with each other. First we try to find a similar topology for alpha topology with ID 50, in the response the definition of similar alpha topology with ID 71 is wrapped in element *specification* as showing in Figure 7.16.

Figure 7.16: Find Similar Alpha Topology-1

Then we try to find similar topology for alpha topology with ID 82, the response with status code '404 Not Found' indicates there is no similar topology in database as we expected.

7.3 Alpha Topology

The screenshot displays a web browser's developer tools interface. At the top, the request method is 'GET' and the URL is 'http://localhost:8080/pertos/similartopology/82'. Below this, there are tabs for 'Authorization', 'Headers (1)', 'Body', and 'Pre-request script'. The 'Authorization' tab is selected, showing 'No Auth'. The main area shows the response status as '404 Not Found' and the time taken as '374 ms'. The response body is displayed in 'HTML' view, showing the following code:

```
i 1 <html>
2   <head>
3     <title>Apache Tomcat/7.0.64 - Error report</title>
4     <style>
5       <!--H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color
6     </style>
7   </head>
8   <body>
9     <h1>HTTP Status 404 - Not Found</h1>
10    <HR size="1" noshade="noshade">
11      <p>
12        <b>type</b> Status report
13      </p>
14      <p>
15        <b>message</b>
16        <u>Not Found</u>
17      </p>
18      <p>
19        <b>description</b>
20        <u>The requested resource is not available.</u>
21      </p>
22      <HR size="1" noshade="noshade">
23      <h3>Apache Tomcat/7.0.64</h3>
24    </body>
25  </html>
```

Figure 7.17: Find Similar Alpha Topology-2

7.4 Gamma Topology

In this section we validate the process to establish a reusable gamma topology by persisting abstract sub-topology, concrete nodes and instance nodes step by step.

7.4.1 Abstract Sub-Topology

First we persist an abstract sub-topology as showing in Figure 7.18.

The screenshot displays a REST client interface for a POST request to `http://localhost:8080/pertos/topology/abstractsubtopology`. The request body is an XML document with the following structure:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <AbstractSubTopology name="WikiMedia" id="WikiMedia_1">
3   <AbstractNode name="ApachePHPModule" id="ApachePHPModule" type="ApachePHPModule">
4     <Level>root</Level>
5   </AbstractNode>
6   <AbstractNode name="AmazonEC2" id="AmazonEC2_2" type="AmazonEC2">
7     <Level>Leaf</Level>
8   </AbstractNode>
9   <AbstractNode name="WindowsAzure" id="WindowsAzure" type="WindowsAzure">
10    <Level>Leaf</Level>
11  </AbstractNode>
12  <AbstractNode name="UbuntuVM" id="UbuntuVM_2" type="UbuntuVM">
13    <Level>2</Level>
14  </AbstractNode>
15  <AbstractNode name="WindowsVM" id="WindowsVM_2" type="WindowsVM">
16    <Level>2</Level>
17  </AbstractNode>
18  <AbstractNode name="ApacheWebserver" id="ApacheWebserver" type="ApacheWebserver">
19    <Level>1</Level>
20  </AbstractNode>
21  <AbstractNode name="MySQLServer" id="MySQLServer" type="MySQLServer">
22    <Level>root</Level>
23  </AbstractNode>
24  <RelationshipOfAbstractNode type="tbt:HostedOn" >
25    <SourceElement ref="UbuntuVM_2"/>
26    <TargetElement ref="AmazonEC2_2"/>

```

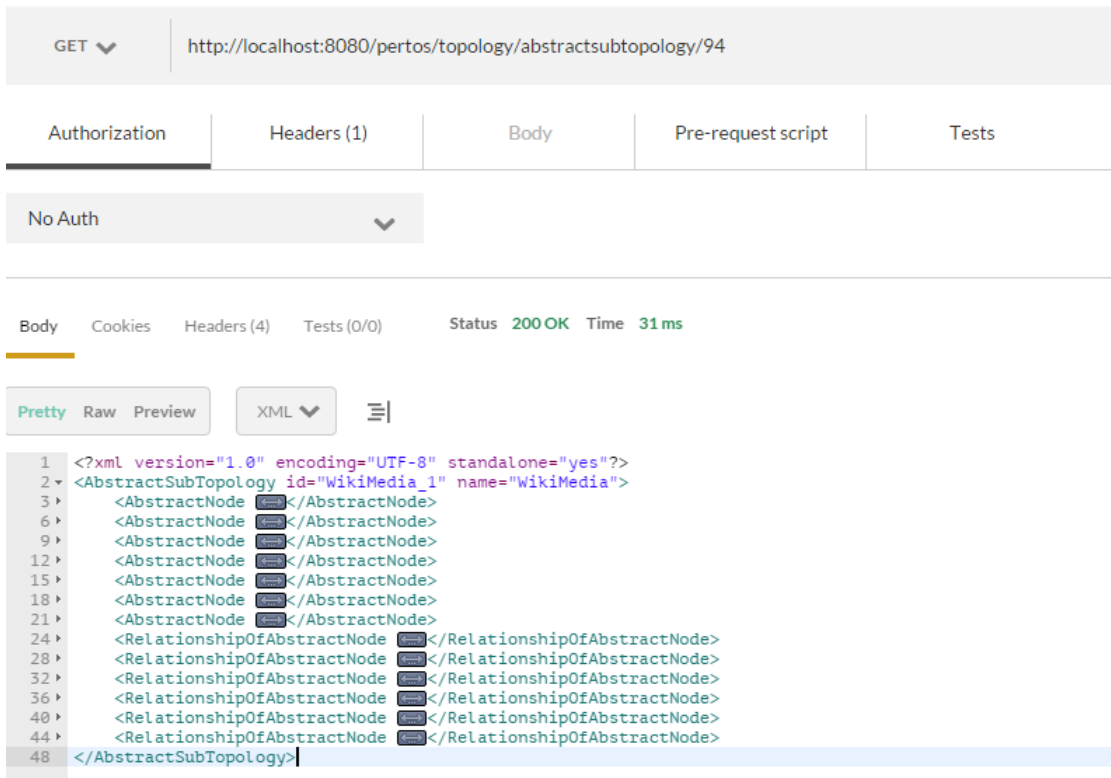
The response status is **201 Created** with a time of **1032 ms**. The response headers include:

- Content-Length → 0
- Date → Tue, 29 Dec 2015 09:04:59 GMT
- Location → `http://localhost:8080/abstractsubtopology/94`
- Server → Apache-Coyote/1.1

Figure 7.18: Request and Response for Persisting an Abstract Sub-Topology

After persisting it, we try to retrieve it by ID 94 as showing in Figure 7.19. Figure 7.20 is the screenshot of the newly created abstract sub-topology, the nodes in gray are abstract nodes and the node in yellow is abstract sub-topology index.

7.4 Gamma Topology



GET ▼ | http://localhost:8080/pertos/topology/abstractsubtopology/94

Authorization | Headers (1) | Body | Pre-request script | Tests

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status **200 OK** | Time **31 ms**

Pretty | Raw | Preview | XML ▼ | ☰

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <AbstractSubTopology id="WikiMedia_1" name="WikiMedia">
3   <AbstractNode />
6   <AbstractNode />
9   <AbstractNode />
12  <AbstractNode />
15  <AbstractNode />
18  <AbstractNode />
21  <AbstractNode />
24  <RelationshipOfAbstractNode />
28  <RelationshipOfAbstractNode />
32  <RelationshipOfAbstractNode />
36  <RelationshipOfAbstractNode />
40  <RelationshipOfAbstractNode />
44  <RelationshipOfAbstractNode />
48 </AbstractSubTopology>
```

Figure 7.19: Retrieve one an Abstract Sub-Topology By ID

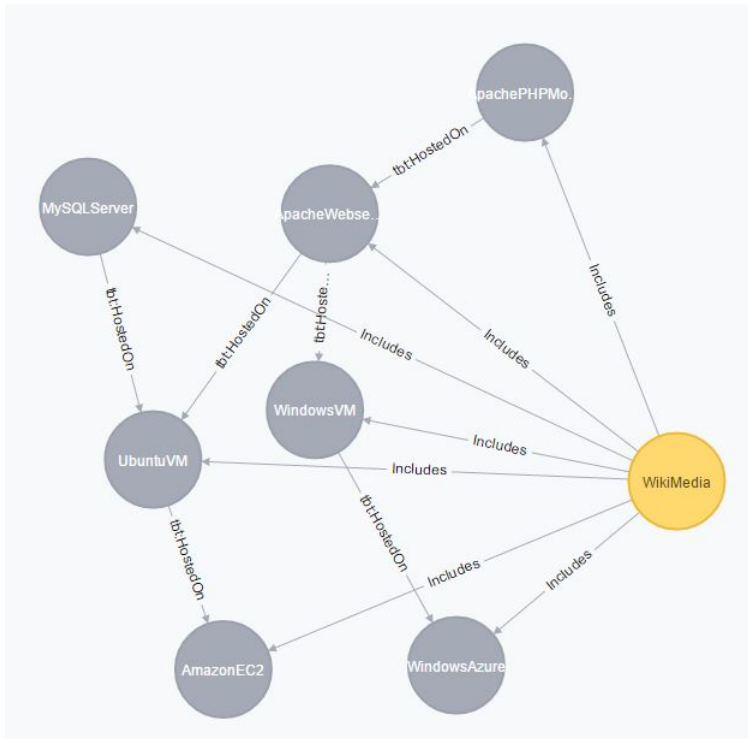


Figure 7.20: Abstract Sub-Topology in Database

POST ▼ | http://localhost:8080/pertos/concretenode/ApachePHPModule

Authorization | Headers (1) | **Body** | Pre-request script

form-data x-www-form-urlencoded raw binary XML (application/xml) ▼

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <tosca:Definitions id="winery-defs-for_ns2-ApachePHPModule" targetNamespace="ht
3 <tosca:Import namespace="http://docs.oasis-open.org/tosca/ns/2011/12/Tosca!
4 <tosca:Import namespace="http://docs.oasis-open.org/tosca/ns/2011/12/Tosca!
5 <tosca:Import namespace="http://docs.oasis-open.org/tosca/ns/2011/12/Tosca!
6 <tosca:NodeType name="ApachePHPModule" targetNamespace="http://docs.oasis-
7 <tosca:documentation>Apache PHP Module</tosca:documentation>
8 <winery:PropertiesDefinition elementname="Properties" namespace="http:,
9 <tosca:DerivedFrom typeRef="tst:ApacheModule" xmlns:tst="http://docs.o
10 <tosca:CapabilityDefinitions>
11 <tosca:CapabilityDefinition name="apaContainer" capabilityType="tst
12 </tosca:CapabilityDefinitions>
13 <tosca:Interfaces>
14 <tosca:Interface name="http://docs.oasis-open.org/tosca/ns/2011/12,
15 <tosca:Operation name="start"/>
16 <tosca:Operation name="uninstall"/>
17 </tosca:Interface>
18 </tosca:Interfaces>
19 </tosca:NodeType>
20 </tosca:Definitions>

```

Body | Cookies | **Headers (4)** | Tests (0/0) | Status **201 Created** Time **3842 ms**

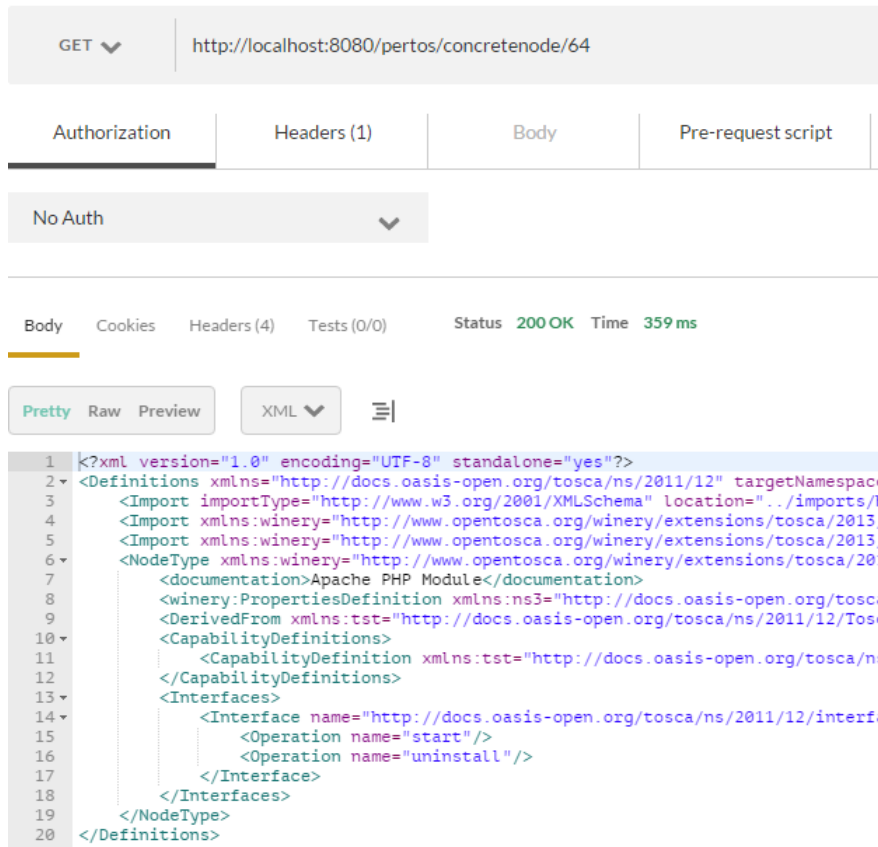
Content-Length → 0
Date → Mon, 28 Dec 2015 21:12:12 GMT
Location → http://localhost:8080/concretenode/64
Server → Apache-Coyote/1.1

Figure 7.21: Request and Response for Persisting a Concrete Node

7.4.2 Concrete Node

After abstract sub-topology is persisted successfully, concrete node can be added for each type of abstract node. Figure 7.21 shows persisting a concrete node for abstract node type *ApachePHPModule*. The location headers of the response contains ID of the newly created concrete node. Figure 7.22 verifies retrieving the concrete node persisted in previous step with ID 64.

7.4 Gamma Topology



```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Definitions xmlns="http://docs.oasis-open.org/tosca/ns/2011/12" targetNamespace="
3   <Import importType="http://www.w3.org/2001/XMLSchema" location="../imports/h
4   <Import xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/
5   <Import xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/
6   <NodeType xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/201
7     <documentation>Apache PHP Module</documentation>
8     <winery:PropertiesDefinition xmlns:ns3="http://docs.oasis-open.org/tosca
9     <DerivedFrom xmlns:tst="http://docs.oasis-open.org/tosca/ns/2011/12/Tosca
10    <CapabilityDefinitions>
11      <CapabilityDefinition xmlns:tst="http://docs.oasis-open.org/tosca/ns
12    </CapabilityDefinitions>
13    <Interfaces>
14      <Interface name="http://docs.oasis-open.org/tosca/ns/2011/12/interfa
15        <Operation name="start"/>
16        <Operation name="uninstall"/>
17      </Interface>
18    </Interfaces>
19  </NodeType>
20 </Definitions>
```

Figure 7.22: Retrieve a Concrete Node By ID

Next step we add concrete nodes for each type of abstract sub-topology, then we verify the result by retrieving all concrete nodes in the database as showing in Figure 7.24.

As defined in TOSCA specification, the relationship between node templates is established by verifying requirements and capabilities pairs of concrete node. So for the leaf nodes of alpha topology, we add two concrete node type, each of which contains one requirements as showing in Figure 7.23. Finally we check the data in database as showing in Figure 7.25. The nodes in dark pink are concrete nodes, the two on top with two capabilities in green.

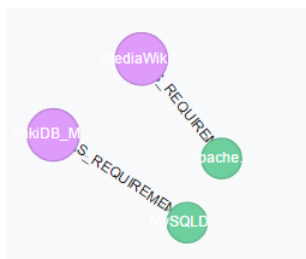


Figure 7.23: Concrete Nodes for Alpha Topology

GET

Authorization Headers (1) Body Pre-request script

No Auth

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 163 ms

Pretty Raw Preview XML

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <ConcreteNodeList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <specification DatabaseId="64">
4     <Definitions </Definitions>
23 </specification>
24 <specification DatabaseId="66">
25   <Definitions </Definitions>
75 </specification>
76 <specification DatabaseId="67">
77   <Definitions </Definitions>
84 </specification>
85 <specification DatabaseId="69">
86   <Definitions </Definitions>
140 </specification>
141 <specification DatabaseId="70">
142   <Definitions </Definitions>
178 </specification>
179 <specification DatabaseId="71">
180   <Definitions </Definitions>
183 </specification>
184 <specification DatabaseId="72">
185   <Definitions </Definitions>
221 </specification>
222 </ConcreteNodeList>

```

Figure 7.24: Retrieving all Concrete Nodes

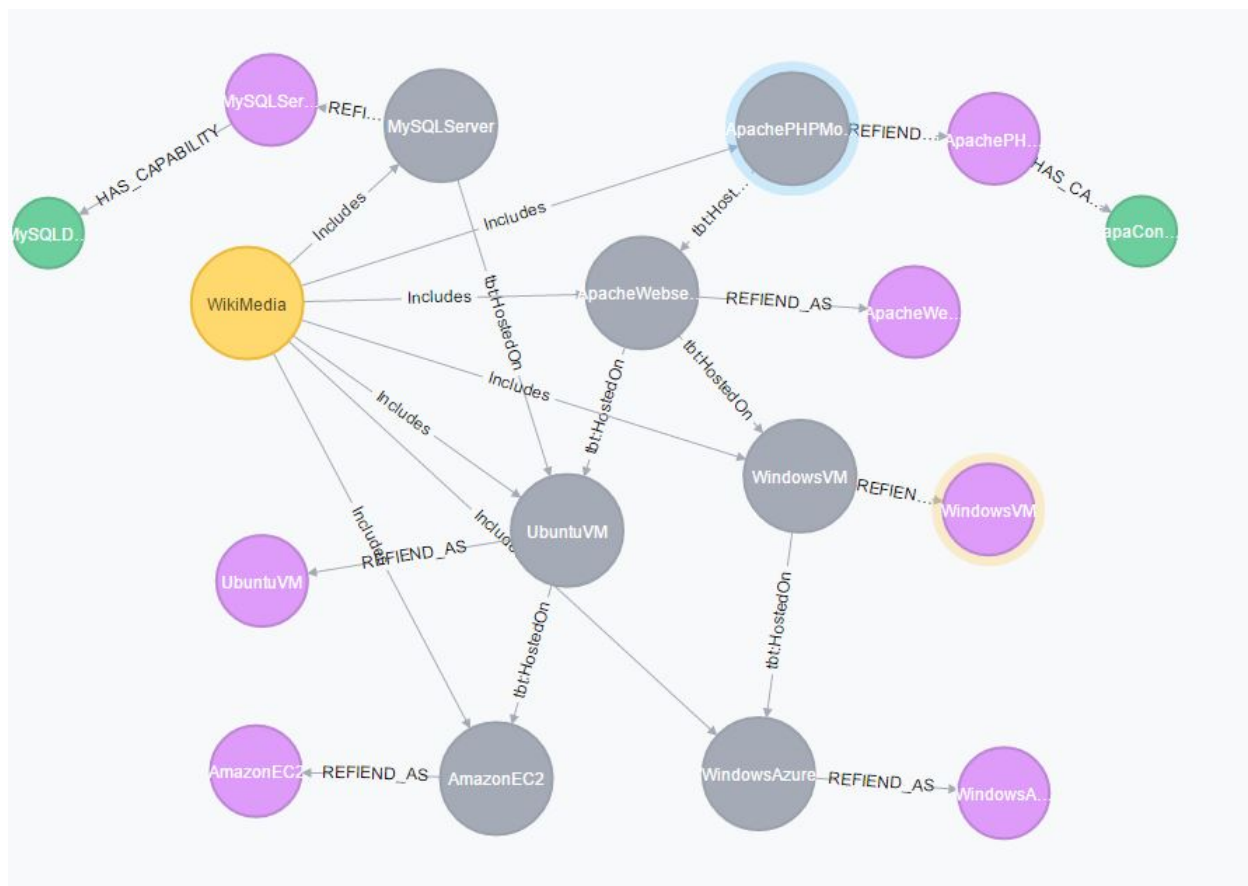


Figure 7.25: Concrete Nodes in Database

7.4.3 Instance Node

POST ▼ | http://localhost:8080/pertos/instancenode

Authorization | Headers (1) | **Body** | Pre-request scrip

form-data x-www-form-urlencoded raw binary XML (application/xml)

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <tosca:Definitions id="winery-defs-for_ex1-MediaWiki" targetNamespace="http://www.example.org/">
3   <tosca:ServiceTemplate id="MediaWiki" name="MediaWiki" targetNamespace="http://www.example.org/">
4     <winery:Properties xmlns:ns1="http://www.eclipse.org/winery/mode">
5     </winery:Properties>
6     <tosca:TopologyTemplate>
7       <tosca:NodeTemplate name="ApachePHPModule" id="ApachePHPModule">
8         <tosca:Properties>
9           <ns3:Properties xmlns:ty="http://docs.oasis-open.org/tosca/ns/2011/12" ty="ApachePHPModule">
10            </ns3:Properties>
11         </tosca:Properties>
12       </tosca:NodeTemplate>
13     </tosca:TopologyTemplate>
14   </tosca:ServiceTemplate>
15 </tosca:Definitions>
```

Body | Cookies | **Headers (4)** | Tests (0/0) | Status **201 Created** | Time **355 ms**

Content-Length → 0
Date → Mon, 28 Dec 2015 21:21:40 GMT
Location → http://localhost:8080/instancenode/74
Server → Apache-Coyote/1.1

Figure 7.26: Request and Response for Persisting an Instance Node

GET ▼ | http://localhost:8080/pertos/instancenode/74

Authorization | **Headers (1)** | Body | Pre-request scrip

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status **200 OK** | Time **3121 ms**

Pretty Raw Preview | XML ▼ | ☰

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Definitions xmlns="http://docs.oasis-open.org/tosca/ns/2011/12" targetNamespace="http://www.example.org/">
3   <ServiceTemplate targetNamespace="http://www.example.org/" name="MediaWiki">
4     <Properties xmlns:winery="http://www.opentosca.org/winery/mode">
5     </Properties>
6     <TopologyTemplate>
7       <NodeTemplate xmlns:tst="http://docs.oasis-open.org/tosca/ns/2011/12" name="ApachePHPModule">
8         <Properties>
9           <ns3:Properties xmlns:ns3="http://docs.oasis-open.org/tosca/ns/2011/12" ns3="ApachePHPModule">
10            </ns3:Properties>
11         </Properties>
12       </NodeTemplate>
13     </TopologyTemplate>
14   </ServiceTemplate>
15 </Definitions>
```

Figure 7.27: Retrieve an Instance Node By ID

GET ⌵ | http://localhost:8080/pertos/instancenode

Authorization | Headers (1) | Body | Pre-request script

No Auth ⌵

Body | Cookies | Headers (4) | Tests (0/0) | Status 200 OK | Time 489 ms

Pretty | Raw | Preview | XML ⌵ | ☰

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <InstanceNodeList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <specification DatabaseId="74">
4     <Definitions targetNamespace="http://www.example.org/" id="winery-defs-for_ex
5       <ServiceTemplate targetNamespace="http://www.example.org/" name="MediaWik
6         <winery:Properties xmlns:winery="http://www.opentosca.org/winery/exte
7           <TopologyTemplate>
8             <NodeTemplate xmlns:tst="http://docs.oasis-open.org/tosca/ns/2011
9               <Properties>
10                <ns3:Properties xmlns:ns3="http://docs.oasis-open.org/tos
11              </Properties>
12            </NodeTemplate>
13          </TopologyTemplate>
14        </ServiceTemplate>
15      </Definitions>
16    </specification>
17  <specification DatabaseId="75">
18    <Definitions         </Definitions>
19  </specification>
20  <specification DatabaseId="76">
21    <Definitions         </Definitions>
22  </specification>
23  <specification DatabaseId="77">
24    <Definitions         </Definitions>
25  </specification>
26  <specification DatabaseId="78">
27    <Definitions         </Definitions>
28  </specification>
29  <specification DatabaseId="79">
30    <Definitions         </Definitions>
31  </specification>
32  <specification DatabaseId="80">
33    <Definitions         </Definitions>
34  </specification>
35 </InstanceNodeList>

```

Figure 7.28: Retrieve all Instance Nodes

Figure 7.26, 7.27 and 7.28 shows persisting and retrieving instance nodes. One thing should be mentioned here is that when persisting instance node, system checks the Node Type of instance node and links the instance node to concrete node automatically. In Figure 7.29, the nodes in red are instance nodes. Up to now, gamma topology is established. we can use this reusable topology to discover viable topologies which is validated in next section. Another useful API is validated here as well. As showing in Figure 7.30, this API can provide all instance nodes which are linked to a concrete node (here ID:64).

7.4 Gamma Topology

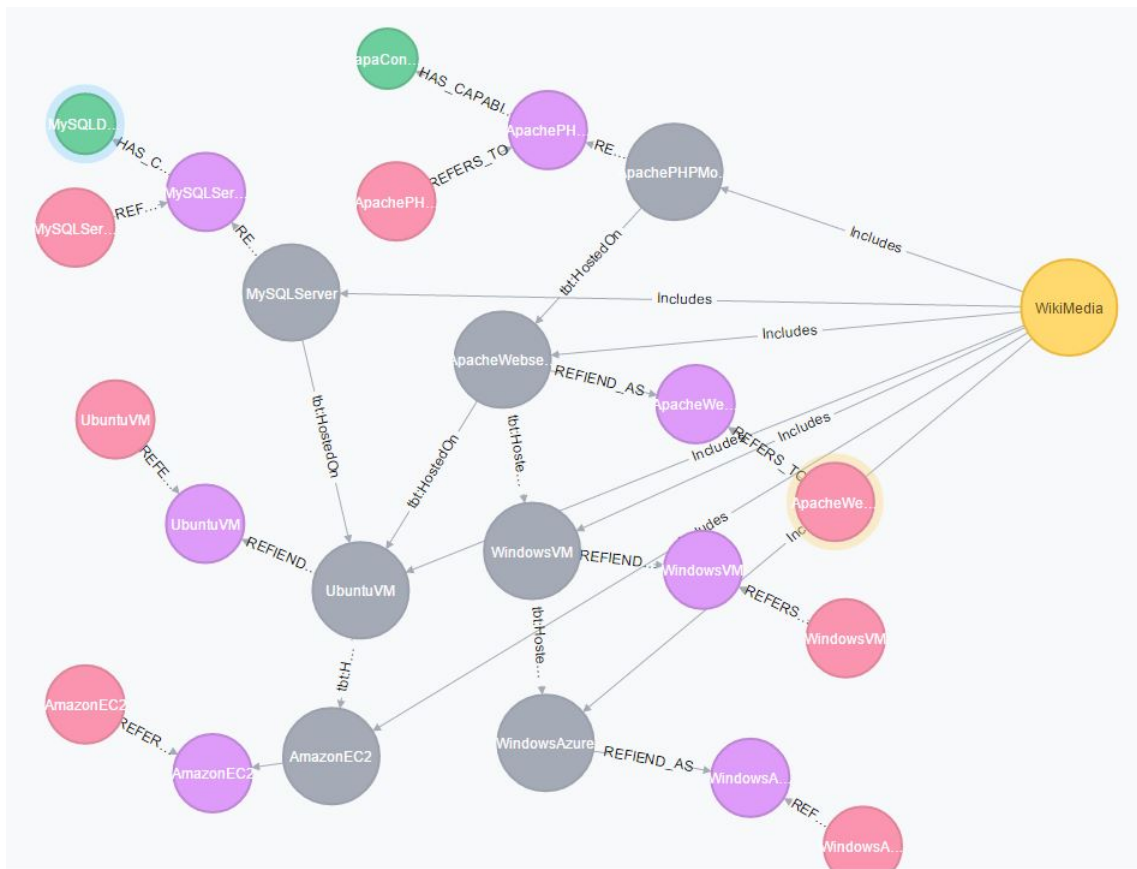


Figure 7.29: Instance Nodes in Database

GET

Authorization	Headers (1)	Body	Pre-request script	Tests
No Auth				

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 376 ms

Pretty Raw Preview XML

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <InstanceNodeList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <specification DatabaseId="74">
4     <Definitions targetNamespace="http://www.example.org/" id="winery-defs-for_ex1-MediaWiki">
5       <ServiceTemplate targetNamespace="http://www.example.org/" name="MediaWiki" id="MediaW:
6         <winery:Properties xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/
7         <TopologyTemplate>
8           <NodeTemplate xmlns:tst="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaSpec:
9             <Properties>
10              <ns3:Properties xmlns:ns3="http://docs.oasis-open.org/tosca/ns/2011/12,
11              </Properties>
12            </NodeTemplate>
13          </TopologyTemplate>
14        </ServiceTemplate>
15      </Definitions>
16    </specification>
17  </InstanceNodeList>
  
```

Figure 7.30: Instance Nodes linked to a Concrete Node

7.5 Topology Enrichments

In this section topology enrichments are validated.

7.5.1 Workload

Figure 7.31, 7.32 and 7.33 show persisting and retrieving workloads. Figure 7.34 shows an alpha topology performs a workload. Figure 7.35 and Figure 7.36 show an alpha topology retrieves and queries its performed workloads.

The screenshot displays a REST client interface for a POST request to `http://localhost:8080/pertos/workload`. The request body is XML, defining a workload with specific parameters. The response is a 201 Created status with a response time of 743 ms.

```
POST http://localhost:8080/pertos/workload
```

Authorization | Headers (1) | **Body** | Pre-request script | Tests

form-data x-www-form-urlencoded raw binary XML (application/xml) ▼

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <Workload id="workload_1" startTime="2015-11-22 00:00:00.000" endTime="2015-11-23 23:59:59.000">
3   <pattern>pattern_1</pattern>
4   <arrival>logarithmic</arrival>
5   <behavioral>gamma</behavioral>
6   <avg_users>200</avg_users>
7   <avg_transactions>5000</avg_transactions>
8 </Workload>
```

Body | Cookies | **Headers (4)** | Tests (0/0) | Status **201 Created** | Time **743 ms**

Content-Length → 0
Date → Tue, 29 Dec 2015 16:54:37 GMT
Location → http://localhost:8080/workload/97
Server → Apache-Coyote/1.1

Figure 7.31: Request and Response for Persisting a Workload

7.5 Topology Enrichments

GET http://localhost:8080/pertos/workload/97

Authorization Headers (1) Body Pre-request script Tests

No Auth

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 89 ms

Pretty Raw Preview XML

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Workload endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="workload_1">
3   <pattern>pattern_1</pattern>
4   <arrival>logarithmic</arrival>
5   <behavioral>gamma</behavioral>
6   <avg_users>200</avg_users>
7   <avg_transactions>5000</avg_transactions>
8 </Workload>
```

Figure 7.32: Retrieve a Workload By ID

GET http://localhost:8080/pertos/workload

Authorization Headers (1) Body Pre-request script Tests

No Auth

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 50 ms

Pretty Raw Preview XML

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <WorkloadList>
3   <workloadWithDatabaseID databaseId="97">
4     <workload endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="workload_1">
5       <pattern>pattern_1</pattern>
6       <arrival>logarithmic</arrival>
7       <behavioral>gamma</behavioral>
8       <avg_users>200</avg_users>
9       <avg_transactions>5000</avg_transactions>
10    </workload>
11  </workloadWithDatabaseID>
12  <workloadWithDatabaseID databaseId="98">
13    <workload endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="workload_2">
14      <pattern>pattern_2</pattern>
15      <arrival>logarithmic</arrival>
16      <behavioral>gamma</behavioral>
17      <avg_users>200</avg_users>
18      <avg_transactions>5000</avg_transactions>
19    </workload>
20  </workloadWithDatabaseID>
21  <workloadWithDatabaseID databaseId="99">
22    <workload endTime="2015-12-25 23:59:59.000" startTime="2015-12-22 00:00:00.000" id="workload_1">
23      <pattern>pattern_1</pattern>
24      <arrival>logarithmic</arrival>
25      <behavioral>gamma</behavioral>
26      <avg_users>200</avg_users>
27      <avg_transactions>5000</avg_transactions>
28    </workload>
29  </workloadWithDatabaseID>
30 </WorkloadList>
```

Figure 7.33: Retrieve all Workloads

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/pertos/topology/alphatopology/50/workload
- Body:**

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <PerformWorkloadID>
3 <id>97</id>
4 </PerformWorkloadID>

```
- Form Data:** form-data, x-www-form-urlencoded, raw (selected), binary, XML (application/xml)
- Status:** 200 OK, Time: 364 ms
- Message:** employ workload ID:97 to AlphaTopology ID:50 Successfully!

Figure 7.34: An Alpha Topology performs a workload

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/pertos/topology/alphatopology/50/workload/all
- Body:**

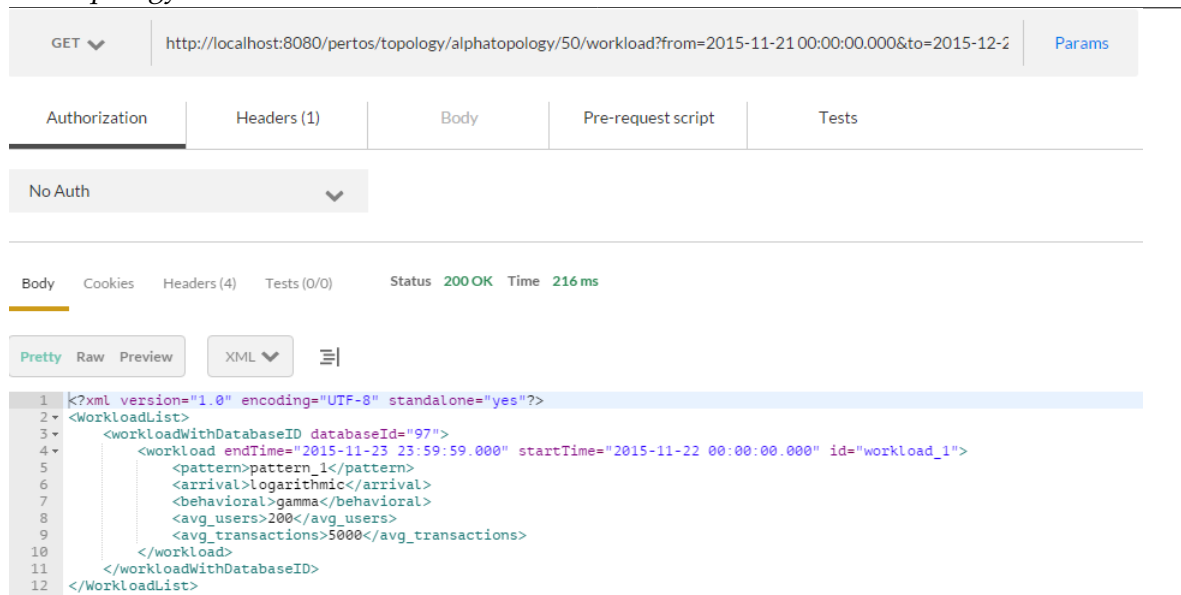
```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <WorkloadList>
3 <workloadWithDatabaseID databaseId="97">
4 <workload endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="workload_1">
5 <pattern>pattern_1</pattern>
6 <arrival>logarithmic</arrival>
7 <behavioral>gamma</behavioral>
8 <avg_users>200</avg_users>
9 <avg_transactions>5000</avg_transactions>
10 </workload>
11 </workloadWithDatabaseID>
12 </WorkloadList>

```
- Form Data:** No Auth
- Status:** 200 OK, Time: 252 ms

Figure 7.35: An alpha topology retrieves all its Workloads

7.5 Topology Enrichments



The screenshot shows a REST client interface. At the top, a GET request is shown to the URL `http://localhost:8080/pertos/topology/alphatopology/50/workload?from=2015-11-21 00:00:00.000&to=2015-12-2`. Below the request bar, there are tabs for Authorization, Headers (1), Body, Pre-request script, and Tests. The Authorization tab is selected, showing "No Auth". Below the tabs, the response details are shown: Status 200 OK, Time 216 ms. The response body is displayed in XML format, showing a `WorkloadList` containing one `workloadWithDatabaseID` element with a `workload` child element. The `workload` element has attributes `endTime="2015-11-23 23:59:59.000"` and `startTime="2015-11-22 00:00:00.000"`, and a `id="workload_1"`. The `workload` element contains several sub-elements: `pattern` (pattern_1), `arrival` (logarithmic), `behavioral` (gamma), `avg_users` (200), and `avg_transactions` (5000).

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <WorkloadList>
3   <workloadWithDatabaseID databaseId="97">
4     <workload endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="workload_1">
5       <pattern>pattern_1</pattern>
6       <arrival>logarithmic</arrival>
7       <behavioral>gamma</behavioral>
8       <avg_users>200</avg_users>
9       <avg_transactions>5000</avg_transactions>
10    </workload>
11  </workloadWithDatabaseID>
12 </WorkloadList>
```

Figure 7.36: An Alpha Topology queries its Workloads

7.5.2 Performance

Figure 7.37, 7.38 and 7.39 show persisting and retrieving performances. Figure 7.40 shows an alpha topology performs a performance. Figure 7.41 and Figure 7.42 show an alpha topology retrieves and queries its performed performances.

POST http://localhost:8080/pertos/performance

Authorization Headers (1) Body Pre-request script Tests

form-data x-www-form-urlencoded raw binary XML(application/xml)

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <Performance id="Performance_4" startTime="2015-11-22 00:00:00.000" endTime="2015-11-23 23:59:59.000">
3   <time_behaviour>
4     <response_time>
5       <min>200</min>
6       <max>200</max>
7       <avg>200</avg>
8       <st>200</st>
9     </response_time>
10    <throughput>
11      <min>200</min>
12      <max>200</max>
13      <avg>200</avg>
14      <st>200</st>
15    </throughput>
16    <processing_time>
17      <min>200</min>
18      <max>200</max>
19      <avg>200</avg>
20      <st>200</st>
21    </processing_time>
22    <avg_read_speed>
23      <min>200</min>
24      <max>200</max>
25      <avg>200</avg>
26      <st>200</st>
  </Performance>

```

Body Cookies Headers (4) Tests (0/0) Status 201 Created Time 106 ms

Content-Length → 0
Date → Tue, 29 Dec 2015 17:06:50 GMT
Location → http://localhost:8080/performance/131
Server → Apache-Coyote/1.1

Figure 7.37: Request and Response for Persisting a Performance

GET http://localhost:8080/pertos/performance/131

Authorization Headers (1) Body Pre-request script Tests

No Auth

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 315 ms

Pretty Raw Preview XML

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Performance endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="Performance_4">
3   <time_behaviour></time_behaviour>
53  <capacity></capacity>
97  <resource_utilization></resource_utilization>
135 <scalability></scalability>
167 <availability></availability>
193 </Performance>

```

Figure 7.38: Retrieve a Performance By ID

7.5 Topology Enrichments

GET ▼ | http://localhost:8080/pertos/performance/

Authorization | Headers (1) | Body | Pre-request script | Tests

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status 200 OK | Time 340 ms

Pretty | Raw | Preview | XML ▼ | ≡

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <PerformanceList>
3   <performanceWithDatabaseID databaseId="100">
4     <performance endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="Performance_2">
5       <time_behaviour>[REDACTED]</time_behaviour>
55      <capacity>[REDACTED]</capacity>
99      <resource_utilization>[REDACTED]</resource_utilization>
137     <scalability>[REDACTED]</scalability>
169     <availability>[REDACTED]</availability>
195   </performance>
196 </performanceWithDatabaseID>
197 <performanceWithDatabaseID databaseId="131">
198   <performance endTime="2015-11-23 23:59:59.000" startTime="2015-11-22 00:00:00.000" id="Performance_4">
199     <time_behaviour>[REDACTED]</time_behaviour>
249     <capacity>[REDACTED]</capacity>
293     <resource_utilization>[REDACTED]</resource_utilization>
331     <scalability>[REDACTED]</scalability>
363     <availability>[REDACTED]</availability>
389   </performance>
390 </performanceWithDatabaseID>
391 </PerformanceList>
```

Figure 7.39: Retrieve all Performances

POST ▼ | http://localhost:8080/pertos/topology/alphatopology/50/performance

Authorization | Headers (1) | Body | Pre-request script

form-data x-www-form-urlencoded raw binary | XML (application/xml) ▼

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <PerformPerformanceID>
3   <id>131</id>
4 </PerformPerformanceID>
```

Body | Cookies | Headers (4) | Tests (0/0) | Status 200 OK | Time 257 ms

Pretty | Raw | Preview | HTML ▼ | ≡

```
1 employ performanace ID:131 to AlphaTopology ID:50 Successfully!
```

Figure 7.40: An Alpha Topology performs a Performance

GET ▼ | http://localhost:8080/pertos/topology/alphatopology/50/performance

Authorization | Headers (1) | Body | Pre-request script

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status **200 OK** | Time **254 ms**

Pretty | Raw | Preview | XML ▼ | ☰

```

1 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 | <PerformanceList>
3 |   <performanceWithDatabaseID databaseId="131">
4 |     <performance />
196 |   </performanceWithDatabaseID>
197 | </PerformanceList>

```

Figure 7.41: An alpha topology retrieves all its Performances

GET ▼ | http://localhost:8080/pertos/topology/alphatopology/50/performance?from=2015-11-21 00:00:00.000&to=2015-11-21 00:00:00.000

Authorization | Headers (1) | Body | Pre-request script | Tests

No Auth ▼

Body | Cookies | Headers (4) | Tests (0/0) | Status **200 OK** | Time **282 ms**

Pretty | Raw | Preview | XML ▼ | ☰

```

1 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 | <PerformanceList>
3 |   <performanceWithDatabaseID databaseId="131">
4 |     <performance />
196 |   </performanceWithDatabaseID>
197 | </PerformanceList>

```

Figure 7.42: An Alpha Topology queries its Performances

7.6 Viable Distribution Topologies Discovery

Figure 7.43 shows the process of discovering viable topology for an alpha topology. Here we try to find all viable topologies for alpha topology with ID 50. There are two discovered topologies in the example. TOSCA definition is wrapped in element *specification*.

To validate if the discovered viable topologies are correct, we use Winery to model the topology. Figure 7.44 and Figure 7.45 are the screenshot of the modeling result for our discovered viable topologies. The result are the same as we expected. The topology structure is decided by abstract sub-topology, and the combination of instance nodes linked to each concrete node compose the final viable topologies.

In the next step, we select one viable topology as the one to be deployed. First we persist it in the database as showing in Figure 7.46. Once the viable topology is persisted, system links the viable topology to the alpha topology automatically and relevant attributes are set like *create date*, *obsolete* and so on. In the example, we persist a second viable topology. It means the topology developer wants another viable topology to be deployed, so the first one is obsolete. Figure 7.47 shows the result when trying to get viable topology history of an alpha topology. The viable topology with databaseID 95 is the first one persisted, once the second viable topology is persisted, it became an *obsolete* viable topology, so the attribute *obsolete* is set to *Yes*.

```

1 k?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <MuTopologList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <specification id="1">
4     <Definitions targetNamespace="http://www.example.org/" id="winery-defs-for_ex1-MediaWiki">
5       <ServiceTemplate <img alt="copy icon" data-bbox="365 765 375 775"/></ServiceTemplate>
102     </Definitions>
103   </specification>
104   <specification id="2">
105     <Definitions targetNamespace="http://www.example.org/" id="winery-defs-for_ex1-MediaWiki">
106       <ServiceTemplate <img alt="copy icon" data-bbox="365 805 375 815"/></ServiceTemplate>
190     </Definitions>
191   </specification>
192 </MuTopologList>

```

Figure 7.43: Discover Viable Topologies

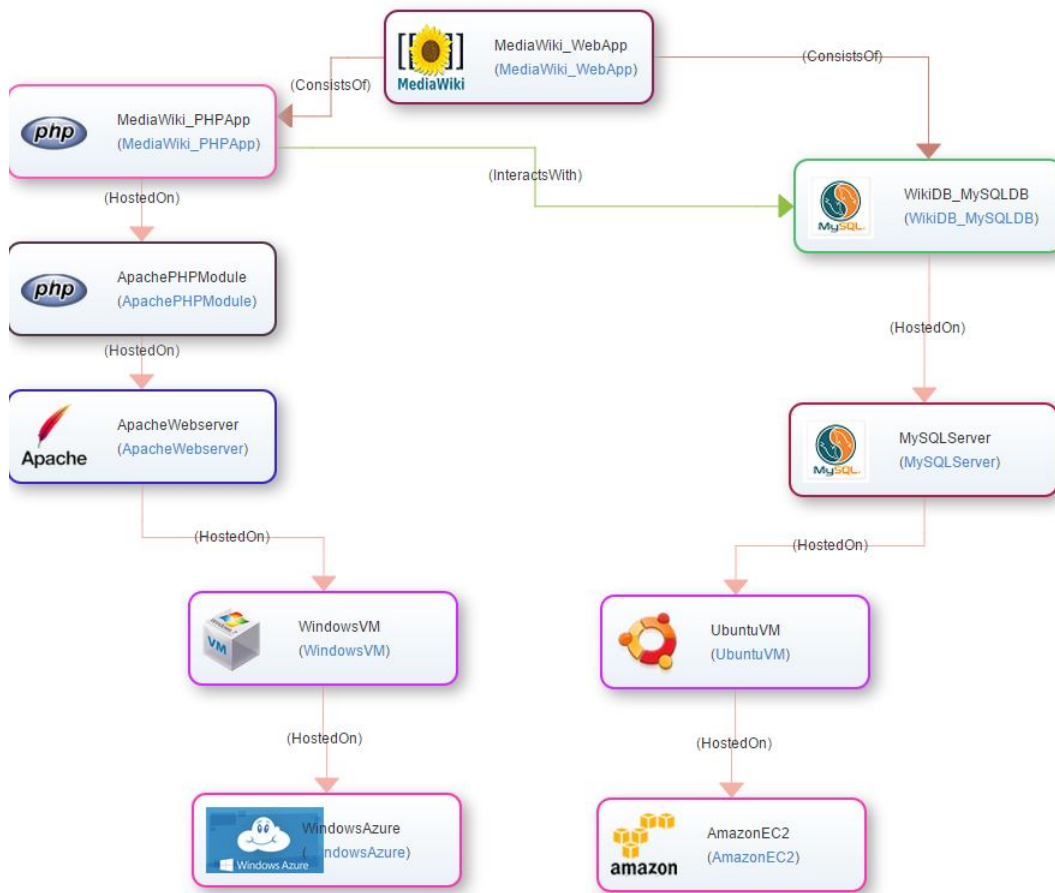


Figure 7.44: Verify the generated Topology in Winery-1

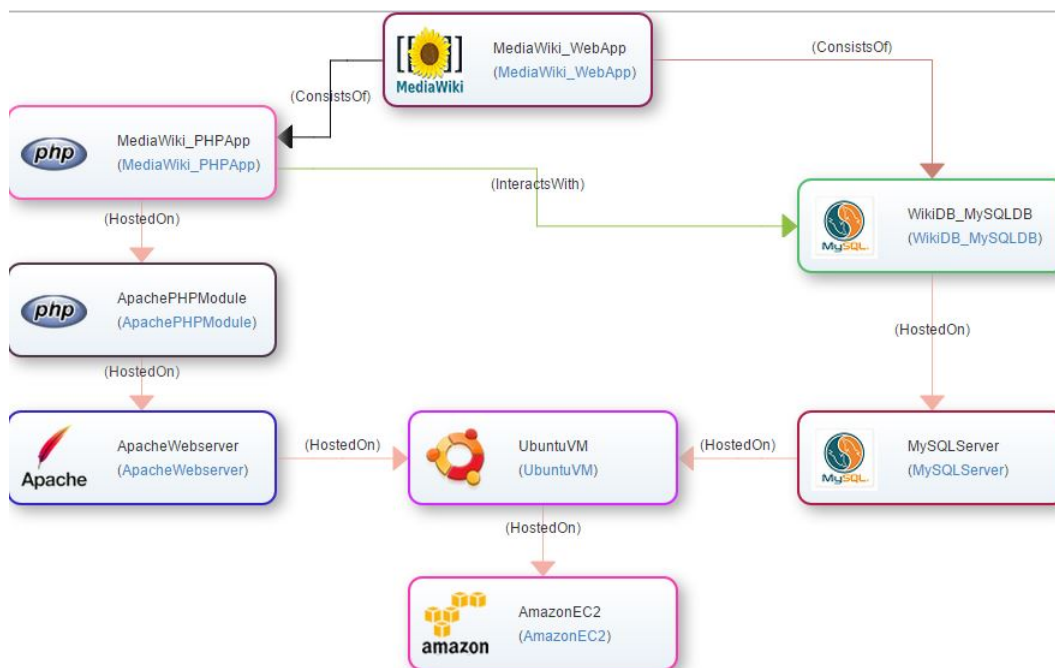


Figure 7.45: Verify the generated Topology in Winery-2

7.6 Viable Distribution Topologies Discovery

POST http://localhost:8080/pertos/viabletopology

Authorization Headers (1) Body Pre-request script

form-data x-www-form-urlencoded raw binary XML (application/xml)

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <tosca:Definitions xmlns:tosca="http://docs.oasis-open.org/tosca/ns/2011/12" targetN
3   <tosca:ServiceTemplate targetNamespace="http://www.example.org/" name="MediaWiki
4     <ns3:Properties xmlns:ns3="http://www.opentosca.org/winery/extensions/tosca/
5     </ns3:Properties>
6     <tosca:TopologyTemplate>
7       <tosca:NodeTemplate xmlns:ns3="http://www.example.com" xmlns:ns4="http://
8         <tosca:Policies>
9           <tosca:Policy xmlns:ns5="http://www.example.org/PolicyTemplates"
10          </tosca:Policy>
11        </tosca:Policies>
12      </tosca:NodeTemplate>
13      <tosca:NodeTemplate xmlns:ns3="http://www.example.com" xmlns:ns4="http://
14        <tosca:Policies>
15          <tosca:Policy xmlns:ns5="http://www.example.org/PolicyTemplates"
16          </tosca:Policy>
17        </tosca:Policies>
18      </tosca:NodeTemplate>
19      <tosca:RelationshipTemplate xmlns:ns3="http://www.example.com" name="cor
20        <tosca:SourceElement ref="MediaWiki_WebApp"/>
21        <tosca:TargetElement ref="WikiDB_MySQLDB"/>
22      </tosca:RelationshipTemplate>
23      <tosca:RelationshipTemplate xmlns:ns3="http://www.example.com" name="cor
24        <tosca:SourceElement ref="MediaWiki_WebApp"/>
25        <tosca:TargetElement ref="MediaWiki_PHPApp"/>
26      </tosca:RelationshipTemplate>

```

Body Cookies Headers (4) Tests (0/0) Status 201 Created Time 624ms

Content-Length → 0
Date → Tue, 29 Dec 2015 09:33:13 GMT
Location → http://localhost:8080/mutopology/95
Server → Apache-Coyote/1.1

Figure 7.46: Persist a Viable Topology

Besides retrieving all history of viable topologies for an alpha topology, the history can be queried. Figure 7.48 shows querying viable topology by providing timestamps as parameters. Figure 7.49 shows the data in database. The two nodes linked to alpha topology with ID 50 are viable topologies.

Figure 7.49 is the full picture in database of all above operations. Up to now, viable topologies for an alpha topology with ID 50 are discovered and persisted, workloads and performances are persisted and performed by the same alpha topology. Through this validation, we design a process and simulate a scenario that how a topology developer works with topology persistence and discovery system. The REST API and data modeling work as we expected.

GET ▼ http://localhost:8080/pertos/topology/alphatopology/50/viabletopology/

Authorization Headers (1) Body Pre-request script Tests

No Auth ▼

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 22448 ms

Pretty Raw Preview XML ▼ ≡

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <ViableTopologyList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <viableTopologyWithDababaseID databaseId="96">
4     <viableTopology>
5       <alphaTopologyId>MediaWiki</alphaTopologyId>
6       <alphaTopologyName>MediaWiki</alphaTopologyName>
7       <alphaTopologyNameSpace>http://www.example.org/</alphaTopologyNameSpace>
8       <Definitions targetNameSpace="http://www.example.org/" id="winezy-defs-for_ex1-MediaWiki">
9         <ServiceTemplate />
10      </Definitions>
107     <obsolete>no</obsolete>
108     <createDate>2015-12-29 10:41:46.694</createDate>
109     <endDate></endDate>
110   </viableTopology>
111 </viableTopologyWithDababaseID>
112   <viableTopologyWithDababaseID databaseId="95">
113     <viableTopology>
114       <alphaTopologyId>MediaWiki</alphaTopologyId>
115       <alphaTopologyName>MediaWiki</alphaTopologyName>
116       <alphaTopologyNameSpace>http://www.example.org/</alphaTopologyNameSpace>
117       <Definitions targetNameSpace="http://www.example.org/" id="winezy-defs-for_ex1-MediaWiki">
118         <ServiceTemplate />
119       </Definitions>
120     <obsolete>yes</obsolete>
121     <createDate>2015-12-29 10:33:12.624</createDate>
122     <endDate>2015-12-29 10:41:46.839</endDate>
123   </viableTopology>
124 </viableTopologyWithDababaseID>
125 </ViableTopologyList>

```

Figure 7.47: Get all Viable Topologies of an Alpha Topology

GET ▼ http://localhost:8080/pertos/topology/alphatopology/50/viabletopology?from=2015-11-21 00:00:00.000&to=2015-11-21 00:00:00.000 Params

Authorization Headers (1) Body Pre-request script Tests

No Auth ▼

Body Cookies Headers (4) Tests (0/0) Status 200 OK Time 411 ms

Pretty Raw Preview XML ▼ ≡

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <ViableTopologyList xmlns="http://docs.oasis-open.org/tosca/ns/2011/12">
3   <viableTopologyWithDababaseID databaseId="95">
4     <viableTopology>
5       <alphaTopologyId>MediaWiki</alphaTopologyId>
6       <alphaTopologyName>MediaWiki</alphaTopologyName>
7       <alphaTopologyNameSpace>http://www.example.org/</alphaTopologyNameSpace>
8       <Definitions />
9       <obsolete>yes</obsolete>
10      <createDate>2015-12-29 10:33:12.624</createDate>
109     <endDate>2015-12-29 10:41:46.839</endDate>
110   </viableTopology>
111 </viableTopologyWithDababaseID>
112 </ViableTopologyList>

```

Figure 7.48: Query Viable Topologies of an Alpha Topology

7.6 Viable Distribution Topologies Discovery

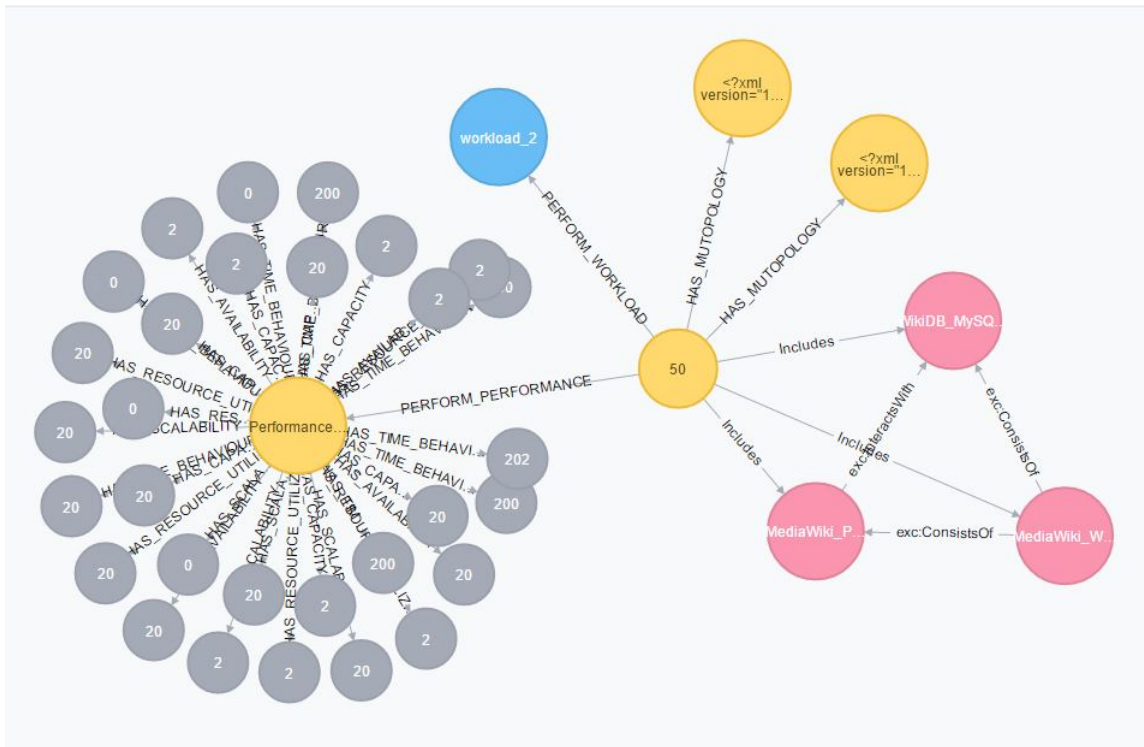


Figure 7.49: An Alpha Topology With its Viable Topologies, workloads and performances

8 Outcome and Future Work

The topology description language makes distributed deployment of cloud application across different service providers possible. When selecting and deciding an optimal application topology, various criteria from different dimensions should be taken into account. Both consumers and application developers will benefit more if they can decide which component of the application hosts on which service offering by considering evolving workload and performance. Basing on this scenario researches are conducted on discovering all potential application topologies and selecting the optimal one by taking into account overall aspects of application. Following the approaches proposed in [ASLW14] and [SAGF15], this Master thesis implements a RESTful-based framework which supports discovering and building reusable Cloud application viable topologies w.r.t evolving aspects of application.

In Chapter 2, the necessary background, related technologies and products using in this thesis are presented. In particular we provide a brief introduction of Graph database, which has native advantages when resolving graph problem compared to relational database. The most popular Graph database in industry is Neo4j which is selected and used in the implementation phase of this thesis.

In Chapter 3 we conduct literature researches which reviews the development of Cloud computing theory, especially in the area of application topology description and discovering. Some approaches are already proposed for decision making of optimal topologies like [BBKL14] and [VAL13], however these approaches ignore distributed deployment of application. In contrast, some approaches like [BFL⁺12] only takes into account automatically discovering topology but without considering other criteria: performance, security, QoS etc. So an overall and comprehensive approach is needed. By comparing with other solutions and tracing the latest progress, finally the approach proposed in [ASLW14] and [SAGF15] is used as theoretical foundation of this thesis.

Basing on the researches in Chapter 2 and Chapter 3, in Chapter 4 we provide the necessary data models to support the definition of application topologies: α , γ , μ - topology and associated enrichments. Compared to relational database modeling approaches like UML, there is no standard way to represent data model of Graph database. So we use four basic elements: Node, Label, Relationship and Property to model topology and its enrichments by using sample application *MediaWikiApp*. Furthermore, we establish concepts for the purpose of better modeling. For instance, we propose the concept of node type tree to model typed Graph with inheritance. Basing on this concept, γ -topology is splitted into three components: abstract sub-topology, concrete node and instance node. The intention of design of γ -topology is to make it easily extensible and maintainable: the structure of an application(abstract sub-topology) is relatively stable compare to its concrete implementation(instance nodes): the configuration, price, capacity etc. is various and changed from time to time. In the rest of this

Chapter, the functional and non-functional requirements the system must fulfill are described. After analyzing the requirements, providing an overview of the system, and specifying the necessary use cases, we move to the design of the prototype in chapter 5.

In the fifth Chapter, we design a RESTful framework capable of persisting and discovering application specific and independent topologies. For such a purpose, we first identify the resource and model relationships among them. Then we design representation of each resource using XML schema. Finally we define the endpoint and action of the resource. We divide service layer into six sub-modules which are responsible for business logical like topology enrichments and interpreter. The core algorithms designed for topology discovery and similar topology matching are presented in the rest of this chapter.

The implementation and validation of the system is introduced in Chapters 6 and 7. We use several technologies. For instance, Maven is used to manage the whole project and split the project into five sub-projects. RESTful API is developed under Jersey framework. When handling XML representation of topology and other topology enrichments, Jersey embedded approach, JAXB and DOM are adopted. To separate data access and business logic, a reusable data access object(DAO) is designed for database accessing. To validate our prototype system, we design a scenario of five steps with a sample application from scratch for validating. Finally, the validation result shows that the prototype of topology persistence and discovery system fulfill the requirement as expected.

The research in this thesis serves as the foundations for enabling the persistence and retrieval of reusable application topologies leveraging Graph database technologies. Furthermore, this thesis provides a approach how to use Graph database to resolve application topology problems. With the help of Graph database, topology discovery algorithm are designed using recursive and topology theory. Topology persistence and discovery system can be extended to support various topology description languages in the future. In addition, this system can be integrated with other systems – topology modeler and provision system to build a comprehensive Cloud application topology development ecosystem. More business logic can be developed by reusing the DAO sub-module of service layer. It can be predicted that the combination of Graph database and Cloud application topology will become the focus of academic research in the near future.

Bibliography

- [Ama] A. Amazon. Cloud Formation. Amazon Web Services. AWS CloudFormation. Available online: <http://aws.amazon.com/de/cloudformation>.
- [ARB12] A.-F. Antonescu, P. Robinson, and T. Braun. Dynamic topology orchestration for distributed cloud-based applications. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 116–123. IEEE, 2012.
- [ARSL14] V. Andrikopoulos, A. Reuter, S. G. Sáez, and F. Leymann. A GENTL Approach for Cloud Application Topologies. In *Service-Oriented and Cloud Computing*, pages 148–159. Springer, 2014.
- [ASLW14] V. Andrikopoulos, S. G. Sáez, F. Leymann, and J. Wettinger. Optimal distribution of applications in the cloud. In *Advanced Information Systems Engineering*, pages 75–90. Springer, 2014.
- [BBH⁺13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner. OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In *11th International Conference on Service-Oriented Computing*, LNCS. Springer, 2013.
- [BBKL13] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann. Automated Discovery and Maintenance of Enterprise Topology Graphs. In *Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA 2013)*, pages 126–134. IEEE Computer Society, December 2013.
- [BBKL14] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann. chapter TOSCA: Portable Automated Deployment and Management of Cloud Applications, pages 527–549. Springer, New York, January 2014.
- [BCJ] H. Bruneliere, J. Cabot, and F. Jouault. Combining model-driven engineering and cloud computing. In *Modeling, Design, and Analysis for the Service Cloud-MDA4ServiceCloud’10: Workshop’s 4th edition (co-located with the 6th European Conference on Modelling Foundations and Applications-ECMFA 2010)*.
- [BEDL⁺03] R. Bardohl, H. Ehrig, J. De Lara, O. Runge, G. Taentzer, and I. Weinhold. *Node type inheritance concept for typed graph transformation*. Technische Universität Berlin, Fakultät IV-Elektrotechnik und Informatik, 2003.
- [BFL⁺12] T. Binz, C. Fehling, F. Leymann, A. Nowak, and D. Schumm. Formalizing the Cloud through Enterprise Topology Graphs. In *Proceedings of 2012 IEEE International Conference on Cloud Computing*. IEEE Computer Society Conference Publishing Services, June 2012.

- [BPM12] E. Brandtzæg, M. Parastoo, and S. Mosser. Towards a domain-specific language to deploy applications in the clouds. In *3rd International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 213–218, 2012.
- [BS13] A. Brogi and J. Soldani. Matching cloud services with TOSCA. In *Advances in Service-Oriented and Cloud Computing*, pages 218–232. Springer, 2013.
- [BS14] A. Brogi and J. Soldani. Reusing cloud-based services with TOSCA. In E. Plödereder, L. Grunske, E. Schneider, and D. Ull, editors, *44. Jahrestagung der Gesellschaft für Informatik, INFORMATIK 2014*, volume 232 of *LNI*, pages 235–246. GI, 2014.
- [Bur13] B. Burke. *RESTful Java with JAX-RS 2.0*. O’Reilly Media, 2013.
- [CFSV04] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367–1372, 2004.
- [de15] db engines. DB-Engines Ranking of Graph DBMS. <http://db-engines.com/en/ranking/graph+dbms>, 2015. [(L1) – (General Deployment Options)].
- [EPN⁺11] E. Escalona, S. Peng, R. Nejabati, D. Simeonidou, J. Garcia-Espin, J. Ferrer, S. Figuerola, G. Landi, N. Ciulli, J. Jimenez, et al. GEYSERS: a novel architecture for virtualization and co-provisioning of dynamic optical networks and IT services. In *Future Network & Mobile Summit (FutureNetw)*, 2011, pages 1–8. IEEE, 2011.
- [Fie00] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
- [Fro14] S. FroberOfBits. Neo4j Embedded vs Server mode. <http://stackoverflow.com/questions/24925800/neo4j-server-vs-embedded-mode>, 2014. [(L1) – (General Deployment Options)].
- [Gar85] C. Gary. *Introductory Graph Theory*, 1985.
- [Hec06] R. Heckel. Graph transformation in a nutshell. *Electronic notes in theoretical computer science*, 148(1):187–198, 2006.
- [KBBL13a] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery – Modeling Tool for TOSCA-based Cloud Applications. In *11th International Conference on Service-Oriented Computing*, LNCS. Springer, 2013.
- [KBBL13b] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery—a modeling tool for TOSCA-based cloud applications. In *Service-Oriented Computing*, pages 700–704. Springer, 2013.
- [LK09] H. J. La and S. D. Kim. A systematic process for developing high quality saas cloud services. In *Cloud computing*, pages 278–289. Springer, 2009.
- [LM12] T. Leidig and C. Momm. USDL service level agreement, 2012.

- [MG11] P. Mell and T. Grance. The NIST definition of cloud computing. 2011.
- [Mie10] R. Mietzner. A method and implementation to define and provision variable composite applications, and its usage in cloud computing. 2010.
- [Neo16] Neo4j. Caches in Neo4j. http://neo4j.com/docs/2.2.0-M03/configuration-caches.html#_file_buffer_cache, 2016. [(L1) – (Java API for RESTful Web Services)].
- [Nie16] J. V. P. Nieves. Master Thesis No. 0838-007 - Knowledge Capturing and Usage of Evolving Cloud Application Topologies. Master’s thesis, University Stuttgart, Germany, 2016.
- [NLPVDH12] D. K. Nguyen, F. Lelli, M. P. Papazoglou, and W.-J. Van Den Heuvel. Blueprinting approach in support of cloud computing. *Future Internet*, 4(1):322–346, 2012.
- [OAS15a] OASIS. TOSCA. <https://www.oasis-open.org/committees/tosca/charter.php>, 2015. [(L1) – (TOSCA Overview)].
- [Oas15b] Oasis. TOSCA Schema. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/schemas/TOSCA-v1.0.xsd>, 2015. [(L1) – (TOSCA)].
- [Ori] OrientDB.
- [PR69] J. L. Pfaltz and A. Rosenfeld. Web grammars. In *Proceedings of the 1st international joint conference on Artificial intelligence*, pages 609–619. Morgan Kaufmann Publishers Inc., 1969.
- [RMVG⁺10] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010.
- [Rou16] M. Rouse. cloud computing definition. <http://searchcloudcomputing.techtarget.com/definition/cloud-computing>, 2016. [(L1) – (Cloud Computing)].
- [RWE15] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases: New Opportunities for Connected Data*. O’Reilly Media, 2015.
- [SAGF15] S. G. Sáez, V. Andrikopoulos, K. Ganguly, and L. Frank. Enriching Cloud Application Topologies with Evolving Performance & Workload Models. *IAAS, University of Stuttgart*, 2015.
- [SBB⁺15] J. Soldani, T. Binz, U. Breitenbücher, F. Leymann, and A. Brogi. TOSCA-MART: A Method for Adapting and Reusing Cloud Applications. 2015.
- [TOS] Topology and Orchestration Specificatoin for Cloud Application Versioin 1.0.
- [Tru13] R. Trudeau. *Introduction to Graph Theory*. Dover Books on Mathematics. Dover Publications, 2013.

-
- [TSB10] W.-T. Tsai, X. Sun, and J. Balasooriya. Service-oriented cloud computing architecture. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 684–689. IEEE, 2010.
- [Uni15] UniStuttgart. OpenTOSCA. <http://www.iaas.uni-stuttgart.de/OpenTOSCA>, 2015. [(L1) – (OpenTOSCA)].
- [VAL13] Z. S. Vasilios Andrikopoulos and F. Leymann. Supporting the Migration of Applications to the Cloud through a Decision Support System. In *Proceedings of the 6th IEEE International Conference on Cloud Computing (CLOUD 2013), June 27-July 2, 2013, Santa Clara Marriott, CA, USA*, pages 565–572. IEEE Computer Society, 2013.
- [VB14] R. Van Bruggen. *Learning Neo4j. Community Experience Distilled*. Packt Publishing, 2014.
- [VB15] B. Varanasi and S. Belida. *Spring REST*. Apress, 2015.
- [VMZ⁺10] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM, 2010.
- [wik15a] wikipedia. annotation. https://en.wikipedia.org/wiki/Java_annotation, 2015. [(L1) – (Java annotation)].
- [wik15b] wikipedia. Data access object. https://en.wikipedia.org/wiki/Data_access_object, 2015. [(L1) – (Data Access Object)].
- [wik15c] wikipedia. JAX-RS. https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services, 2015. [(L1) – (Java API for RESTful Web Services)].
- [wik16] wikipedia. Cloud Computing. https://en.wikipedia.org/wiki/Cloud_computing, 2016. [(L1) – (Cloud Computing)].
- [Wil12] B. Wilder. *Cloud architecture patterns: using microsoft azure*. "O'Reilly Media, Inc.", 2012.

All links were last followed on January 14, 2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, January 14, 2016

(Name)