Project Acronym:	**OPTIMIS**
Project Title:	**Optimized Infrastructure Services**
Project Number:	**257115**
Instrument:	**Integrated Project**
Thematic Priority:	**ICT-2009.1.2 – Internet of Services, Software and Virtualisation**

ID2.2.2 Service Manifest Specification

Activity 2:

WP 2.2: *Service Manifest*

Due Date:		
Submission Date:		
Start Date of Project:		01/06/2010
Duration of Project:		
Organisation Responsible for the Deliverable:		SCAI
Version:		0.1
Status		Draft
Author(s):	Angela Rumpl Hassan Rasheed Oliver Waeldrich	SCAI
Reviewer(s)		

Table of Contents

1 Introduction

There are three steps in the OPTIMIS service lifecycle: construction of the service, deployment of the service, and operation of the service. Services are developed, orchestrated, and configured by SPs for deployment on IPs. The SP writes specification and configuration of the service manifest describing the functional and non-functional parameters of the service. Information relevant to the service manifest includes: VM images, thresholds for TREC factors the SP requests, location and cost constraints, capacity and elasticity requirements, KPIs to monitor, etc.

The OPTIMIS service manifest basically describes the requirements of the service provider for an infrastructure service provisioning process. The Service Manifest is therefore an abstract definition of the infrastructure services as expected by the service provider. It is possible to specify multiple infrastructure services in one manifest file. All aspects of these infrastructure services must be described in detail in the manifest. Figure 1 service manifest structure shows a high level overview of the service manifest structure.

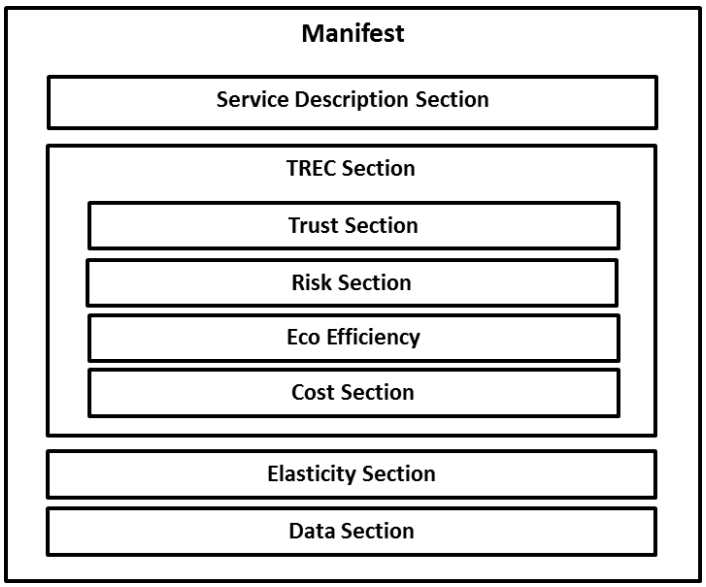

Figure 1 service manifest structure

The remainder of the document is structured as follows; first we give a high level overview of the manifest component model and we describe how references to service components are realized in the service manifest. Then we give an overview how the manifest can be extended with additional information in a particular domain, i.e. the service provider domain or the infrastructure provider domain. Next we provide a detailed description of the data types used in the service manifest in detail.

2 The Component Model

Applications that are deployed in the cloud often consist of more than one component; a web-application for example may require a web server, an application server and a database server to run. In order to perform efficiently, each application component may have different requirements in terms of numbers of allocated instances, CPU speed, memory, etc. Therefore, the OPTIMIS service provider must be able to describe multiple components of a service that it wishes to run along with the constraints for each component in a single service manifests.

The service manifest supports this requirement by a simple component model. Basically, each service may consist of one or more components. A service is described in the service description section of the service manifest. The components of the service are specified in this section as *Service Component* elements. Each service component must be identified by a component id which must be unique within a single manifest instance.

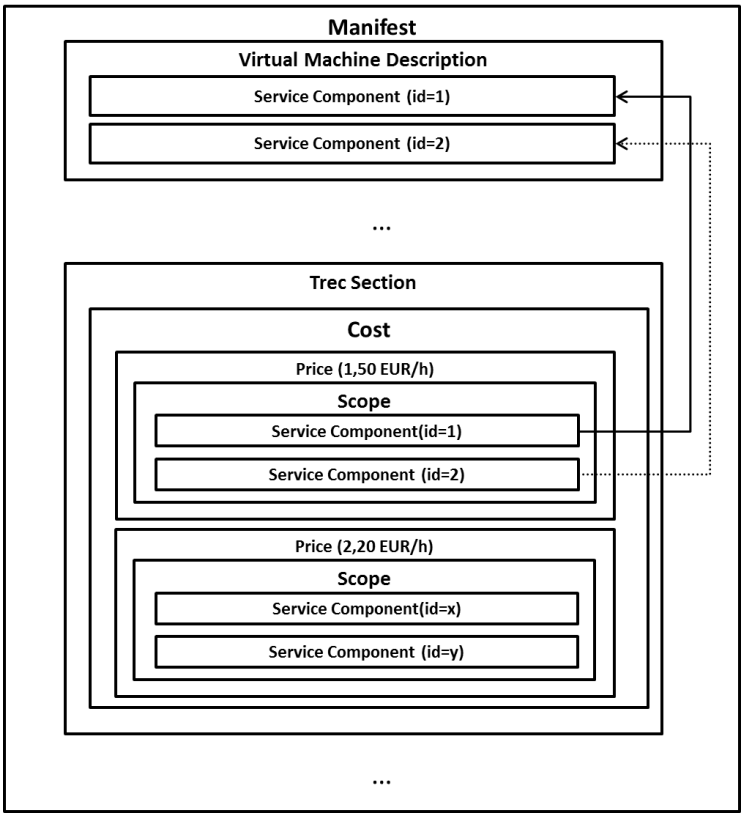

Figure 2: example of a service component reference

Service components must be referable by other sections in the service manifest. It must for example be possible to specify a different price for the different components of a service. Therefore the different sections in the manifest may specify a component scope. In general we distinguish between two types of sections in the manifest: *global sections* and *scoped sections*. Global sections apply to the general service provisioning process, in other words these sections apply to the aggregated service. Scoped sections apply to at least one service component, but they may refer to multiple service components. If a scoped section refers to a particular

service component, it must refer to a valid service component id. Entities that process the service manifest must take these references into account. Figure 2 illustrates this behavior on the example of the cost section.

3 Annotating the Service Manifest

By default the service manifest only contains the information that is exchanged between service provider and infrastructure provider, i.e. a description of the different types of virtual machines that should be provided by the infrastructure provider, the number of instances that can be allocated for each VM type, the trust, risk, eco efficiency and cost parameters, etc. This information is usually insufficient at multiple stages of the service provisioning process. The service provider for example wants to specify software dependencies for a particular service component in the manifest. A set of contextualization tools allow the service provider to install these software dependencies into a bare VM image offered by the infrastructure provider. The resulting customized VM image is then uploaded to the infrastructure provider and used for the service provisioning process. Another example is the provisioning of the requested VMs in the infrastructure provider domain. After the service manifest was received by the infrastructure provider the IP must generate a document which describes all VM instances for all components defined in the service manifest service description section. This document must be passed along with the service manifest. Multiple components need to process the VM instance description document for different purposes, i.e. the Cloud Optimizer needs to specify which VM instance is deployed on which host, the IP contextualization tool adopt for example the network configuration of the virtual machine to sensible values with respect of the allocated infrastructure, data management might change the references to the VM images according to the physical file names of the images which were transferred to the execution system. These examples already show that there is a need to specify a wide set of information in the service manifest. In general we can distinguish three categories of information:

1. Information only used in the service provider domain
2. Information only used in the infrastructure provider domain
3. Shared information

The shared information is basically what is defined in the core service manifest. This information is passed from the SP to the IP when the service is contracted, i.e. when the SLA for a service provisioning process is created. Information that is only used in one of these domains, i.e. for preparing a VM image of for deploying and configuring a service, can be included in the service manifest as an extension document. This allows passing this type of information between the different services within one domain.

Note: Extension documents are not passed between domains. They are not part of the contract between service provider and infrastructure provider. Components from other domains than the one for which the extension document was defined are completely unaware of the existence of this information. This is also true for different infrastructure providers, for example in a cloud bursting scenario.

This specification defines two types of extension documents, one for infrastructure providers and one for service providers. Each extension document is defined in a separate namespace. The definition of the namespaces is provided below.

Namespaces

Service Provider Namespace	*http://schemas.optimis.eu/optimis/sp-extensions*
Infrastructure Provider Namespace	*http://schemas.optimis.eu/optimis/ip-extensions*

As mentioned before, the domain-specific extension documents are included in the service manifest and can therefore be passed along with the manifest between the services of one domain. For that purpose the service manifest defines the appropriate extension points (xs:any elements, see XML Schema specification). Figure 3 illustrates the extension mechanism on the example of an infrastructure provider extension.

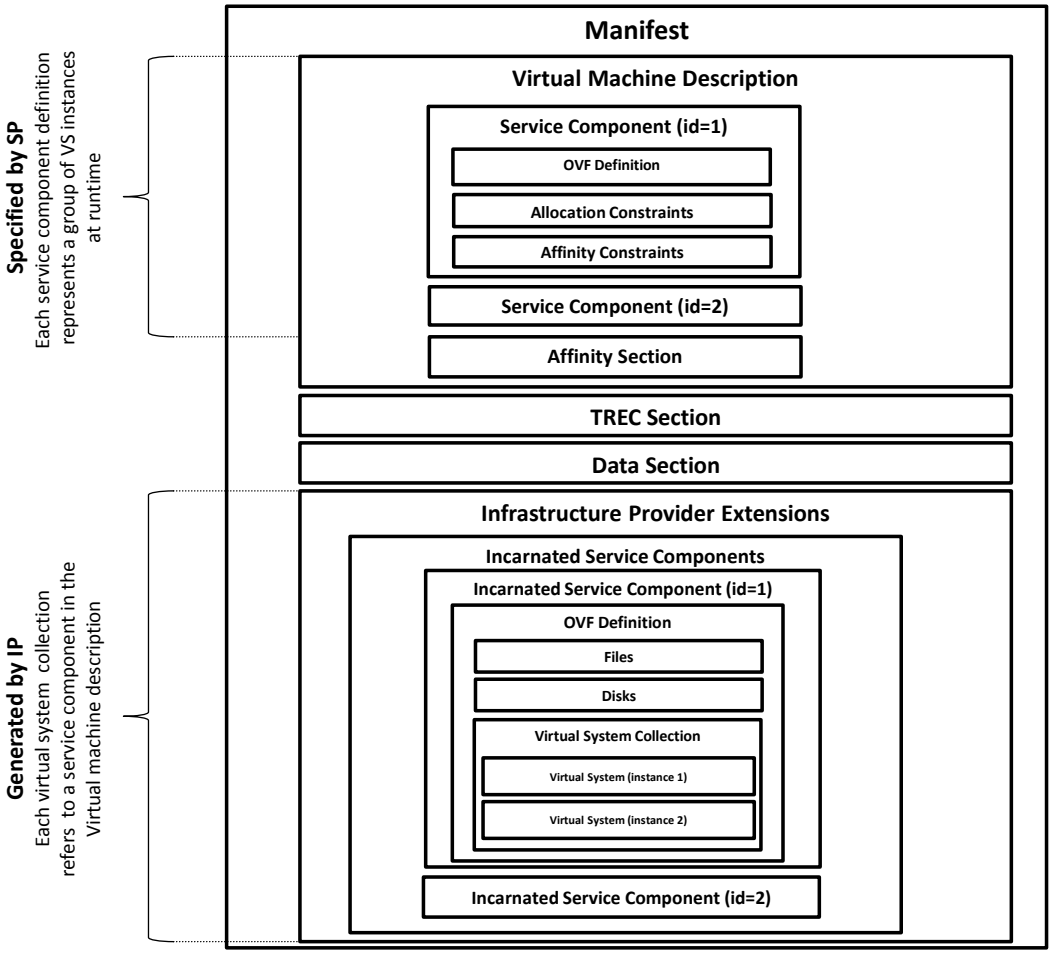

Figure 3: example of an infrastructure provider extension in the manifest

4 Service Manifest Schema

The following section gives a detailed definition of the types defined for the service manifest schema. It defines the main purpose of the different sections of the manifest and describes the use of the elements within these sections.

4.1 Service Description Section

This section provides a definition of an OPTIMIS Service. A service MAY consist of multiple components. At least one service component MUST be defined in the service description section. In principle different types of service components can be defined for a manifest. By default this specification defines a virtual machine service component that can be used to deploy one particular virtual system in an OPTIMIS IP infrastructure (see Virtual Machine Description).

```
<opt:ServiceDescriptionSection>
    <opt:ServiceComponent componentId=”xs:string”> +
</opt:ServiceDescriptionSectionType>
```

/opt:ServiceDescriptionSection/opt:ServiceComponent

This element describes one component of a service that is deployed in the OPTIMIS infrastructure; i.e. one type of VMs with a given number of instances. At least one service component element MUST be specified in a service description section.

/opt:ServiceDescriptionSection/opt:ServiceComponent/@opt:componentId

The REQUIRED attribute “*componentId*” represents the ID of the service component. The id MUST be unique in the service manifest. It is used to link other sections in the service manifest with one particular service component, i.e. in order to associate specific costs with that component.

4.2 Virtual Machine Description

A virtual machine description is a specific representation of an OPTIMIS service description. It is used to describe a set of virtual machine components that are deployed in an OPTIMIS IP infrastructure. It also describes the affinity of the components.

```
<opt:VirtualMachineDescription>
   <opt:VirtualMachineComponent>
        opt:VirtualMachineComponent
   </opt:VirtualMachineComponent>
   <opt:AffinitySection>
      opt:AffinitySection
   </opt:AffinitySection >
</opt:VirtualMachineDescription
```

4.2.1 Virtual Machine Component

A virtual machine component is a specific representation of an OPTIMIS service component. It is used to describe one particular class of virtual machines that are deployed in an OPTIMIS IP infrastructure. A class of virtual machines is defined as a set of virtual machine instances that are based on the same *virtual system definition*. A virtual system definition is provided as OVF description. This description MUST define exactly one virtual system. Virtual system collection is NOT supported.

The virtual machine description section (Figure 4) consists of a number of service components that together form a service described in the manifest.

Each service component comprises an OVF description section, an allocation constraints section and an affinity section.

- The *OVFDefinition* element (see 4.2.1.1) is a template for creating instances of this component at the IP site. It provides information on location, format, network connection and virtual system description to be used for creating component instances.
- The *AllocationConstraints* element (see 4.2.1.2) is used to define the maximum and minimum number of component instances.
- *AffinityConstraints* (see 4.2.1.3) describe the level of affinity the incarnated instances must have.

For further information on how the component instances are created see section 5 on IP Extensions.

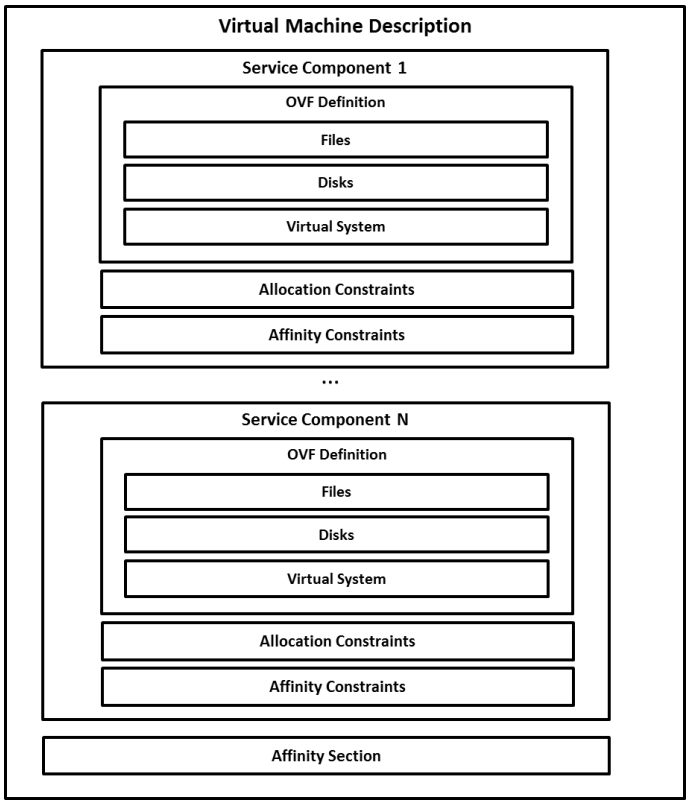

Figure 4: virtual machine description

4.2.1.1 OVF Definition

The OPTIMIS virtual machine service component is described in the Open Virtualization Format (OVF). The OVF specification is a standard being developed within the Distributed Management Task Force (DMTF) association to promote an open, secure, portable, efficient, and extensible format for the packaging and distribution of software to be run in virtual machines.

The service component description MUST provide only the following elements:

- A list of file references to all external files that are part of the OVF package, defined by the *References* element and its *File* child elements. These are typically virtual disk files, ISO images, and internationalization resources. The specified files refer to the appropriate files in the OPTIMIS VM repository. The IP uses this information at runtime to stage in the referenced files to the allocated execution systems.
- A *DiskSection* that describes meta-information about virtual disks in the OVF package. Virtual disks and their metadata are described outside the virtual hardware to facilitate sharing between virtual machines within an OVF package. A virtual disk MUST link to a file definition in the references section.
- A description of the virtual machine, more specifically a *VirtualSystem* element. The *VirtualSystem* definition MUST be contained in a valid OPTIMIS service component description. Only one virtual system is allowed. Virtual System Collections are NOT supported in the service component definition.

4.2.1.2 Allocation Constraints

In this section we define how many instances of a component can be created. An allocation constraint is directly associated with a particular service component. It defines the boundaries of the virtual system instances that can be started for a particular service component.

/opt:AllocationConstraintType/opt:LowerBound

The lower bound defines the minimum number of VM instances that must be provided. The infrastructure provider should not allocate less VM instances as defined by the lower bound.

/opt:AllocationConstraintType/opt:UpperBound

The upper bound defines the maximum number of VM instances that can be provided. The infrastructure provider must not allocate more VM instances as defined by this bound.

/opt:AllocationConstraintType/opt:Initial

The initial number of instances that are allocated

4.2.1.3 Affinity Constraints

This describes the level of affinity between incarnated instances of a service component. A service provider may specify one of the following values: *Low*, *Medium* or *High*. The values have the following definition:

Low: The service component instances don't impose special deployment or communication restrictions. Therefore the component instances can be distributed across different data centers.

Medium: The service component instances impose a medium coupling with respect to deployment or communication. Therefore the component instances must be deployed within one data center.

High: The service component instances impose high restrictions with respect to deployment or communication. Therefore the component instances must be deployed within one cluster.

4.2.2 Affinity Section

```
<opt:AffinitySectionType>
   <opt:Rule>
       <opt:Scope>
            opt:ScopeArray
       </opt:Scope>
       <opt:AffinityLevel>
               {  HIGH | MEDIUM | LOW  }
        </opt:AffinityLevel>
    </opt:Rule> +
</opt:AffinitySectionType
```

In the OPTIMIS service manifest there can be two levels of affinity. First the affinity between a service component's instances, as described in section 4.2.1.3, and second the affinity between service components. This section describes the level of affinity between service components. It therefore uses the scope array to make a reference to the service components the affinity level applies to.

opt:Rule/opt:Scope/opt:ScopeArray

Specifies a set of components on which a specific affinity level is applied. The components are identified by their component ids.

opt:Rule/opt:AffinityLevel

A service provider may specify one of the following values: *Low*, *Medium* or *High*. The values have the following definition:

Low: The service components don't impose special deployment or communication restrictions. Therefore the components can be distributed across different data centers.

Medium: The service components impose a medium coupling with respect to deployment or communication. Therefore the components must be deployed within one data center.

High: The service components impose high restrictions with respect to deployment or communication. Therefore the components must be deployed within one cluster.

4.3 TREC Section

```
<opt:TRECSectionType>
    <opt:TrustSection>
        opt:TrustSection
    </opt:TrustSection>  ?
    <opt:RiskSection>
        opt:RiskSection
    </opt:RiskSection>  ?
    <opt:EcoEfficiencySection>
        opt:EcoEfficiencySection
    </opt:EcoEfficiencySection>  ?
    <opt:CostSection>
        opt:CostSection
    </opt:CostSection>  ?
</opt:TRECSectionType>
```

4.3.1 Trust

```
<opt:TrustSectionType>
    <opt:TrustLevel>
        opt:TrustLevel
    </opt:TrustLevel>
    <xsd:any namespace="##other">  *
</opt:TrustSectionType>
```

Specifies the OPTIMIS trust parameters in a TREC section.

/opt:TrustLevelType/xsd:int

Specifies the OPTIMIS trust level that is used for delegation in a federated cloud scenario

4.3.2 Risk

```
<opt:RiskSectionType>
    <opt:RiskLevel>opt:RiskLevel</opt:RiskLevel>  ?
    <opt:AvailabilityArray>
        <opt:Availability>
            <opt:Availability AssessmentInterval="xsd:duration">
                xs:int
            </opt:Availability>
        </opt:Availability>  *
    </opt:AvailabilityArray>  ?
    <xsd:any namespace="##other">  *
</opt:RiskSectionType>
```

/opt:RiskLevelType/xsd:int

Specifies the OPTIMIS risk level that is used for delegation in a federated cloud scenario

/opt:RiskSectionType/opt:AvailabilityArray

Contains one or more Availability elements, each availability element specifies the availability of a virtual machine in a given assessment interval.

/opt:AvailabilityArrayType/opt:Availability

Specifies the guaranteed availability of a virtual machine in a given assessment interval

The following describes the attributes and elements listed in the schema above:

/opt:AvailabilityType/@opt:AssessmentInterval

Defines the duration of the assessment interval in seconds

4.3.3 EcoEfficiency

```
<opt:EcoEfficiencySectionType>
   <opt:LEEDCertification>
       opt:LEEDCertificationConstraint
   </opt:LEEDCertification> ?
   <opt:BREEAMCertification>
       opt:BREEAMCertificationConstraint
   </opt:BREEAMCertification> ?
   <opt:EuCoCCompliant>xsd:boolean</opt:EuCoCCompliant> ?
   <opt:EnergyStarRating>opt:EnergyStarRating</opt:EnergyStarRating> ?
   <xsd:any namespace="##other">  *
</opt:EcoEfficiencySectionType>
```

The following describes the attributes and elements listed in the schema above:

/opt:EcoEfficiencySectionType/opt:LEEDCertificationConstraintType

Provides enumerated values as follows:

NotRequired, Certified, Silver, Gold, Platinum

/opt:EcoEfficiencySectionType/opt:BREEAMCertification

Provides enumerated values as follows:

NotRequired, Pass, Good, VeryGood, Excellent, Outstanding

/opt:EcoEfficiencySectionType/opt:EuCoCCompliant

Specifies Boolean value

/opt:EcoEfficiencySectionType/opt:EnergyStarRating

Specifies range value between 1 and 100

/opt:EcoEfficiencySectionType/{any##other}

4.3.4 Cost

```
<opt:CostSection>
      <opt:PricePlan opt:currency="xs:string" opt:planCap="0.0" opt:planFloor="0.0">
        <opt:Scope>
          <opt:componentId>opt:componentId</opt:componentId>
        </opt:Scope>
```

```
        <opt:PlanComponents>
          <opt:PriceComponent opt:componentCap="0.0" opt:componentFloor="0.0">
            <opt:ComponentLevels>
              <opt:PriceLevel>
                <opt:AbsoluteAmount>0.0</opt:AbsoluteAmount>
                <opt:PriceMetrics>
                  <opt:PriceMetric opt:factor="0.0">opt:PriceMetric</opt:PriceMetric>
                </opt:PriceMetrics>
                <opt:LevelFences>
                  <opt:PriceFence>
                    <opt:BusinessTerm>opt:BusinessTerm</opt:BusinessTerm>
                       <opt:BusinessTermExpression>
                          opt:BusinessTermExpression
                       </opt:BusinessTermExpression>
                       <opt:QuantityLiterals>
                          <opt:Quantity opt:id="xs:string">
                            <opt:Amount>0.0</opt:Amount>
                            <opt:TypeReference>opt:TypeReference</opt:TypeReference>
                          </opt:Quantity>
                       </opt:QuantityLiterals>
                  </opt:PriceFence>
                </opt:LevelFences>
              </opt:PriceLevel>
            </opt:ComponentLevels>
            <opt:Multiplier>opt:Multiplier</opt:Multiplier>
          </opt:PriceComponent>
        </opt:PlanComponents>
      </opt:PricePlan>
    </opt:CostSection>
```

In OPTIMIS, cost is an explicit parameter throughout the full service lifecycle. The OPTIMIS tools will incorporate economics-related features and thus will e.g., facilitate comparisons of alternative configurations for a service, giving rise to cost efficient services. These terms and examples are adopted from the Unified Service Description Language (USDL) Pricing Module.

- PricePlan

A PricePlan is a set of charges associated with a network-provisioned entity. Alternative sets of fees (i.e. alternative PricePlans) of the same service provision may be made available for the consumer to choose from, for example to offer the consumer the choice between a flat price scheme and a usage-based scheme (a common practice in the telecommunication industry).

Several PricePlans may exist for the same service in order to suit different user profiles and charge them appropriately (e.g. heavy- and light-usage users), or as a key price customization instrument to individually match diverse service valuations. There are three attributes associated with the PricePlan term:

1. currency, as a name string, EString: the currency for all price amounts within this PricePlan, e.g. , EUO.
2. planCap, as a float num., EFloat: providing this maximum PricePlan value prevents from charging the user a higher total price, regardless of the cumulative total price the components and adjustments within this PricePlan may eventually amount to. Example: A cap may be used to set an upper limit in a strictly usage-based plan.
3. planFloor, as a float num., EFloat: providing this minimum PricePlan value prevents from charging the user a lower total price, regardless of the cumulative total price the components and adjustments within this PricePlan may eventually amount to. Example: A floor may be used to set a lower limit to discounts that may result in an excessively low price.

- PriceComponent

PriceComponents are fees included in a PricePlan, which subject to conditions (expressed as PriceFences) may contribute to the total amount charged. Components within the same plan are summed together in order to get the total amount (price of the service). Common examples of PriceComponents that may coexist in the same PricePlan are: startup or membership charges (to access the service), periodic subscription fees (with a certain recurrence - e.g. monthly - as long as committed to by the contract), pay-per-unit charges (whose total will be proportional to the metered usage), options or feature dependent charges. The final value of the component will depend on the active PriceLevel (determined by the evaluation of the relative PriceFences) and the PriceAdjustments that may apply (e.g. discounts). There are two attributes associated with the PriceComponent term:

1. componentCap, as a float num., EFloat: providing this maximum PriceComponent value prevents the component final price from exceeding a certain amount, regardless of its levels and the parameters they are indexed to. Example: A cap may be used to set an upper limit for a component whose levels vary with usage.
2. componentFloor, as a float num., EFloat: providing this minimum PriceComponent value prevents the component final price from falling below a certain amount, regardless of its levels and the parameters they are indexed to. Example: A floor may be used to set a lower limit for a component whose levels vary with usage.

- PriceLevel

PriceLevel captures amounts charged by a PriceComponent. Since each PriceComponent may assume several values depending on the provider's price segmentation strategies, it is allowed to contain multiple PriceLevels. This allows shaping charged amounts according to customers' behavior and aligning usage with capacity or incurred costs (just like utilities do by offering different electricity rates for different times of day).

- PriceMetric

PriceMetric represents the unit of measurement by which the customer is charged for the consumption of the service or bundle. Metrics can be abstract/un-typed (e.g. per invocation) or typed (e.g. per MByte). The latter are covered by the sub-class TypedPriceMetric. The attributes that defines the PriceMetric is the factor, as a float num., EFloat: the minimum block of units that is priced, i.e. the step increase the price metric may take. It may also be a fraction. Examples: - A Gigabyte metric could be expressed equivalently as a Megabyte metric with a

factor 1024 - A professional service priced with hourly rates but charged in 15 minutes increments (factor would be 0.25).

- TypedPriceMetric

TypedPriceMetric represents a concretely typed price metric, i.e. a metric associated with a defined unit of measurement. It is defined by one attribute:

typeReference: a pointer to an entity in a type schema that formally specifies the structure of the metric .

- PriceFence

PriceFence represents a conditional expression evaluated to determine if a price element (i.e. *PricePlan, PriceComponent* or *PriceLevel*) applies. Within a *PriceFence* a certain business entity (represented by the *businessTerm)* is compared to a certain value (or set of values - the literals available to account for the different dimensions of the service provision process).

4.4 Elasticity Section

```
        opt:Rule
```

The following describes the attributes and elements listed in the schema above:

/opt:ElasticityArraySectionType/opt:Rule

4.4.1 Elasticity Rules

```
<opt:ElasticityArraySectionType>
    <opt:Rule>
        <opt:Scope>
            opt:ScopeArray
        </opt:Scope>
        <opt:KPIName>xsd:string</opt:KPIName>
        <opt:Window>xsd:duration</opt:Window>
        <opt:Frequency>xsd:positiveInteger</opt:Frequency>
        <opt:Quota>xsd:positiveInteger</opt:Quota>
        <opt:Tolerance>opt:PositiveDecimal</opt:Tolerance>  ?
    </opt:Rule>  +
</opt:ElasticityArraySectionType>
```

The following describes the attributes and elements listed in the schema above:

/opt:RuleType/opt:KPIName

/opt:RuleType/opt:Window

/opt:RuleType/opt:Frequency

/opt:RuleType/opt:Quota

/opt:RuleType/opt:Tolerance

/opt:RuleType/opt:Scope

Specifies a set of VM instances on which a specific Elasticity Action is invoked. Specific elasticity actions are defined for a specific KPI. The elasticity actions are invoked for the VMs in the same sequence as specified in this list.

```
<opt:ScopeArrayType>
    <opt:ComponentId>
        xsd:string
    </opt:ComponentId>  +
</opt:ScopeArrayType>
```

4.5 Data Protection Section

```
<opt:DataProtectionSectionType>
    <opt:EligibleCountryList>
        <opt:Country>opt:ISO3166Alpha2</opt:Country> +
    </opt:EligibleCountryList>  ?
    <opt:NonEligibleCountryList>
        <opt:Country>opt:ISO3166Alpha2</opt:Country> +
    </opt:NonEligibleCountryList>  ?
    <opt:DataProtectionLevel>
        opt:DataProtectionLevel
    </opt:DataProtectionLevel>  ?
    <opt:DataEncryptionLevel>
        <opt:EncryptionAlgorithm>
        <opt:EncryptionKeySize>
        <opt:CustomEncryptionLevel>
    </opt:DataEncryptionLevel>  ?
    <xsd:any namespace="##other">  *
</opt:DataProtectionSectionType>
```

The following describes the attributes and elements listed in the schema above:

/opt:EncryptionLevelType/opt:EncryptionAlgorithmType

Enumeration values are: NotApplicable, AES, Twofish, AES-Twofish, AES-Twofish-Serpent, Serpent-AES, Serpent-Twofish-AES, Twofish-Serpent

/opt: EncryptionLevelType/opt:CustomEncryptionLevel

5 IP Extensions

The IP extension document contains additional information for deploying services in an OPTIMIS IP infrastructure based on the service specification provided by the service provider. The IP extension document is generated by the OPTIMIS IP before deploying the services requested by the SP. The IP extension document is dynamically included in the manifest and available for all services in the IP domain. It is passed along with the manifest and the different

IP components can read the extension document and process its content during a service provisioning process. The IP extension document is only valid in the domain of the particular IP that generated the document; i.e. this information will not be passed to subcontractors such as other IPs in a cloud bursting scenario.

IP Extension Document Processing Model

The SP specifies the services it needs in order to run an application as service components in the manifest *Service Description Section*. Each service components acts as a prototype for the virtual system instances that are finally provided by the OPTIMIS IP; this means the OVF descriptions of the virtual systems that are finally deployed in the OPTIMIS IP infrastructure are derived from the abstract service component definitions. The number of virtual system instances that are provided per service component is defined in the allocation section of a service component.

The IP extensions document is an incarnation of the abstract services component definition specified in the manifest. It basically contains a set of *incarnated service definitions*. An incarnated service definition is an OVF document which describes all virtual systems that can be deployed for one service component. Each incarnated service refers to one service component definition in the manifest. An incarnated service is basically represented as an OVF document which contains a Virtual System Collection with the virtual systems that can be deployed for a service component with respect to the allocation constraints. Figure 5 shows the IP extension document which is the result of an incarnation process.

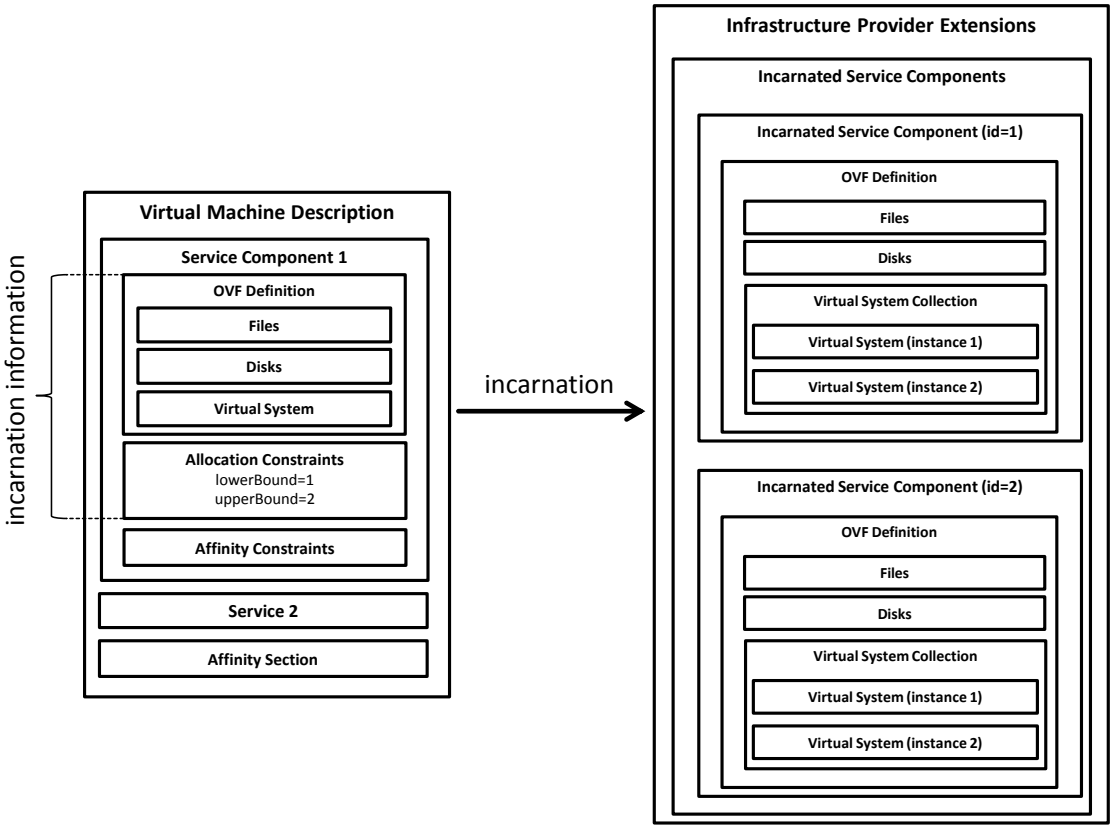

Figure 5: generated extension document during the service incarnation process

Incarnation Model

The incarnation process is the process of generating a concrete OVF document from the abstract OVF template specified in the service component definition. The incarnation process uses the allocation constraints defined in the service component's allocation constraints section to determine the maximum number of instances of the service component. For each possible VM instance a new virtual system is generated in the resulting OVF file. The virtual systems are grouped in a Virtual System Group element. In order to successfully incarnate a service from the OVF template the template document must be valid, i.e. all reference in the template must be valid.

The incarnation process follows a simple algorithm illustrated below.

- Copy all references in the OVF template to the incarnated OVF
- Update the Ids of all copied references such as: newId = Id + "instance_" + i
- Copy all disks form the OVF template to the incarnated OVF
- Update the Ids of all copied disks such as: newId = Id + "instance_" + i
- Update the file references of all copied disks such as: newRef = Ref + "instance_" + i
- Copy the virtual system definition from the template to the incarnated virtual system group
- Update the Id of the copied virtual system such as: newId = Id + "instance_" + i
- Update the disk references of the copied virtual system such as: newRef = Ref + "instance_" + i

6 SP Extensions

- Include a short description of the software dependencies
- Description of mapping software dependencies to a particular service/VM image?

7 Appendix

7.1 Service Manifest

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3               xmlns:opt="http://schemas.optimis.eu/optimis/"
4               xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
5               targetNamespace="http://schemas.optimis.eu/optimis/"
6               elementFormDefault="qualified" attributeFormDefault="qualified">
7       <xs:import namespace="http://schemas.dmtf.org/ovf/envelope/1"
8 schemaLocation="http://schemas.dmtf.org/ovf/envelope/1/dsp8023_1.1.0.xsd"/>
9
10
11      <xs:element name="ServiceManifest" type="opt:ManifestType"/>
12      <xs:element name="ElasticitySection" type="opt:ElasticityArraySectionType"/>
13      <xs:element name="DataProtectionSection" type="opt:DataProtectionSectionType"/>
14      <xs:element name="TRECSection" type="opt:TRECSectionType"/>
15
16      <xs:element name="ServiceDescriptionSection"
17                  type="opt:ServiceDescriptionSectionType"/>
18      <xs:element name="ServiceComponent" type="opt:ServiceComponentType"/>
19      <xs:element name="VirtualMachineDescription"
20                  type="opt:VirtualMachineDescriptionType"
21                  substitutionGroup="opt:ServiceDescriptionSection"/>
22      <xs:element name="VirtualMachineComponent"
type="opt:VirtualMachineComponentType"
23                  substitutionGroup="opt:ServiceComponent"/>
24
25      <xs:complexType name="ManifestType">
26         <xs:annotation>
27            <xs:documentation>
28               Type definition of the OPTIMIS service manifest.
29            </xs:documentation>
30         </xs:annotation>
31         <xs:sequence>
32            <xs:element ref="opt:ServiceDescriptionSection"/>
33            <xs:element ref="opt:TRECSection" minOccurs="0"/>
34            <xs:element ref="opt:ElasticitySection" minOccurs="0"/>
35            <xs:element ref="opt:DataProtectionSection" minOccurs="0"/>
36            <xs:any minOccurs="0" maxOccurs="unbounded" processContents="strict"
37                    namespace="##other"/>
38         </xs:sequence>
39         <xs:attribute name="manifestId" use="required">
40            <xs:annotation>
```

```
41                <xs:documentation>
42                   The manifest id is composed of the SLA name and SLA version. The
43                   values are separated by a colon.
44
45                   Remark: this pattern might change in future in case requirements
46                   change.
47                </xs:documentation>
48             </xs:annotation>
49             <xs:simpleType>
50                <xs:restriction base="xs:string">
51                   <xs:pattern value="\w[\w_\-]*\:\d+"/>
52                </xs:restriction>
53             </xs:simpleType>
54          </xs:attribute>
55          <xs:attribute name="serviceProviderId" type="xs:string"/>
56       </xs:complexType>
57
58       <xs:complexType name="ServiceDescriptionSectionType" abstract="true">
59          <xs:annotation>
60             <xs:documentation>
61                Base type of an OPTIMIS Service Description. All service descriptions
62                inherit from this type. Additional service descriptions MAY be defined
63                for OPTIMIS and can be included into the service manifest as XSD
64                substitution group.
65             </xs:documentation>
66          </xs:annotation>
67          <xs:sequence>
68             <xs:element ref="opt:ServiceComponent" minOccurs="1"
maxOccurs="unbounded"/>
69          </xs:sequence>
70          <xs:attribute name="serviceId" type="xs:string" use="required"/>
71       </xs:complexType>
72
73       <xs:complexType name="ServiceComponentType" abstract="true">
74          <xs:annotation>
75             <xs:documentation>
76                Base type of an OPTIMIS Service Component. All service components
77                inherit from this type. Additional service components MAY be defined
for
78                OPTIMIS and can be included into the service manifest as XSD
79                substitution group.
80             </xs:documentation>
81          </xs:annotation>
82          <xs:attribute name="componentId" type="xs:string" use="required"/>
83       </xs:complexType>
```

```
84
85      <xs:complexType name="AbstractVirtualMachineDescriptionType" abstract="true">
86         <xs:annotation>
87            <xs:documentation>
88               Provisioning of plain virtual machines is the default OPTIMIS use case.
89               The VirtualMachineServiceDescription specifies the VMs that are
provided
90               to a customer once an SLA is created.
91            </xs:documentation>
92         </xs:annotation>
93         <xs:complexContent>
94            <xs:extension base="opt:ServiceDescriptionSectionType">
95               <xs:sequence>
96                  <!--<xs:element ref="opt:VirtualMachineComponent" minOccurs="1"-->
97                  <!--maxOccurs="unbounded"/>-->
98                  <xs:element name="AffinitySection" type="opt:AffinitySectionType"
99                              maxOccurs="1"/>
100                 <xs:any namespace="##other" processContents="strict" minOccurs="0"
101                         maxOccurs="unbounded"/>
102              </xs:sequence>
103           </xs:extension>
104        </xs:complexContent>
105     </xs:complexType>
106     <xs:complexType name="VirtualMachineDescriptionType">
107        <xs:annotation>
108           <xs:documentation>
109              Provisioning of plain virtual machines is the default OPTIMIS use case.
110              The VirtualMachineServiceDescription specifies the VMs that are
provided
111              to a customer once an SLA is created.
112           </xs:documentation>
113        </xs:annotation>
114        <xs:complexContent>
115           <xs:restriction base="opt:AbstractVirtualMachineDescriptionType">
116              <xs:sequence>
117                 <xs:element ref="opt:VirtualMachineComponent" minOccurs="1"
118                             maxOccurs="unbounded"/>
119                 <xs:element name="AffinitySection" type="opt:AffinitySectionType"
120                             maxOccurs="1"/>
121                 <xs:any namespace="##other" processContents="strict" minOccurs="0"
122                         maxOccurs="unbounded"/>
123              </xs:sequence>
124           </xs:restriction>
125        </xs:complexContent>
126     </xs:complexType>
```

```
127
128     <xs:element name="OVFDefinition" type="ovf:EnvelopeType"/>
129     <xs:element name="AllocationConstraints"
130                 type="opt:AllocationConstraintType"/>
131     <xs:element name="AffinityConstraints" type="opt:AffinityConstraintType"/>
132     <xs:complexType name="VirtualMachineComponentType">
133        <xs:annotation>
134           <xs:documentation>
135              It is used to describe one particular class of virtual machines that
are
136              deployed in an OPTIMIS IP infrastructure.
137           </xs:documentation>
138        </xs:annotation>
139        <xs:complexContent>
140           <xs:extension base="opt:ServiceComponentType">
141              <xs:sequence>
142                 <xs:element ref="opt:OVFDefinition" maxOccurs="1"/>
143                 <xs:element ref="opt:AllocationConstraints" maxOccurs="1"/>
144                 <xs:element ref="opt:AffinityConstraints" maxOccurs="1"/>
145              </xs:sequence>
146           </xs:extension>
147        </xs:complexContent>
148     </xs:complexType>
149
150     <xs:complexType name="AllocationConstraintType">
151        <xs:annotation>
152           <xs:documentation>
153              Defines the scaling constraints for a specific component.
154           </xs:documentation>
155        </xs:annotation>
156        <xs:sequence>
157           <xs:element name="LowerBound" type="xs:int"/>
158           <xs:element name="UpperBound" type="xs:int"/>
159           <xs:element name="Initial" type="xs:int"/>
160        </xs:sequence>
161     </xs:complexType>
162     <!--
163        Definition of OPTIMIS TREC parameters.
164     -->
165     <xs:complexType name="TRECSectionType">
166        <xs:sequence>
167           <xs:element name="TrustSection" type="opt:TrustSectionType"
minOccurs="0"/>
168           <xs:element name="RiskSection" type="opt:RiskSectionType" minOccurs="0"/>
```

```
169           <xs:element name="EcoEfficiencySection"
type="opt:EcoEfficiencySectionType"
170                       minOccurs="0"/>
171           <xs:element name="CostSection" type="opt:CostSectionType" minOccurs="0"/>
172        </xs:sequence>
173     </xs:complexType>
174     <xs:complexType name="TrustSectionType">
175        <xs:annotation>
176           <xs:documentation>
177              Specifies the OPTIMIS trust parameters in a TREC section.
178           </xs:documentation>
179        </xs:annotation>
180        <xs:sequence>
181           <xs:element name="TrustLevel" type="opt:TrustLevelType"/>
182           <xs:any namespace="##other" processContents="strict" minOccurs="0"
183                   maxOccurs="unbounded"/>
184        </xs:sequence>
185     </xs:complexType>
186     <xs:simpleType name="TrustLevelType">
187        <xs:annotation>
188           <xs:documentation>
189              Specifies the OPTIMIS Trust Level that is used for delegation in a
190              federated cloud scenario.
191
192              TODO: is there a specification of the different Trust Levels in
OPTIMIS?
193           </xs:documentation>
194        </xs:annotation>
195        <xs:restriction base="xs:int">
196           <xs:minInclusive value="0"/>
197        </xs:restriction>
198     </xs:simpleType>
199     <!--
200        Definition of OPTIMIS Risk Constraints.
201     -->
202     <xs:complexType name="RiskSectionType">
203        <xs:sequence>
204           <xs:element name="RiskLevel" type="opt:RiskLevelType" minOccurs="0"/>
205           <xs:element name="AvailabilityArray" type="opt:AvailabilityArrayType"
206                       minOccurs="0"/>
207           <xs:any namespace="##other" processContents="strict" minOccurs="0"
208                   maxOccurs="unbounded"/>
209        </xs:sequence>
210     </xs:complexType>
211     <xs:complexType name="AvailabilityArrayType">
```

```
212         <xs:sequence>
213            <xs:element name="Availability" type="opt:AvailabilityType" minOccurs="0"
214                        maxOccurs="unbounded"/>
215         </xs:sequence>
216      </xs:complexType>
217      <xs:complexType name="AvailabilityType">
218         <xs:simpleContent>
219            <xs:extension base="xs:double">
220               <xs:attribute name="AssessmentInterval" type="xs:duration"/>
221            </xs:extension>
222         </xs:simpleContent>
223      </xs:complexType>
224      <xs:simpleType name="RiskLevelType">
225         <xs:annotation>
226            <xs:documentation>
227               Specifies the OPTIMIS Risk Level that is used for delegation in a
228               federated cloud scenario.
229
230               TODO: see comment TrustLevelType.
231            </xs:documentation>
232         </xs:annotation>
233         <xs:restriction base="xs:int">
234            <xs:minInclusive value="0"/>
235         </xs:restriction>
236      </xs:simpleType>
237
238      <!--
239         Definition of OPTIMIS EcoEfficiency Constraints.
240      -->
241      <xs:complexType name="EcoEfficiencySectionType">
242         <xs:sequence>
243            <xs:element name="LEEDCertification"
244                        type="opt:LEEDCertificationConstraintType"
245                        default="NotRequired"/>
246            <xs:element name="BREEAMCertification"
247                        type="opt:BREEAMCertificationConstraintType"
248                        default="NotRequired"/>
249            <xs:element name="EuCoCCompliant" type="xs:boolean" default="false"/>
250            <xs:element name="EnergyStarRating" type="opt:EnergyStarRatingType"
251                        default="No"/>
252            <xs:any namespace="##other" processContents="strict" minOccurs="0"
253                    maxOccurs="unbounded"/>
254         </xs:sequence>
255      </xs:complexType>
```

```
256
257
258     <xs:simpleType name="LEEDCertificationConstraintType">
259        <xs:restriction base="xs:string">
260           <xs:enumeration value="NotRequired"/>
261           <xs:enumeration value="Certified"/>
262           <xs:enumeration value="Silver"/>
263           <xs:enumeration value="Gold"/>
264           <xs:enumeration value="Platinum"/>
265        </xs:restriction>
266     </xs:simpleType>
267     <xs:simpleType name="BREEAMCertificationConstraintType">
268        <xs:restriction base="xs:string">
269           <xs:enumeration value="NotRequired"/>
270           <xs:enumeration value="Pass"/>
271           <xs:enumeration value="Good"/>
272           <xs:enumeration value="VeryGood"/>
273           <xs:enumeration value="Excellent"/>
274           <xs:enumeration value="Outstanding"/>
275        </xs:restriction>
276     </xs:simpleType>
277     <xs:simpleType name="EnergyStarRatingType">
278        <xs:union>
279           <xs:simpleType>
280              <xs:restriction base="xs:string">
281                 <xs:enumeration value="No"/>
282              </xs:restriction>
283           </xs:simpleType>
284           <xs:simpleType>
285              <xs:restriction base="xs:int">
286                 <xs:minInclusive value="1"/>
287                 <xs:maxInclusive value="100"/>
288              </xs:restriction>
289           </xs:simpleType>
290        </xs:union>
291     </xs:simpleType>
292     <!--
293        Definition of OPTIMIS Cost constraints.
294
295        TODO: Additional input required.
296     -->
297     <xs:complexType name="CostSectionType">
298        <xs:sequence>
299           <xs:element name="PricePlan" maxOccurs="unbounded"
```

```
300                      type="opt:PricePlanType"/>
301         </xs:sequence>
302      </xs:complexType>
303      <xs:complexType name="PricePlanType">
304         <xs:annotation>
305            <xs:documentation>
306               A PricePlan is a set of charges associated with a network-provisioned
307               entity. Alternative sets of fees (i.e. alternative PricePlans) of the
308               same service provision may be made available for the consumer to choose
309               from, for example to offer the consumer the choice between a flat price
310               scheme and a usage-based scheme (a common practice in the
311               telecommunication industry). Several PricePlans may exist for the same
312               service in order to suit different user profiles and charge them
313               appropriately (e.g. heavy- and light-usage users), or as a key price
314               customization instrument to individually match diverse service
315               valuations. There are three attributes associated with the PricePlan
316               term:
317               <br/>
318               1. currency, as a name string, EString: the currency for all price
319               amounts within this PricePlan, e.g. EUR.
320               <br/>
321               2. planCap, as a float num., EFloat: providing this maximum PricePlan
322               value prevents from charging the user a higher total price, regardless
323               of the cumulative total price the components and adjustments within
this
324               PricePlan may eventually amount to. Example: A cap may be used to set
an
325               upper limit in a strictly usage-based plan.
326               <br/>
327               3. planFloor, as a float num., EFloat: providing this minimum PricePlan
328               value prevents from charging the user a lower total price, regardless
of
329               the cumulative total price the components and adjustments within this
330               PricePlan may eventually amount to. Example: A floor may be used to set
331               a lower limit to discounts that may result in an excessively low price.
332            </xs:documentation>
333         </xs:annotation>
334         <xs:sequence>
335            <xs:element name="Scope" type="opt:ScopeArrayType"/>
336            <xs:element name="PlanComponents" type="opt:PlanComponentArrayType"/>
337         </xs:sequence>
338         <xs:attribute name="planCap" type="xs:float"/>
339         <xs:attribute name="planFloor" type="xs:float"/>
340         <xs:attribute name="currency" type="xs:string"/>
341      </xs:complexType>
```

```
342
343     <xs:complexType name="PlanComponentArrayType">
344        <xs:sequence>
345           <xs:element name="PriceComponent" type="opt:PriceComponentType"/>
346        </xs:sequence>
347     </xs:complexType>
348     <xs:complexType name="PriceComponentType">
349        <xs:annotation>
350           <xs:documentation>
351              PriceComponents are fees included in a PricePlan, which subject to
352              conditions (expressed as PriceFences) may contribute to the total
amount
353              charged. Components within the same plan are summed together in order
to
354              get the total amount (price of the service). Common examples of
355              PriceComponents that may coexist in the same PricePlan are: startup or
356              membership charges (to access the service), periodic subscription fees
357              (with a certain recurrence - e.g. monthly - as long as committed to by
358              the contract), pay-per-unit charges (whose total will be proportional
to
359              the metered usage), options or feature dependent charges. The final
360              value of the component will depend on the active PriceLevel (determined
361              by the evaluation of the relative PriceFences) and the PriceAdjustments
362              that may apply (e.g. discounts). There are two attributes associated
363              with the PriceComponent term:
364              <br/>
365              1. componentCap, as a float num., EFloat: providing this maximum
366              PriceComponent value prevents the component final price from exceeding
a
367              certain amount, regardless of its levels and the parameters they are
368              indexed to. Example: A cap may be used to set an upper limit for a
369              component whose levels vary with usage.
370              <br/>
371              2. componentFloor, as a float num., EFloat: providing this minimum
372              PriceComponent value prevents the component final price from falling
373              below a certain amount, regardless of its levels and the parameters
they
374              are indexed to. Example: A floor may be used to set a lower limit for a
375              component whose levels vary with usage.
376           </xs:documentation>
377        </xs:annotation>
378        <xs:sequence>
379           <xs:element name="ComponentLevels" type="opt:ComponentLevelArrayType"/>
380           <xs:element name="Multiplier" type="xs:string"/>
381        </xs:sequence>
382        <xs:attribute name="componentCap" type="xs:float"/>
```

```
383         <xs:attribute name="componentFloor" type="xs:float"/>
384      </xs:complexType>
385      <xs:complexType name="ComponentLevelArrayType">
386         <xs:sequence>
387            <xs:element name="PriceLevel" type="opt:PriceLevelType"
388                        maxOccurs="unbounded"/>
389         </xs:sequence>
390      </xs:complexType>
391
392      <xs:complexType name="PriceLevelType">
393         <xs:annotation>
394            <xs:documentation>
395               PriceLevel captures amounts charged by a PriceComponent. Since each
396               PriceComponent may assume several values depending on the provider's
397               price segmentation strategies, it is allowed to contain multiple
398               PriceLevels. This allows shaping charged amounts according to
customers'
399               behavior and aligning usage with capacity or incurred costs (just like
400               utilities do by offering different electricity rates for different
times
401               of day).
402            </xs:documentation>
403         </xs:annotation>
404         <xs:sequence>
405            <xs:element name="AbsoluteAmount" type="xs:decimal"/>
406            <xs:element name="PriceMetrics" type="opt:PriceMetricArrayType"/>
407            <xs:element name="LevelFences" type="opt:LevelFencesArrayType"/>
408         </xs:sequence>
409      </xs:complexType>
410
411      <xs:complexType name="PriceMetricArrayType">
412         <xs:sequence>
413            <xs:element name="PriceMetric" type="opt:PriceMetricType"
414                        maxOccurs="unbounded"/>
415         </xs:sequence>
416      </xs:complexType>
417
418      <xs:complexType name="PriceMetricType">
419         <xs:annotation>
420            <xs:documentation>
421               PriceMetric represents the unit of measurement by which the customer is
422               charged for the consumption of the service or bundle. Metrics can be
423               abstract/un-typed (e.g. per invocation) or typed (e.g. per MByte). The
424               latter are covered by the sub-class TypedPriceMetric. The attributes
```

```
425              that defines the PriceMetric is the factor, as a float num., EFloat:
the
426              minimum block of units that is priced, i.e. the step increase the price
427              metric may take. It may also be a fraction.
428              <br/>
429              Examples: - A Gigabyte metric could be expressed equivalently as a
430              Megabyte metric with a factor 1024 - A professional service priced with
431              hourly rates but charged in 15 minutes increments (factor would be
432              0.25).
433           </xs:documentation>
434        </xs:annotation>
435        <xs:simpleContent>
436           <xs:extension base="xs:string">
437              <xs:attribute name="factor" type="xs:float"/>
438           </xs:extension>
439        </xs:simpleContent>
440     </xs:complexType>
441
442     <xs:complexType name="LevelFencesArrayType">
443        <xs:sequence>
444           <xs:element name="PriceFence" type="opt:PriceFenceType"
445                       maxOccurs="unbounded"/>
446        </xs:sequence>
447     </xs:complexType>
448
449     <xs:complexType name="PriceFenceType">
450        <xs:annotation>
451           <xs:documentation>
452              PriceFence represents a conditional expression evaluated to determine
if
453              a price element (i.e. PricePlan, PriceComponent or PriceLevel) applies.
454              Within a PriceFence a certain business entity (represented by the
455              businessTerm) is compared to a certain value (or set of values - the
456              literals available to account for the different dimensions of the
457              service provision process).
458           </xs:documentation>
459        </xs:annotation>
460        <xs:sequence>
461           <xs:element name="BusinessTerm" type="xs:string"/>
462           <xs:element name="BusinessTermExpression" type="xs:string"/>
463           <xs:element name="QuantityLiterals" type="opt:QuantityLiteralsArrayType"/>
464        </xs:sequence>
465     </xs:complexType>
466
467     <xs:complexType name="QuantityLiteralsArrayType">
```

```
468         <xs:sequence>
469            <xs:element name="Quantity" type="opt:QuantityType"
maxOccurs="unbounded"/>
470         </xs:sequence>
471      </xs:complexType>
472
473      <xs:complexType name="QuantityType">
474         <xs:sequence>
475            <xs:element name="Amount" type="xs:decimal"/>
476            <xs:element name="TypeReference" type="xs:string"/>
477         </xs:sequence>
478         <xs:attribute name="id" type="xs:string" use="required"/>
479      </xs:complexType>
480
481      <!--
482
483         Definition of the ElasticityArray. The definition of the RuleType is based on
the
484         Reservoir Elasticity Array. (see schema:
http://schemas.telefonica.com/claudia/ovf)
485
486         TODO: Review required!
487      -->
488      <xs:element name="Rule" type="opt:RuleType"/>
489
490      <xs:complexType name="ElasticityArraySectionType">
491         <xs:sequence>
492            <xs:element ref="opt:Rule" maxOccurs="unbounded"/>
493         </xs:sequence>
494      </xs:complexType>
495      <xs:complexType name="RuleType">
496         <xs:sequence>
497            <xs:element name="Scope" type="opt:ScopeArrayType"/>
498            <xs:element name="KPIName" type="xs:string"/>
499            <xs:element name="Window" type="xs:duration"/>
500            <xs:element name="Frequency" type="xs:positiveInteger"/>
501            <xs:element name="Quota" type="xs:positiveInteger"/>
502            <xs:element name="Tolerance" type="opt:PositiveDecimalType"
minOccurs="0"/>
503         </xs:sequence>
504      </xs:complexType>
505      <xs:complexType name="ScopeArrayType">
506         <xs:sequence>
507            <xs:element name="componentId" type="xs:string" maxOccurs="unbounded"/>
508         </xs:sequence>
509      </xs:complexType>
```

```
    <xs:simpleType name="PositiveDecimalType">
       <xs:restriction base="xs:decimal">
          <xs:minExclusive value="0"/>
       </xs:restriction>
    </xs:simpleType>
    <!--
       Definition of the OPTIMIS data protection constraints.
    -->
    <xs:complexType name="DataProtectionSectionType">
       <xs:sequence>
          <xs:element name="EligibleCountryList" type="opt:CountryListType"
                      minOccurs="0"/>
          <xs:element name="NonEligibleCountryList" type="opt:CountryListType"
                      minOccurs="0"/>
          <xs:element name="DataProtectionLevel" type="opt:DataProtectionLevelType"
                      minOccurs="0"/>
          <xs:element name="DataEncryptionLevel" type="opt:EncryptionLevelType"
                      minOccurs="0"/>
          <xs:any namespace="##other" processContents="strict" minOccurs="0"
                  maxOccurs="unbounded"/>
       </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="DataProtectionLevelType">
       <xs:annotation>
          <xs:documentation>
             DataProtectionLevel specifies the level of protection that is
guaranteed
             by a service provider regarding data management. In general it defines
             to which countries data may be transfered by the provider. Countries
are
             divided into countries that have a sufficient level of protection
(known
             as Data Protection Area-DPA) and countries that do not meet these
             levels. Transferring sensitive data to the latter is a violation and
the
             cloud providers engaged in federations should have the necessary
             framework to prevent this from happening. By law, the Cloud Provider
             does not have the obligation to keep the data in one particular country
             of the DPA. The DataProtectionLevelType specifies whether the data
             included in the service under consideration is sensitive or not. If
not,
             there are no limitations to their transfer. If yes, they should be
             restricted to countries that are part of the DPA. The list of the DPA
             countries is the following:

             - all 27 EU Member States - all countries of the European Economic Area
```

```
551               (Iceland, Liechtenstein, Norway) - Switzerland - Canada - Argentina -
552               Guernsey - Isle of Man - US organisations who take part in the US safe
553               harbour program - And the state of Israel.
554
555               TODO: Is there a maintained reference list of DPA countries online
556               available?
557            </xs:documentation>
558         </xs:annotation>
559         <xs:restriction base="xs:string">
560            <xs:enumeration value="DPA"/>
561            <xs:enumeration value="None"/>
562         </xs:restriction>
563      </xs:simpleType>
564      <xs:simpleType name="ISO3166Alpha2">
565         <xs:annotation>
566            <xs:documentation>
567               Two-letter (alpha-2) ISO 3166-1 code for one of the 243 countries.
These
568               codes are subject to change. For valid values refer to
569               http://www.iso.org/iso/list-en1-semic-3.txt
570            </xs:documentation>
571         </xs:annotation>
572         <xs:restriction base="xs:string">
573            <xs:whiteSpace value="collapse"/>
574            <xs:pattern value="[A-Z]{2}"/>
575         </xs:restriction>
576      </xs:simpleType>
577      <xs:complexType name="CountryListType">
578         <xs:sequence>
579            <xs:element name="Country" type="opt:ISO3166Alpha2"
maxOccurs="unbounded"/>
580         </xs:sequence>
581      </xs:complexType>
582      <xs:complexType name="EncryptionLevelType">
583         <xs:choice>
584            <xs:sequence>
585               <xs:element name="EncryptionAlgoritm"
586                           type="opt:EncryptionAlgoritmType"/>
587               <xs:element name="EncryptionKeySize" type="xs:int" default="128"
588                           minOccurs="0"/>
589            </xs:sequence>
590            <xs:sequence>
591               <xs:element name="CustomEncryptionLevel" type="xs:anyType"/>
592            </xs:sequence>
593         </xs:choice>
```

```
    </xs:complexType>
    <xs:simpleType name="EncryptionAlgoritmType">
       <xs:restriction base="xs:string">
          <xs:enumeration value="NotApplicable"/>
          <xs:enumeration value="AES"/>
          <xs:enumeration value="Twofish"/>
          <xs:enumeration value="AES-Twofish"/>
          <xs:enumeration value="AES-Twofish-Serpent"/>
          <xs:enumeration value="Serpent-AES"/>
          <xs:enumeration value="Serpent-Twofish-AES"/>
          <xs:enumeration value="Twofish-Serpent"/>
       </xs:restriction>
    </xs:simpleType>

    <!--
       Definition of the AffinitySection.

       TODO: Review required!
    -->
    <xs:element name="AffinityRule" type="opt:AffinityRuleType"/>
    <xs:complexType name="AffinitySectionType">
       <xs:sequence>
          <xs:element ref="opt:AffinityRule" maxOccurs="unbounded"/>
       </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AffinityRuleType">
       <xs:sequence>
          <xs:element name="Scope" type="opt:ScopeArrayType"/>
          <xs:element name="AffinityConstraints" type="opt:AffinityConstraintType"/>
       </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="AffinityConstraintType">
       <xs:restriction base="xs:string">
          <xs:enumeration value="High"/>
          <xs:enumeration value="Medium"/>
          <xs:enumeration value="Low"/>
       </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

7.2 Infrastructure Provider Extensions

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
2    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:opt-
ip="http://schemas.optimis.eu/optimis/infrastructure"
3               xmlns:opt="http://schemas.optimis.eu/optimis/"
4               targetNamespace="http://schemas.optimis.eu/optimis/infrastructure"
elementFormDefault="qualified"
5               attributeFormDefault="qualified">
6       <xs:import namespace="http://schemas.optimis.eu/optimis/"
schemaLocation="./optimis.xsd"/>
7
8       <xs:element name="InfrastructureProviderExtensions" type="opt-
ip:InfrastructureProviderExtensionType"
9                   nillable="true"/>
10      <xs:element name="IncarnatedServiceComponents" type="opt-
ip:IncarnatedServiceComponentsType"/>
11      <xs:element name="IncarnatedServiceComponent" type="opt:ServiceComponentType"/>
12
13      <xs:element name="IncarnatedVirtualMachineComponents" type="opt-
ip:IncarnatedVirtualMachineComponentsType"
14                  substitutionGroup="opt-ip:IncarnatedServiceComponents"/>
15      <xs:element name="IncarnatedVirtualMachineComponent" type="opt-
ip:IncarnatedVirtualMachineComponentType"
16                  substitutionGroup="opt-ip:IncarnatedServiceComponent"/>
17
18      <xs:complexType name="InfrastructureProviderExtensionType">
19         <xs:annotation>
20            <xs:documentation>
21               Provisioning of extensions for an Infrastructure Provider.
22            </xs:documentation>
23         </xs:annotation>
24         <xs:sequence>
25            <xs:element ref="opt-ip:IncarnatedServiceComponents" minOccurs="0"/>
26            <xs:any namespace="##other" processContents="strict" minOccurs="0"
maxOccurs="unbounded"/>
27         </xs:sequence>
28      </xs:complexType>
29      <xs:complexType name="IncarnatedServiceComponentsType" abstract="true">
30         <xs:sequence>
31            <xs:element ref="opt-ip:IncarnatedServiceComponent" minOccurs="1"
maxOccurs="unbounded"/>
32         </xs:sequence>
33      </xs:complexType>
34
35      <xs:complexType name="IncarnatedVirtualMachineComponentsType">
36         <xs:complexContent>
37            <xs:restriction base="opt-ip:IncarnatedServiceComponentsType">
38               <xs:sequence>
```

```
39                  <xs:element ref="opt-ip:IncarnatedVirtualMachineComponent"
minOccurs="1" maxOccurs="unbounded"/>
40               </xs:sequence>
41            </xs:restriction>
42         </xs:complexContent>
43      </xs:complexType>
44
45      <xs:complexType name="IncarnatedVirtualMachineComponentType">
46         <xs:complexContent>
47            <xs:extension base="opt:ServiceComponentType">
48               <xs:sequence>
49                  <xs:element ref="opt:OVFDefinition" maxOccurs="1"/>
50               </xs:sequence>
51            </xs:extension>
52         </xs:complexContent>
53      </xs:complexType>
54 </xs:schema>
```

7.3 Service Provider Extensions

```
1     <?xml version="1.0" encoding="UTF-8"?>
2     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3                xmlns:opt-sp="http://schemas.optimis.eu/optimis/service"
4                targetNamespace="http://schemas.optimis.eu/optimis/service"
elementFormDefault="qualified"
5                attributeFormDefault="qualified">
6
7        <xs:import namespace="http://schemas.dmtf.org/ovf/envelope/1"
8
schemaLocation="http://schemas.dmtf.org/ovf/envelope/1/dsp8023_1.1.0.xsd"/>
9        <xs:import namespace="http://schemas.optimis.eu/optimis/"
schemaLocation="./optimis.xsd"/>
10
11       <xs:element name="ServiceProviderExtensions" type="opt-
sp:ServiceProviderExtensionType"/>
12       <xs:complexType name="ServiceProviderExtensionType">
13          <xs:annotation>
14             <xs:documentation>
15                Provisioning of extensions for a Service Provider.
16             </xs:documentation>
17          </xs:annotation>
18          <xs:sequence>
19             <xs:element name="SoftwareDependencies" type="opt-
sp:SoftwareDependenciesType" maxOccurs="unbounded"
20                         minOccurs="0"/>
21          </xs:sequence>
```

```
22      </xs:complexType>
23
24      <xs:complexType name="SoftwareDependenciesType">
25         <xs:sequence>
26            <xs:element name="Dependency" minOccurs="0" maxOccurs="unbounded">
27               <xs:complexType>
28                  <xs:all>
29                     <xs:element name="groupId" type="xs:string"/>
30                     <xs:element name="artifactId" type="xs:string"/>
31                     <xs:element name="version" type="xs:string"/>
32                  </xs:all>
33               </xs:complexType>
34            </xs:element>
35         </xs:sequence>
36         <xs:attribute name="componentId" type="xs:string"/>
37      </xs:complexType>
38
39   </xs:schema>
```