A Holistic, Innovative Framework for the Design, Development and Orchestration of 5G-ready Applications and Network Services over Sliced Programmable Infrastructure

DELIVERABLE D1.5

DEPLOYMENT AND RUNTIME POLICY METAMODEL

Due Date of Delivery:	M9 *Mx* (28/02/2018 *dd/mm/yyyy*)
Actual Date of Delivery:	05/03/2018 *dd/mm/yyyy*
Workpackage:	WP1 – MATILDA Reference Architecture, Conceptualization and Use Cases
Type of the Deliverable:	OTHER
Dissemination level:	PU
Editors:	UBITECH
Version:	1.0

Co-funded by the Horizon 2020 Framework Programme of the European Union

Call:
H2020-ICT-2016-2

Type of Action:
IA

Project Acronym:
MATILDA

Project ID:
761898

Duration:
30 months

Start Date:
01/06/2017

Project Coordinator:
Name: Franco Davoli
Phone: +39 010 353 2732
Fax: +39 010 353 2154
e-mail: franco.davoli@cnit.it

Technical Coordinator
Name: Panagiotis Gouvas
Phone: +30 216 5000 503
Fax: +30 216 5000 599
e-mail: pgouvas@ubitech.eu

List of the Authors	
CNIT	Consorzio Nazionale Interuniversitario per le Telecomunicazioni
Franco Davoli, Roberto Bruschi	
ATOS	ATOS Spain SA
Javier Melian	
INTRA	INTRASOFT INTERNATIONAL SA
Kostas Thivaios, Marios Logothetis	
UBITECH	GIOUMPITEK Meleti Schediasmos Ylopoiisi kai Polisi Ergon Pliroforikis EPE
Panagiotis Gouvas, Anastasios Zafeiropoulos, Eleni Fotopoulou, Thanos Xirofotos	
SUITE5	SUITE5 Data Intelligence Solutions
Fenareti Lampathaki, George Sideratos, Dimitris Panopoulos	
UPRC	UNIVERSITY OF PIRAEUS RESEARCH CENTER
Angeliki Alexiou, Eftychia-Asimina Vorila	

Disclaimer

The information, documentation and figures available in this deliverable are written by the MATILDA Consortium partners under EC co-financing (project H2020-ICT-761898) and do not necessarily reflect the view of the European Commission.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.

Copyright

The MATILDA Consortium consists of:

CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI

ATOS SPAIN SA (ATOS)

ERICSSON TELECOMUNICAZIONI (ERICSSON)

INTRASOFT INTERNATIONAL SA (INTRA)

COSMOTE KINITES TILEPIKOINONIES AE (COSM)

ORANGE ROMANIA SA (ORO)

EXXPERTSYSTEMS GMBH (EXXPERT)

GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH)

INTERNET INSTITUTE, COMMUNICATIONS SOLUTIONS AND CONSULTING LTD (ININ)

INCELLIGENT IDIOTIKI KEFALAIOUCHIKI ETAIREIA (INC)

SUITE5 DATA INTELLIGENCE SOLUTIONS LIMITED (SUIT5)

NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS" (NCSRD)

UNIVERSITY OF BRISTOL (UNIVBRIS)

AALTO-KORKEAKOULUSAATIO (AALTO)

UNIVERSITY OF PIRAEUS RESEARCH CENTER (UPRC)

ITALTEL SPA (ITL)

BIBA - BREMER INSTITUT FUER PRODUKTION UND LOGISTIK GMBH (BIBA).

Table of Contents

1 Executive Summary

This deliverable provides an overview of the policies metamodel defined within the MATILDA project. Policies in MATILDA are related to deployment and runtime policies and they are both associated with a developed application. Deployment policies regard mainly deployment constraints defined on behalf of application developers, while runtime policies regard set of expressions (event-condition-action rules) guiding operational decisions for optimal execution of the applications, considering the overall execution and infrastructural status context. Both policy types are of importance in the scope of the project, and their enforcement is tackled by different components in the MATILDA architectural approach.

Deployment policies denoted by software developers are included within the produced slice intent that is provided to telecom/infrastructure providers for the creation and management of the appropriate network slice and the management of the allocated resources. Runtime policies are denoted by service providers (e.g. cloud service providers, vertical industries application providers) and regard the real-time and dynamic management of the application control functions, including actions realised in the service mesh level as well as actions realised in the network slice level. For the latter, the set of actions that can be realised depends on the actions advertised on behalf of the communication service providers towards the application/service providers through a northbound API.

Under this perspective, a set of facets are conceptualized for the description of policies expressions, covering conditions and actions associated with orchestration components in the various layers. Such policies expressions are included within an overall policies descriptor, binded to an application. During deployment, the active policies are enforced leading to enhanced and more intelligent operation of the orchestration mechanisms. These facets along with the descriptor and indicative usage examples are detailed in this deliverable.

2 Introduction

2.1 Scope of the Deliverable

The main scope of this deliverable regards the conceptualization of the policy metamodel in MATILDA. Policies may consider deployment or runtime policies, each one of which aiming to serve set of objectives. Deployment policies are used for optimal placement of an application over a programmable infrastructure, taking into account set of defined constraints, mainly for resource allocation purposes. Runtime policies are used for dynamic enforcement of actions by the various orchestration mechanisms during the execution time of an application.

Deployment policies are denoted by software developers and made available in the form of a descriptor accompanying the overall application graph descriptor. Such policies are interpreted and lead to the production of the slice intent, that can be interpreted by the Slice Manager during initial deployment of an application. Taking advantage of optimisation mechanisms, an optimal (or close to optimal) deployment plan is produced, aiming to satisfy the imposed constraints. Runtime policies are denoted by service providers and software developers and made available in the form of a descriptor. Software developers, having detailed knowledge of the developed software, are able to suggest set of expressions (event/condition-action) that can be enforced during runtime and optimise the software operation. Such expressions are accompanying the software and may be exploited by service providers towards the definition of more advanced policies. For instance, a software developer-defined scaling policy based on a resources-usage metric may be slightly modified by a service provider through having a more holistic view of the overall infrastructure that this application is going to be deployed.

Under this perspective, in this deliverable, focus is given on the definition of the runtime policies metamodel consisted of the set of facets taken into account towards the definition of expressions as well as the overall descriptor incorporating the defined expressions. As already mentioned, with regards to the deployment policies, their description is included in the form of constraints in the application graph and slice intent metamodels, as detailed in D1.2 [3] and D1.4 [5].

It should be noted that -in addition to the metamodel defined in this section- within WP3, a formal language for the validation of the appropriate specification of policies expressions is going to be provided. Upon validation, the policy is going to be translated to the appropriate format in order to be easily interpretable by the runtime policies engine.

2.2 Structure of the Document

The structure of the document is as follows: in section 3, the architectural approach followed for runtime policies enforcement is provided, along with the exact positioning of the policies components within the MATILDA architectural framework; in section 4, the main set of facets considered for the specification of conditions and actions within the policies expressions are detailed, along with the specification of the overall runtime policies descriptor and the documentation of indicative examples regarding the way that defined policies can be translated to Drools (the rule-based expression format adopted and being interpretable by the policy engine); in section 5, overall conclusions and plans for usage of the defined metamodel in the various WPs are provided.

3 Policies Management Architectural Approach

3.1 Policies Management Positioning in the MATILDA Architecture

Polices management and enforcement in MATILDA is realised over the deployed application graphs over the instantiated network slices. Based on monitoring information collected through streams provided via the service mesh data and control plane, as well as the communication service provider network monitoring infrastructure, inference mechanisms are applied over the set of defined expressions consisting a policy. Various metrics and functions are considered for both the conditions and actions part per policy, including metrics related to application components configuration, resources usage and allocation, service mesh functionality, component characterisation and network links monitoring and management.

It should be noted that policies management of the activated network services for supporting a deployed application graph is considered as part of relevant frameworks developed within various NFV Orchestrators (NFVOs) and, thus, it will be also tackled in terms of adoption and potential extension of a relevant NFVO. However, policies enforcement over the deployed application graph is considering actions requested by the network management systems of a communication service provider, as they are provided through northbound interfaces to the application/services providers.

The positioning of the Runtime Policies Manager within the MATILDA architecture, as detailed at D1.1 [1] is depicted at Figure 3.1 (the main components and interactions denoted in blue colour). Policies formulation is realized in the Policies Editor that is accessible by application/service providers. The outcome of a policy formulation is the descriptor provided in section 4.2. The descriptor is going to be validated in terms of correctness and compatibility with a set of formal rules, as they are going to be included in a policies Domain Specific Language (DSL) that is under specification in the framework of WP3 activities. A validated policy description is going to be translated to Drools [2] and loaded to the Policy Manager (Rule Engine) for supporting runtime policies enforcement upon the deployment of an application graph. Monitoring streams providing data to be evaluated by the rule engine are sent from monitoring mechanisms being active at the service mesh level (e.g. telemetry data), the monitoring infrastructure of the communication service provider, as well as the analytics engine (e.g. real-time analytic processes results).

Runtime policies management mechanisms in MATILDA are going to provide policies enforcement over the deployed application graphs following a continuous match-resolve-act approach. Specifically, the match phase regards the mapping of the set of applied rules that are satisfied based on the data streams coming from the monitoring mechanisms, the resolve phase regards the process of conflict resolution -if any- among the satisfied rules taking into account the pre-defined salience of each rule, while the act phase regards the provision of a set of suggested actions to the various orchestration components through the Execution Manager. Policies enforcement is going to be realised through a rule-based framework that attempts to derive execution instructions based on the current set of data and the active rules; rules associated with the deployed service graphs over at each point of time. Specifically, Drools is going to be used that is a Business Rules Management System (BRMS) solution [2].

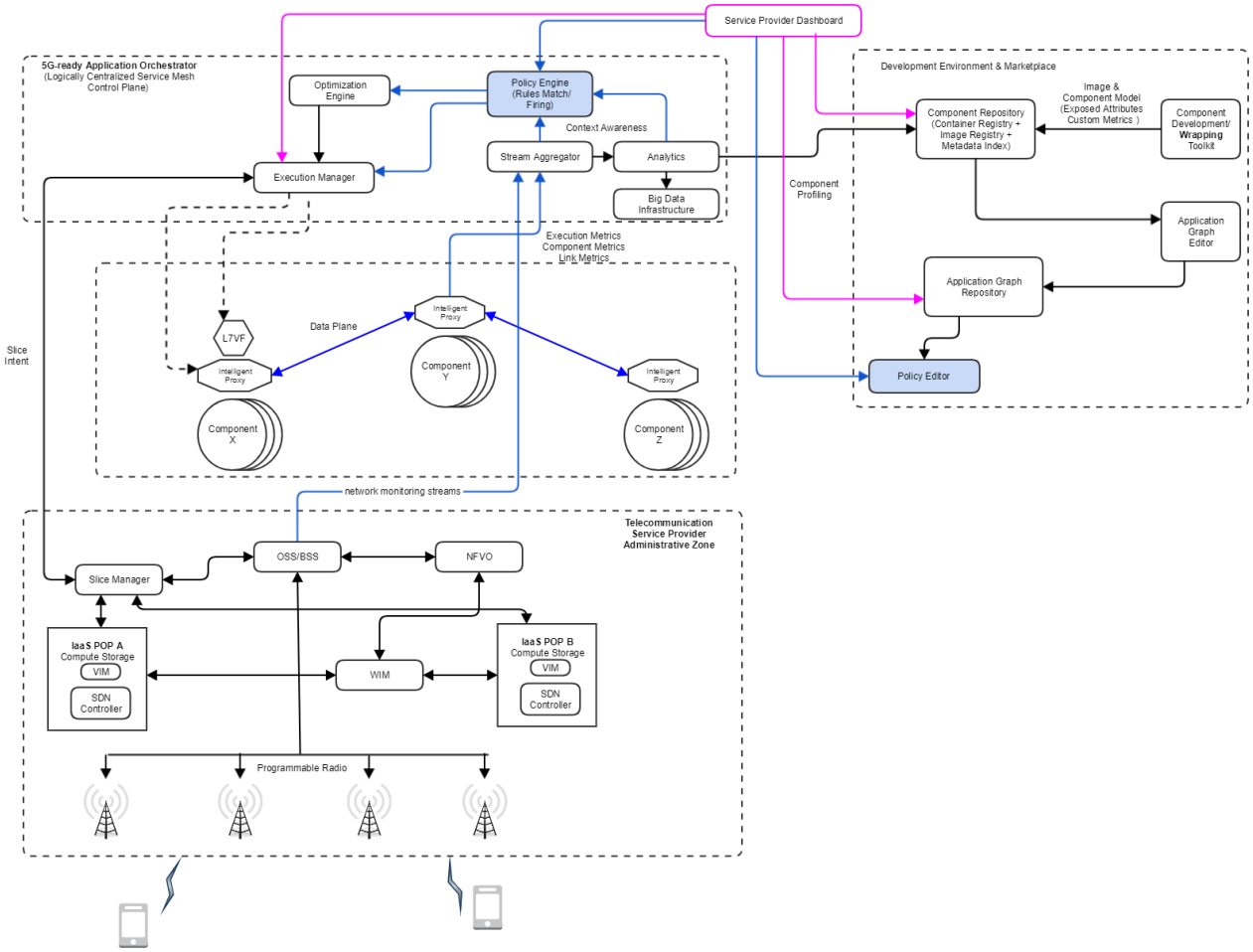

Figure 3.1 Policies Management Framework within Matilda

During the operational state of the service graph a set of independent, monitoring streams are aggregated by a monitoring server. As it is depicted, these streams relate a) to measurements that are provided by the telco provider per se (e.g. instrumentation of VIMs) and b) to measurements that are pulled by the Service Graph orchestrator. These measurements are directed to a Complex Event Processing (a.k.a. CEP) engine which is responsible to execute time-window-based operations in the form of rules. The set of rules that are active per stream-evaluation is addressed as Policy. Although many policies may be defined for one graph only one can be active.

Based on the rules' execution the orchestrator may fire some actions. These actions are classified in two categories. The first type of action can be realized by the telco provider itself. As depicted on Figure 1, the telco provider exposes a northbound interface to the service orchestrator which 'proxies' the functional capabilities of telco programmability. These capabilities span from allocation of virtualized resources (at the data center or at the edge level) to the provisioning of specific quality class on the network traffic between UEs and component interfaces. On the other hand, the second class of actions that will be supported are telco-agnostic. According to the Service Mesh paradigm each component that participates in the graph is programmable through a specific proxy. This proxy can handle multiple commands such as Layer-7 balancing etc. The type of conditions and actions that will be

supported are describe on Deliverable X.X[ref]. After examining the flow of the metamodels' usage we will delve into the details of the Slice Intent and the Slice metamodel respectively.

Based on the Drools engine, the overall policies enforcement framework consists of the working memory; facts based on the provided data, the production memory; set of defined rules, and an inference engine that supports reasoning and conflict resolution over the provided set of facts and rules as well as triggering of the appropriate actions (Figure 3.2). Data is fed to the working memory through the various monitoring mechanisms. The production memory is also fed by policies associated with the deployed application graphs, as provided through the Policies Editor – the editor made available to application/service providers for policies definition. The Policy Engine dynamically handles and converts the collected data to working memory facts. Such facts can then be matched with already defined rules on the active policies. An application graph may be associated with a set of policies, however only one can be active during its deployment and execution time.

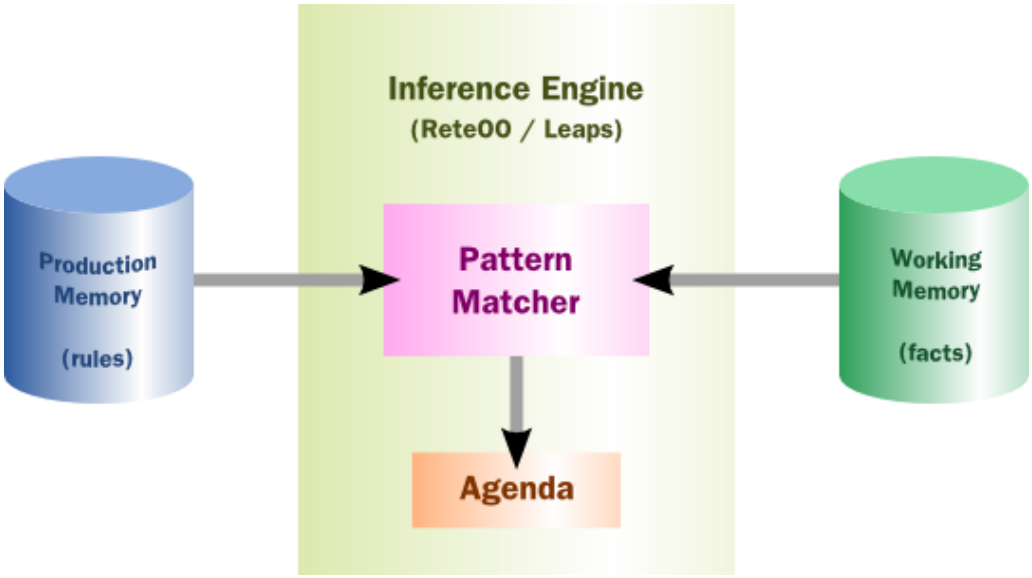

Figure 3.2 High-level View of a Production Rule System [2]

3.2 Policies Metamodel Relationship with MATILDA Metamodels

Before delving into the details of the Policy Metamodel, we provide an overview of the positioning and usage of this model with regards with the set of developed metamodels. The overall metamodel usage is graphically depicted on Figure 3.3 where the basic architectural components of the MATILDA framework and their relationship with the various models is provided.

An application graph placement flow starts with the selection of a vertical application that has to be deployed and supported by a communication service provider. As clearly stated on the architectural deliverable (D1.1[[1]]), MATILDA will support state of the art distributed applications. Therefore, a vertical application in MATILDA consists of multiple components that can be deployed on top of programmable infrastructure. These components when combined to each other they formulate a direct acyclic graph (a.k.a. DAG) which represents a vertical application. In other words, a vertical application is represented by a graph where

components are vertexes and edges are the component-links. For the sake of clarity, MATILDA imposed a formal metamodel of this graph which is analysed in Deliverable 1.2 [3].

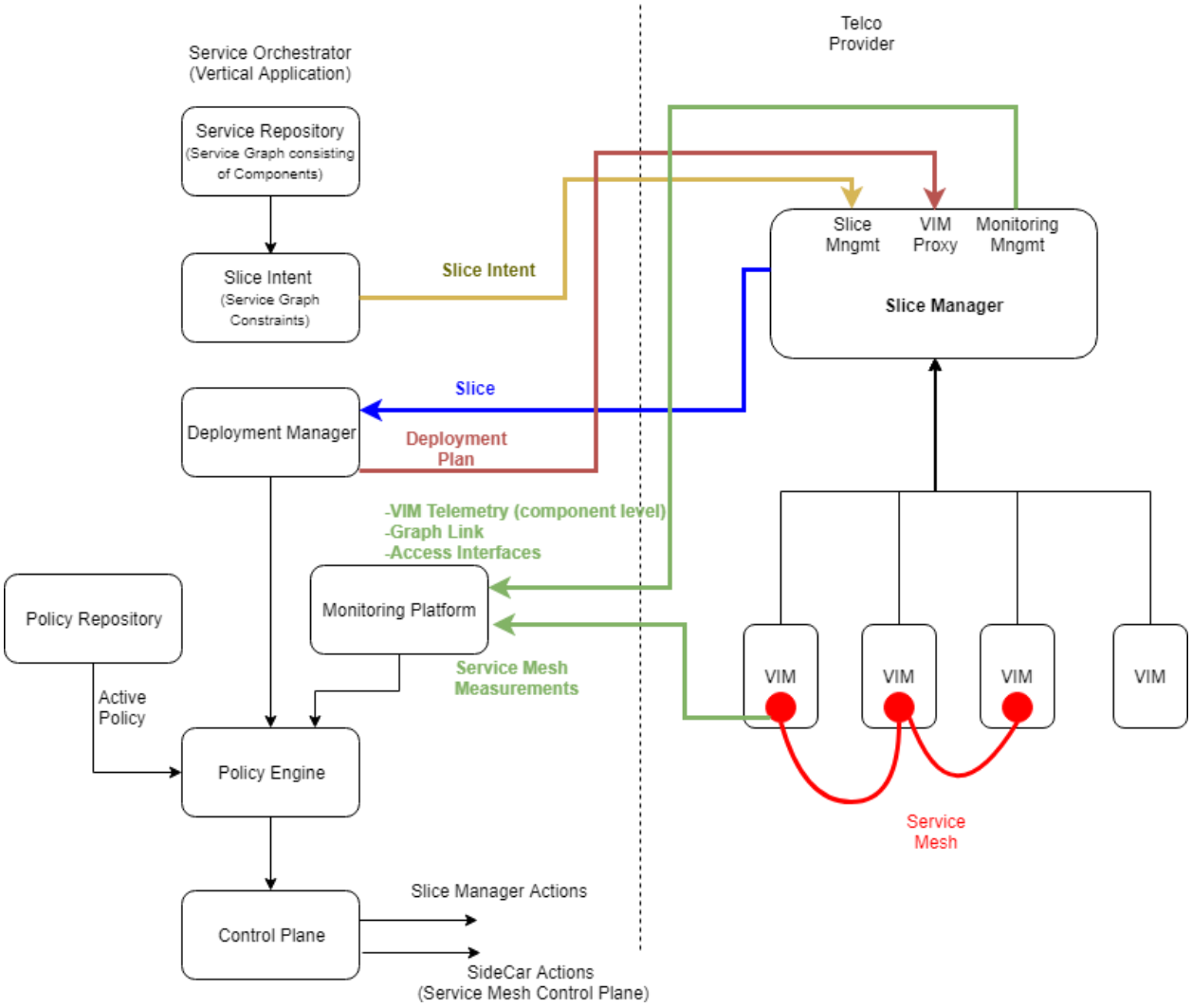

Figure 3.3 Usage of MATILDA Metamodels

The Service Repository (see Figure 3.3) contains all the instances of the service graphs that have been registered. The flow initiates by the selection of service graph by an application/service provider. As depicted in Figure 3.2, there are two distinct administrative zones. On the left part resides the administrative zone of the vertical application/service orchestrator while on the right part the administrative zone of the telco (communication service) provider. Hence, each administrative zone contains its own orchestration entity with clear responsibilities. The orchestration entity on the left is responsible to instantiate a vertical application that meets specific requirements on the virtualized resources that will be provided by the orchestration mechanism of the right.

Taking under consideration the scope of the two orchestrators, we can easily infer that the Application/Service Orchestrator and the Telco Orchestration mechanisms follow a request/response pattern according to which the Service Orchestrator asks for a specific "setup" that is capable to satisfy some characteristics/requirements and the telco provider responds with the details of the environment that has to be used for the appropriate setup.

The first request is addressed as Slice Intent while the latter as the offered Slice. Both specifications are provided in the relevant metamodels, as detailed in D1.4 [5].

As a third step, the telco provider receives the slice intent and tries to find/create a proper setup that will satisfy the set of denoted requirements in step 2. The solution that satisfies the constraints will be announced back to the Service Graph Orchestrator. The solution will be an instance of the Slice Metamodel. Part of the requirements request during deployment or runtime may regard the activation or configuration of network services, able to provide the requested network functionalities. Such services are provided by a NFVO, while the representation is realised based on the metamodel defined in D1.3 [4].

4 Runtime Policies Metamodel

4.1 Runtime Policies Facets

In MATILDA, each runtime policy consists of a set of expressions indicating the conditions over which one or more actions are triggered. Conditions as well as actions may be related with various stakeholders and mechanisms. Following, short reference to the basic set of conditions and actions supported is provided. However, it should be noted that the provided list is indicative and under continuous evaluation and extension during the project lifetime, taking into account the requirements of the various demonstrators to be realised as well as feedback collected via the design and development of the orchestration mechanisms.

Regarding the **set of metrics to be included in the conditions part**, a high-level view is depicted at Figure 4.1. Metrics can be associated with functionality managed in the service mesh level, configuration options of each component, resources usage metrics, component profiling information, application graph metrics focusing mainly on virtual links QoS characteristics as well as overall resources usage metrics of the instantiated network slices.

At the **service mesh level**, the metrics are monitored through the service mesh data plane and may be related with any of the supported service mesh functions (e.g. load balancing, authentication/authorization, health checking). Actually, each designed service may be associated with a set of metrics that can be evaluated during runtime in order to guide potential decisions, targeting mainly at the optimal operation of the service mesh functions. Indicative metrics regard the number of workers managed by a load balancer, the incoming/outgoing traffic measured through a telemetry service etc.

Component configuration metrics are related to metrics denoted on behalf of the application developer as component specific metrics and made available through the application graph descriptor. Such metrics can be also monitored during runtime, given that monitoring mechanisms are implemented in the provided software and the metrics are exposed as component configuration metrics. The variety of such metrics is huge, since they regard implementation business logic of each application component. Indicative metrics regard the number of served users, active sessions, average HTTP response time etc.

Resource usage metrics regard the average/min/max consumption of resources from the deployed component image over the virtualized infrastructure. Such metrics are usually monitored through the resource management entity of the communication service provider. In MATILDA, resource usage metrics are going to be provided by the communication service provider through a well-defined northbound interface. Indicative metrics include CPU usage, memory consumption, allocated storage space, incoming/outgoing traffic rate etc.

Component characterization metrics regard the profiling result of a component and its mapping with one or more characteristics. Profiling of a component may regard different aspects, including -among others- characterization in terms of resource usage, operational and reliability status, energy efficiency, security aspects etc. Indicative characteristics include malicious, energy efficient, CPU intensive, network traffic intensive etc.

With regards to **network link monitoring metrics**, a set of QoS characteristics are considered, as they can be mainly provided through the monitoring infrastructure of communication service providers. Such metrics are provided through network monitoring

functions supported by the communication service provider. These functions can be part of a VNF or developed network monitoring mechanisms (e.g. providing network monitoring data to an OSS/BSS system). Once again, the provided monitoring data is going to be made available through well-defined northbound interfaces. Indicative metrics include end to end delay, packet loss, throughput etc.

Finally, a set of metrics may also regard **network slice management metrics**, applied mainly to actions related to dynamic management of network slice resources. These metrics will be aggregate values of the overall resources consumption/usage metrics of the set of application graphs and network services served through a network slice. Indicative metrics include overall network slice resource (CPU, memory) consumption etc.

With regards to the **actions part of a policy definition**, a set of actions are defined based on the targeted entity to apply them, namely application components, application graphs and network monitoring and management systems of communication service providers. Prior to detailing the set of potential actions, it should be noted that no specific binding between conditions and actions exists, however some combinations may not be available, taking into account that there is no related business logic.

Application component actions may be associated with a service mesh functionality (e.g. change a load balancing policy, spawn or deprovision a number of VMs), a change in the configuration of custom metrics of a component (e.g. change in the transcoding quality level), a change in the resource allocated to the component (e.g. vertical scalability, migration actions) etc.

Application graph actions are mainly service-mesh oriented and regard the application of a function in the service mesh control plane (e.g. activate RBAC mechanisms, change traffic management policy). Such actions are going to be triggered mainly by consuming information collected via the service mesh data plane from various application components associated proxies.

Network monitoring and management actions regard requests towards the OSS/BSS systems of communication services providers and are based on the functions disseminated by them through a northbound interface. Such actions are going to be triggered taking into account the type of the action as well as the set of mechanisms supported by a communication service provider (as they are going to be disseminated through a northbound interface). Indicative actions regard the activation of an end to end monitoring mechanisms (e.g. for end to end delay among two connection endpoints), deployment of a network function or service, establishment of a VPN etc.

As already stated, the list of the aforementioned conditions and actions regard an elaborated but not complete set of the potential conditions and actions that may take part in policies expressions. This list is going to be open and extensible during the lifetime of the project, based on the requirements and needs of the various demonstration activities that are going to take place.

Figure 4.1 Policies Conditions High Level View

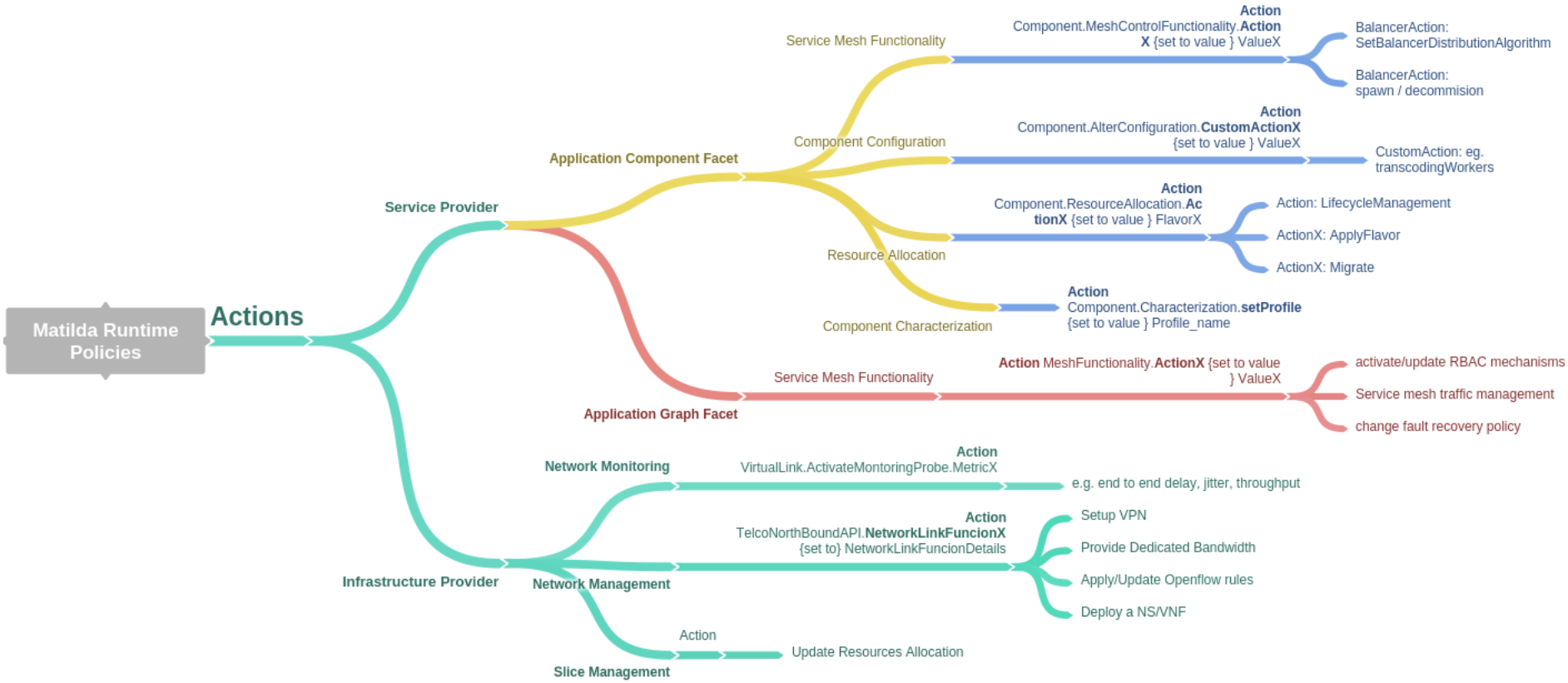

Figure 4.2 Policies Actions High Level View

4.2 Runtime Policies Descriptor

In this section, the runtime policies descriptor in YAML format, along with an example with an instance of such a descriptor based on an indicative policy is provided.

Runtime Policies Descriptor:

```
---
$schema: "http://json-schema.org/draft-04/schema#"
title: "Policy Descriptor Schema"
version: 0.1
description: "The core schema for Matilda policy descriptors."

##
## Some definitions used later on.
##
definitions:
  time_units:
    enum:
      - "s"    # seconds
      - "m"    # minutes
      - "h"    # hours
      - "d"    # days
  aggregation_function:
    enum:
      - "avg"    # average
      - "min"    # min
      - "max"    # max
  operator:
    enum:
      - "less"    # less
      - "equal"    # equal
      - "greater"    # greater
  input:
    enum:
      - "number"    # number
      - "select"    # select
  logicalOperator:
    enum:
      - "AND"    # AND
      - "OR"    # OR
```

```
    profile_tag:
      enum:
        - "under_attack"    # under_attack
        - "mis_performed"    # mis_performed
        - "low_energy"    # mis_performed
        - "Green"    # mis_performed
    action_type:
      enum:
        - "MeshActionType"    # MeshActionType
        - "MonitoringAction"    # MonitoringAction
        -   "NetworkMechanismType"        #   NetworkMechanismType   activated   by
TelcoNorthBoundAPI
        - "InfrastructureType"    # InfrastructureType
        - "Orchestration"    # OrchestrationAction.
        - "LifecycleManagement"    # LifecycleManagementAction
        - "AlterConfiguration"    # AlterConfigurationAction
        - "ProfileType"    # Add/remove a new profile
        - "Log"    # Log
    action_description:
      enum:
        - "ApplyFlavour"    # InfrastructureType
        - "Migrate"    # InfrastructureType
        - "LoadBalancer.setBalancingAlgorithm"    # MeshActionType
        - "Spawn"    # MeshActionType
        - "Deprovision"    # MeshActionType
        - "ChangeFaultRecoveryPolicy"    # MeshActionType
        - "ApplyRBACKMechanism"    # MeshActionType
        - "start"    # LifecycleManagementAction
        - "stop"    # LifecycleManagementAction
        - "restart"    # LifecycleManagementAction
        - "EndToEndDelay"    # MonitoringAction
        - "Jitter"    # MonitoringAction
        - "Throughput"    # MonitoringAction
        - "ProvideDedicatedBandwidth"    # NetworkMechanismType
        - "SetupVPN"    # NetworkMechanismType
        - "DeployNS"    # NetworkMechanismType
        - "DeployVPN"    # NetworkMechanismType
        - "setProfile"    # ProfileType
    expression_type:
```

```
      enum:
        - "ResourceUsageMetricType"    # ResourceUsageMetricType
        - "MeshMetricType"    # MeshMetricType
        - "QoSMetricType"    # QoSMetricType.
        - "CustomMetricType"    # CustomMetricType.
    expression_description:
      enum:
        - "TotalMemory"    # ResourceUsageMetricType
        - "CPULoadUtilization"    # ResourceUsageMetricType
        - "VCPU"    # ResourceUsageMetricType
        - "RAMUtilization"    # ResourceUsageMetricType
        - "hasProfile"    # ProfileType
        - "EndToEndDelay"    # QoSMetricType
        -          "Security_TotalConnectionsIgnored"                                  #
Security_TotalConnectionsIgnored
        - "Telemetry_InTraffic"    # MeshMetricType
    expression:
      type: "object"
      properties:
        id:
         description: "The name of the expression parameter. The name has to be
supported by the service platform or the FSM."
         type: "string"
        field_0:
         description: "The field name of the expression parameter."
         type: "string"
        field_1:
         description: "A condition type"
         $ref: "#/definitions/expression_type"
        field_2:
         description: "A condition type"
         $ref: "#/definitions/expression_description"
        type:
         description: "The type of the parameter."
         type: "string"
        input:
         description: "The input type of the parameter."
         $ref: "#/definitions/input"
        operator:
```

```
          description: "The operator of the expression."
          $ref: "#/definitions/operator"
        value:
          description: "The threshold value of the parameter."
          type: "string"
      required:
        - id
        - field_0
        - field_1
        - field_2
        - type
        - input
        - operator
        - value
    fullExpression:
      description: "A set of expressions bind between them with logical operators."
      type: "object"
      properties:
        condition:
          description: "The operator AND / OR."
          $ref: "#/definitions/logicalOperator"
        rules:
          description: "The set of expressions."
          type: "array"
          items:
            description: "An FSM object of this VNF. FSMs are always Docker containers."
            $ref: "#/definitions/expression"
      required:
        - condition
        - rules
    setOfExpressions:
      description: "An expression with the optional condition."
      type: "object"
      properties:
        rules:
          description: "The set of expressions."
          type: "array"
```

```
        items:
          description: "An FSM object of this VNF. FSMs are always Docker containers."
          $ref: "#/definitions/fullExpression"
      required:
        - rules


  ##
  ## The actual document description.
  ##
  type: "object"
  properties:
    descriptor_schema:
      description: "Reference to the schema corresponding to the descriptor (e.g., URL or local path)."
      type: "string"
    name:
      description: "The name of the policy description."
      type: "string"
      pattern: "^[a-z0-9\\-_.]+$"
    description:
      description: "A longer description of the policy."
      type: "string"
    policyRules:
      description: "A list of Policy rules  used to compose this Policy."
      type: "array"
      items:
        description: "A rule of the current policy"
        type: "object"
        properties:
          name:
            description: "The name of the policy rule."
            type: "string"
          salience:
            description: "The salience of the policy rule."
            type: "number"
          inertia:
            description: "The inertia period of the policy rule."
            type: "object"
```

```
            properties:
              value:
                description: "The duration value of the inertia period."
                type: "number"
              duration_unit:
                description: "The unit of the duration."
                ref: "#/definitions/time_units"
            required:
              - value
              - duration_unit
          duration:
            description: "The duration the condition has to be met before an
event is fired."
            type: "object"
            properties:
              value:
                description: "The duration value."
                type: "number"
              duration_unit:
                description: "The unit of the duration."
                ref: "#/definitions/time_units"
            required:
              - value
              - duration_unit
          aggregation:
            description: "The unit of the duration."
            ref: "#/definitions/aggregation_function"
          conditions:
            description: "The set of conditions, that must be met to fire the
event."
            ref: "#/definitions/setOfExpressions"
          actions:
            description: "A list of notifications that are fired when the
condition is met."
            type: "array"
            items:
              type: "object"
              properties:
                action_type:
                  description: "The type of the action that is send to the
```

```
message bus."
                  ref: "#/definitions/action_type"
                name:
                  description: "The description of the action"
                  ref: "#/definitions/action_description"
                value:
                  description: "The value of the action"
                  type: "string"
                details:
                  description: "The details of the action. Here are presented
custom actions"
                  type: "string"
                profile_tag:
                  description: "A profile tag"
                  ref: "#/definitions/profile_tag"
                target:
                  description: "The component to apply the action"
                  type: "string"
                stability_period:
                  description: "The inertia period of the policy rule."
                  type: "object"
                  properties:
                    value:
                      description: "The duration value of the inertia period."
                      type: "number"
                    duration_unit:
                      description: "The unit of the duration."
                      ref: "#/definitions/time_units"
                  required:
                    - value
                    - duration_unit
              required:
                - action_type
                - target
        required:
          - name
          - duration
          - aggregation
          - conditions
```

```
          - actions
      uniqueItems: true
      minItems: 1
  required:
    - descriptor_schema
    - name
    - policyRules
  additionalProperties: false
```

Runtime Policies Descriptor Instance:

```
  # default YAML description of an policy example



  ---
  descriptor_schema:                "https://gitlab.com/matilda-project/matilda-
metamodels/tree/master/policy-metamodel/policy-schema.yml"


  name: "samplepolicyv1"


  policyRules:
    - name: "highResourcesUtilization"
      salience: 1
      inertia:
        value: 30
        duration_unit: "m"
      duration:
        value: 10
        duration_unit: "m"
      aggregation : "avg"
      conditions:
        condition: AND
        rules:
         - id: componentX.ResourceUsageMetricType.CPULoadUtilization
           field_0: componentX
           field_1: ResourceUsageMetricType
           field_2: CPULoadUtilization
           type: double
           input: number
```

```
        operator: greater
        value: '80'
    actions:
     - action_type: "MeshActionType"
       name: "Spawn"
       value: "2"
       target: "componentX"
  - name: "badQoS"
    salience: 1
    inertia:
      value: 30
      duration_unit: "m"
    duration:
      value: 10
      duration_unit: "m"
    aggregation : "avg"
    conditions:
      condition: AND
      rules:
       - id: virtualLinkX.QoSMetricType.EndToEndDelay
         field_0: virtualLinkX
         field_1: QoSMetricType
         field_2: EndToEndDelay
         type: double
         input: number
         operator: greater
         value: '40'
       - id: componentX.MeshMetricType.Telemetry_InTraffic
         field_0: componentX
         field_1: MeshMetricType
         field_2: Telemetry_InTraffic
         type: integer
         input: select
         operator: greater
         value: '100'
    actions:
     - action_type: "NetworkMechanismType"
       name: "ProvideDedicatedBandwidth"
```

```
        value: "200"
        target: "virtualLinkX"
      - action_type: "Log"
        value: "Provide Dedicated Bandwith to virtualLinkX"
        target: "virtualLinkX"
  - name: "possibleSecurityAttack"
    duration:
      value: 10
      duration_unit: "m"
    aggregation : "avg"
    conditions:
      condition: AND
      rules:
        - id: componentX.MeshMetricType.Security_TotalConnectionsIgnored
          field_0: componentX
          field_1: MeshMetricType
          field_2: Security_TotalConnectionsIgnored
          type: double
          input: number
          operator: greater
          value: '5'
    actions:
     - action_type: "Profile"
       name: "setProfile"
       value: "under_attack"
       target: "componentX"
  - name: "applyFlavor"
    duration:
      value: 10
      duration_unit: "m"
    aggregation : "avg"
    conditions:
      condition: AND
      rules:
        - id: componentX.CustomMetricType.dbResponseTime
          field_0: componentX
          field_1: CustomMetricType
          field_2: dbResponseTime
```

```
            type: double
            input: number
            operator: greater
            value: '100'
      actions:
       - action_type: "InfrastructureType"
         name: "ApplyFlavour"
         value: "3"
         target: "componentX"
```

4.3 Rule-based Expressions

Upon the specification and validation of a policy descriptor, the policy expressions are going to be translated to Drools and imported in the policy engine. Following, indicative examples with policies denoted in the Drools format are provided.

```
  rule "highResourcesUtilization"
  when
      $tot0 := java.lang.Double( $tot0 >=80 ) from accumulate(
      $m0 := ComponentResourceUsageMetric( componentid== "componentX" &&
resourceUsageMetricType == ResourceUsageMetricType.CPULoadUtilization ) over
window:time(1m)from entry-point "MonitoringStream" ,
          average( $m0.getValue() )  )
  then
      insertLogical( new
ComponentMeshAction("componentX",MeshActionType.SPAWN,"2"));
end
```

```
  rule "badQoS"
  when
      (
      $tot0 := java.lang.Double( $tot0 >=40 ) from accumulate(
      $m0 := GraphQoSMetric( virtualLinkid== "virtualLinkX" && qoSMetricType ==
QoSMetricType.EndToEndDelay ) over window:time(1m)from entry-point
"MonitoringStream" ,
          average( $m0.getValue() )  ) and
      $tot1 := java.lang.Double( $tot1 >=100 ) from accumulate(
      $m1 := ComponentMeshMetric( componentid== "componentX" && meshMetricType==
MeshMetricType.Telemetry_InTraffic ) over window:time(1m)from entry-point
"MonitoringStream" ,
          average( $m1.getValue() )  ) )
  then
      insertLogical( new
NetworkManagementAction("virtualLinkX",NetworkLinkFunction.ProvideDedicatedBandw
idth,"200"));
  end
```

```
  rule "possibleSecurityAttack"
  when
      $tot0 := java.lang.Double( $tot0 >=5 ) from accumulate(
      $m0 := ComponentMeshMetric( componentid== "componentX" && meshMetricType
== MeshMetricType.Security_TotalConnectionsIgnored ) over window:time(1m)from
```

```
entry-point "MonitoringStream" ,
          average( $m0.getValue() )  )
  then
      insertLogical( new ComponentTagAction("componentX",TagType.UNDER_ATTACK));
  end
```

```
  rule "applyFlavor"
  when
      $tot0 := java.lang.Double( $tot0 >=100 ) from accumulate(
      $m0 := ComponentCustomMetric( componentid== "componentX" && metricName ==
"dbResponseTime" ) over window:time(1m)from entry-point "MonitoringStream" ,
          average( $m0.getValue() )  )
  then
      insertLogical( new
ComponentResourceAllocationAction("componentX",ResourceAllocationType.APPLY_FLAV
OR,"3"));
  end
```

5 Conclusions

This deliverable provides the main foundation of the facets that are going to be included in the runtime policies description in MATILDA, along with the format of the policies descriptor, as well as some indicative examples of instantiation of the descriptor and the final description of policies in Drools.

The outcome of this deliverable constitutes main input for the design and development of all the runtime policies editing, management and enforcement mechanisms that are going to be developed within WP2 and WP3. Furthermore, it constitutes a starting point towards the definition of the runtime policies to be applied in each one of the MATILDA demonstrators, as they are going to be detailed at D1.6 as well as within WP6.

The provided facets, including the set of conditions and actions, constitute an elaborated version of the potential conditions and actions, however the overall list is going to be open and extensible, targeting at fulfilling all the requirements that may be arisen during the project lifetime.

References

[1] D1.1 – MATILDA Framework and Reference Architecture, MATILDA H2020 Project, Available Online: http://www.matilda-5g.eu/index.php/outcomes

[2] Drools Business Rules Management System, Available Online: https://www.drools.org/

[3] D1.2 – Chainable Application Component & 5G-ready Application Graph Metamodel, MATILDA H2020 Project, Available Online: http://www.matilda-5g.eu/index.php/outcomes

[4] D1.3 – VNF/PNF & VNF Forwarding Graph Metamodel, MATILDA H2020 Project, Available Online: http://www.matilda-5g.eu/index.php/outcomes

[5] D1.4 – Network Slice Intent and Instance Metamodel, MATILDA H2020 Project, Available Online: http://www.matilda-5g.eu/index.php/outcomes