*Assignments*

# Microservices Architecture Modeling

Department of Information Engineering, Computer Science and Mathematics

*Università degli Studi dell 'Aquila, Italy*

Submitted to Professor **Alfonso Pierantonio** & **Juri Di Rocco**

Course Titled as **Model Driven Engineering**

---

| | |
|---|---|
| **Version** | 5.1 |

| | |
|---|---|
| **Date** | January 6th, 2022 |



| Name and Surname | M. Number | Email Address |
|---|---|---|
| Abdul Qadir Ahmed Abbasi | 281728 | abdulqadirahmed.abbasi@student.univaq.it |

## Purpose

This document provides a comprehensive overview of the selected domain i.e. ***Microservices Architecture***, using a number of different tools & technologies to showcase covered aspects of the system required for modelling the above mentioned domain. It is intended to capture and convey the significant implementations which have been made for defining the domain specific languages for the ***Microservices*** based systems.

## Scope

The report provides implementation details of the **Microservices Architecture Modelling (MSA)**.

This document has been generated based on the assignments of Model Driven Engineering 2021/22 course provided to us.

## Tools & Technologies Used

All the content inside this document has been generated using following tools & technologies:

**JetBrains MPS 2021** (*for Metaclasses & Metamodels*)

**Eclipse Modelling Tools 2021-12** (*as Integrated Development Environment*)

**Eclipse Modelling Framework - EMF** (*for Metaclasses & Metamodels*)

**Object Constraint Language - OCLInEcore** (*for Metamodel Constraints*)

**Atlas Transformation Language - ATL** (*for Model to Model Transformation – M2M*)

**Acceleo** (*for Model to Text Transformation – M2T*)

**Xtext** (*for Textual Editor*)

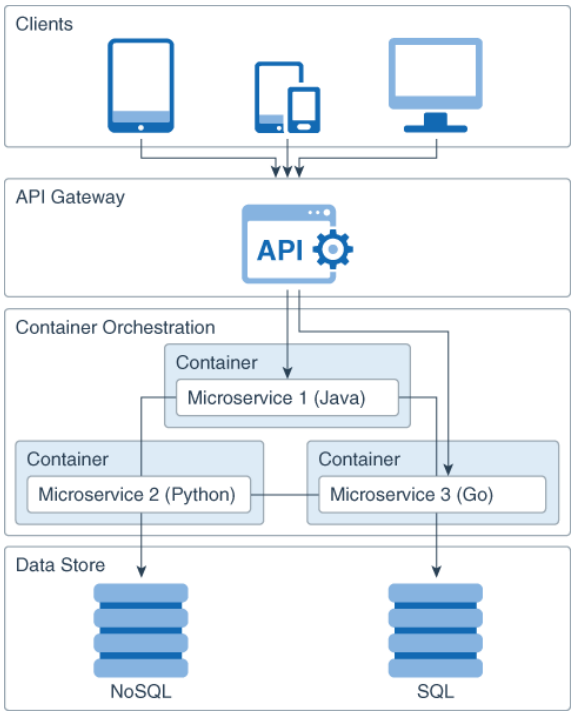
**Sirius** (*for Graphical Editor*)

# Table of Contents

## Domain Overview: Microservices

A microservices architecture is a type of application architecture where the application is developed as a collection of services. It provides the framework to develop, deploy, and maintain microservices independently.

Within a microservices architecture, each microservice is a single service built to accommodate an application feature and handle discrete tasks. Each microservice communicates with other services through simple interfaces to solve business problems.




*(**Figure 1.** Microservice Architecture Overview)*

As shown in ***Figure 1.*** we can see that each microservice is independent from others and focuses on some specific business domain. This allows us to develop microservices using different programming languages & databases. This also provides us flexibility to deploy & manage microservices in a convenient manner.

Some of the benefits of microservices architecture are: improved scalability, high availability, high availability, independent deployment & loosely coupled.

***NOTE: The core focus of this assignment is given to the deployment side. All the efforts done can help users in defining & visualizing the deployment of microservices along with finding estimates in both hardware & software requirement perspectives.***

# Asignment1: Metamodel Definition, Editor & Constraints using MPS

## Task A1.1: Definition of Metamodel

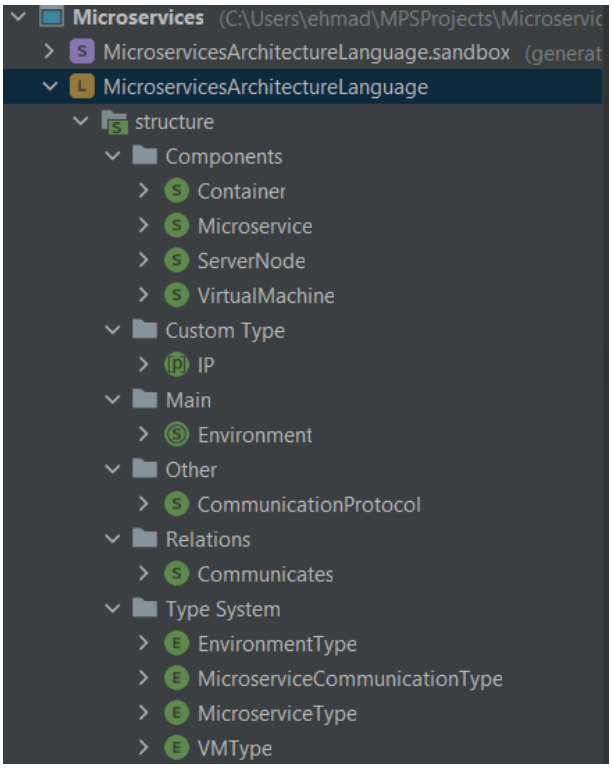
Following **metaclasses** are defined in the Meta model to cover the different aspects of the chosen domain i.e. Microservices:

- *Environment (root class in which the whole microservice architecture is deployed)*
- *ServerNode (class dedicated for hardware server to be hosted inside Environment)*
- *VirtualMachine (class for describing VM which can host Containers inside it)*
- *Container (a class for hosting Microservice & to be deployed inside a VirtualMachine)*
- *Microservice (a class for microservice itself)*

**Enumerations** used by the above mentioned classes are:

- *EnvironmentType (type of hosting environment)*
- *VMType (type of operating system in VM)*
- *MicroserviceType (inter or external)*
- *MicroserviceCommunnicationType (type of communication protocol)*

As one microservice can communicate with another microservice hence there is a **relation** named as "*Communicates*" between them. Additionally a **custom data type** is used for saving a valid "IP" address attributed associated to a "*Container*" class. All the classes **inherit** from a base concept provided by MPS itself implemented by "*INamedConcept*" **interface** for the "*Name*" attribute.



*(**Figure 2.** Folder Structure)*

## Codes

Codes of all the above explanations are given below:

***Figure 3.*** shows that an "*Environment*" can have **one to many children** of "*ServerNode*" type & can also have **one to many relations** of type "*Communicates*". "*EnvironmentType*" **enumeration** is also used.

```
Environment

concept Environment extends    BaseConcept
                    implements INamedConcept

  instance can be root: true
  alias: env
  short description: Meta class for environment hosting microservice architecture based application

  properties:
  Type     : EnvironmentType
  Location : string

  children:
  NodeList       : ServerNode[0..n]
  Communications : Communicates[0..n]

  references:
  << ... >>
```

*(**Figure 3.** Environment Metaclass)*

***Figure 4.*** shows that a "*ServerNode*" can have **one to many children** of "*VirtualMachine*" type.

```
ServerNode

concept ServerNode extends    BaseConcept
                   implements INamedConcept

  instance can be root: false
  alias: node
  short description: A meda class for hardware server node to be deployed in environment

  properties:
  RAM       : string
  Processor : string
  Disk      : string

  children:
  VMList : VirtualMachine[0..n]

  references:
  << ... >>
```

(***Figure 4.*** *ServerNode Metaclass)*

***Figure 5.*** shows that a "*VirtualMachine*" can have **one to many children** of "*Container*" type. "*VMType*" **enumeration** is also used.

```
VirtualMachine

concept VirtualMachine extends    BaseConcept
                       implements INamedConcept

  instance can be root: false
  alias: VM
  short description: A meta class for virtual machine for holding containers and to be deployed on server node

  properties:
  Type : VMType

  children:
  ContainterList : Container[0..n]

  references:
  << ... >>
```

(***Figure 5.*** *VirtualMachine Metaclass)*

***Figure 6.*** shows that a "*Container*" can have **zero or one children** of "*Microservice*" type and **custom datatype** "*IP*" is also used here.

```
Container

concept Container extends    BaseConcept
                  implements INamedConcept

  instance can be root: false
  alias: container
  short description: A meta class for container which will host docker image having microservice

  properties:
  IP   : IP
  Port : integer

  children:
  Microservice : Microservice[0..1]

  references:
  << ... >>
```

(***Figure 6.*** *Container Metaclass)*

***Figure 7.*** shows that a "*Microservice*" metaclass represents a Microservice itself and "*VMType*" enumeration is also used here.

```
Microservice

concept Microservice extends    BaseConcept
                     implements INamedConcept

  instance can be root: false
  alias: microservice
  short description: A meta class for microservice to be hosted inside container

  properties:
  Type   : MicroserviceType
  Health : boolean

  children:
  << ... >>

  references:
  << ... >>
```

(***Figure 7.*** *Microservice Metaclass)*

***Figure 8, 9, 10 & 11*** show definitions of different **enumerations** defined in the under discussion context.

```
EnvironmentType

enumeration EnvironmentType

members:
  • Development   <default presentation>
  • Test          <default presentation>
  • UAT           <default presentation>
  • Demo          <default presentation>
  • Production    <default presentation>   (default)

default member: Production
```

(***Figure 8.*** *EnvironmentType Enumeration)*

```
MicroserviceCommunicationType

enumeration MicroserviceCommunicationType

members:
  • RPC      <default presentation>
  • REST     <default presentation>   (default)
  • PubSub   <default presentation>

default member: REST
```

(***Figure 9.*** *CommunicationType Enumeration)*

```
MicroserviceType

enumeration MicroserviceType

members:
  • Internal   <default presentation>   (default)
  • External   <default presentation>

default member: Internal
```

(***Figure 10.*** *MicroserviceType Enumeration)*

```
VMType

enumeration VMType

members:
  • Linux     <default presentation>   (default)
  • RedHat    <default presentation>
  • Windows   <default presentation>
  • CentOS    <default presentation>

default member: Linux
```

(***Figure 11.*** *VMType Enumeration)*

***Figure 12.*** shows definitions of "*Communicates*" **relation** defined for showing communication between two instances of "*Microservices*" type.

```
Communicates

concept Communicates extends    CommunicationProtocol
                     implements <none>

  instance can be root: false
  alias: comm
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  microservice1 : Microservice[1]
  microservice2 : Microservice[1]
```

(***Figure 12.*** *Communicates Relation)*

***Figure 13.*** shows definitions of "*IP*" **custom datatype** defined for holding a valid IP address value by matching the regex in the "*Container*" type instance.

```
IP

constrained string datatype: IP

    matching regexp: \\b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\\.|$)){4}\\b
```

(***Figure 13.*** *IP Custom Datatype)*

## Task A1.2: Concrete Syntax & Associated Editor of Metamodel + Model Instantiation

Following associated **editors** are defined each metaclass in the Meta model:

- *Environment Editor (for instantiating object of Environment type)*
- *ServerNode Editor (for instantiating object of ServerNode type)*
- *VirtualMachine Editor (for instantiating object of VirtualMachine type)*
- *Container Editor (for instantiating object of Container type)*
- *Microservice Editor (for instantiating object of Microservice type)*
- *Communicates Editor (for instantiating Communicates relation)*

Different **colors** are associated with **data inputs, constants** & **opening/closing brackets** to increase readability and keep the code structured.

## Codes

Codes of all the above explanations are given below:

***Figure 14.*** shows definition of "*Environment Editor*" for defining instance of "*Environment*" type. It can set *name*, *Type* & *Location* along with list of *serverNodes* inside "*Environment*". User can also set the different "*communications*" among defined microservices.

```
Environment_Editor

<default> editor for concept Environment
  node cell layout:
    [-
    set environment { name } ( type = { Type } , location = ' { Location } ' ) {
      [-
      allocate serverNodes() {
        [-
        (- % NodeList % /empty cell: <default> -)
        -]
      }
      -]
    }
    set communications () {
      [-
      (- % Communications % /empty cell: <default> -)
      -]
    }
    -]
```

(***Figure 14.*** *Environment Editor*)

***Figure 15.*** shows definition of "*ServerNode Editor*" for defining instance of "*ServerNode*" type. It can set *name*, *RAM*, *Processor* & *Disk* of "*ServerNode*" along with list of "*VMs*".

```
ServerNode_Editor

<default> editor for concept ServerNode
  node cell layout:
    [-
    node { name } ( RAM = ' { RAM } ' , Processor = ' { Processor } ' , Disk = ' { Disk } ' ) {
      [-
      install VMs() {
        [-
        (- % VMList % /empty cell: <default> -)
        -]
      }
      -]
    }
    -]
```

(***Figure 15.*** *ServerNode Editor)*

***Figure 16.*** shows definition of "*VirtualMachine Editor*" for defining instance of "*VirtualMachine*" type. It can set *name* & *Type* of "*VirtualMachine*" along with list of "*Containers*".

```
VirtualMachine_Editor

<default> editor for concept VirtualMachine
  node cell layout:
    [-
    vm { name } ( type = { Type } ) {
      [-
      host Containers {
        [-
        (- % ContainterList % /empty cell: <default> -)
        -]
      }
      -]
    }
    -]
```

(***Figure 16.*** *VirtualMachine Editor)*

***Figure 17.*** shows definition of "*Container Editor*" for defining instance of "*Container*" type. It can set *name*, *IP* & *Port* of "*Container*" along with a "*Microservice*" type instance.

```
Container_Editor

<default> editor for concept Container
  node cell layout:
    [-
    docker { name } ( IP = { IP } , port = { Port } ) {
      [- % Microservice % -] <constant>
    }
    -]
```

(***Figure 17.*** *Container Editor)*

***Figure 18.*** shows definition of "*Microservice Editor*" for defining instance of "*Microservice*" type. It can set *name*, *Type* & *Health* of "*Microservice*" instance.

```
Microservice_Editor

<default> editor for concept Microservice
  node cell layout:
    [- microservice { name } ( type = { Type } , health = { Health } ) -]
```

(***Figure 18.*** *Microservice Editor)*

***Figure 19.*** shows definition of "*Communicates Editor*" for defining instance of "*Communicates*" relation. It can set *Type* of "*Communicates*" relation along with a "*Microservice1*" & "*Microservice2*" instances of type "*Microservice*".

```
Communicates_Editor

<default> editor for concept Communicates
  node cell layout:
    [- communicates { Type } ( ( % microservice1 % -> { name } ) , ( % microservice2 % -> { name } ) ) -]
```

(***Figure 19.*** *Communicates Editor)*

## Two Concrete Metamodel Instances Instantiation

***Figure 20.*** shows 1st instance of Microservices domain meta model.

```
BookingSystem.Prod

set environment BookingSystem.Prod ( type = Production , location = ' Rome ' ) {
  allocate serverNodes() {
    node Server1 ( RAM = ' 10 GB ' , Processor = ' 2.4 GHz ' , Disk = ' 40 GB ' ) {
      install VMs() {
        vm VM1 ( type = Linux ) {
          host Containers {
            docker Container1 ( IP = 192.168.1.20 , port = 8080 ) {
              microservice UserManagementService ( type = Internal , health = true )
            }
            docker Container2 ( IP = 192.168.1.10 , port = 8081 ) {
              microservice PaymentService ( type = External , health = true )
            }
            docker Container3 ( IP = 192.168.1.30 , port = 8082 ) {
              microservice AnalyticsService ( type = Internal , health = true )
            }
          }
        }
      }
    }
  }
}
set communications () {
  communicates REST ( AnalyticsService , UserManagementService )
  communicates REST ( AnalyticsService , PaymentService )
}
```

(***Figure 20.*** *Booking System Instance)*

***Figure 21.*** shows 2nd instance of Microservices domain meta model.

```
SmartBuildingManagementSystem.Test

set environment SmartBuildingManagementSystem.Test ( type = Test , location = ' Bahrain ' ) {
  allocate serverNodes() {
    node Server1 ( RAM = ' 6 GB ' , Processor = ' 2.4 Ghz ' , Disk = ' 10 GB ' ) {
      install VMs() {
        vm VM1 ( type = Windows ) {
          host Containers {
            docker Container1 ( IP = 172.10.10.41 , port = 1119 ) {
              microservice AuthorizationService ( type = External , health = true )
            }
            docker Container2 ( IP = 172.10.20.42 , port = 2020 ) {
              microservice IOTGateway ( type = Internal , health = true )
            }
          }
        }
      }
    }
  }
}
set communications () {
  communicates RPC ( AuthorizationService , IOTGateway )
}
```

(***Figure 21.*** *Smart Building Management System)*

## Task A1.3: Metamodel Constraints

Following two **constraints** are defined for the Meta models along with error notifications:

- *Restricted Port Number*
- *Invalid for Communication*

***Figure 22 & 23.*** shows *Restricted Port Number* constraints on "*Container*" meta model and its error output in editor.

```
Container_ConstraintRules

@API.Experimental
constraint rules for Container
  rules block for CanBeChild
    rule RestrictedPortNumber  is  childNode.Port > 1024 -> when fails show Error: Usage of restricted port is not allowed.
      applies always
```

(***Figure 22.*** *Restricted Port Number Constraint)*

```
BookingSystem.Prod
set environment BookingSystem.Prod ( type = Production , location = ' Rome ' ) {
  allocate serverNodes() {
    node Server1 ( RAM = ' 10 GB ' , Processor = ' 2.4 GHz ' , Disk = ' 40 GB ' ) {
      install VMs() {
        vm VM1 ( type = Linux ) {
          host Containers {
            docker Container1 ( IP = 192.168.1.20 , port = 1 ) {
              microservice UserManagementService ( type = Internal , health = true )
            }
            docker Container2 ( IP = 192.168.1.10 , port = 8081 ) {
              microservice PaymentService ( type = External , health = true )
            }
            docker Container3 ( IP = 192.168.1.30 , port = 8082 ) {
              microservice AnalyticsService ( type = Internal , health = true )
            }
          }
        }
      }
    }
  }
}
Error: Usage of restricted port is not allowed.
```

(***Figure 23.*** *Restricted Port Number Constraint Error)*

***Figure 24 & 25.*** shows *Invalid for Communication* constraints on "*Microservice*" meta model and its error output in editor.

```
Microservice_ConstraintRules
@API.Experimental
constraint rules for Microservice
  rules block for CanBeChild
    rule MicroserviceHealth  is  childNode.Health :eq: true -> when fails show
      applies always
Error: Communication cannot be done because microservice is down.
```

(***Figure 24.*** *Invalid for Communication Constraint)*

```
BookingSystem.Prod
set environment BookingSystem.Prod ( type = Production , location = ' Rome ' ) {
  allocate serverNodes() {
    node Server1 ( RAM = ' 10 GB ' , Processor = ' 2.4 GHz ' , Disk = ' 40 GB ' ) {
      install VMs() {
        vm VM1 ( type = Linux ) {
          host Containers {
            docker Container1 ( IP = 192.168.1.20 , port = 8080 ) {
              microservice UserManagementService ( type = Internal , health = true )
            }
            docker Container2 ( IP = 192.168.1.10 , port = 8081 ) {
              microservice PaymentService ( type = External , health = true )
            }
            docker Container3 ( IP = 192.168.1.30 , port = 8082 ) {
              microservice AnalyticsService ( type = Internal , health = false )
            }
          }
        }
      }
    }
  }
}
Error: Communication cannot be done because microservice is down.
```

(***Figure 25.*** *Invalid for Communication Constraint Error)*

# Asignment2: Metamodel Definition, Model Instantiation & Constraints using EMF + OCL

## Task A2.1: Definition of Metamodel

Following **metaclasses** are defined in the Meta model to cover the different aspects of the chosen domain i.e. Microservices:

- *Environment (root class in which the whole microservice architecture is deployed)*
- *ServerNode (class dedicated for hardware server to be hosted inside Environment)*
- *VirtualMachine (class for describing VM which can host Containers inside it)*
- *Container (a class for hosting Microservice & to be deployed inside a VirtualMachine)*
- *Microservice (a class for microservice itself)*

**Enumerations** used by the above mentioned classes are:

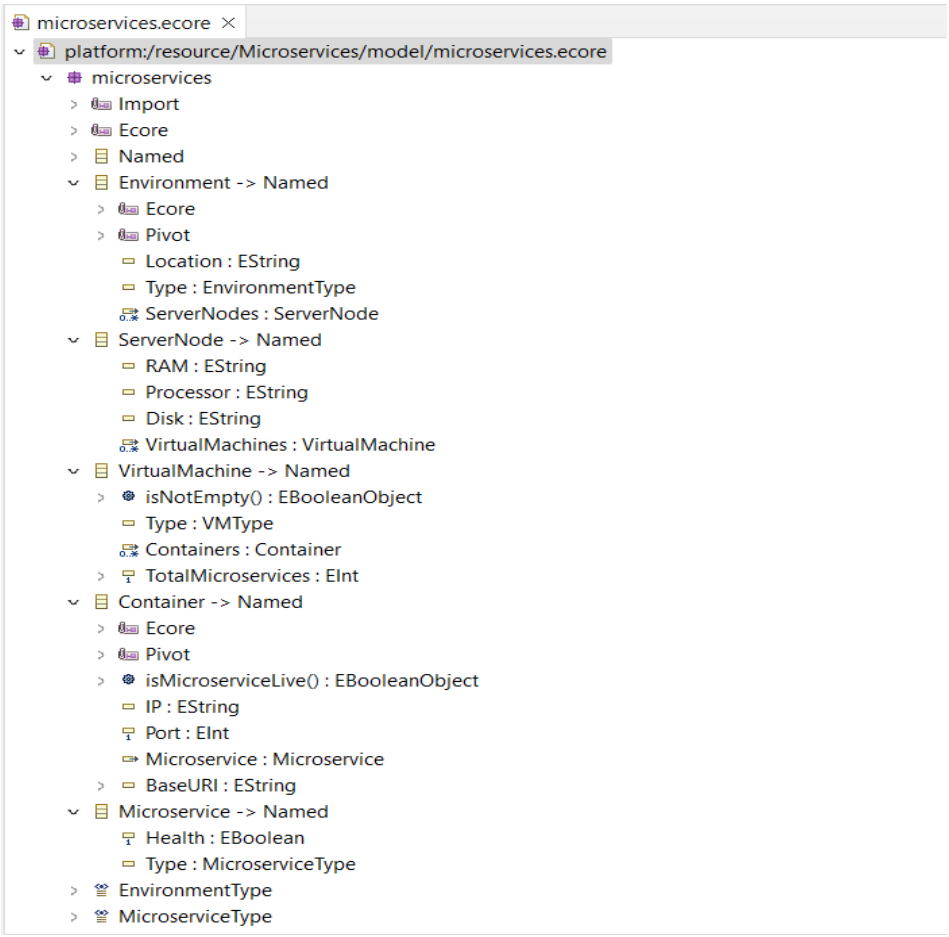
- *EnvironmentType (type of hosting environment)*
- *VMType (type of operating system in VM)*
- *MicroserviceType (inter or external)*

All the classes **inherit** from a base class named as "*Named*". A list of "*ServerNode*" is **contained** inside the "*Environment*". A list of "*VirtualMachine*" is **contained** inside the "*ServerNode*". A list of "*Container*" is **contained** inside the "*VirtualMachine*". A **zero or many** "*Microservice*" is **contained** inside the "*Container*".

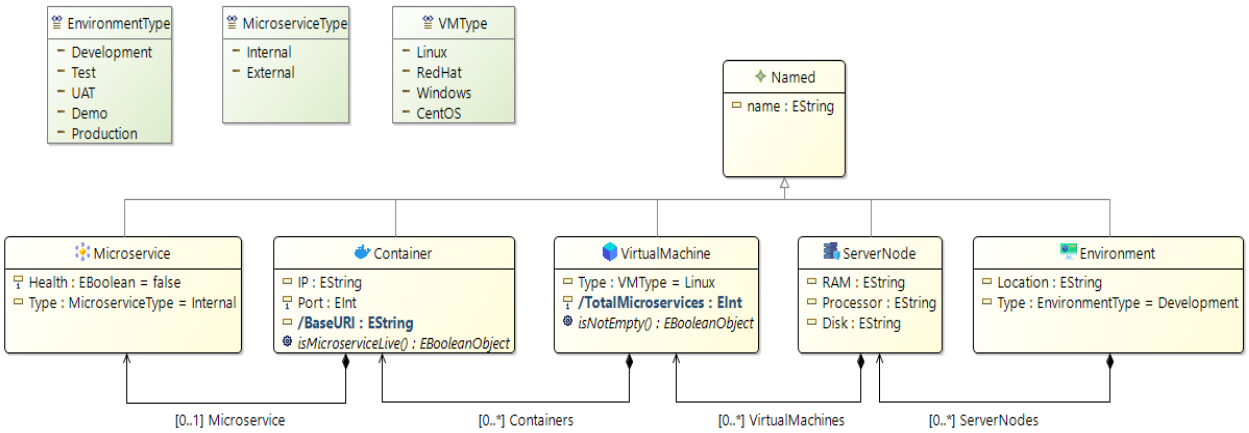
Following icons are associated for representation purposes in the coming sections:

- Environment
  
- ServerNode
  
- VirtualMachine
  
- Container
  
- Microservice
  

Each class has associated set of **attributes** and all the explanations are reflected in the ***Figure 26 & 27*** which represent the Ecore Meta Model & Class Diagram of Microservices domain.



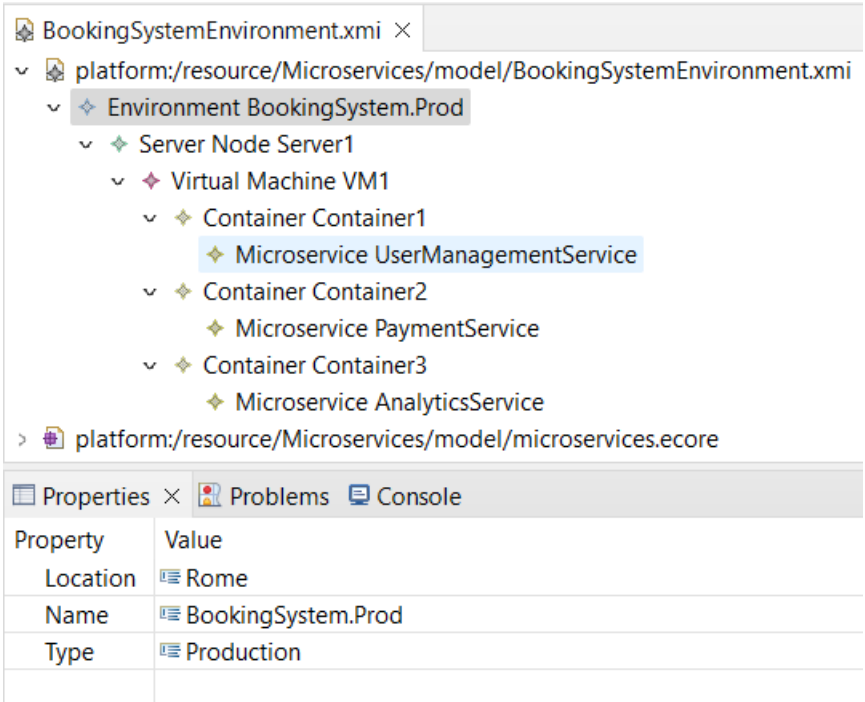

(***Figure 26.*** *EMF Ecore Metamodel of Microservices)*



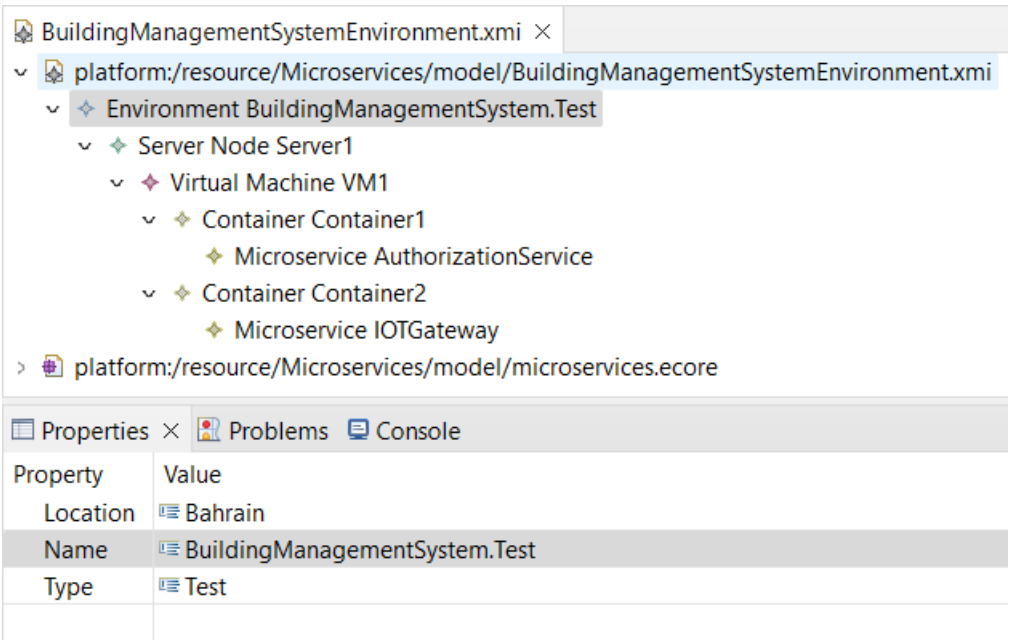

(***Figure 27.*** *Class Diagram of Microservices Metamodel)*

## Task A2.2: Two Concrete Metamodel Instances Instantiation

Following **two** instance confirming the Meta model are instantiated and showed in ***Figure 28 & 29***:




(***Figure 28.*** *Booking System Instance)*




(***Figure 29.*** *Building Management System Instance)*

## Task A2.3: Metamodel Constraints, Operations & Derived Fields

Following three **constraints** are defined for the Metamodel along with errors on validations:

- *Valid Port Assigned (for Container)*
- *Valid IP Address (for Container)*
- *Different Server Names (for Environment)*

Following two **operations** are defined for the Metamodel:

- *isNotEmpty() (for VirtualMachine)*
- *isMicroserviceLive() (for Container)*

Following two **derived fields** are defined for the Metamodel:

- *TotalMicroservices (for VirtualMachine)*
- *BaseURI (for Container)*

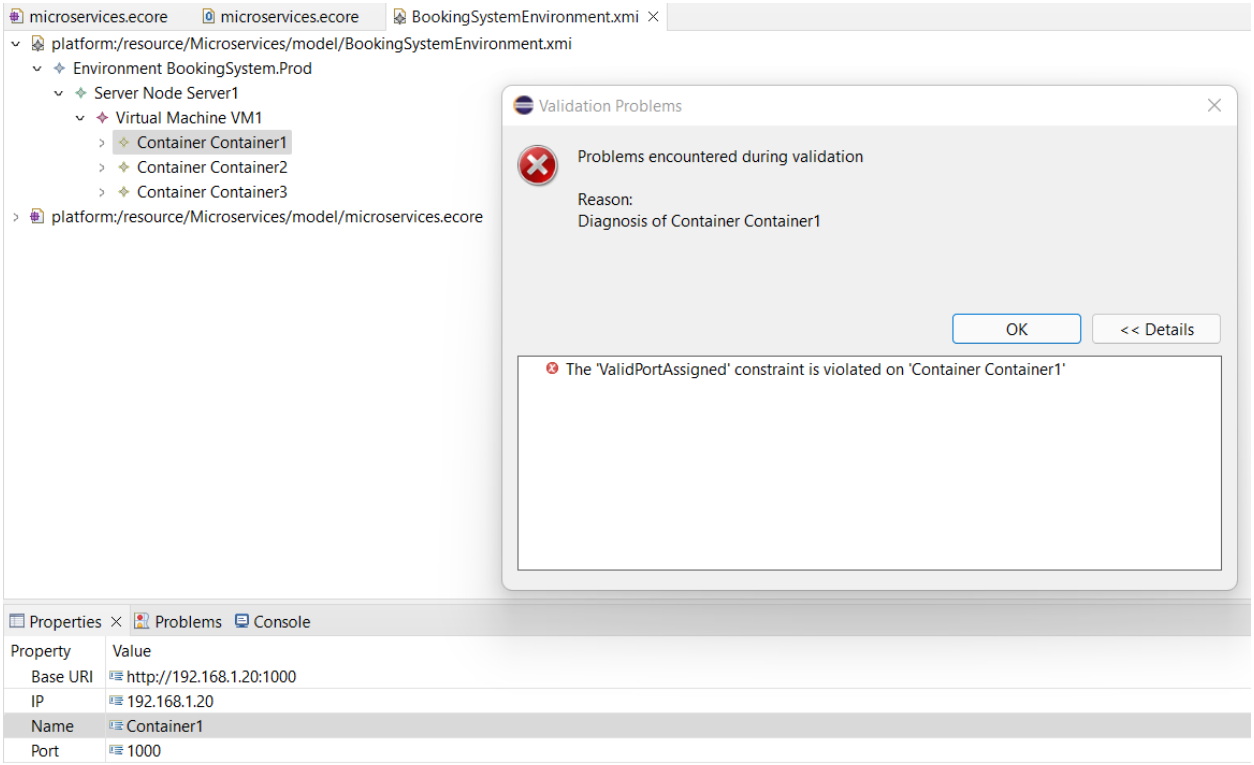
### Code

The code for the above mentioned constraints, operations & derived fields is given below:

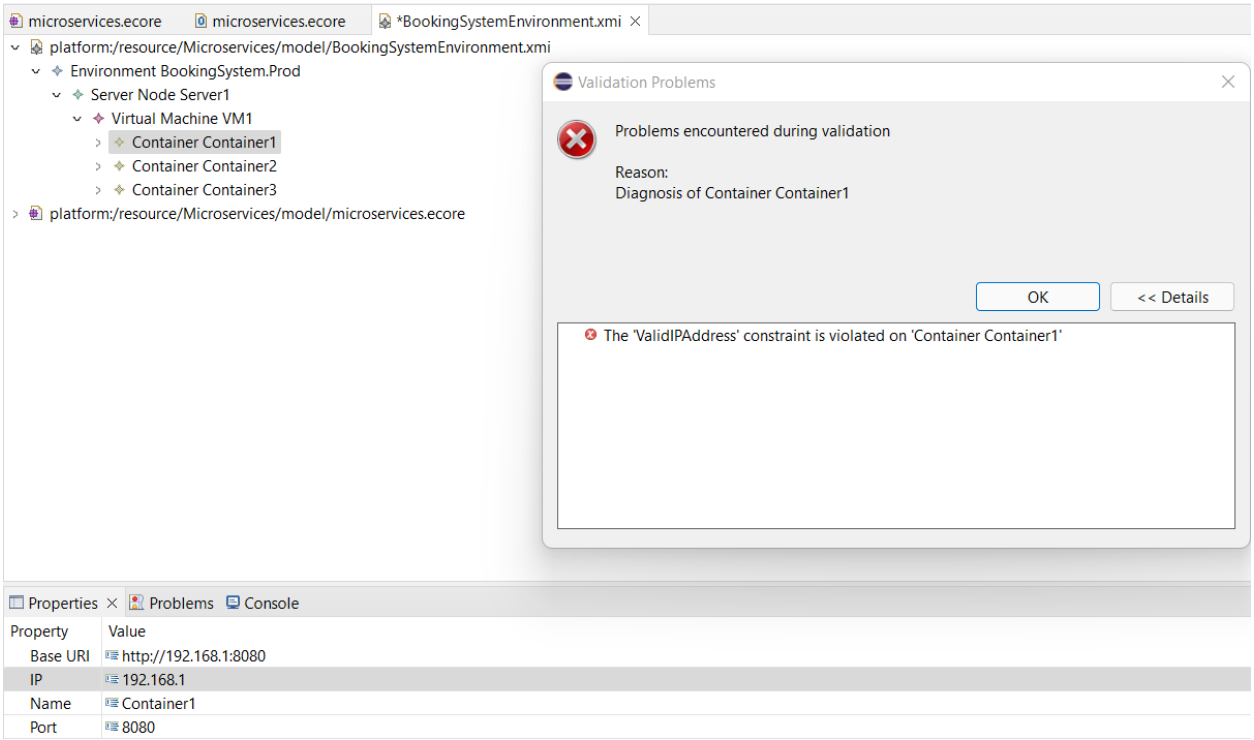
```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore';

package microservices : microservices = 'http://www.example.org/microservices'
{
	class Named
	{
		attribute name : String[?];
	}
	class Environment extends Named
	{
		attribute Location : String[?];
		attribute Type : EnvironmentType[?];
		property ServerNodes : ServerNode[*|1] { ordered composes };
		invariant DifferentServerNames: ServerNodes->isUnique(x | x.name);
	}
	class ServerNode extends Named
	{
		attribute RAM : String[?];
		attribute Processor : String[?];
		attribute Disk : String[?];
		property VirtualMachines : VirtualMachine[*|1] { ordered composes };
	}
	class VirtualMachine extends Named
	{
		attribute Type : VMType[?];
		property Containers : Container[*|1] { ordered composes };
		--Derived Property
		property TotalMicroservices  : ecore::EInt[1]{ derived,volatile }
	{
```

```
		derivation: Containers->collectNested(Microservice) -> size();
	}
	--Operation
		operation isNotEmpty() : Boolean[?]
		{
			body: Containers -> notEmpty();
		}
	}
	class Container extends Named
	{
		attribute IP : String[?];
		attribute Port : ecore::EInt[1];
		property Microservice : Microservice[?] { composes };
		--Derived Property
		property BaseURI  : String[?] { derived,volatile }
	{
		derivation: 'http://' + IP + ':' + Port.toString();
	}
	--Constraints
	invariant ValidPortAssigned: Port > 1024;
		invariant ValidIPAddress: IP.matches('\\b((25[0-5]|2[0-4][0-9]|[01]?[0-
9][0-9]?)(\\.|$)){4}\\b') = true;
		--Operation
		operation isMicroserviceLive() : Boolean[?]
		{
			body: self.Microservice.Health;
		}
	}
	class Microservice extends Named
	{
		attribute Health : Boolean[1];
		attribute Type : MicroserviceType[?];
	}
	enum EnvironmentType { serializable }
	{
		literal Development;
		literal Test = 1;
		literal UAT = 2;
		literal Demo = 3;
		literal Production = 4;
	}
	enum MicroserviceType { serializable }
	{
		literal Internal;
		literal External = 1;
	}
	enum VMType { serializable }
	{
		literal Linux;
		literal RedHat = 1;
		literal Windows = 2;
		literal CentOS = 3;
	}
}
```

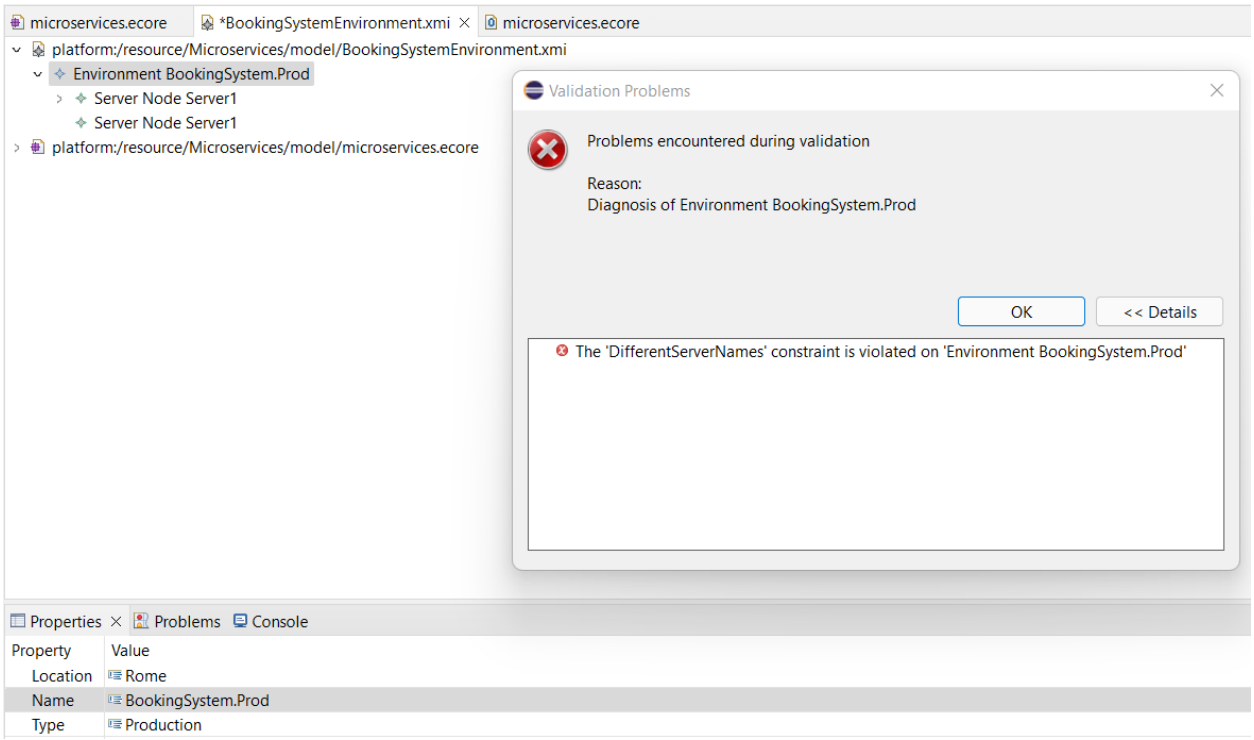
***Figure 30, 31 & 32.*** show **constraints** error output notification while validating the meta model.




(***Figure 30.*** *Valid Port Assigned Constraint on Container)*

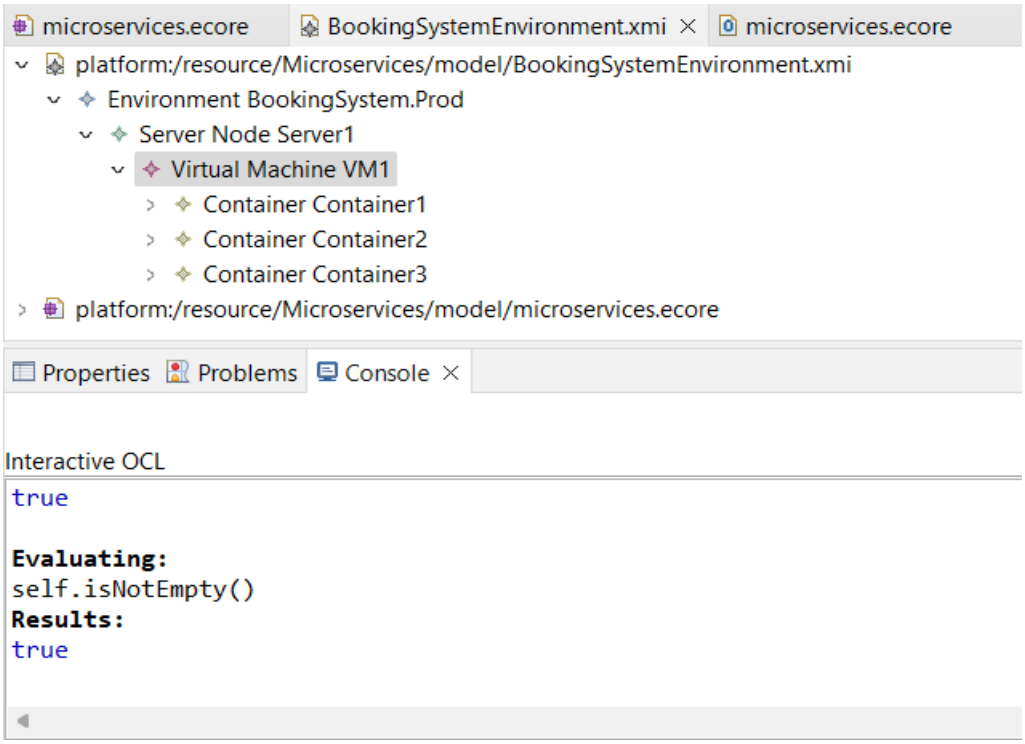



(***Figure 31.*** *Valid IP Address Constraint on Container)*

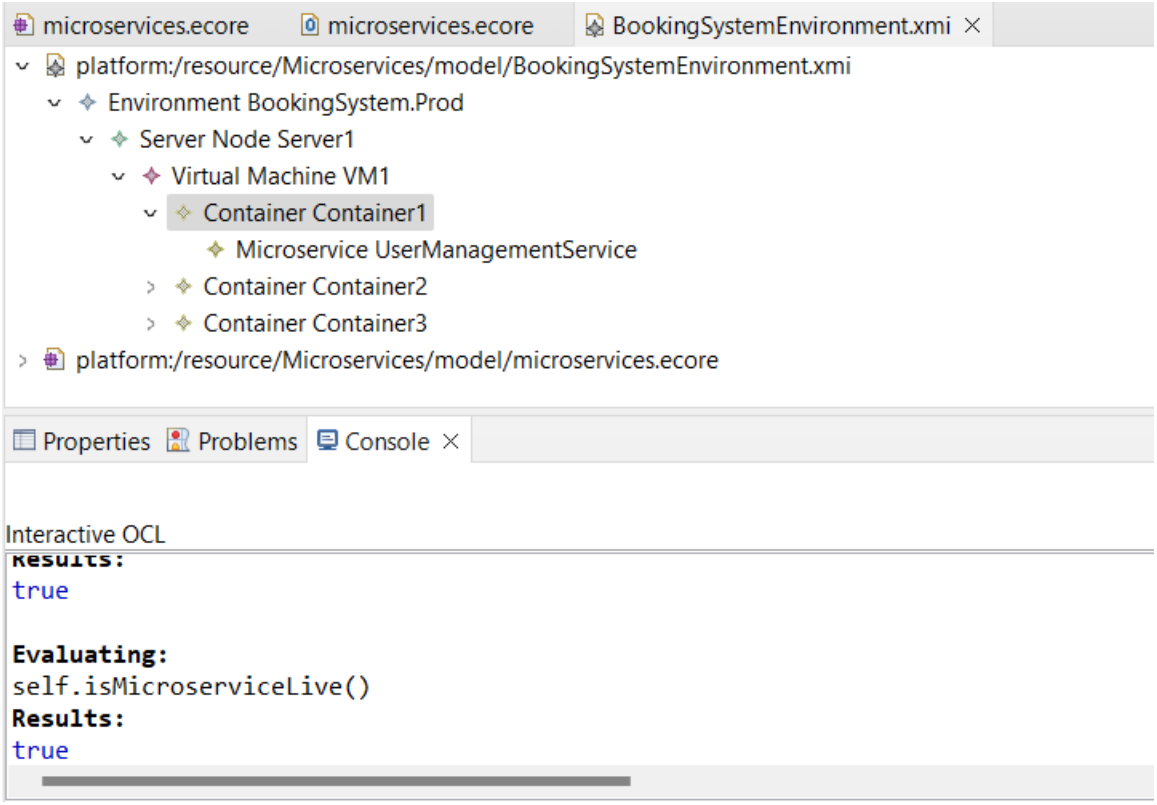



(***Figure 32.*** *Different Server Names Constraint on Environment)*

***Figure 33 & 34.*** show **operations** defined for the meta model along with outputs in OCL console.

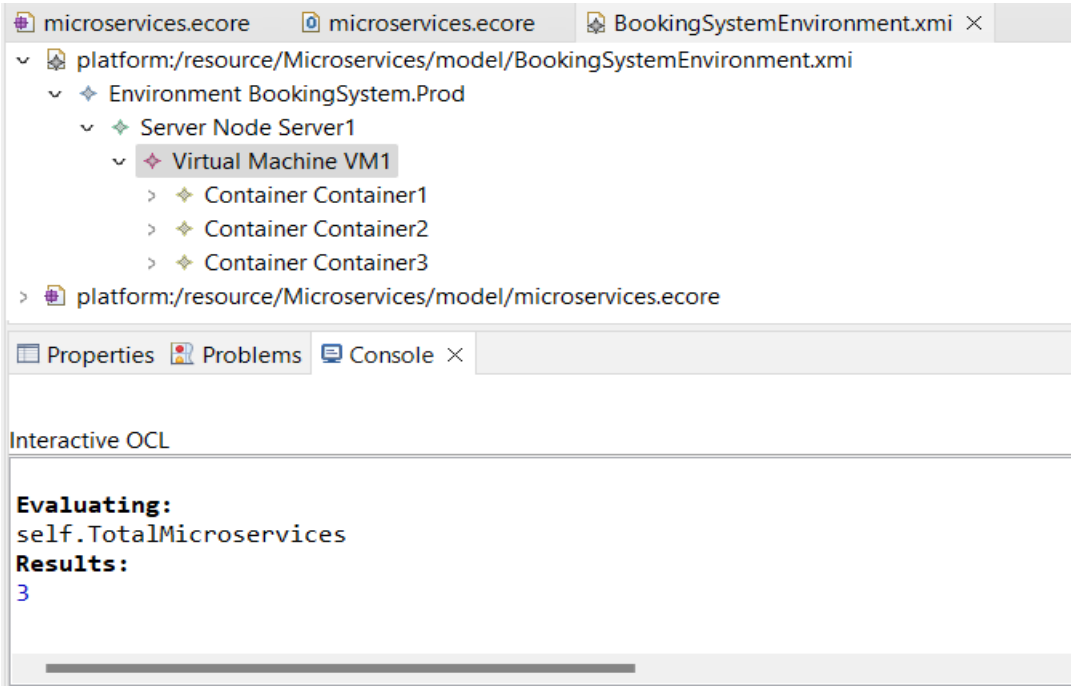



(***Figure 33.*** *isNotEmpty() Operation for VirtualMachine)*

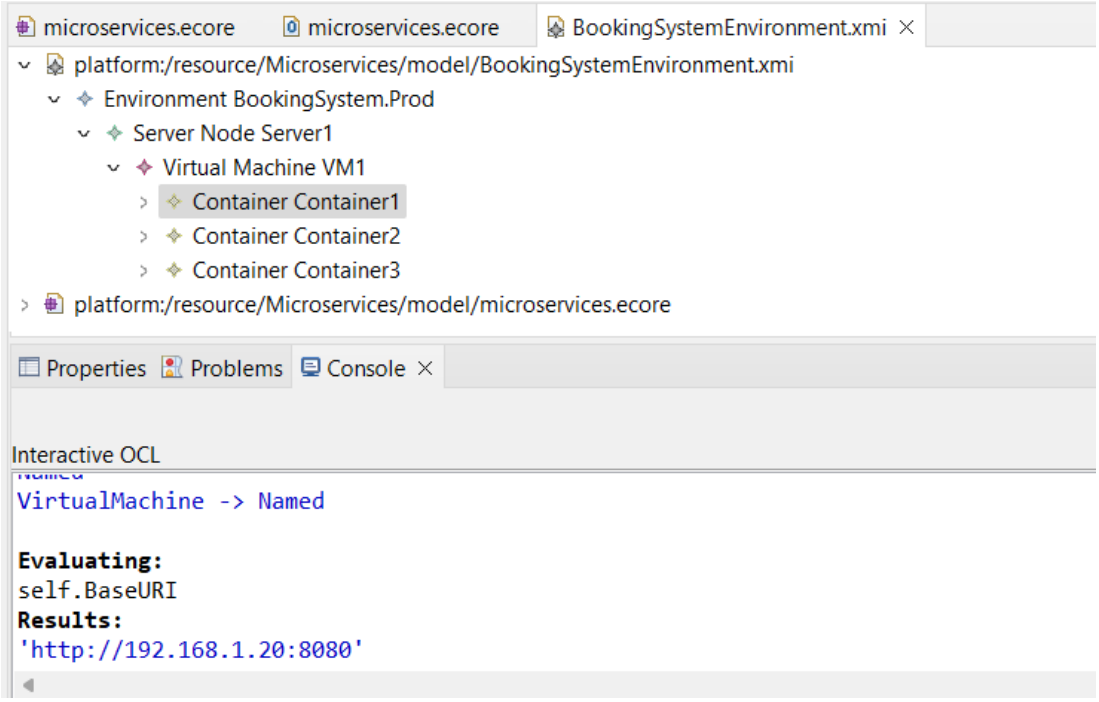



(***Figure 34.*** *isMicroserviceLive() Operation for Container)*

***Figure 35 & 36.*** show **derived fields** defined for the meta model along with outputs in OCL console.




(***Figure 35.*** *TotalMicroservices Derived Field for VirtualMachine)*




(***Figure 36.*** *BaseURI Derived Field for Container)*

---

# Asignment3: Model to Model & Model to Text Transformation using ATL + Acceleo

## Task A3.1: Refactoring of Metamodel

Following **metaclasses** are defined in the refactored Meta model to cover the different aspects of the chosen domain i.e. Microservices. Some **concepts** are **deleted** & new **concepts** are **added**. Few concepts have been **renamed**. **Structural** changes are also made i.e. **removal** of **attributes** and **references**. Some **attributes** are also **changed** All the refactoring are highlighted i.e. green as new, red as old & ~~red~~ as deleted:

- *Environment (no change in root class)*
- *Node (name of class changed from ServerNode to Node)*
- ~~*VirtualMachine*~~ *(class has been deleted)*
- *Docker (name of class changed from Container to Docker)*
- *Microservice (now this class inherits from both Entity & DataTransferObject classes)*
- *DataTransferObject (new concept added)*

**Enumerations** used by the above mentioned classes are:

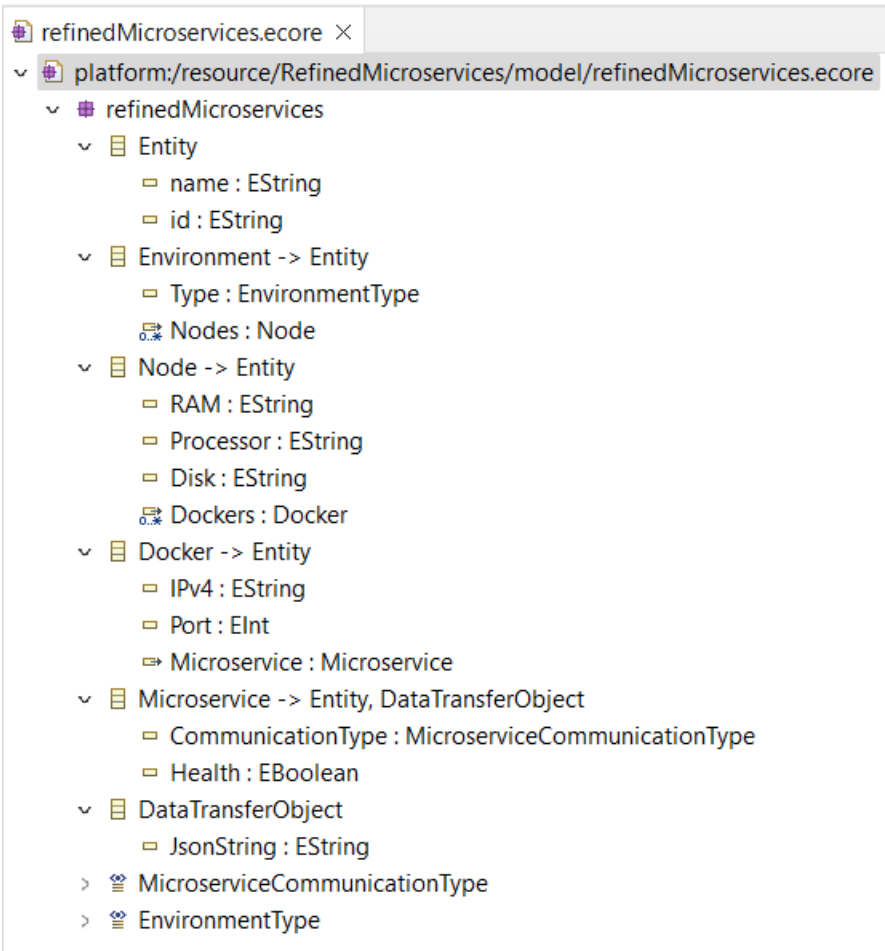
- *EnvironmentType (type of hosting environment)*
- ~~*VMType*~~ *this enumeration is deleted)*
- ~~*MicroserviceType*~~ *(this enumeration is deleted)*

- *MicroserviceCommunicationType (a new enumeration added)*

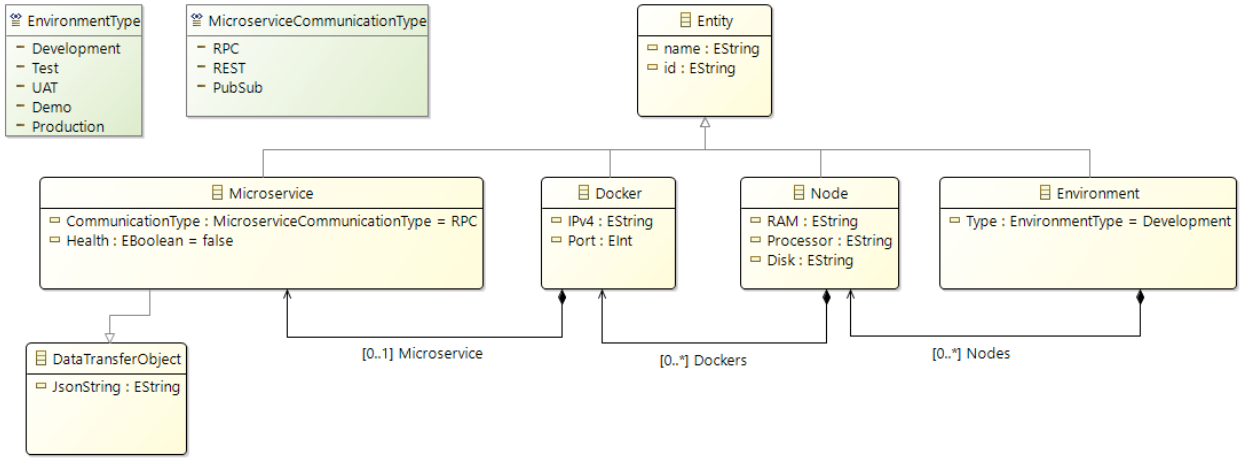
All the classes now **inherit** from a new base class named as "*Entity*" having both *Id* & *Name* attributes instead of "*Named*" which was having only *Name* attribute. A list of "*Node*" is **contained** inside the "*Environment*". A list of "*Docker*" is **contained** inside the "*Node*" and the concept & relation of "~~*VirtualMachine*~~" is deleted. A list of "*Container*" is changed to list of "*Docker*". A **zero or many** "*Microservice*" is **contained** inside the "*Docker*".

The attribute named "*IP*" is changed to "*IPv4*". A new attribute of "*CommunicationType*" is added to the "*Microservice*" and the older "~~*Type*~~" attribute is deleted.

***Figure 37 & 38*** represent the Ecore Meta Model & Class Diagram of Refactored Microservices domain.




(***Figure 37.*** *EMF Ecore Refined Metamodel of Microservices)*




*(**Figure 38.** Class Diagram of Refined Microservices Metamodel)*

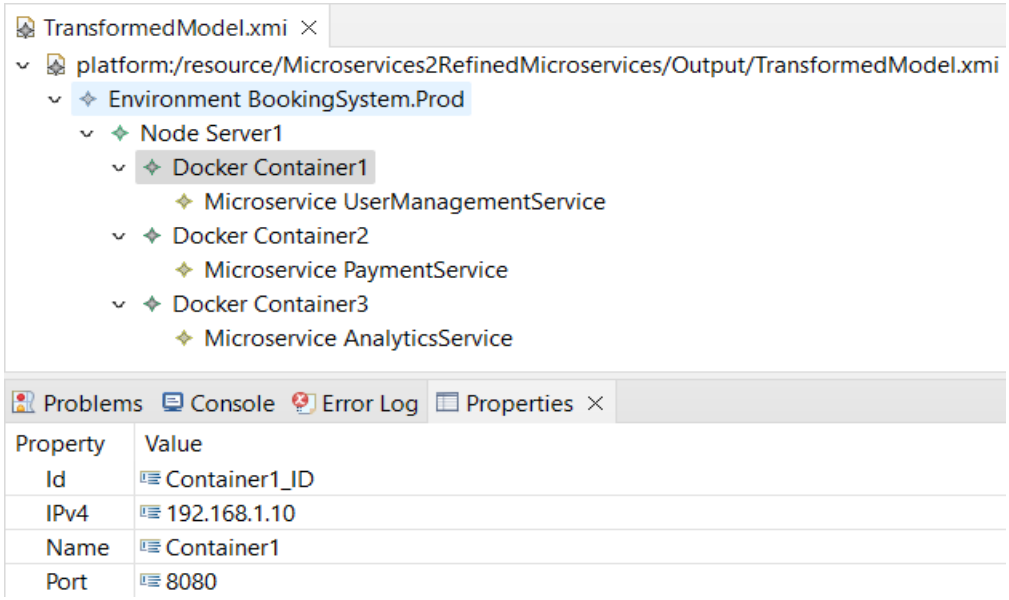
## Task A3.2: Model to Model Transformation

Model to Model transformation for *Microservices* to *RefinedMicroservices* model is done using ATL and the code is given below. **Matched rules** for each concept are defined and one **helper** is also defined to get the ID from the Name attribute.

### Code

```
-- @path MM=/Microservices/model/microservices.ecore
-- @path OUTMM=/RefinedMicroservices/model/refinedMicroservices.ecore

module Microservices2RefinedMicroservices;
create OUT : OUTMM from IN : MM;

helper def : getIDFromName(name : String): String = name + '_ID';

rule Environment2Environment {
   from
         src : MM!Environment
   to
         trg : OUTMM!Environment (
                   id <- thisModule.getIDFromName(src.name),
                   name <- src.name,
                   Type <- src.Type,
                   Nodes <- src.ServerNodes
         )
 }

rule ServerNode2Node {
   from
         src : MM!ServerNode
   to
         trg : OUTMM!Node (
                   id <- thisModule.getIDFromName(src.name),
```

```
                    name <- src.name,
                    RAM <- src.RAM,
                    Processor <- src.Processor,
                    Disk <- src.Disk,
                    Dockers <- src.VirtualMachines -> collect(vm | vm.Containers) ->
flatten()
          )
 }

rule Container2Docker {
   from
          src : MM!Container
   to
          trg : OUTMM!Docker (
                    id <- thisModule.getIDFromName(src.name),
                    name <- src.name,
                    IPv4 <- src.IP,
                    Port <- src.Port,
                    Microservice <- src.Microservice
          )
 }

rule Microservice2Microservice {
   from
          src : MM!Microservice
   to
          trg : OUTMM!Microservice (
                    id <- thisModule.getIDFromName(src.name),
                    name <- src.name,
                    Health <- src.Health,
                    JsonString <- '{}'
          )
 }
```




*(**Figure 39.** Output of Transformed Model)*

***Figure 39.*** shows the output of the transformed model from the instance of "*Microservices*" to the instance of "*RefinedMicroservices*" using the code defined in ATL given above.

## Task A3.3: Model to Text Transformation

Model to text transformation for *Microservices* model is done using Acceleo and the code is given below. One HTML index page is generated along with dedicated pages for the concepts.

### Code (generate.mtl)

```
[comment encoding = UTF-8 /]
[**
 * The documentation of the module generate.
 */]
[module generate('http://www.example.org/microservices')]
[import org::eclipse::acceleo::module::microservices::main::generateDetailsPage /]

[**
 * The documentation of the template generateElement.
 * @param anEnvironment
 */]
[template public generateElement(anEnvironment : Environment)]

      [comment @main/]
      [file (anEnvironment.name.replaceAll('', ' ') + '.html', false, 'UTF-8')]
      <html>
                  <head>
                        <style>
                              table, th, td {
                                border: 1px solid black;
                              }
                              label {
                                font-weight:bold
                              }
                              h1 {
                                width:500px;
                             margin: 0 auto;
                             background: gray;
                             text-align: center;
                              }
                              a.button {
                                  -webkit-appearance: button;
                                  -moz-appearance: button;
                                  appearance: button;

                                  text-decoration: none;
                                  color: initial;
                                     background: gray;
                                     float: right;
                                     margin-left: 5px;
                                     border-radius: 3px;
```

```
                                        padding: 5px;
                                    }
                                    .red-dot
                                    {
                                       height: 10px;
                                       width: 10px;
                                       background-color: red;
                                       border-radius: 50%;
                                       display: inline-block;
                                    }
                                    .green-dot
                                    {
                                       height: 10px;
                                       width: 10px;
                                       background-color: green;
                                       border-radius: 50%;
                                       display: inline-block;
                                    }
                             </style>
                             <title>[anEnvironment.name/] Home</title>
                      </head>
                      <body>
                             <h1>Microservices Archictecture used for
"[anEnvironment.name/]" Application
                             <h2>Details</h2>
                             <label>Name : </label> <span>[anEnvironment.name/]</span>
<br>
                             <label>Type : </label> <span>[anEnvironment.Type/]</span>
<br>
                             <label>Location : </label>
<span>[anEnvironment.Location/]</span> <br>
                             <div>It consists of <strong>[anEnvironment.ServerNodes-
>size()/]</strong> server/s.</div>

                             <h2>Servers</h2>
                             <table>
                                    <thead>
                                           <tr>
                                                  <th>Name</th>
                                                  <th>RAM</th>
                                                  <th>Processor</th>
                                                  <th>Disk</th>
                                                  <th>Total Virtual Machines</th>
                                                  <th>Virtual Machines</th>
                                           <tr>
                                    </thead>
                                    <tbody>

                                    [for (node : ServerNode |
anEnvironment.ServerNodes)]
                                           [generateServerRow(node)/]
                                    [/for]

                                    </tbody>
                             </table>
```

```
                    </body>
        </html>
        [/file]

[/template]


[template private generateServerRow(aServerNode : ServerNode)]
<tr>
        <td>[aServerNode.name/]</td>
        <td>[aServerNode.RAM/]</td>
        <td>[aServerNode.Processor/]</td>
    <td>[aServerNode.Disk/]</td>
    <td>[aServerNode.VirtualMachines -> size()/]</td>
        <td>
            [for (vm : VirtualMachine | aServerNode.VirtualMachines)]
                    [vm.name/]<a target="_blank" href="[vm.name.replaceAll(' ', '') +
'.html'/]" class="button">Details</a>
                    [vm.generateVirtualMachineDetailsPage()/]
              [/for]
        </td>
</tr>
[/template]
```

## Code (generateDetailsPage.mtl)

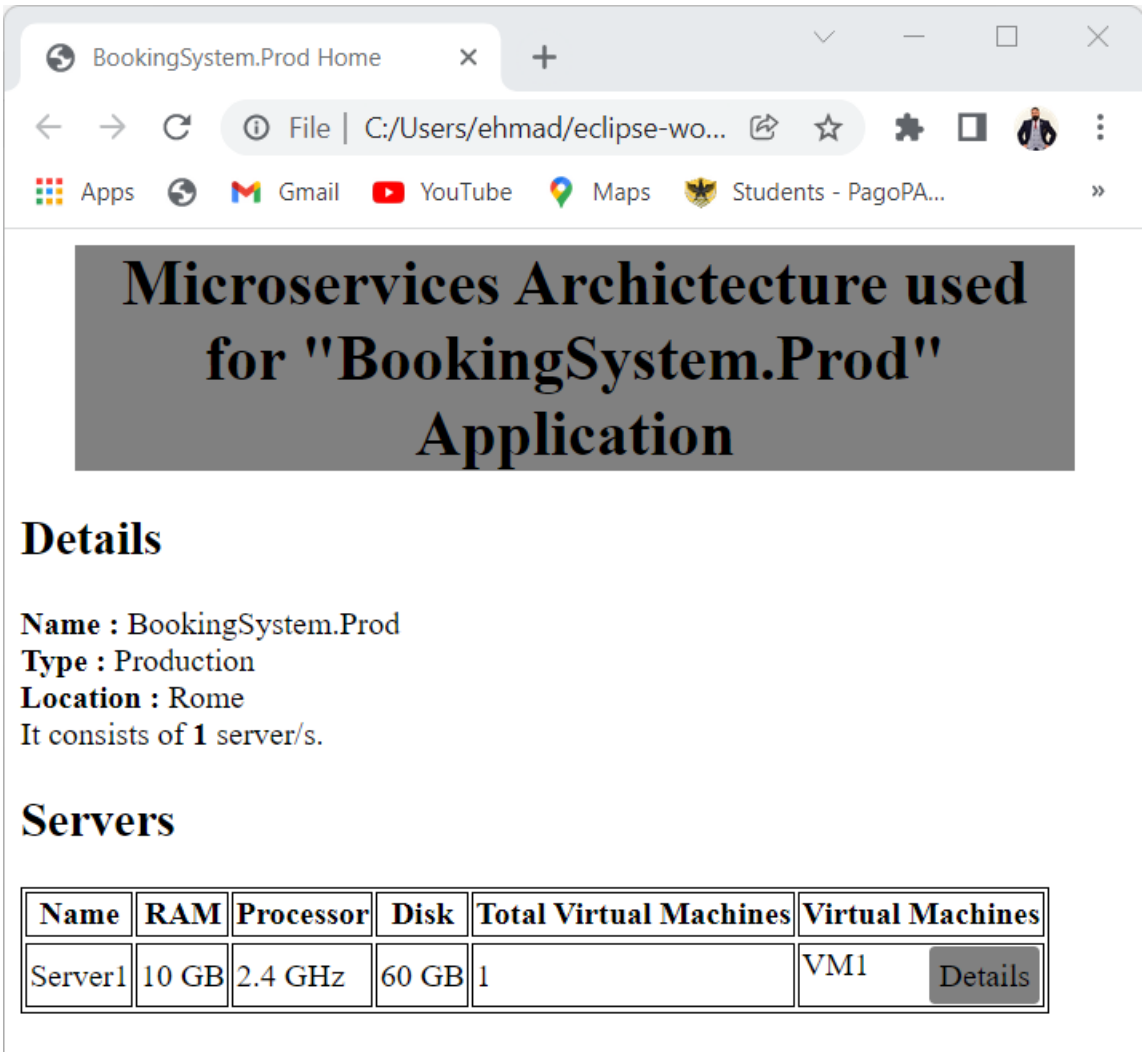
```
[comment encoding = UTF-8 /]
[module generateDetailsPage('http://www.example.org/microservices')/]

[template public generateVirtualMachineDetailsPage(aVirtualMachine : VirtualMachine)]

        [comment @main /]
        [file (aVirtualMachine.name.replaceAll(' ', '') + '.html', false, 'UTF-8')]
              <head>
                    <style>
                          table, th, td {
                            border: 1px solid black;
                          }
                          label {
                            font-weight:bold
                          }
                          h1 {
                            width:500px;
                         margin: 0 auto;
                         background: gray;
                         text-align: center;
                          }
                          a.button {
                              -webkit-appearance: button;
                              -moz-appearance: button;
                              appearance: button;
```

```
                    text-decoration: none;
                    color: initial;
                      background: gray;
                      float: right;
                      margin-left: 5px;
                      border-radius: 3px;
                      padding: 5px;
                }
                .red-dot
                {
                   height: 10px;
                   width: 10px;
                   background-color: red;
                   border-radius: 50%;
                   display: inline-block;
                }
                .green-dot
                {
                   height: 10px;
                   width: 10px;
                   background-color: green;
                   border-radius: 50%;
                   display: inline-block;
                }
            </style>
            <title>[aVirtualMachine.name/] Details</title>
        </head>
        <body>
            <h1>"[aVirtualMachine.name/]" Vitual Machine
            <h2>Details</h2>
            <label>Name : </label> <span>[aVirtualMachine.name/]</span> <br>
            <label>Type : </label> <span>[aVirtualMachine.Type/]</span> <br>
            <div>It consists of <strong>[aVirtualMachine.Containers ->
collectNested(Microservice) -> size()/]</strong> microservice/s.</div>

            <h2>Containers</h2>
            <table>
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>IP</th>
                        <th>Port</th>
                        <th>BaseURI</th>
                        <th>Microservice</th>
                        <th>isMicroserviceLive</th>
                    <tr>
                </thead>
                <tbody>

                [for (docker : Container | aVirtualMachine.Containers)]
                    [generateContainerRow(docker)/]
                [/for]

                </tbody>
```

```
                    </table>
              </body>
        [/file]

[/template]

[template private generateContainerRow(aContainer : Container)]
<tr>
        <td>[aContainer.name/]</td>
        <td>[aContainer.IP/]</td>
        <td>[aContainer.Port/]</td>
     <td>[aContainer.BaseURI/]</td>
        <td>
              [aContainer.Microservice.name/]<a target="_blank"
href="[aContainer.Microservice.name.replaceAll(' ', '') + '.html'/]"
class="button">Details</a>
              [aContainer.Microservice.generateMicroserviceDetailsPage()/]
        </td>

        <td style="text-align:center">
              [if (aContainer.isMicroserviceLive())]
                    <span class="green-dot"
title="[aContainer.isMicroserviceLive()/]"></span></td>
         [else]
              <span class="red-dot"
title="[aContainer.isMicroserviceLive()/]"></span></td>
         [/if]
</tr>
[/template]

[template public generateMicroserviceDetailsPage(aMicroservice : Microservice)]

        [comment @main /]
        [file (aMicroservice.name.replaceAll(' ', '') + '.html', false, 'UTF-8')]
              <head>
                    <style>
                          label {
                            font-weight:bold
                          }
                          h1 {
                            width:500px;
                         margin: 0 auto;
                         background: gray;
                         text-align: center;
                          }
                    </style>
                    <title>[aMicroservice.name/] Details</title>
              </head>
              <body>
                    <h1>"[aMicroservice.name/]" Microservice
                    <h2>Details</h2>
                    <label>Name : </label> <span>[aMicroservice.name/]</span> <br>
                    <label>Type : </label> <span>[aMicroservice.Type/]</span> <br>
                    <label>Health : </label> <span>[aMicroservice.Health/]</span>
<br>
```

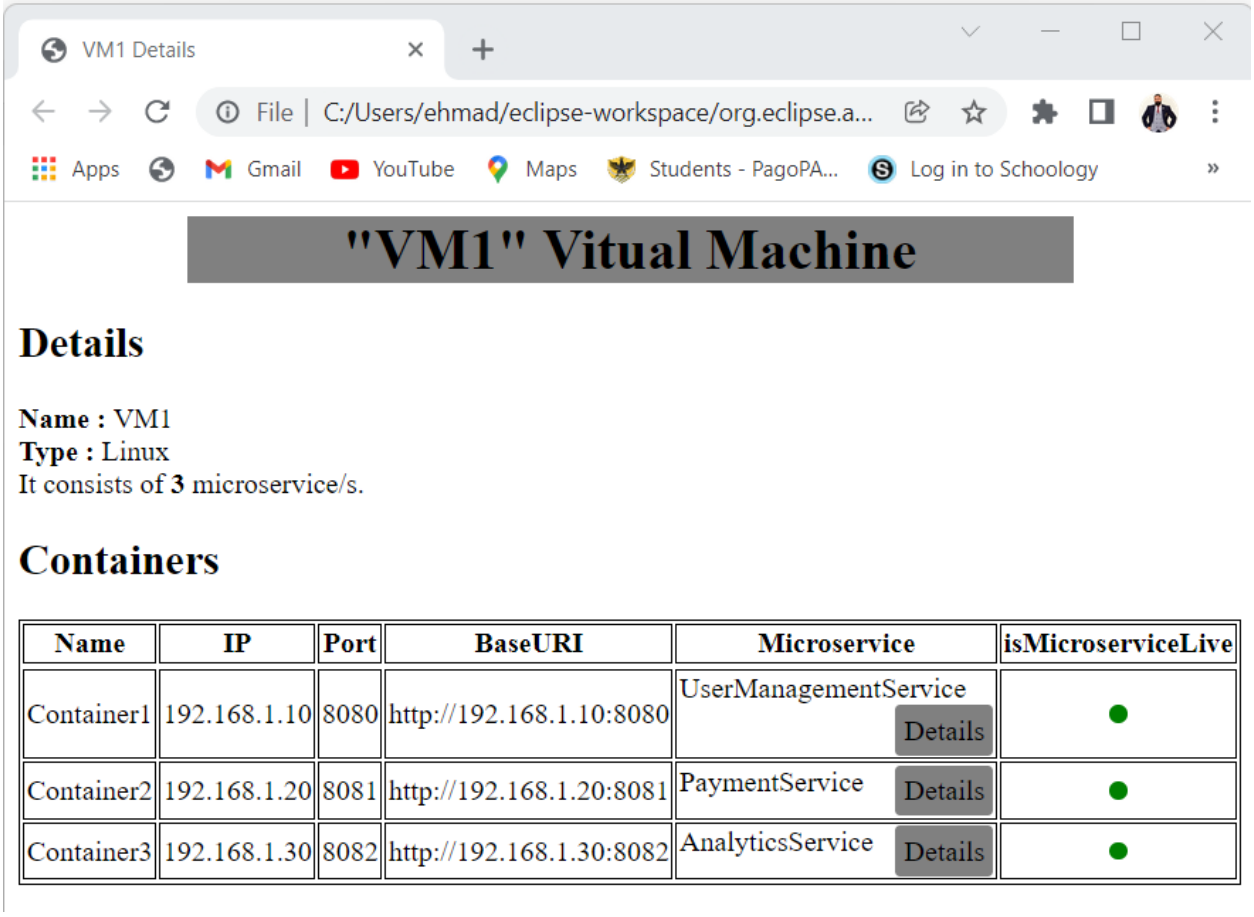
```
        </body>
    [/file]

[/template]
```

***Figure 40, 41, 42, 43 & 44.*** shows the output of the model to text transformation from the instance of "*Microservices*" named as "*BookingSystem.Prod*" using the code defined in Acceleo given above.



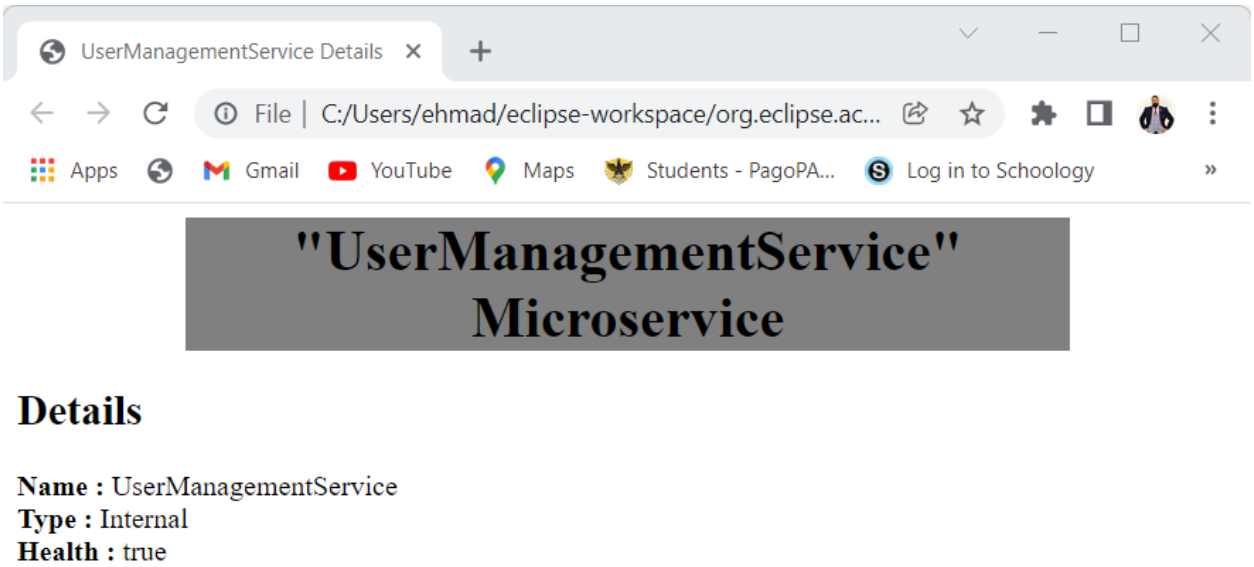

(***Figure 40.*** *Output of Transformed "BookingSystem.Prod" Environment)*

This figure also show the list of "*ServerNode*" available in "*Environment*".

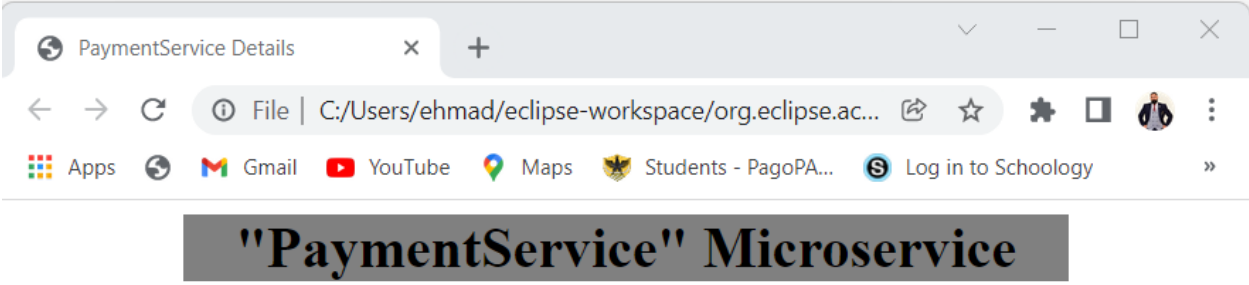



*(**Figure 41.** Output of Transformed "VM1" VirtualMachine)*

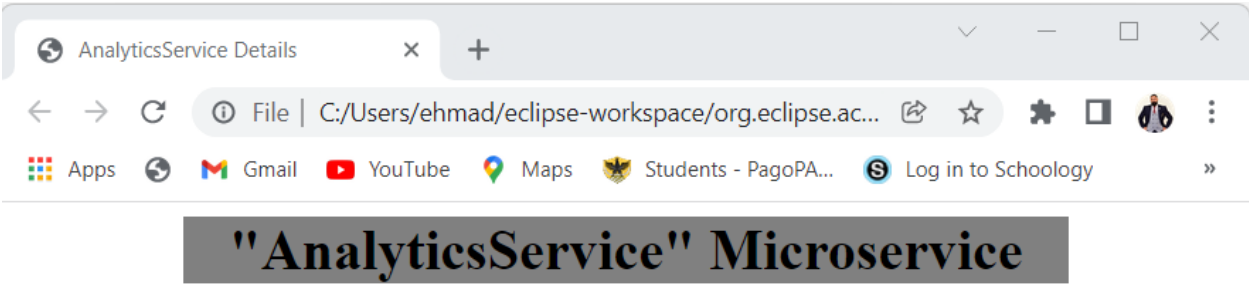
This figure also show the list of "*Container*" available in "*VirtualMachine*".




*(**Figure 42.** Output of Transformed "UserManagementService" Microservice)*




(***Figure 43.*** *Output of Transformed "PaymentService" Microservice)*




(***Figure 44.*** *Output of Transformed "AnalyticsService" Microservice)*

# Asignment4: Textual & Graphical Editor Definition using Xtext + Sirius

## Task A4.1: Textual Editor

Code given below shows the definition of the concrete syntax defined for the textual editor using Xtext. ***Figure 45.*** shows the example usage of the editor.

### Code

```
grammar org.xtext.microservices.mydsl.MyDsl with org.eclipse.xtext.common.Terminals

generate myDsl "http://www.xtext.org/microservices/mydsl/MyDsl"

EnvironmentModel returns EnvironmentModel:
	environment=Environment;  //only zero or one enviroment can be initialised

Environment returns Environment:
	 'environment' name=ID
	'{'
		('id' '=' id=STRING ';')?
		('name' '=' name=STRING ';')?
		('type' '=' Type=EnvironmentType ';')?
		('location' '=' name=STRING ';')?
		('serverNodes' '=' '{' nodes+=ServerNode (',' nodes+=ServerNode)* '}'
';')? //only zero or many serverNodes can be initialised
	'}';

enum EnvironmentType returns EnvironmentType:
			Development = 'Development' | Test = 'Test' | UAT = 'UAT' | Demo
= 'Demo' | Production = 'Production';


ServerNode returns ServerNode:
	'node' name=ID
	'{'
		('id' '=' id=STRING ';')?
		('name' '=' name=STRING ';')?
		('RAM' '=' name=STRING ';')?
		('Processor' '=' name=STRING ';')?
		('Disk' '=' name=STRING ';')?
		('virtualMachines' '=' '{' vms+=VirtualMachine (','
vms+=VirtualMachine)* '}' ';')?
	'}';

VirtualMachine returns VirtualMachine:
	'vm' name=ID
	'{'
		('id' '=' id=STRING ';')?
		('name' '=' name=STRING ';')?
		('type' '=' Type=VMType ';')?
		('containers' '=' '{' dockers+=Container (',' dockers+=Container)* '}'
';')?
```

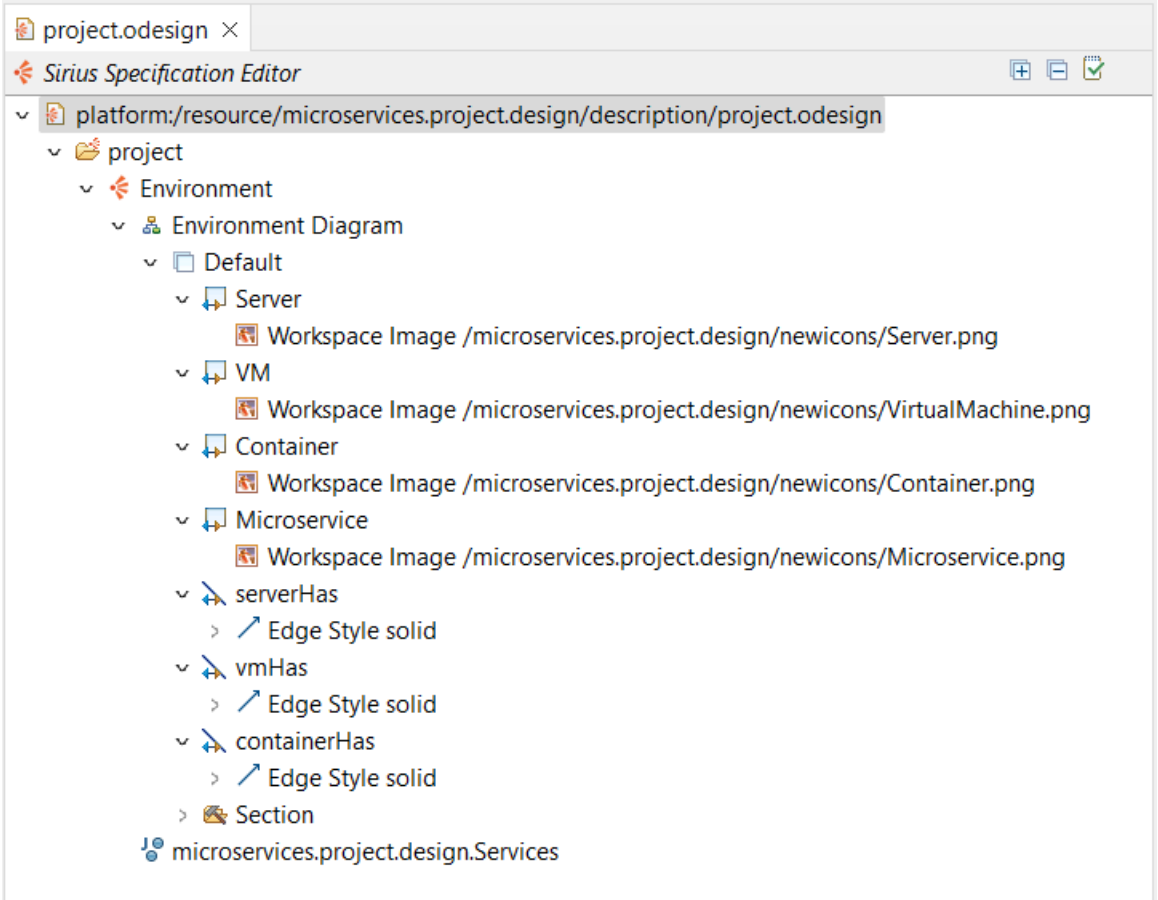
```
        '}';

enum VMType returns VMType:
                    Linux = 'Linux' | RedHat = 'RedHat' | Windows = 'Windows' |
CentOS = 'CentOS';

Container returns Container:
        'docker' name=ID
        '{'
                ('id' '=' id=STRING ';')?
                ('name' '=' name=STRING ';')?
                ('IP' '=' name=STRING ';')?
                ('Port' '=' Port=INT  ';')?
                ('Microservice' '=' Microservice=Microservice ';')? //only zero or one
microservice can be initialised inside one container
        '}';

Microservice returns Microservice:
        'service' name=ID
        '{'
                ('id' '=' id=STRING ';')?
                ('name' '=' name=STRING ';')?
                ('type' '=' Type=MicroserviceType ';')?
        '}';

enum MicroserviceType returns MicroserviceType:
                    Internal = 'Internal' | External = 'External';
```

```
BookingSystem.msa
environment BookingSystemProduction{
    id = "1"; name = "BookingSystem.Prod"; type = Production; location = "Rome";
    serverNodes = {
        node server1 {
            id = "1"; name = "Name"; RAM = "Name"; Processor = ""; Disk = "Name";
            virtualMachines = {
                vm vm1 {
                    id = "1"; name = "vm1"; type =  Linux;
                    containers = {
                        docker docker1 {
                            id = "1"; name = "docker1"; IP = "192.168.1.20"; Port = 8081;
                            Microservice = service PaymentService {
                                id = "1"; name = "Payment"; type = Internal;
                            };
                        },
                        docker docker2 {
                            id = "2"; name = "docker2"; IP = "192.168.1.21"; Port = 8082;
                            Microservice = service AnalyticsService {
                                id = "1"; name = "Analytics"; type = Internal;
                            };
                        }
                    };
                }
            };
        }
    };
}
```

(***Figure 45.*** *Textual Editor Example for Microservices DSL)*

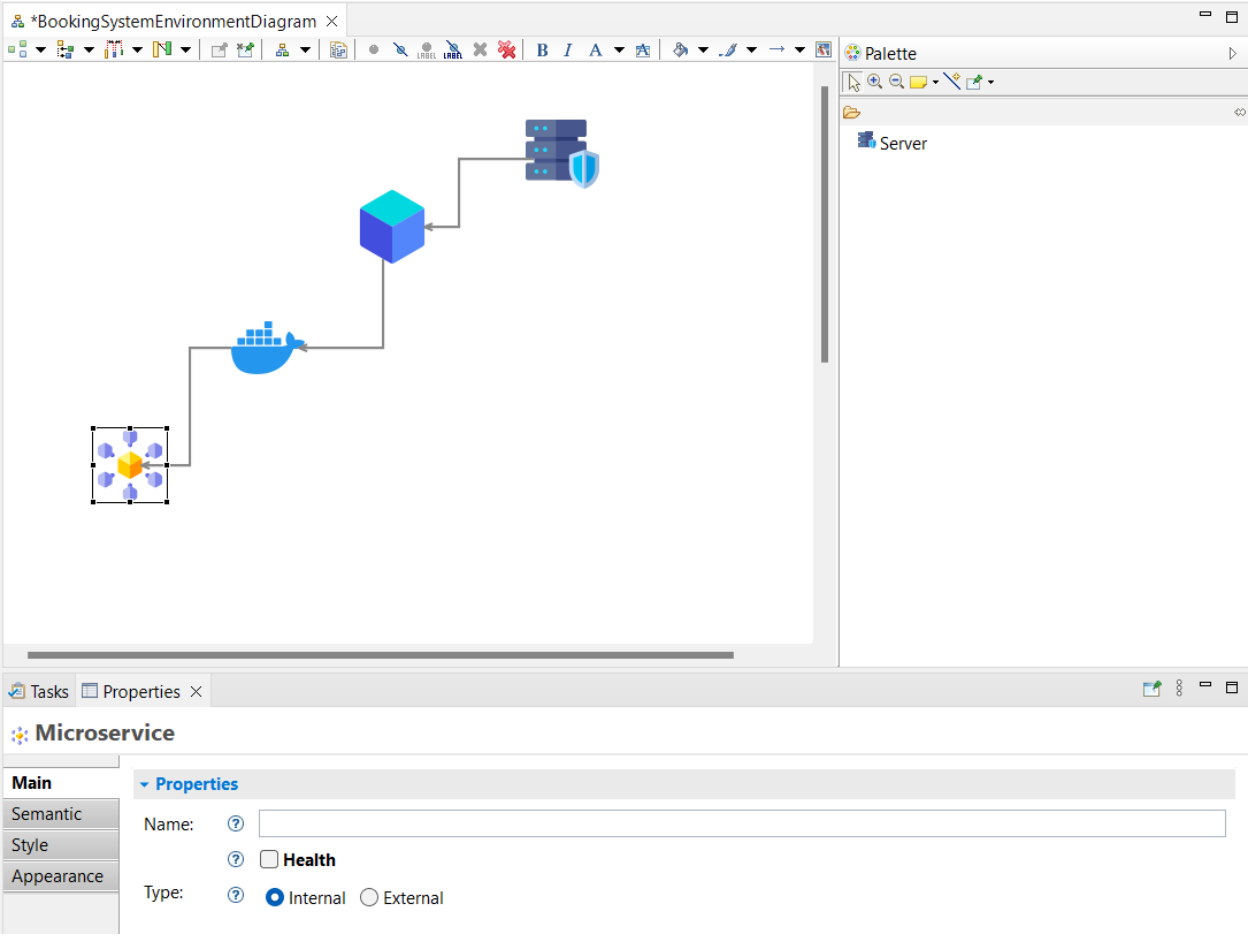
## Task A4.2: Graphical Editor

***Figure 46.*** shows the specifications defined for the graphical editor using Sirius. It has **node** representation**, edge** representation & section for **palette** items cover all the concepts of Microservices DSL.



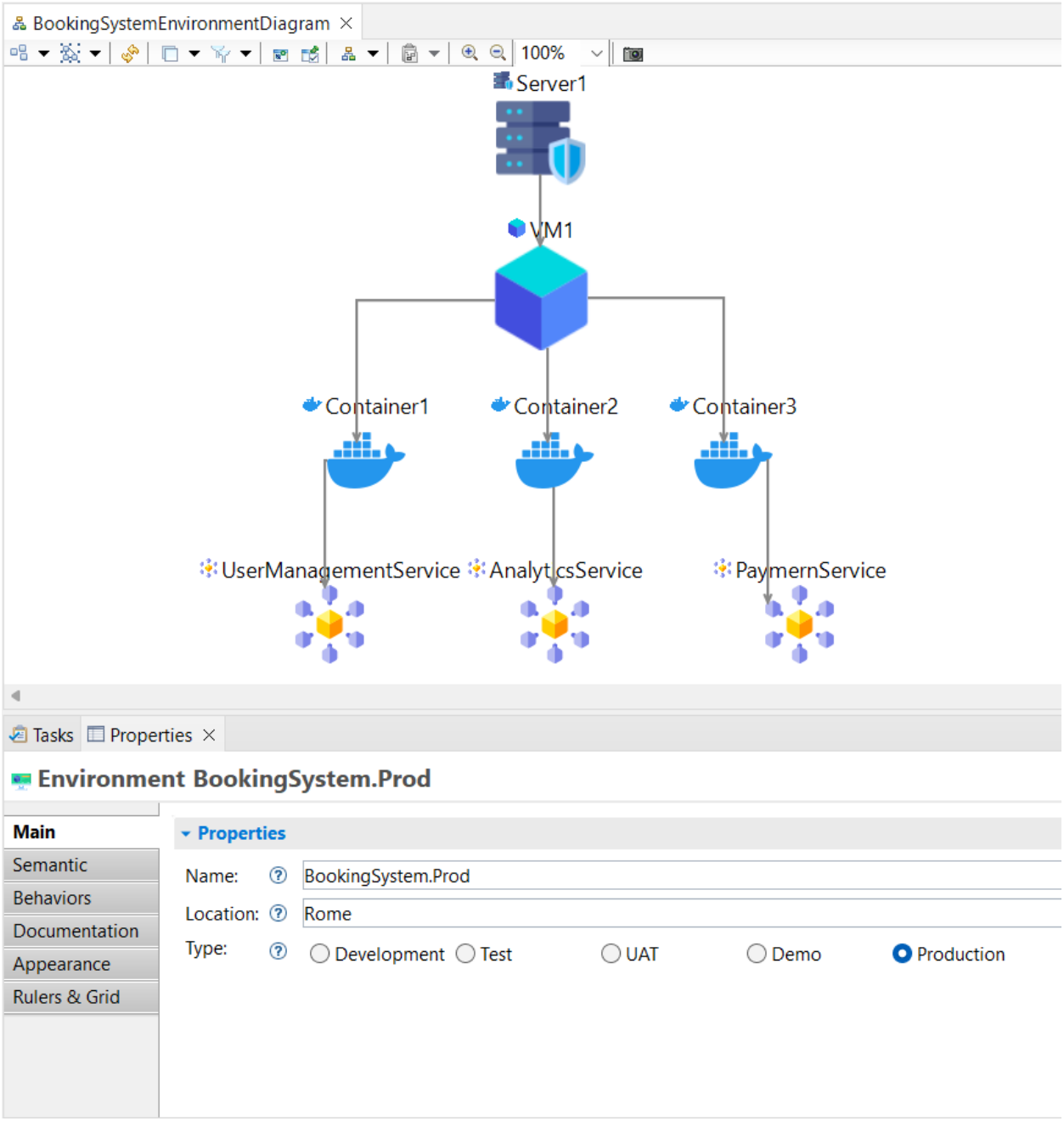

(***Figure 46.*** *Specifications for Microservices DSL)*

***Figure 47.*** shows the palette items where we can click and add "*ServerNode*" in the graphical editor. By double clicking, user can add "*VirtualMachine*" to the diagram and similar operation is required for adding "*Container*" under "*VirtualMachine*" and "*Microservice*" under "*Container*". Edges between the concepts are automatically drawn. User can enter additional details in the properties section of the selected node.




(***Figure 47.*** *Palette Operations for Microservices DSL)*

***Figure 48.*** shows an example of a "*BookingSystem.microservices*" instance created using the graphical editor defined above.




*(**Figure 45.** Graphical Editor Example for Microservices DSL)*