PIACERE

DOML Specification

Editor(s):	Sergio Canzoneri, Bin Xiang, Adrian Noguero
Responsible Partner:	POLIMI, Go4It
Status-Version:	V3.1
Date:	15.09.2023
Distribution level (CO, PU):	Public

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	Annex to D3.3
Due Date of Delivery to the EC	31.05.2023

Workpackage responsible for the Deliverable:	WP3 - Plan and create Infrastructure as Code
Editor(s):	POLIMI, Go4It
Contributor(s):	POLIMI, Go4It
Reviewer(s):	Alfonso de la Fuente (Prodevelop)
Approved by:	All Partners
Recommended/mandatory readers:	WP4, WP5, WP6, WP7

Abstract:	This document is an annex to Deliverable D3.3 - PIACERE Abstractions, DOML and DOML-E - v1. It includes the detailed specification of the DOML concepts. It will be updated periodically with every new release of DOML. All releases are publicly available on the DOML website.
Keyword List:	DOML, Modelling abstractions, DOML concepts
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the authors' views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.01	08.10.2021	First draft version	GO4IT
v0.02	10.12.2021	Completed the DOML specification v1.0. Added the concrete layer and all DOML-E mechanisms. Completed the properties and updated examples	GO4IT, POLIMI
V0.1	20.12.2021	Syntax definition added, revision of the whole content	POLIMI
V2.0	30.06.2022	Updates to DOML V2.0	POLIMI
V2.0	01.07.2022	Review of DOML specification v2.0	POLIMI
V2.0	04.07.2022	Release of DOML v2.0	POLIMI
V2.1	05.10.2022	Release of DOML v2.1	POLIMI
V2.2	18.02.2023	Release of DOML v2.2	POLIMI
V2.2.1	04.03.2023	Release of DOML v2.2.1	POLIMI
V3.0	29.05.2023	Release of DOML v3.0	POLIMI
V3.1	15.09.2023	Release of DOML v3.1	POLIMI

Table of contents

List of figures

Terms and abbreviations

CSP	Cloud Service Provider
DevOps	Development and Operation
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
IaC	Infrastructure as Code
IEP	IaC execution platform
IOP	IaC Optimization
KPI	Key Performance Indicator
SW	Software

Executive Summary

This document contains the main concepts of the DOML language specification, as well as its extension mechanisms (DOML-E). The goal of the document is to serve as cornerstone for the implementation of DOML-based solutions in PIACERE, ranging from the IDE to the optimization algorithms.

1 Description of DOML

DOML specifies a common language for addressing the definition, deployment, and operation of complex cloud-based applications inside the PIACERE framework. DOML is intended to be used by users with different degrees of expertise, therefore, it has been conceived to be easy to use by non-expert users, but also expressive enough to allow expert users to get the most out of it.

DOML is a declarative language, thus, each of the layers describe what the application and infrastructure should look like after all the deploying is done. However, DOML allows the user to integrate imperative scripts, to actually describe some specific configurations whenever needed. The main goal of DOML is to serve as a bridge to the many IaC languages that currently exist (e.g., Terraform, TOSCA, Ansible), providing a degree of expressiveness that allows the PIACERE framework to generate IaC code easily in the specific formats.

In addition, DOML is intended to be used with the PIACERE optimization mechanisms. To achieve this DOML allows the user to define different application deployment configurations, as well as different infrastructure configurations, and it includes a specific layer to define optimization objectives and constraints.

Finally, DOML is envisaged as an evolving entity capable of coping with the constant advancements in the cloud computing state-of-the-art. As such, DOML includes extension mechanisms built inside that allow the user and the tools using DOML to create new concepts for any of the layers in DOML, as well as extend existing ones with new properties and attributes. These extension mechanisms are collectively called DOML-E.

This specification is intended to be used as a reference guide for the implementation of all DOML related tools.

1.1 DOML Layers

The DOML language specification is split into several packages, referred to as “layers”, which incrementally enrich the description of the cloud-based applications that will be managed inside PIACERE. Each layer provides a unique point of view of the applications; yet all the layers build up for a comprehensive application description.

The **Commons Layer** contains the main abstract application agnostic concepts that are shared among different layers. The DOML extension mechanisms (DOML-E) are also addressed in this layer by setting up the basic elements that will allow creating new concepts and properties in the top layers.

The **Application Layer** contains the information to describe the components and building blocks that compose the applications, as well as the functional requirements of each of them in terms of software interfaces and APIs. Finally, this layer describes how the application is deployed into the different infrastructure components.

The **Infrastructure Layer** defines the abstract infrastructure elements that will be used to deploy the application components. Concepts in this layer will include information that is relevant to meet the requirements of the applications. However, most of the concepts in this layer will require a concretization, or in other words, a more concrete instance they will be mapped on. For example, a virtual machine in this layer must be mapped to a concrete virtual machine instance, be it a VM from AWS or a specific VM deployed by the user.

The **Concrete Layer** provides the tools to concretize the infrastructure elements in the Infrastructure Layer and map them onto specific infrastructure instances either provided by cloud runtime providers, such as AWS or Google Cloud, or provided by the users.

The **Optimization Layer** defines all the information required for the optimizers to locate the best configurations for the cloud applications described in the DOML, as well as means to capture the optimization solutions.

2 Commons Layer

The following diagram (see Fig. 1) shows the main elements of the Commons Layer in DOML:

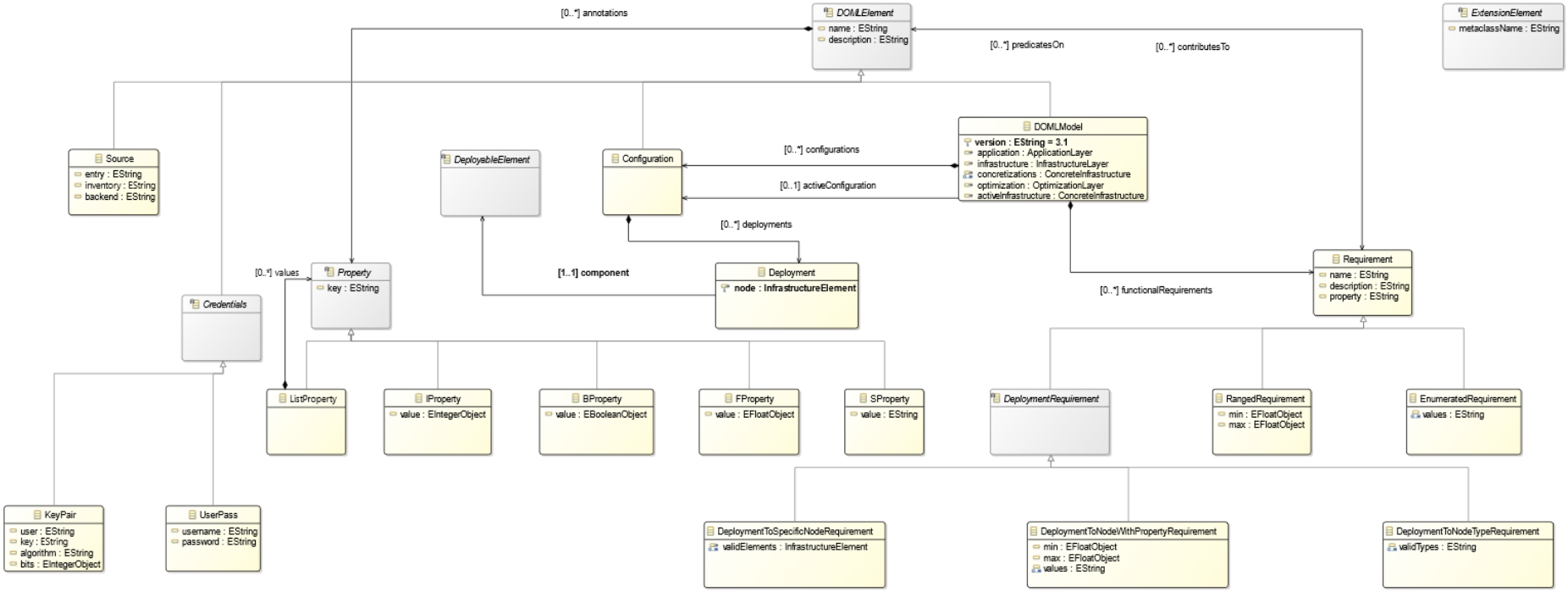

Figure 1. Commons Layer diagram

2.1 DOMLElement Class (abstract)

A DOMLElement represents any element inside the DOML language and it is intended to be the top meta-element of the DOML.

Attributes

name: String [1]	An identifier for this DOML Element
description: String [0..1]	An optional textual description of the Element. Used for documenting the element or similar purposes.

Associations

annotations: Property [0..*]	A set of properties used to modify the semantics of this particular element. These properties will add to the final semantics of the element, refining the DOML element to which they are applied. These properties will serve as the main extension mechanisms for DOML.
contributesTo: Requirement [0..*]	The set of requirements this DOMLElement contributes to achieving. This association is derived from the predicatesOn association of the Requirement class.

Constraints

* All properties added to a DOML element must have different keys.

Usage

DOMLElement is the common parent of all elements in DOML except the Property class. It is also the enabler for DOML-E extensions through the use of the Property elements.

2.2 Property Class (abstract)

A Property represents an additional information added to any DOML Element to further refine their meaning or semantics.

Attributes

key: String [1]	An identifier for this Property

Constraints

* All properties owned by a DOML element must have different keys.

Usage

Instances of the Property class are used to add information to DOML elements that cannot be described using that element's attributes and associations. Properties can be used as any or both attributes and associations to refine any DOML element.

2.3 IProperty Class

A Property with Integer value.

Superclass

Property

Attributes

value: Integer	Integer value of this IProperty

2.4 SProperty Class

A Property with String value.

Superclass

Property

Attributes

value: String	String value of this SProperty

2.5 FProperty Class

A Property with Float value.

Superclass

Property

Attributes

value: Float	Float value of this FProperty

2.6 BProperty Class

A Property with Boolean value.

Superclass

Property

Attributes

value: Boolean	Boolean value of this BProperty

2.7 ListProperty Class

A Property to group a set of sub-properties.

Superclass

Property

Associations

value: Property [0..*]	The set of properties belonging to this ListProperty

2.8 DOMLModel Class

A DOMLModel represents the design and development space for a cloud application or set of applications. The DOML model provides access to the different points of view of this space through the use of the model layers.

Superclass

DOMLElement

Attributes

version: readonly String [1];	A readonly string to mark the current DOML version ("3.1").

Associations

application: ApplicationLayer [0..1]	A reference to the Application Layer instance associated with this model.
infrastructure: InfrastructureLayer [0..1]	A reference to the Concrete Infrastructure Layer instance associated with this model.
concretizations: ConcreteInfrastructure [0..*]	A list of concrete infrastructures that map on to abstract infrastructure layer elements.
activeInfrastructure: ConcreteInfrastructure [0..1]	The ConcreteInfrastructure considered active for the current DOML specification
optimization:OptimizationLayer [0..1]	A reference to the Optimization Layer instance associated with this model.
configurations: Configuration [0..*]	All possible configurations of the current DOML specification
activeConfiguration: Configuration [0..1]	The Configuration instance considered as active
functionalRequirements: Requirement [0..*]	The set of functional requirements that are applicable to the current DOML specification.

Usage

A DOML model is intended to be used as the container for all the DOML layers defined in a particular design space. Each of those layers will provide a different point of view of the design space of the application. This element should be used as the root element of any DOML model.

2.9 Configuration Class

A Configuration describes how an application is intended to be deployed on top of the cloud infrastructure, and what credentials, parameters, etc. will apply to each of the application and/or infrastructure elements.

Superclass

DOMLElement

Associations

deployments: [0..*]	Deployment	A set of Deployment instances describing each of the links between an application component and a node of the infrastructure.

Constraints

* There must not be two Deployment instances inside a Configuration element with the same source and target elements.

Usage

A configuration must fully describe how an application will operate on top of a particular infrastructure. The parameters associated to each DOML element, whether it is an application component or an infrastructure node, will differ. So, the configuration element will use the Property list to include them, using the reference Association of the Property to describe the model element that a particular parameter affects.

2.10 Deployment Class

A Deployment element describes an association between an application component (e.g. a web application) and the infrastructure element that will host it (e.g. a Virtual Machine).

Associations

source: [1]	ApplicationComponent	The application component that will be deployed.
target: [1]	InfrastructureElement	The infrastructure element that will host/support the application component.

Usage

The deployment is designed to establish 1 to 1 relationships between application and infrastructure elements.

2.11 ExtensionElement Class (abstract)

A ExtensionElement abstract metaclass is used as a common meta-type for all the classes that are part of DOML-E extension mechanisms.

Attributes

metaclassName: String [1]	The name of the metaclass that will be added to DOML by using the extension class instance.

Usage

The extension element class must never be instantiated nor subclassed. Instead, all extension metaclasses in DOML (i.e. ExtApplicationComponent, ExtInfrastructureElement, ExtConcreteElement and ExtOptimizationObjective) extend this metaclass, in addition to other extensions.

2.12 Requirement Class

A Requirement represents an objective to be achieved by the current DOML specification. Requirements, whether they are functional, non-functional or optimization objectives, must be described in plain text and also annotations can be used to further qualify it, if needed.

Attributes

title: String [0..1]	An optional meaningful title for the requirement.
description: String [0..1]	A text further specifying the requirement.
property: String [0..1]	The property of the DOMLElement instances this requirement predicates on.

Associations

predicatesOn: DOMLElement [0..*]	A reference to the set of DOMLElement instances this requirement predicates on.

Constraints

* All requirements in a DOML model must have different identifiers.

Usage

Requirements are used to model objectives and restrictions the current DOML design must meet. These objectives should be as formal as possible; however, they can also be used in a less formal way using the textual attributes. The way to define them in a formal way is by using the “property” and the “predicatesOn” members. The Requirement class is also the parent of all formal requirements defined in DOML. The following diagram shows the requirements section of the commons layer in DOML.

2.13 RangedRequirement Class

A RangedRequirement is a formal requirement instance which establishes a range of valid values to a property in a set of DOMLElements.

Superclass

Requirement

Attributes

min: Float [0..1]	The minimum value of the property.
max: Float [0..1]	The maximum value of the property.

Constraints

* The property attribute of a RangedRequirement must always be set.

* The predicatesOn association must always be linked to at least one DOMLElement for a RangedRequirement

* At least the max or the min attributes of a RangedRequirement mus be set.

* A ranged requirement can only be applied to numeric properties.

Usage

A ranged requirement should be used to establish limits to the numeric properties that need them.

2.14 EnumeratedRequirement Class

A EnumeratedRequirement describes a formal requirement that restricts the number of valid values that a property of a certain DOML element may take.

Superclass

Requirement

Attributes

values: String [1..*]	The set of values that are valid for the property referred by this requirement.

Constraints

* The property attribute of the EnumeratedRequirement must always be set.

* The predicatesOn association must always be linked to at least one DOMLElement for a EnumeratedRequirement

* At least one value must be set in the attribute values.

Usage

An enumerated requirement is used to set a list of valid values for a particular property.

2.15 DeploymentRequirement Class (abstract)

A DeploymentRequirement class describes a restriction to be applied to the definition of configurations in the current DOML.

Superclass

Requirement

Constraints

* The predicatesOn association must always be linked to at least one DOMLElement for a DeploymentRequirement and they must all be ApplicationComponent instances.

Usage

A DeploymentRequirement is used as the common parent class to all deployment related formal requirements in DOML.

2.16 DeploymentToNodeTypeRequirement Class

A DeploymentToNodeTypeRequirement describes a formal requirement that restricts types of infrastructure elements that an application component can be deployed to.

Superclass

DeploymentRequirement

Attributes

validTypes: String [1..*]	The set of valid meta-types the application components (that this requirement predicates on) can be deployed to.

Constraints

* At least one value must be set in the validTypes attribute.

* Values in validTypes must all be valid names of meta-classes in DOML infrastructure layer that extend the InfrastructureElement class.

Usage

A requirement of this kind is used to make an application component or a set of components deployable only into certain types of infrastructure elements (for example, make a software package only deployable to physical nodes).

2.17 DeploymentToNodeWithPropertyRequirement Class

A DeploymentToNodeWithPropertyRequirement describes a formal requirement that restricts the infrastructure elements an application component can be deployed to, according to the value of a property.

Superclass

DeploymentRequirement

Attributes

min: Float [0..1]	The minimum value of the property.
max: Float [0..1]	The maximum value of the property.
values: String [0..*]	The set of values that are valid for the property referred by this requirement.

Constraints

* The property attribute of a DeploymentToNodeWithPropertyRequirement must always be set.

* At least the max, the min or the values attributes of a requirement of this kind must be set.

* If values is not empty, then min and max cannot be set.

* If min and/or max are set, then values has to be empty.

Usage

A DeploymentToNodeWithPropertyRequirement is used to restrict the valid infrastructure nodes an application component can be deployed to, according to the value of a property of the target infrastructure element (for example, a software interface can only be attached to a

network interface with a minimum speed of 1Gbps, or a dbms component can only be deployed to a node with location equal to Europe).

2.18 DeploymentToSpecificNodeRequirement Class

A DeploymentToSpecificNodeRequirement describes a formal requirement that restricts the set of valid infrastructure element an application component can be deployed to a specific list.

Superclass

DeploymentRequirement

Associations

validElements: InfrastructureElement [1..*]	The set of elements the application component referred to by this requirement can be deployed to.

Usage

A DeploymentToSpecificNodeRequirement provides a valid set of infrastructure elements to be used to deploy an application component or a set of application components.

2.19 Credentials Class (abstract)

The Credentials class represents the credentials for computing node.

Superclass

DOMLElement

2.20 KeyPair Class

The KeyPair class represents the key credentials for computing node.

Superclass

Credentials

Attributes

user: String	The user name.
key: String	The public key.
algorithm: String	The encryption algorithm name.
bits: Integer	The number of the bits used in the algorithm.

Usage

The KeyPair class is used to define the key pair generated in a specific computing node with the encryption algorithm, which will be used for login.

2.21 UserPass Class

The UserPass class represents the password credentials for computing node.

Superclass

Credentials

Attributes

username: String	The user name.
password: String	The password.

Usage

The UserPass class is used to define the user and password for login to a specific computing node.

2.22 DeployableElement Class

DeployableElement marks and enforces a class extending from it is deployable.

2.23 Source Class

Source describes the source code of a specific software component.

Superclass

DOMLElement

Attributes

entry: String [0..1]	Entry point of the source.
backend: String [0..1]	Backend for executing the source.
inventory: String [0..1]	Custom inventory for the source.

3 Application Layer

The following diagram (see Fig. 2) shows the main elements of the Application Layer in DOML:

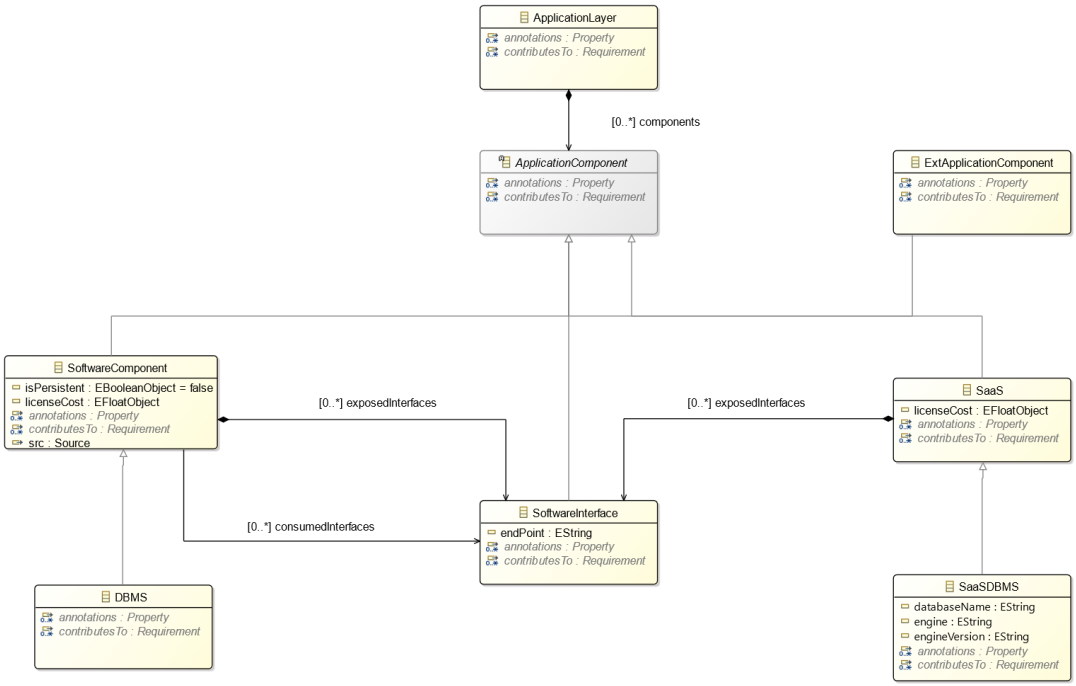

Figure 2. Application Layer diagram

3.1 ApplicationLayer Class

The Application class represent the container for all the components of the application in a DOML design. The representation of the Application Layer, and all the functional elements of the cloud application to be deployed, must be defined as application components inside it.

Superclass

DOMLElement

Associations

components: ApplicationComponent [0..*]	A containment reference to all the application components that will be part of the current application layer.

Usage

The Application is designed to be a container for ApplicationComponent instances.

3.2 ApplicationComponent Class (abstract)

The ApplicationComponent describes anything meaningful to the application being deployed in DOML from the functional perspective (e.g. software components, services or APIs). Each application component is susceptible of being deployed to an infrastructure element in the infrastructure model.

Superclass

DOMLElement, DeployableElement

Usage

The ApplicationComponent class is intended to be the common parent class for all elements in the application layer. Any common properties must always be specified on this class.

3.3 SoftwareComponent Class

The SoftwareComponent class describes any of the functional software components that conform an application in DOML. A software component may use or provide software interfaces, creating this way links among components, APIs and other functional elements in the application layer.

Superclass

ApplicationComponent

Attributes

isPersistent: Boolean [1]	A flag to indicate whether this component persists any information/state during operation. By default the value of this property is *false*.
licenseCost: Float [0..1]	An optional license cost (in Euro) associated to this software component.
src: Source	Source for the software component

Associations

exposedInterfaces: SoftwareInterface [0..*]	A set of software interfaces provided by this component for other software components to use.
consumedInterfaces: SoftwareInterface [0..*]	The set of software interfaces required by this component to fulfil its role.

Constraints

* Consumed interfaces must always refer to software interfaces exposed by other components or SaaS instances.

Usage

The SoftwareComponent class is intended to describe the main functional components or an application (e.g. web server, a REST API, etc.). It is important to note that software packages should be part of the components to be deployed in and are susceptible of having requirements attached to them.

3.4 SaaS Class

The SaaS class models an API that is external to our application, but relevant for functional purposes.

Superclass

ApplicationComponent

Attributes

licenseCost: Float [0..1]	An optional license cost (in Euro) associated to this SaaS.

Associations

exposedInterfaces: SoftwareInterface [0..*]	A set of software interfaces provided by this component for other software components to use.

Usage

The SaaS class is intended to describe APIs that are external to the current application, but are used by the software components inside it. SaaS components must not have requirements associated to them, the user has no control over them. SaaS instances may, however, define properties related to expected performance, response time, etc. if those are relevant for the current DOML model.

3.5 SoftwareInterface Class

The SoftwareInterface class models a software interface (e.g., a REST API, a TCP/IP connection, etc.) that connects two different application components in the application model.

Superclass

ApplicationComponent

Attributes

endPoint: String	The IP address / hostname / URL through which the service is accessed

Constraints

* A software interface must always be provided by one application component and used by at least one application component in the DOML model.

Usage

The SoftwareInterface class is intended to describe a connector between two different application components.

3.6 DBMS Class

The DBMS describes a software component that includes a Data Base Management System.

Superclass

SoftwareComponent

Constraints

* The isPersistent attribute of a DBMS component must always be set to *true*.

Usage

The DBMS is just a convenient subclass of the more generic SoftwareComponent class to model specifically DBMS.

3.7 SaaSDBMS Class

The SaaSDBMS describes an external API that will provide the DataBase Management System Functionality.

Superclass

SaaS

Attributes

databaseName: String [0..1]	The database name.
engine: String [0..1]	The engine name.
engineVersion: String [0..1]	The engine version.

Usage

The SaaSDBMS class is just a convenient subclass of the more generic SaaS class to model specifically a DBMS providing SaaS.

3.8 ExtApplicationComponent Class

The ExtApplicationComponent describes an instance of a new application layer concept. This class is part of DOML-E extension mechanisms.

Superclasses

ApplicationComponent, ExtensionElement

Usage

The ExtApplicationComponent class should be used to create instances of concepts and metaclasses not currently available in DOML.

4 Infrastructure Layer

The infrastructure layer describes the abstract infrastructure elements that will be supporting the execution of the application described in the ApplicationLayer. It is important to note that this abstract representation of the infrastructure is intended to be reused, mapping the elements on this layer to concrete instances in the infrastructure (e.g. an abstract virtual machine described in this layer will be mapped to a concrete VM instance, provided by a specific runtime provider, such as AWS or GoogleCloud).

The following diagram (see Fig. 3) shows the main elements of the Infrastructure Layer in DOML related to the infrastructure nodes:

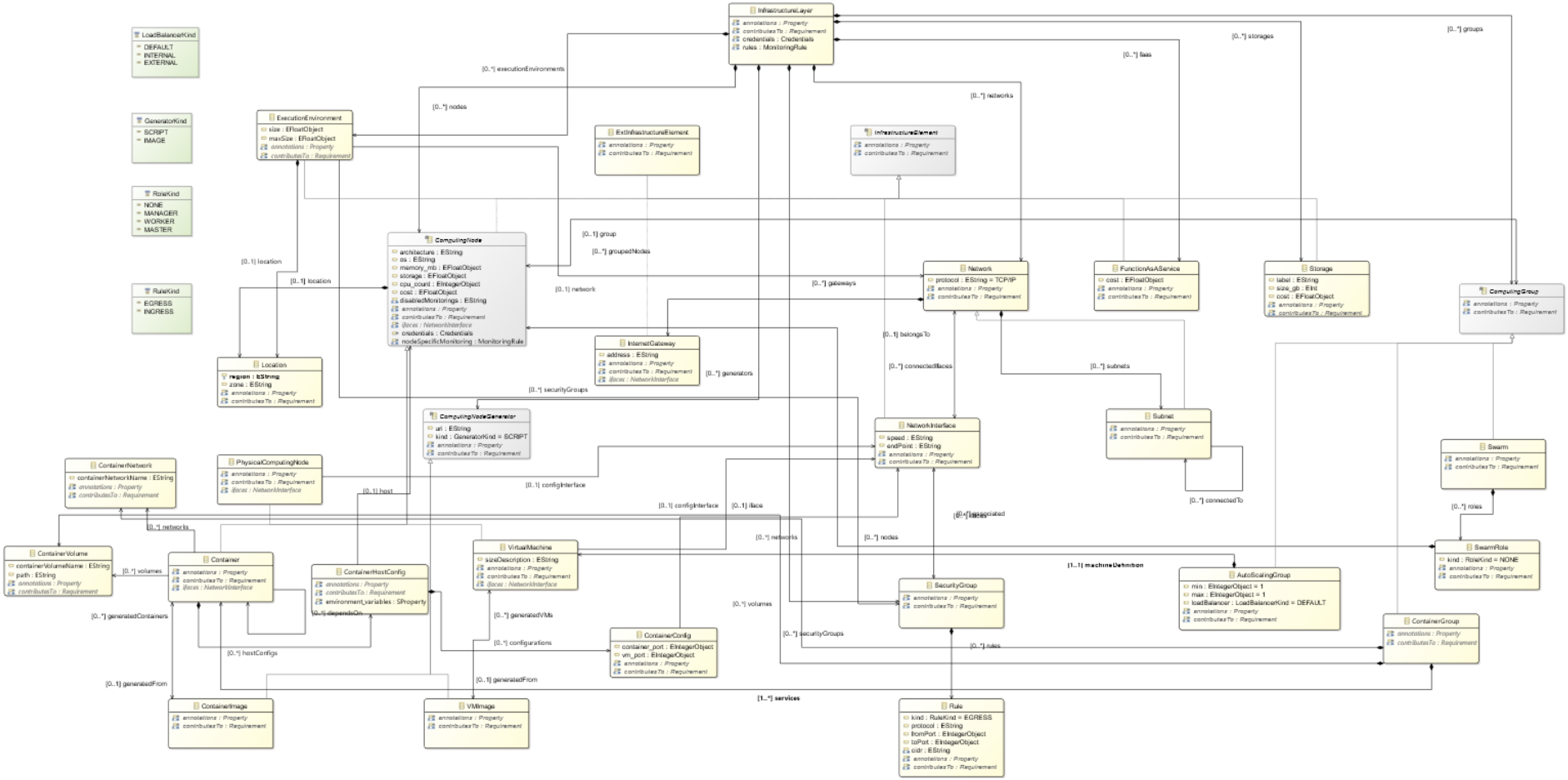

Figure 3. Infrastructure layer diagram

4.1 InfrastructureLayer Class

The InfrastructureLayer class is the container for the catalog of infrastructure elements that will be available to the current DOML model.

Superclass

DOMLElement

Associations

nodes: ComputingNode [0..*]	The list of independent computing nodes (not attached to a provider) available in the catalogue
generators: ComputingNodeGenerator [0..*]	The list of virtual machine and container images available in the catalogue
groups: ComputingGroup [0..*]	The list of computing groups
securityGroups: SecurityGroup [0..*]	The list of security groups
networks: Network [0..*]	The list of independent networks (not attached to a runtime provider) available in the catalogue
storages: Storage [0..*]	The list of storage resources that will be part of this abstract infrastructure model
faas: FunctionAsAService [0..*]	The list of FaaS services part of this DOML model infrastructure.
credentials: Credentials [0..*]	The list of credentials.
executionEnvironments: ExecutionEnvironment [0..*]	The list of execution environments that will be part of this abstract infrastructure model.
rules: MonitoringRule [0..*]	The list of monitoring rules defined for the abstract infrastructure

Usage

The InfrastructureLayer contains abstract infrastructure elements for the deployment of the cloud application.

4.2 InfrastructureElement Class (abstract)

The InfrastructureElement class represents all infrastructure elements that can have an application component deployed to them.

Superclass

DOMLElement, DeployableElement

Usage

The InfrastructureElement is intended to be used as the parent for more concrete elements of the infrastructure model.

4.3 Node (abstract)

The Node class represents both computing and network nodes, having some associated network interfaces.

Superclass

InfrastructureElement

Associations

ifaces: NetworkInterface [0..*]	The network interfaces owned by this computing node.

Usage

The Node class is intended to be used as the parent for both computing and network nodes.

4.4 ComputingNode Class (abstract)

The ComputingNode class represents any element that can be used for computing, from a dedicated host to an IoT node.

Superclass

Node

Attributes

architecture: String [0..1]	A string describing the internal architecture of the computing node (e.g. x86, x64, etc.).
os: String [0..1]	A string describing the operating system of this node (e.g., Windows 10, Ubuntu 20.04, etc.).
memory_mb: Float [0..1]	A float describing the total memory of this node in MB (e.g. 4096 MB).
storage: Float [0..1]	A float describing the total storage available in this node in GB (e.g., 512.0).
cpu_count: Integer [0..1]	An integer describing the number of CPU of the computing node.
cost: Float [0..1]	An optional cost value (in Euro).
disabledMonitorings: String [0..*]	A set of strings defining the monitorings that will not apply for the current computing node. Refer to MonitoringRule class for further information.

Associations

group: ComputingGroup [0..1]#groupedNodes	A link to the computing group that owns this computing node. The bidirectional reference is indicated through “groupedNodes”.

location: Location [0..1]	An optional location for this infrastructure element.
credentials: Credentials [0..1]	The credentials for this computing node.
nodeSpecificMonitoring: MonitoringRule [0..*]	A set of monitoring rules that are to be applied to this computing node.

Usage

The ComputingNode class is intended to be the common parent for all the infrastructure elements capable of executing code.

4.5 PhysicalComputingNode Class

The PhysicalComputingNode class represents a dedicated physical server.

Superclass

ComputingNode

Associations

configInterface: NetworkInterface [0..1]	A reference to the network interface used to configure this computing node.

4.6 ComputingNodeGenerator Class (abstract)

The ComputingNodeGenerator class represents all infrastructure elements that describe a virtual computing node.

Superclass

DOMLModel

Usage

The ComputingNodeGenerator is intended to be used as the commong parent for all elements defining the characteristics of virtual computing nodes. Often these generators rely on a file which defines them.

Usage

The PhysicalComputingNode is used to describe physical computing nodes available for the owner of a cloud application that are going to be used as part of the cloud deployment.

4.7 VirtualMachine Class

The VirtualMachine class represents a virtual computing node running on top of a supervisor software.

Superclass

ComputingNode

Attributes

sizeDescription: String [0..1]	An optional string describing the size of the VM.

Associations

generatedFrom: VMImage [0..1]	The image used to generate this virtual machine.
configInterface: NetworkInterface [0..1]	A reference to the network interface used to configure this computing node.

Usage

The VirtualMachine is used to describe virtual computing nodes running on a supervisor software. In order to be automatically configurable, the virtual machine must define the image that will generate it.

4.8 Location Class

The Location class represents the place where a computing node should be.

Superclass

DOMLModel

Attributes

region: String [1]	A string describing the region for this location.
zone: String [0..1]	An optional attribute to refine the location if the region is not precise enough.

Usage

The Location is intended to describe the location of infrastructure elements, more concretely virtual machines and physical machines.

4.9 ContainerConfig Class

The ContainerConfig class represents a specific configuration for a container on a hosted on a computing node, including the port mapping and the network interface.

Superclass

DOMLElement

Attributes

container_port: Integer [0..1]	A container port to be exposed.
vm_port: Integer [0..1]	A port on a computing node where this container will map to.

Associations

iface: NetworkInterface [0..1]	The network interface connected to this container.

4.10 ContainerHostConfig Class

The ContainerHostConfig class represents a host configuration for a container, including the computing node where it's hosted, the environmental variables and the specific configurations on that host.

Superclass

DOMLElement

Associations

host: ComputingNode [0..1]	The computing node host for this container.
environment_variables: SProperty [0..*]	The list of environmental variables to be set for this host.
configurations: ContainerConfig [0..*]	The list of configurations for this host.

4.11 Container Class

The Container class represents a virtual computing node running on top of another computing node.

Superclass

ComputingNode

Associations

generatedFrom: ContainerImage [0..1]	The image used to generate this container.
hostConfigs: ContainerHostConfig [0..*]	The list of host configurations for this container.
networks: ContainerNetwork [0..*]	The list of container networks this container connects to.
volumes: ContainerVolume [0..*]	The list of container volumes this container has access to.
dependsOn: Container [0..*]	The list of containers on which this container depends.

Constraints

Networks, volumes and container dependencies can be specified only within a container group.

Usage

The Container is used to describe virtual computing nodes, such as Docker containers.

4.12 ContainerGroup Class

The ContainerGroup class represents a group of containers sharing some resources (e.g., networks, volumes) and possibly having dependency relationships between them.

Superclass

ComputingGroup

Associations

services: Container [1..*]		The container services of this container group.
networks: [0..*]	ContainerNetwork	The container networks to which services of this container group can connect.
volumes: [0..*]	ContainerVolume	The container volumes that can be accessed by services of this container group.

Usage

The ContainerGroup is used to describe groups of virtual computing nodes (e.g., Docker Compose).

4.13 ContainerNetwork Class

The ContainerNetwork class represents a virtual container network.

Superclass

DOMLElement

Attributes

containerNetworkName: [0..1]	String	The name of this container network.

Usage

The ContainerNetwork is used to describe virtual networks among containers.

4.14 ContainerVolume Class

The ContainerVolume class represents a volume for persisting data, to be mounted on containers.

Superclass

DOMLElement

Attributes

containerVolumeName: [0..1]	String	The name of this container volume.

path: String [0..1]	The container path on which this volume is to be mounted.

Usage

The ContainerVolume is used to describe a mechanism for persisting data generated by and used by containers.

4.15 ExecutionEnvironment Class

The ExecutionEnvironment class represents properly configurable environments to host specific software services (e.g., DBaaS).

Superclass

InfrastructureElement

Attributes

size: Float [0..1]	The available storage.
maxSize: Float [0..1]	The maximum available storage for environments with scalable capability.

Associations

location: Location [0..1]	The location for this infrastructure element.
network: Network [0..1]	The network element the hosted service needs to be connected to.
securityGroups: SecurityGroup [0..*]	The list of security groups associated with this element.

Usage

The ExecutionEnvironment is used to describe proper environment configurations for executing specific software services.

4.16 GeneratorKind Enum

The GeneratorKind enumeration describes the different computing node generation kinds.

Values

SCRIPT, IMAGE

4.17 ComputingNodeGenerator Class (abstract)

The ComputingNodeGenerator class represents all infrastructure elements that describe a virtual computing node.

Superclass

DOMLModel

Attributes

uri: String [0..1]	An URI to the file containing this computing node generation image or file.
kind: GeneratorKind [0..1]	An optional attribute to define whether this generator uses a node image (i.e. a VM image) or a file (i.e. docker file) to generate the computing node.

Usage

The ComputingNodeGenerator is intended to be used as the commong parent for all elements defining the characteristics of virtual computing nodes. Often these generators rely on a file which defines them.

4.18 VMImage Class

The VMImage class represents the image (i.e. the set of attributes and parameters) that can be used to generate a virtual machine.

Superclass

ComputingNodeGenerator

Associations

generatedVMs: VirtualMachine [0..*]	The set of virtual machines that will be created using this image.

Usage

The VMImage is used for generation purposes, allowing the ICG to generate the scripts to generate VMs from a VM defining image.

4.19 ContainerImage Class

The ContainerImage class represents the image (i.e. the set of attributes and parameters) that can be used to generatae a container.

Superclass

ComputingNodeGenerator

Attributes

generatedContainers: Container [0..*]	The set of containers that have been generated using this conatiner image.

Usage

The ContainerImage is used for generation purposes, allowing the ICG to generate the scripts to generate containers from the container defining image.

4.20 ComputingGroup Class (abstract)

The ComputingGroup class represents a group of computing nodes.

Superclass

DOMLElement

Associations

groupedNodes: ComputingNode [0..*]#group	A group of computing nodes, which has a bidirectional reference with the ComputingNode through the “group” field.

Usage

The ComputingGroup class allows to configure a set of nodes to act as a group.

4.21 AutoScalingGroup Class

The AutoScalingGroup class represents a group of computing nodes with the auto scaling property.

Superclass

ComputingGroup

Attributes

min: Integer	The minimum number of computing nodes
max: Integer	The maximum number of computing nodes
loadBalancer: LoadBalancerKind	The load balance used in the auto scaling group

Associations

machineDefinition: VirtualMachine [1]	The VM template

Usage

The AutoScalingGroup class allows to configure a set of nodes to act as a group supporting auto scaling functionality.

4.22 LoadBalancerKind Enum

The LoadBalancerKind enumeration describes the different load balancer kinds.

Values

DEFAULT, INTERNAL, EXTERNAL

4.23 Storage Class

The Storage class represents an infrastructure node that aims at incrementing the overall storage available to the computing nodes in the infrastructure.

Superclass

InfrastructureElement

Attributes

label: String [0..1]	The label of the storage
size_gb: Integer [0..1]	The size of the strorage in GB
cost: Float [0..1]	The cost of this storage service in Euro

Usage

The Storage class allows to define a node that increments the storage of the application. The node cannot support any other functionality other than providing storage space.

4.24 FunctionAsAService Class

The FunctionAsAService class represents a pure software infrastructure component capable of executing functional algorithms through an API.

Superclass

InfrastructureElement

Attributes

cost: Float [0..1]	The cost of this service in Euro

Usage

The FunctionAsAService class allows to define a service used to execute pure business logic/algorithms on a set of input data.

4.25 ExtInfrastructureElement Class (abstract)

The ExtInfrastructureElement class is just used to represent an instance of a new infrastructure element concept that the user wants to add to DOML. This class is part of the DOML-E extension mechanisms.

Superclass

InfrastructureElement, ExtensionElement

Usage

The ExtInfrastructureElement class should be used to create instances of concepts and metaclasses not currently available in DOML.

4.26 Network Class

The Network class represents the means to interconnect computing nodes. The concepts related to the network, as well as associations among them.

Superclass

DOMLElement

Attributes

protocol: String [0..1]	A string defining the protocol of this network (default is "TCP/IP").

Associations

connectedIfaces: NetworkInterface [0..*]	The set of network interfaces connected to this network. This is a derived association.
subnets: Subnet [0..*]	The set of sub networks of the current one.
gateways: InternetGateway [0..*]	The Internet gateways.

Usage

The Network describes the means to interconnect computing nodes as part of a cloud architecture.

4.27 Subnet Class

The Subnet class models a partition of a main network. A subnet is also a network.

Superclass

Network

Associations

connectedTo: Subnet [0..*]	The list of subnets connected.

Usage

The Subnet is used to describe partitions of main networks.

4.28 NetworkInterface Class

The NetworkInterface class represents the means to interconnect computing nodes.

Superclass

InfrastructureElement

Attributes

endPoint: String [0..1]	A string defining the endpoint (i.e. address) of this network interface inside the network.
speed: String [0..1]	A string defining the maximum speed of this network interface.

Associations

belongsTo: Network [1]	A reference to the network associated to this interface.
associated: SecurityGroup [0..*]	The associated security groups.

Usage

The Network describes the means to interconnect computing nodes as part of the cloud architecture.

4.29 InternetGateway Class

The InternetGateway class represents the gateway for the access of Internet.

Superclass

Node

Attributes

address: String [0..1]	A string defining the IP address for the gateway.

4.30 SecurityGroup Class

The SecurityGroup class represents the group of the security rules for the network.

Superclass

DOMLElement

Associations

ifaces: NetworkInterface [0..*]#associated	The list of the network interfaces, which has a bidirectional reference with the NetworkInterface through the field “associated”.
rules: Rule [0..*]	The security rules for the network.

Usage

The SecurityGroup describes a resource used to secure the access to a specific network.

4.31 Rule Class

The Rule class defines the security rule for the security group.

Superclass

DOMLElement

Attributes

kind: RuleKind	The kind of security rule, e.g., ingress or egress.
protocol: String	The protocol for this rule, e.g., https.
fromPort: Integer	The start port.
toPort: Integer	The end port.
cidr: String [0..*]	The CIDR block of IP.

Usage

The Rule class describes the security rule as part of the security group.

4.32 RuleKind Enum

The RuleKind enumeration describes the different security rule kinds.

Values

INGRESS, EGRESS

4.33 Swarm Class (not implemented)

The Swarm class represents the swarm of computing nodes, e.g., dockerswarm.

Note: This class is present for future development purposes, but swarms are currently not interpreted by the ICG.

Superclass

ComputingGroup

Attributes

roles: SwarmRole [0..*]	The list of roles in the swarm.

Usage

The Swarm class is used to describe the computing swarm, e.g., dockerswarm, which defines a group of docker services of a cloud application.

4.34 SwarmRole Class (not implemented)

The Swarm class represents the swarm of computing nodes, e.g., dockerswarm.

Note: This class is present for future development purposes, but swarms are currently not interpreted by the ICG.

Superclass

DOMLElement

Attributes

kind: RoleKind	The kind of a role for the computing node in the swarm.

4.35 RoleKind Enum (not implemented)

The RoleKind enumeration describes the different swarm role kinds.

Note: This class is present for future development purposes, but swarms are currently not interpreted by the ICG.

Values

MANAGER, WORKER, MASTER

4.36 MonitoringRule Class

The MonitoringRule class represents a set of configuration parameters valid for the PIACERE monitoring module.

Superclass

DOMLElement

Attributes

condition: String [1]	Formal string, dependent of the strategy attribute, that defines the condition that will trigger the monitoring.
strategy: String [1]	The name of the monitoring strategy this rule will use.
strategyConfigurationString: String [0..1]	Parameters valid for the given monitoring strategy.

Usage

Monitoring rules are to be defined at infrastructure layer, to configure the monitoring services in PIACERE.

5 Concrete Layer

The following diagram (see Fig. 4) shows the main elements of the Concrete Layer in DOML:

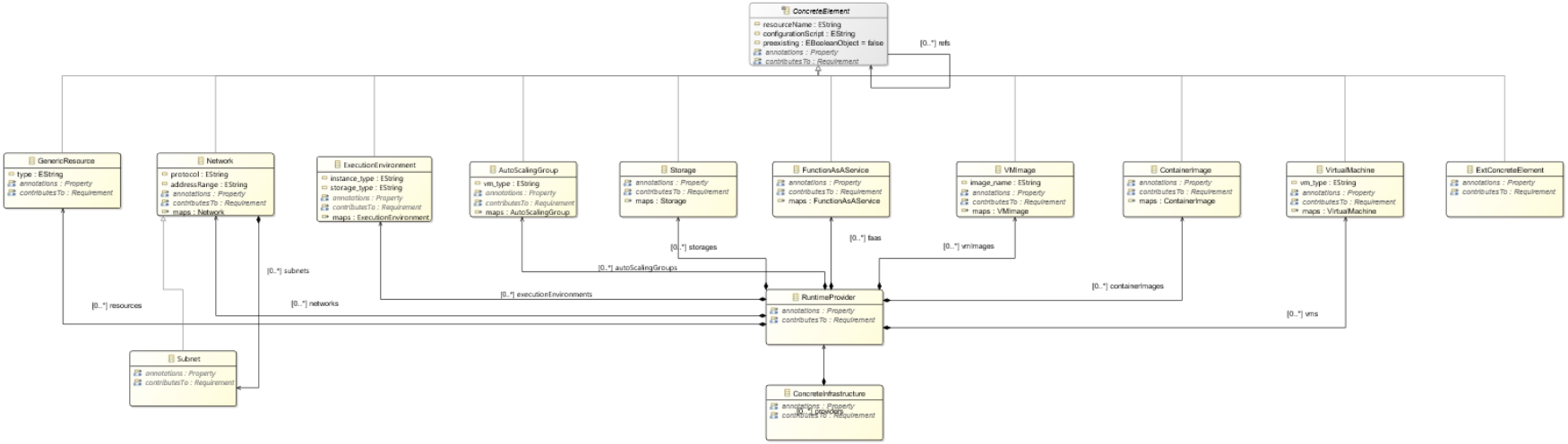

Figure 4. Concrete Infrastructure Layer diagram

5.1 ConcreteInfrastructure Class

The ConcreteInfrastructure class is the container for the catalog of concrete infrastructure elements that will be available to the current DOML configuration. Several concrete infrastructure instances may exist at the same time, each of them being part of a particular DOML solution.

Superclass

DOMLElement

Associations

providers: [0..*]	RuntimeProvider	The list of runtime providers available in the catalogue

Usage

The ConcreteInfrastructure is a container element used to deploy the final cloud application using DOML for a particular solution.

5.2 ConcreteElement Class (abstract)

The ConcreteElement class represents all concrete infrastructure elements that can have an abstract infrastructure element component mapped onto them.

Superclass

DOMLElement

Attributes

resourceName: String [0..1]	An optional name for this resource.
configurationScript: String [0..1]	An optional URI to the script that has to be executed to correctly configure a node.
preexisting: Boolean [0..1]	Mark if it's a preexisting resource.

Associations

refs: ConcreteElement [0..*]	References to other concrete elements.

Usage

The ConcreteElement is intended to be used as the parent for more concrete elements of the concrete infrastructure model.

5.3 GenericResource Class

The GenericResource class is used to support some concrete generic resources that are specialized in different IaC languages like Terraform, etc.

Superclass

ConcreteElement

Attributes

type: String [0..1]	Type of this generic resource.

5.4 RuntimeProvider Class

The RuntimeProvider class describes a cloud resources provider (e.g. AWS).

Superclass

DOMLElement

Associations

autoScalingGroups: AutoScalingGroup [0..*]	The list of autoscaling groups.
resources: GenericResource [0..*]	The list of concrete generic resources.
vms: VirtualMachine [0..*]	The virtual machines that will be provided by the runtime provider.
executionEnvironments: ExecutionEnvironment [0..*]	The list of execution environments to host specific software services.
vmImages: VMImage [0..*]	The list of images for VM.
containerImages: ContainerImage [*]	The list of images for container.
networks: Network [0..*]	The networks requested to the runtime provider.
storages: Strorage [0..*]	The storages offered by this particular provided.
faas:FunctionAsAService [0..*]	The FaaS services offered by this runtime provider

Usage

The RuntimeProvider is intended to model all the parameters related to a specific cloud IaaS provider.

5.5 VirtualMachine Class

The VirtualMachine class in the concrete layer represents a specific VM instance either provided by a runtime provider or configured by the user on their own infrastructure.

Superclass

ConcreteElement

Attributes

vm_type: String [0..1]	The desired machine type.

Associations

maps: infrastructure.VirtualMachine	The VM on the abstract infrastructure layer this concrete VM maps on.

Usage

The VirtualMachine is intended to be used as the concrete counterpart of the abstract VM defined in the infrastructure layer.

5.6 VMImage Class

The VMImage class represents the image for a specific VM.

Superclass

ConcreteElement

Attributes

image_name: String [0..1]	The image name, mostly used for available image names in the provider catalogue (e.g., “ami-000e50175c5f86214" on AWS provider).

Associations

maps: infrastructure. VMImage	The map to the VMImage defined in the abstract infrastructure layer for this concrete VMImage.

Usage

The VMImage is intended to be used as the concrete counterpart of the abstract VMImage defined in the infrastructure layer.

5.7 ContainerImage Class

The ContainerImage class represents the image for a specific Container.

Superclass

ConcreteElement

Associations

maps: infrastructure. ContainerImage	The map to the ContainerImage defined in the abstract infrastructure layer for this concrete ContainerImage.

Usage

The ContainerImage is intended to be used as the concrete counterpart of the abstract ContainerImage defined in the infrastructure layer.

5.8 ExecutionEnvironment Class

The ExecutionEnvironment class in the concrete layer represents a specific environment instance to host a software service and its configuration.

Superclass

ConcreteElement

Attributes

instance_type: String [0..1]	The desired machine type for the VM hosting the service.
storage_type: String [0..1]	The desired storage type (e.g., General Purpose SSD types).

Associations

maps: infrastructure.ExecutionEnvironment	The ExecutionEnvironment on the abstract infrastructure layer this concrete element maps on.

Usage

The ExecutionEnvironment is intended to be used as the concrete counterpart of the abstract ExecutionEnvironment defined in the infrastructure layer.

5.9 Network Class

The Network class in the concrete layer represents a specific network instance either provided by a runtime provider or configured by the user on their own infrastructure.

Superclass

ConcreteElement

Attributes

protocol: String [0..1]	A string defining the protocol of this network.
addressRange: String [0..1]	A string describing the valid addresses in this particular network.

Associations

subnets: concrete.Subnet [0..*]	The subnets defined for this concrete network.
maps: infrastructure.Network	The network on the abstract infrastructure layer this concrete element maps on.

Usage

The Network is intended to be used as the concrete counterpart of the abstract Network defined in the infrastructure layer.

5.10 Subnet Class

The Subnet class in the concrete layer represents a specific subnet instance defined within a concrete network.

Superclass

Network

Usage

The Subnet is intended to be used as the concrete counterpart of the abstract Subnet defined in the infrastructure layer.

5.11 Storage Class

The Storage class in the concrete layer represents a specific storage service either provided by a runtime provider or configured by the user on their own infrastructure.

Superclass

ConcreteElement

Associations

maps: infrastructure.Storage	The storage service on the abstract infrastructure layer this concrete storage maps on.

Usage

The Storage is intended to be used as the concrete counterpart of the abstract Storage defined in the infrastructure layer.

5.12 FunctionAsAService Class

The FunctionAsAService class in the concrete layer represents a specific functional logic service instance either provided by a runtime provider or configured by the user on their own infrastructure.

Superclass

ConcreteElement

Associations

maps: infrastructure.FunctionAsAService	The FaaS instance on the abstract infrastructure layer this concrete element maps on.

Usage

The FunctionAsAService is intended to be used as the concrete counterpart of the abstract FunctionAsAService defined in the infrastructure layer.

5.13 AutoScalingGroup Class

The AutoScalingGroup class in the concrete layer represents a specific autoscaling group instance either provided by a runtime provider or configured by the user on their own infrastructure.

Superclass

ConcreteElement

Attributes

vm_type: String [0..1]	The desired machine type.

Associations

maps: infrastructure.AutoScalingGroup	The autoscaling group on the abstract infrastructure layer this concrete group maps on.

Usage

The AutoScalingGroup is intended to be used as the concrete counterpart of the abstract AutoScalingGroup defined in the infrastructure layer.

5.14 ExtConcreteElement Class

The ExtConcreteElement class is just used to represent an instance of a new infrastructure element concept that the user wants to add to DOML. This class is part of the DOML-E extension mechanisms.

Superclass

ConcreteElement, ExtensionElement

Usage

The ExtConcreteElement class should be used to create instances of concepts and metaclasses not currently available in DOML.

6 Optimization Layer

The following diagram (see Fig. 5) shows the main elements of the Optimization Layer in DOML:

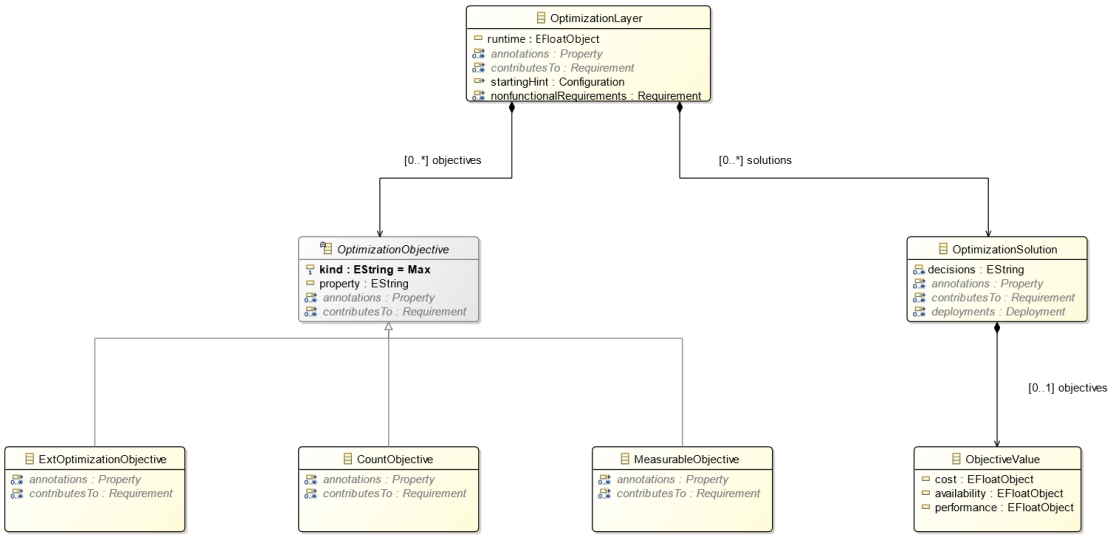

Figure 5. Optimization Layer diagram

6.1 OptimizationLayer Class

The OptimizationLayer class is the main container for all the elements related to the definition and usage of the optimization algorithms in DOML.

Superclass

DOMLModel

Attributes

runtime: Float [0..1]	The optimizer runtime in seconds, returned after performing the model optimization.

Associations

objectives: OptimizationObjective [0..*]	The set of objectives for the optimization algorithms.
solutions: OptimizationSolution [0..*]	All the solutions generated by the optimization algorithm.
nonfunctionalRequirements: commons.Requirement [0..*]	The list of nonfunctional requirements for this optimization.
startingHint: Configuration [0..1]	An optional configuration instance that will be used as a hint by the optimization algorithm.

Constraints

* At least one optimization objective should be provided to use the model for optimization purposes.

Usage

The OptimizationLayer is intended to be used as a container for objectives and solutions associated to the optimization algorithms for a DOML model.

6.2 OptimizationObjective Class (abstract)

The OptimizationObjective class represents a formal objective for an optimization algorithm. This objective will afterwards be used by the algorithms as an input to obtain a solution for the application deployment into the cloud infrastructure.

Superclass

DOMLElement

Attributes

property: String [1]	The property associated to this optimization objective.
kind: String [1]	The kind of objective, which can be either “max” or “min”.

Constraints

* The kind attribute may only have the “min” or “max” values.

Usage

The OptimizationObjetive is made abstract to serve as the basis for more concrete optimization objectives, such as objectives that measure the property, or objectives that are related to counting the number of different values of a property.

6.3 CountObjective Class

The CountObjective class represents an optimization objective that will count the different number of values associated to the property specified on them.

Superclass

OptimizationObjetive

Usage

The CountObjective is used to define optimization objectives which want to maximize or minimize the total number of values a property may take (e.g. minimize the number of locations for all the servers in a DOML solution).

6.4 MeasurableObjective Class

The MeasurableObjective class represents an optimization objective associated to the measurement of a particular property.

Superclass

OptimizationObjetive

Usage

The MeasurableObjective is used to define an optimization objective related directly to the value of a particular property (e.g. minimize the cost or maximize the throughput of a DOML solution).

6.5 OptimizationSolution Class

The OptimizationSolution class represents a Configuration of the current DOML model obtained through the usage of optimization algorithms.

Superclass

Configuration

Attributes

objectives: ObjectiveValue [1]	The list of objective values obtained from the optimization.
decisions: String [1..*]	The list of decision values obtained from the optimization.

Usage

The OptimizationSolution is a subclass of the main Configuration class in the commons package, as it is foreseen that any information related to the results obtained by the optimization algorithms (e.g. parameters, used requirements, etc.) could be added as additional information to this kind of Configuration instances.

6.6 ObjectiveValue Class

The ObjectiveValue class represents the data structure for storing the objective function value. Note that it could be defined as a list of float values representing any different optimization targets, while it is currently concretized to the following three widely used targets, i.e., cost, availability, performance, which is mainly tailored for PIACERE.

Attributes

cost: Float [0..1]	The cost target defined for the optimization.
availability: Float [0..1]	The availability target defined for the optimization.
performance: Float [0..1]	The performance target defined for the optimization.

6.7 ExtOptimizationObjective Class (abstract)

The ExtOptimizationObjective class is just used to represent an instance of a new optimization objective concept that the user wants to add to DOML. This class is part of the DOML-E extension mechanisms.

Superclass

OptimizationObjective, ExtensionElement

Usage

The ExtIOptimizationObjective class should be used to create instances of concepts and metaclasses not currently available in DOML.

7 DOML Text Syntax

The following figures 6-14 define the current DOML syntax. This will evolve in the future releases based on the feedback by the end users and the other PIACERE technical partners.

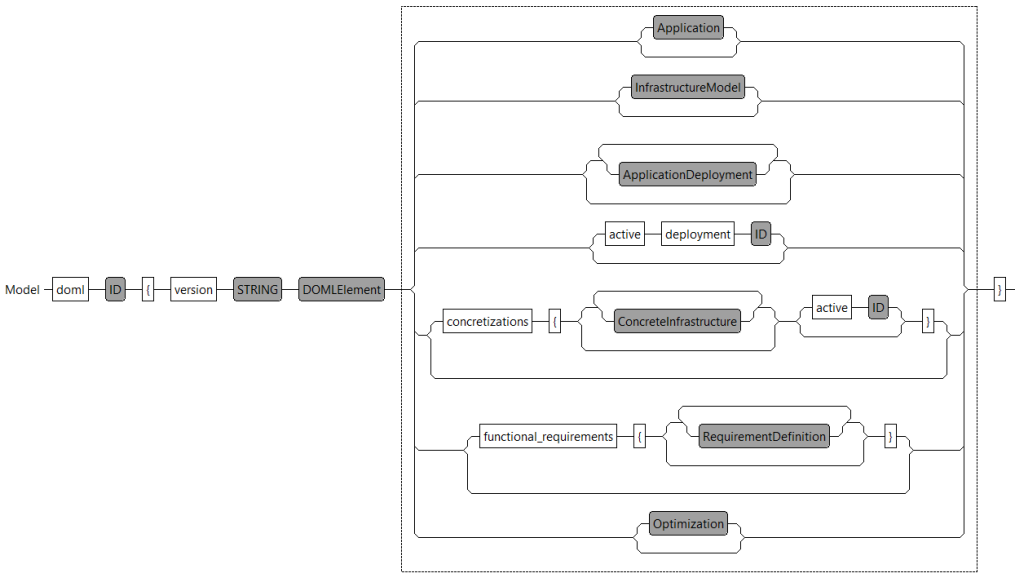

Figure 6. DOML top level model.

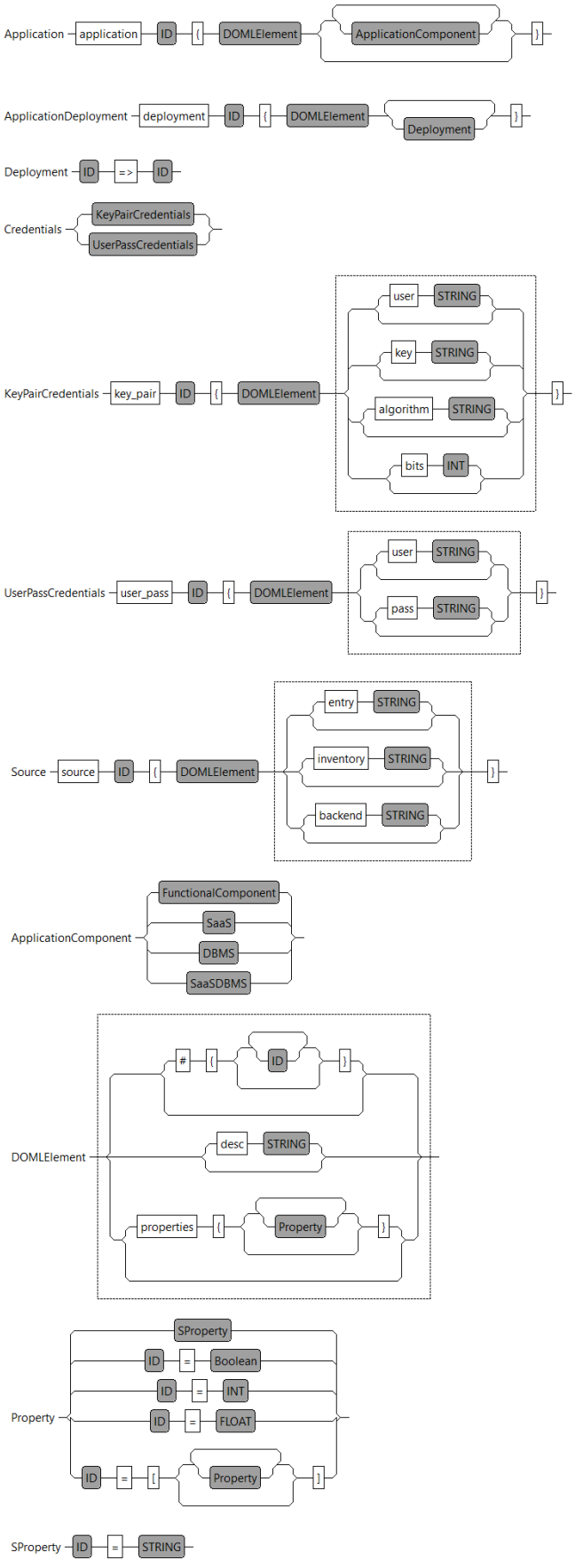

Figure 7. Application layer model (part 1/2).

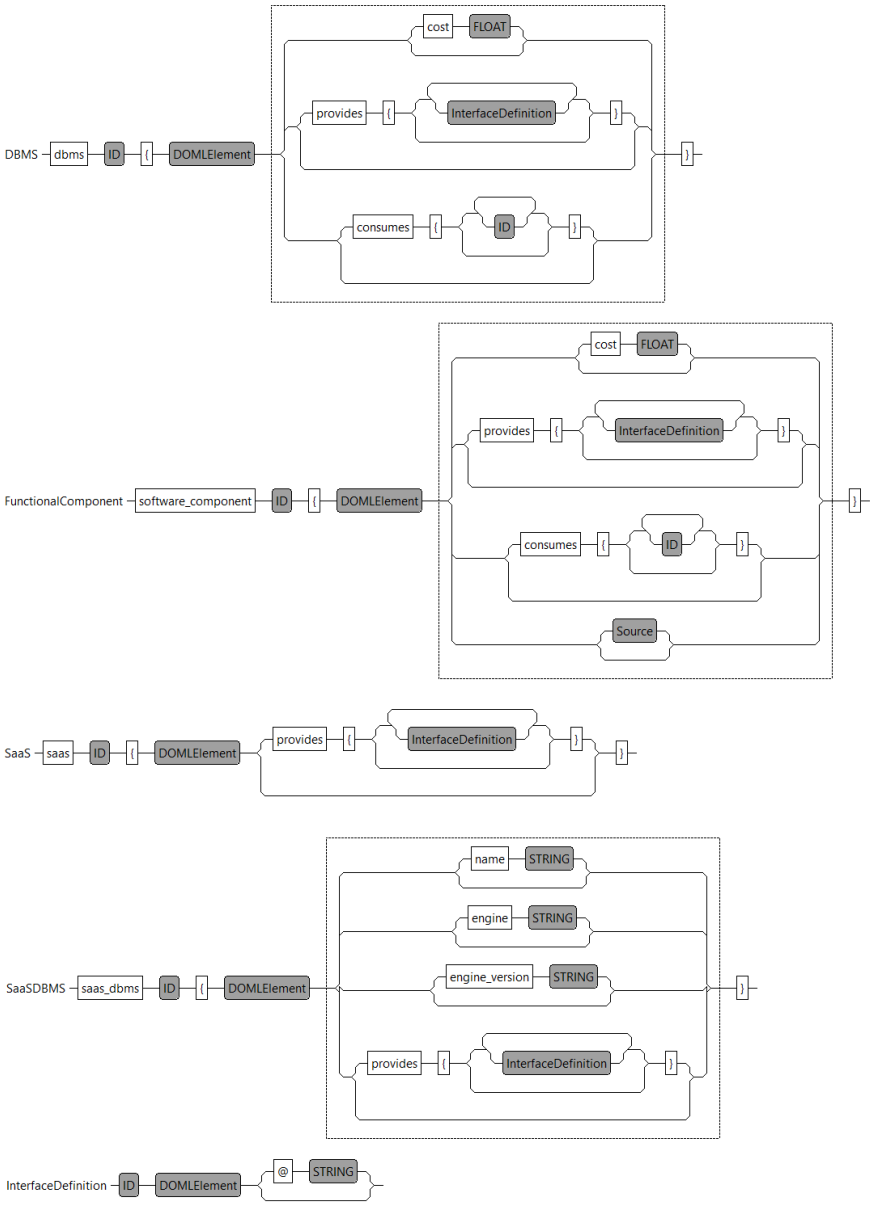

Figure 8. Application layer model (part 2/2).

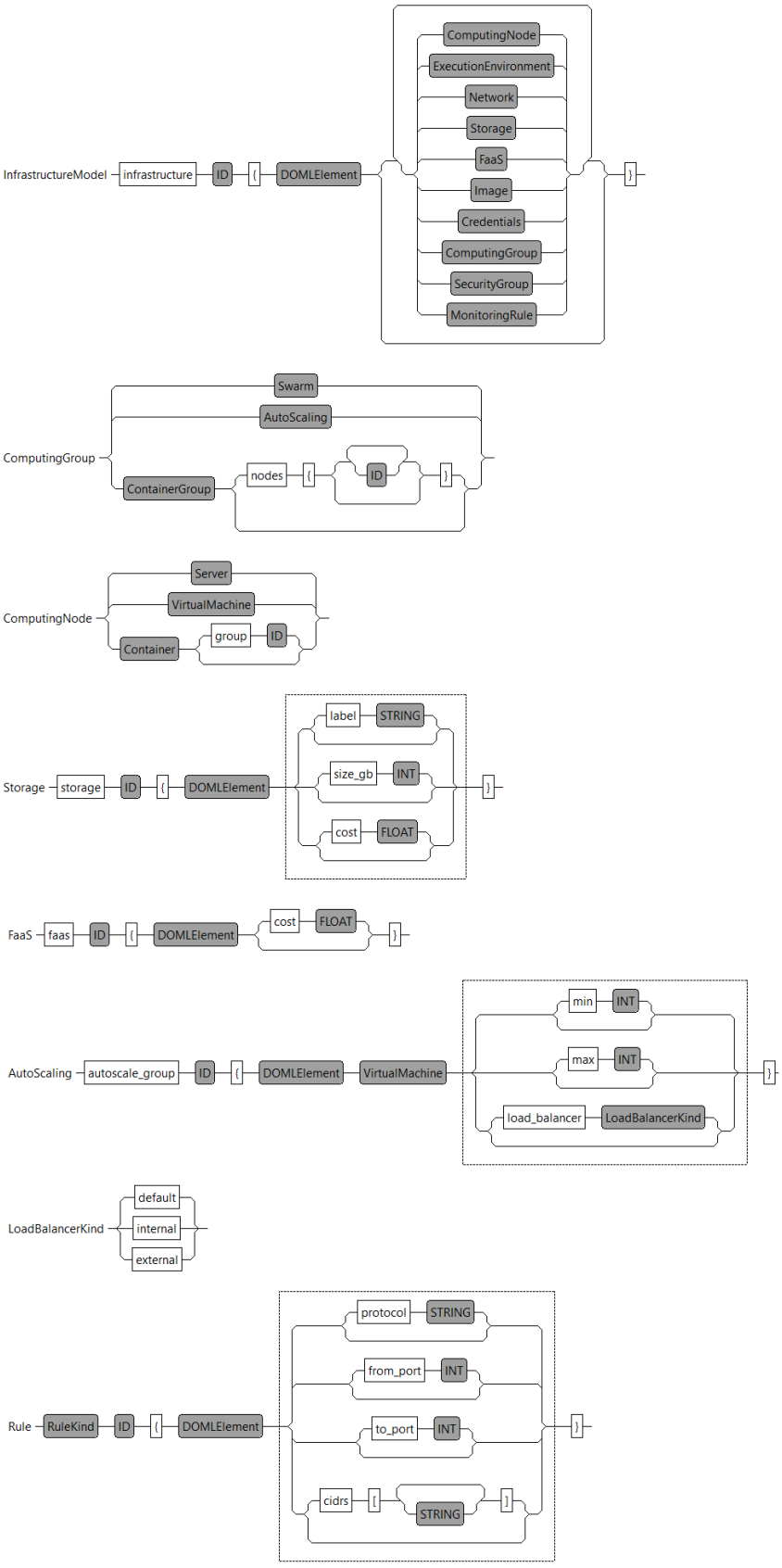

Figure 9. Abstract Infrastructure layer model (part 1/5).

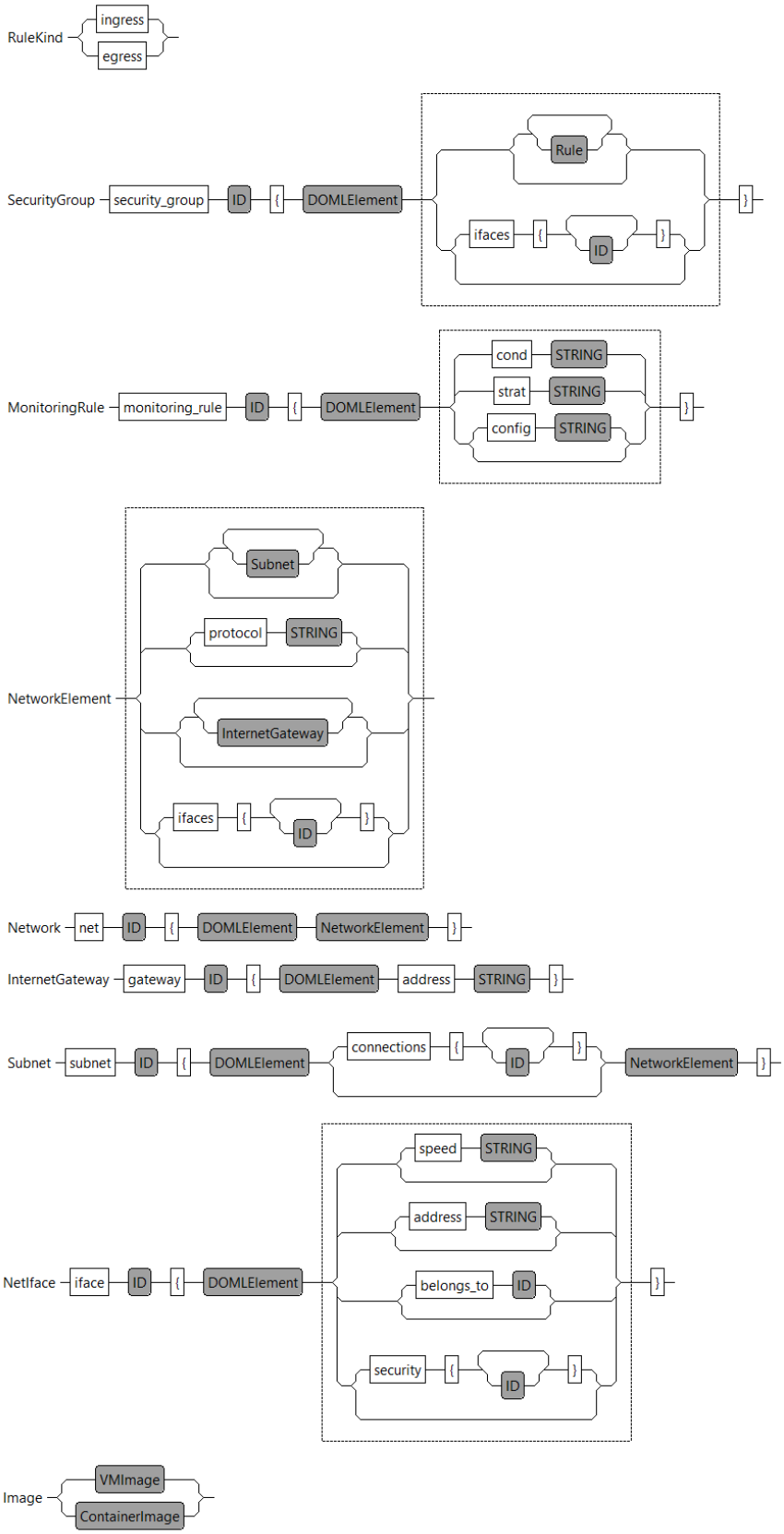

Figure 10. Abstract Infrastructure layer model (part 2/5).

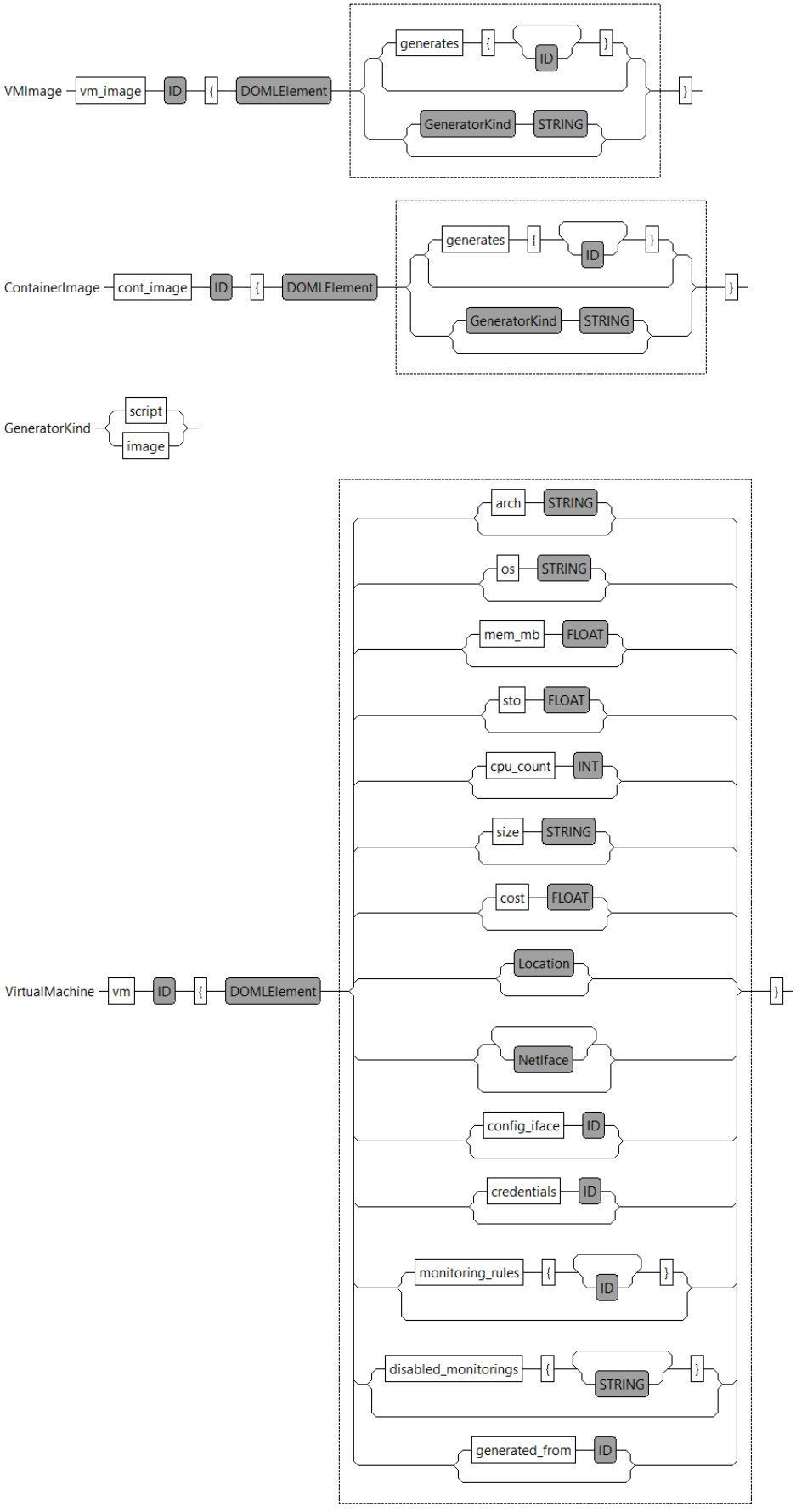

Figure 11. Abstract Infrastructure layer model (part 3/5).

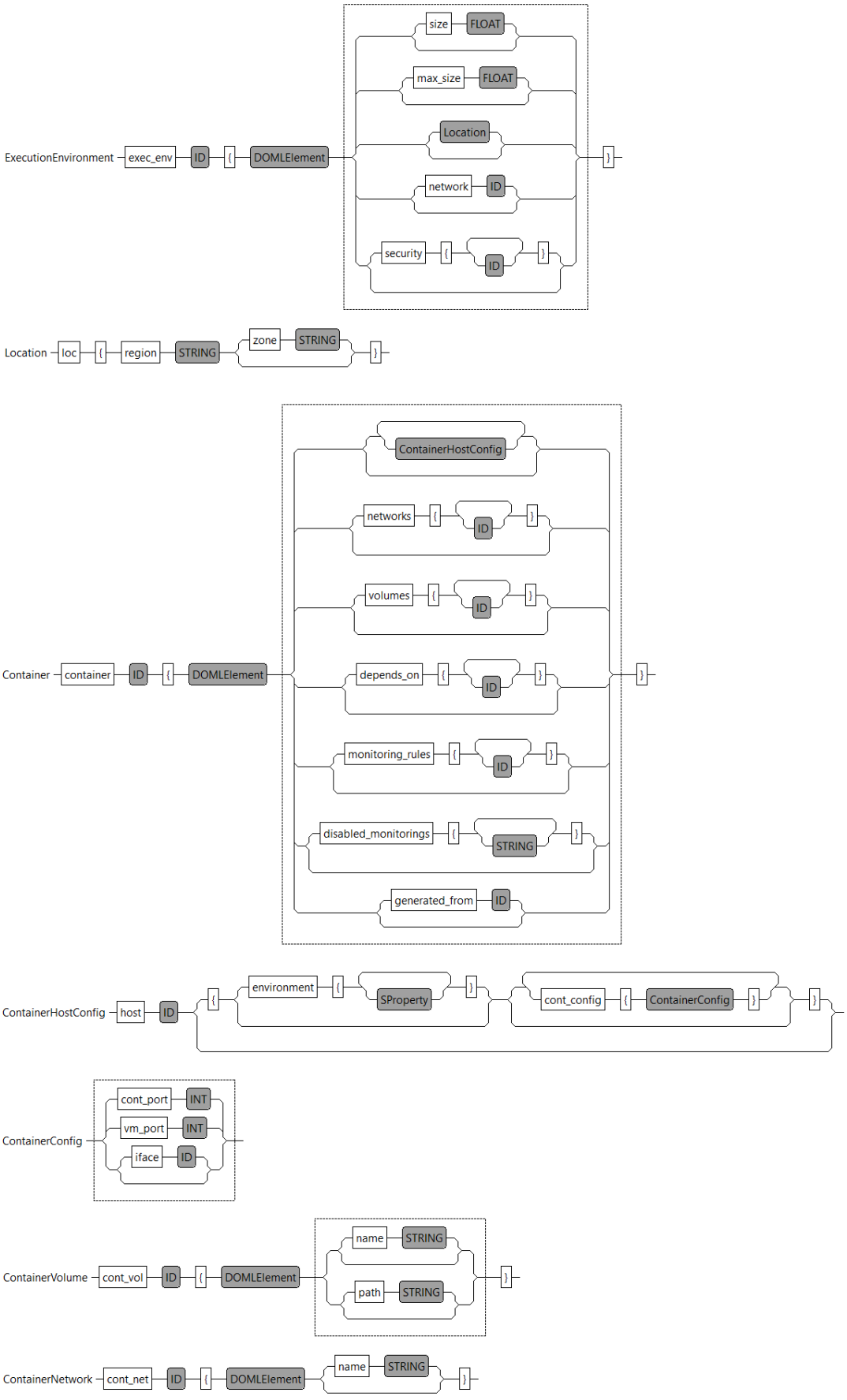

Figure 12. Abstract Infrastructure layer model (part 4/5).

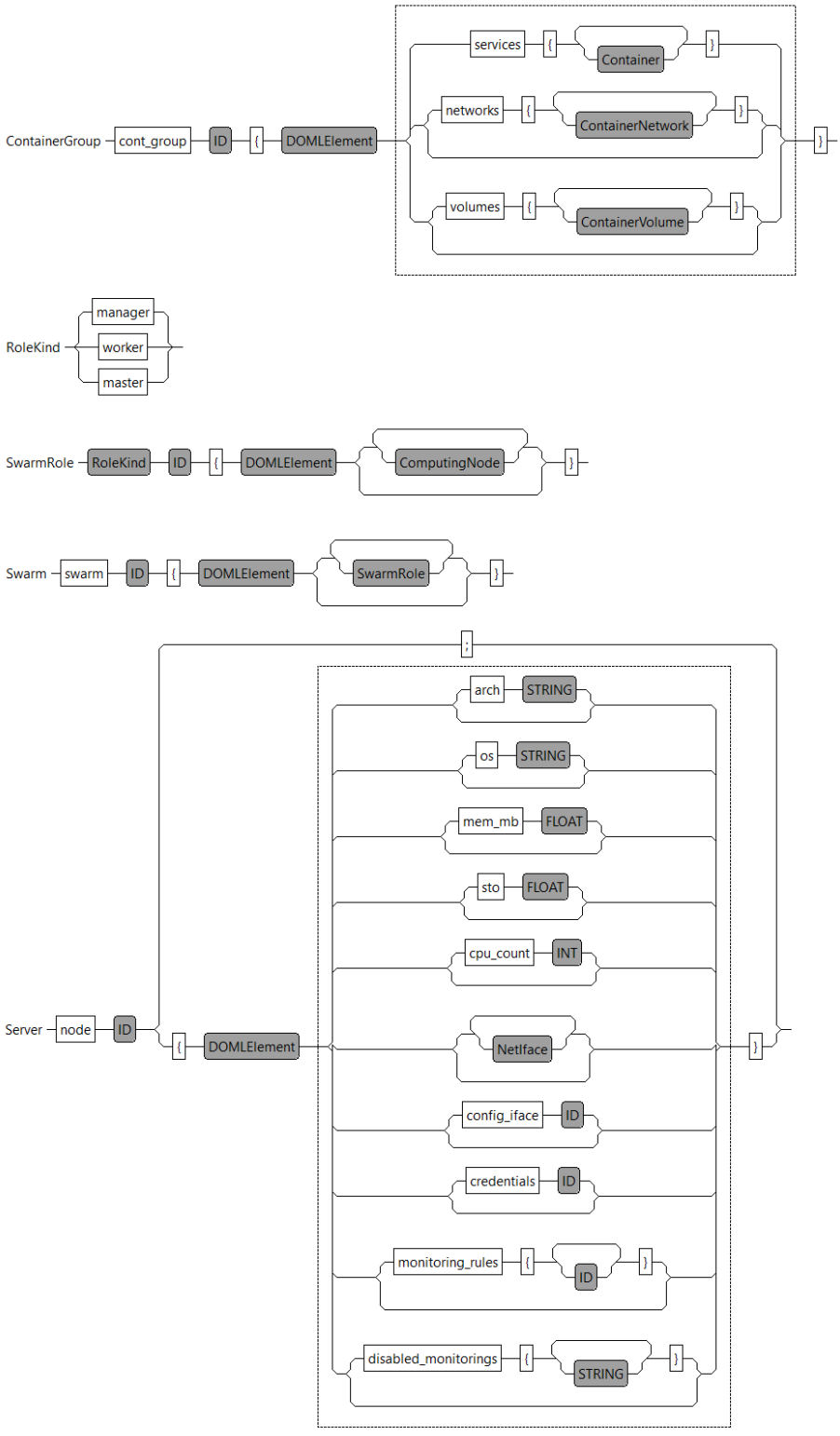

Figure 13. Abstract Infrastructure layer model (part 5/5)

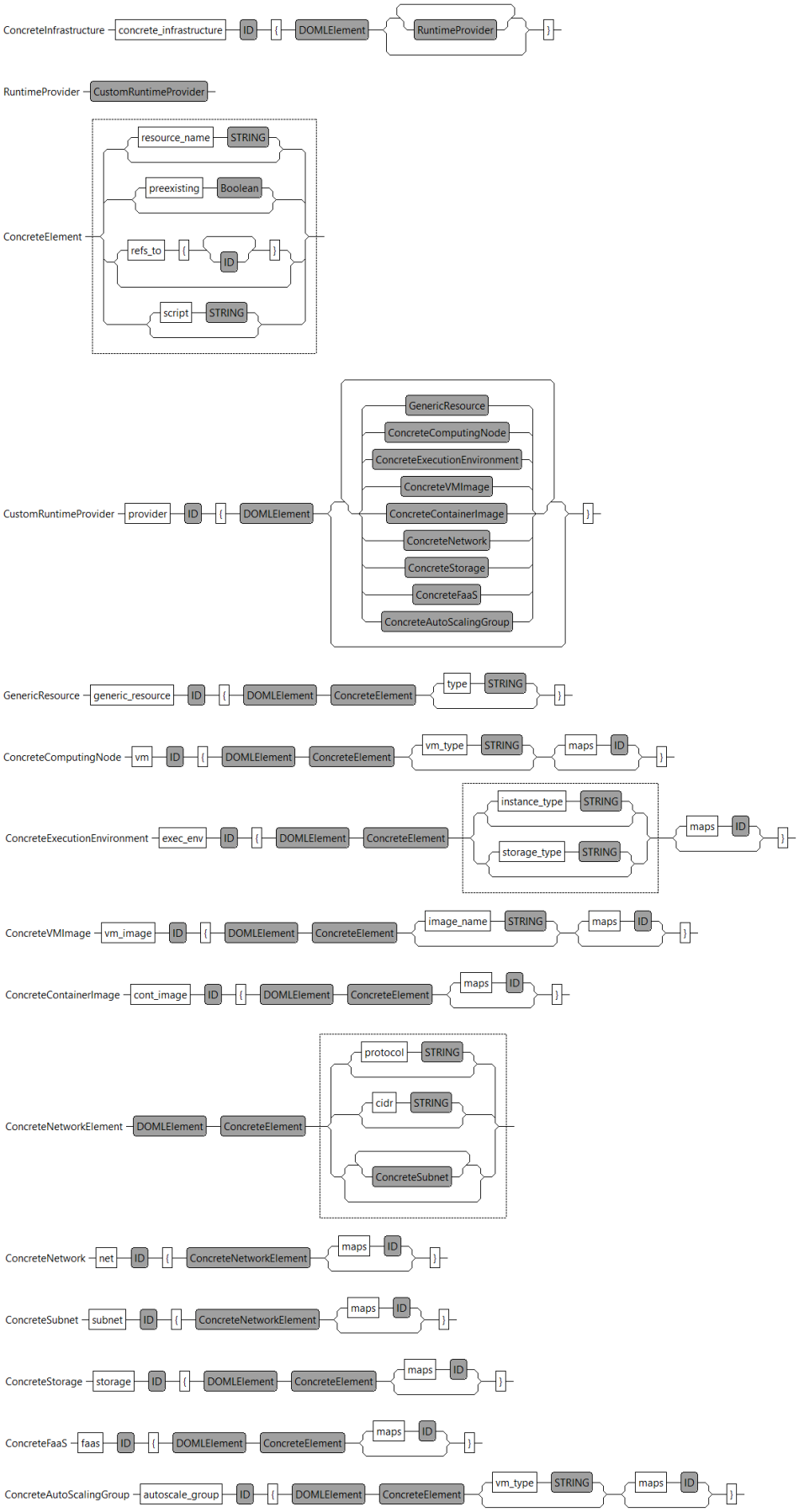

Figure 14. Concrete Infrastructure layer model.

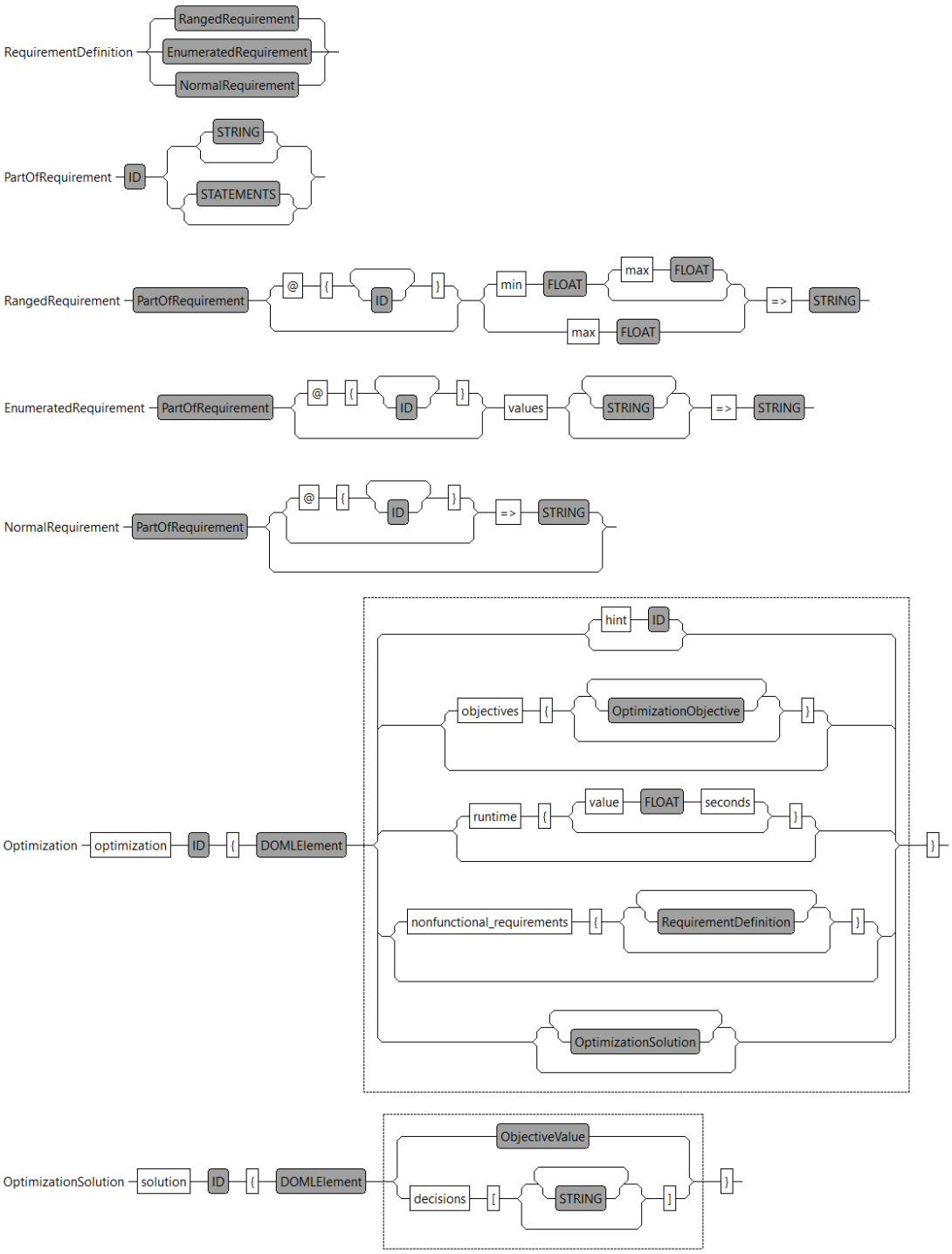

Figure 15. Optimization layer model (with Requirement definitions) (part 1/2).

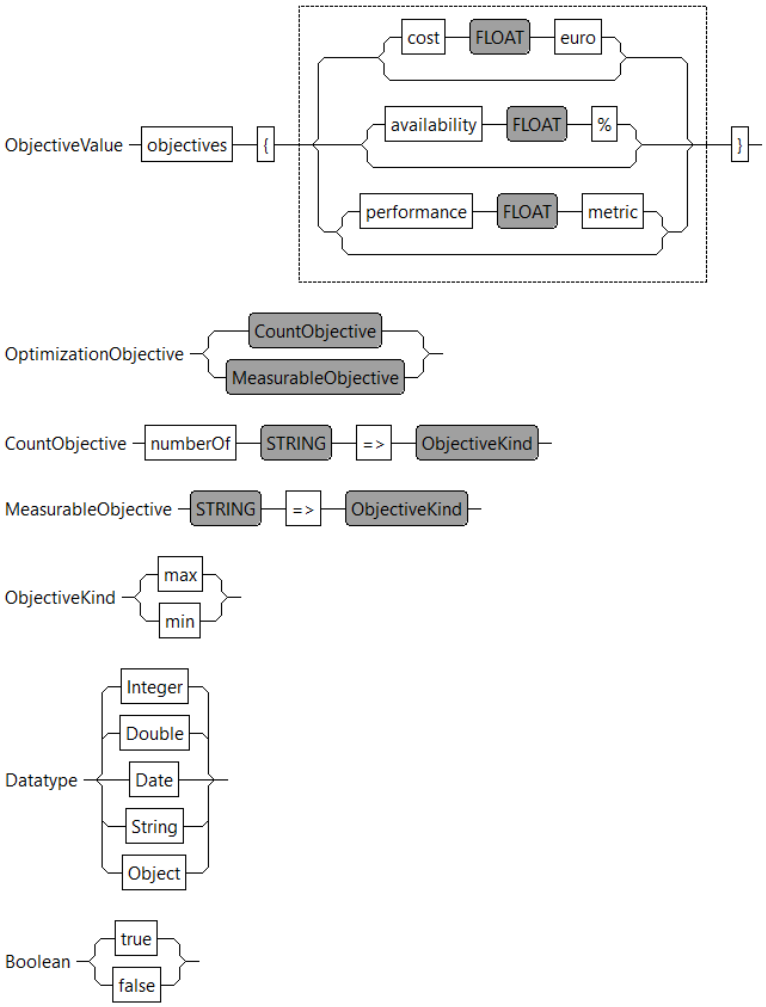

Figure 16. Optimization layer model (with Requirement definitions) (part 2/2).

8 DOML Examples

In this section we provide two simple DOML examples, showing their definition in the textual syntax and the corresponding translation in an XMI notation. Also, other examples are available in Deliverable D3.1.

8.1 Simple Web Application

This example describes a very simple DOML model with a web application that accesses an external API and a database and 2 IoT nodes that provide information to the application. The main software components will be running on virtual machines provided by a runtime provider, while the iotUnits will be running on physical nodes. The configuration has been made manually by the user.

A possible textual representation of this DOML model would be as follows:

```
doml iot_simple_app {
      version "3.1"

      application iot_simple_app {
            dbms oracle {
                  provides { sql_interface }
            }
```

```
            software_component web_server {
                  provides { sensor_info }
                  consumes { sql_interface get_weather }
            }

            software_component iot_provider {
                  consumes { sensor_info }
            }

            saas external_meteo {
                  provides { get_weather @ "https://api.mymeteo.com/get" }
            }
      }

      infrastructure infra {
            vm vm1 { size "micro" }
            vm vm2 {}
            node iot_device1 {}
            node iot_device2 {}
      }

      deployment config1 {
            oracle => vm1
            web_server => vm2
            iot_provider => iot_device1
            iot_provider => iot_device2
      }

      active deployment config1

      concretizations {
            concrete_infrastructure con_infra1 {
                  provider aws {
                        vm concrete_vm1 {
                              properties {}
                              maps vm1
                        }
                        vm concrete_vm2 {
                              properties {}
                              maps vm2
                        }
                  }
            }
            active con_infra1
      }
}
```

The example above shows the 2 main layers of DOML: application and infrastructure, as well as some elements in the commons layer (i.e., the configuration). As described in the example, the application layer contains 4 application components:

- The Oracle database software, defined by the DBMS metaclass. This component provides an sql_interface to access the database.
- The main web server software component, defined by the SoftwareComponent metaclass. As described in the example definition, the component consumes the database interface and the external_meteo API service. It also provides an interface sensor_info for IoT components to send messages.
- The IoT software component, which will be deployed to all IoT nodes, uses the sensor_info interface of the web application to upload their data.
- The external API required by the web application, described by the SaaS metaclass. In the case of the external API, the service provides an interface with an URL to access the weather information.

The abstract infrastructure layer defines all the infrastructural elements for the software components in the application layer:

- Two virtual machines are defined.
- Two physical nodes are modelled as the IoT devices deployed with the application.

Then, the configuration is done by simply linking the application component to the corresponding infrastructure elements, and that one is configured as the active configuration.

Finally, the concrete infrastructure layer defines the concretization for the abstract layer:

- The AWS runtime provider provisions the two concrete VM instances.

The XML representation of the above model is shown below:

```
<?xml version="1.0" encoding="ASCII"?>
<commons:DOMLModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:app="http://www.piacere-
project.eu/doml/application" xmlns:commons="http://www.piacere-project.eu/doml/commons"
xmlns:infra="http://www.piacere-project.eu/doml/infrastructure" name="iot_simple_app"
version="3.1" activeConfiguration="//@configurations.0"
activeInfrastructure="//@concretizations.0">
  <application name="iot_simple_app">
    <components xsi:type="app:DBMS" name="oracle" isPersistent="false">
      <exposedInterfaces name="sql_interface"/>
    </components>
    <components xsi:type="app:SoftwareComponent" name="web_server" isPersistent="false"
consumedInterfaces="//@application/@components.0/@exposedInterfaces.0
//@application/@components.3/@exposedInterfaces.0">
      <exposedInterfaces name="sensor_info"/>
    </components>
    <components xsi:type="app:SoftwareComponent" name="iot_provider"
isPersistent="false"
consumedInterfaces="//@application/@components.1/@exposedInterfaces.0"/>
    <components xsi:type="app:SaaS" name="external_meteo">
      <exposedInterfaces name="get_weather" endPoint="https://api.mymeteo.com/get"/>
    </components>
  </application>
  <infrastructure name="infra">
    <nodes xsi:type="infra:VirtualMachine" name="vm1" sizeDescription="micro"/>
    <nodes xsi:type="infra:VirtualMachine" name="vm2"/>
    <nodes xsi:type="infra:PhysicalComputingNode" name="iot_device1"/>
    <nodes xsi:type="infra:PhysicalComputingNode" name="iot_device2"/>
  </infrastructure>
  <concretizations name="con_infra1">
    <providers name="aws">
      <vms name="concrete_vm1" preexisting="false" maps="//@infrastructure/@nodes.0"/>
      <vms name="concrete_vm2" preexisting="false" maps="//@infrastructure/@nodes.1"/>
    </providers>
  </concretizations>
  <configurations name="config1">
    <deployments component="//@application/@components.0"
node="//@infrastructure/@nodes.0"/>
    <deployments component="//@application/@components.1"
node="//@infrastructure/@nodes.1"/>
    <deployments component="//@application/@components.2"
node="//@infrastructure/@nodes.2"/>
    <deployments component="//@application/@components.2"
node="//@infrastructure/@nodes.3"/>
  </configurations>
</commons:DOMLModel>
```

8.2 Optimization Problem Example

The example shows that a DOML model can be fed to an optimization service to obtain an optimal configuration for the cloud application according to the defined objectives and requirements. For simplicity, we only demonstrate the fragment of the optimization layer defined in a DOML model, as follows:

```
optimization opt {
      objectives {
            "cost" => min
            "availability" => max
            "performance" => max
      }
      nonfunctional_requirements {
            req1 "Cost <= 200" max 200.0 => "cost"
            req2 "Availability >= 98%" min 98.0 => "availability"
            req3 "Region" values "00EU" => "region"
            req4 "Provider" values "aws" => "provider"
            req5 "elements" => "VM VM"
      }
}
```

The optimization layer contains the definition of objectives and requirements. In the above fragment it defines 3 objectives, i.e., minimizing “cost”, maximizing “availability” and “performance”, and 5 requirements: the “cost” should be less than 200 (euro), the “availability” should be greater than 98%, the “Region” of VM should be located at “00EU” (based on IEC), the provider should be “aws” (Amazon Web Services), and two VMs should be returned as elements to be deployed.

The corresponding XML definition is shown below:

```
<optimization name="opt">
    <objectives xsi:type="optimization:MeasurableObjective" kind="min" property="cost"/>
    <objectives xsi:type="optimization:MeasurableObjective" kind="max"
property="availability"/>
    <objectives xsi:type="optimization:MeasurableObjective" kind="max"
property="performance"/>
    <nonfunctionalRequirements xsi:type="commons:RangedRequirement" name="req1"
description="Cost &lt;= 200" property="cost" max="200.0"/>
    <nonfunctionalRequirements xsi:type="commons:RangedRequirement" name="req2"
description="Availability >= 98%" property="availability" min="98.0"/>
    <nonfunctionalRequirements xsi:type="commons:EnumeratedRequirement" name="req3"
description="Region" property="region">
      <values>00EU</values>
    </nonfunctionalRequirements>
    <nonfunctionalRequirements xsi:type="commons:EnumeratedRequirement" name="req4"
description="Provider" property="provider">
      <values>aws</values>
    </nonfunctionalRequirements>
    <nonfunctionalRequirements name="req5" description="elements" property="VM VM"/>
  </optimization>
```

9 Conclusions

This document has described the specification of the DOML language concepts. The DOML has been conceived as a declarative language to make it easier for non-expert users, but it includes mechanisms to include imperative scripts and advanced features for expert user profiles.

DOML has also been designed considering the fast evolution of the cloud computing state-of-the-art, including mechanisms to extend itself easily, adding more concepts and properties to the existing ones.