Unified Service Description Language (USDL) Pricing Module

December 28, 2009

Edited by SAP Research

Abstract. This document describes what is called the Pricing module in the third version of the Unified Service Description Language (USDL). USDL was developed to describe services in business networks and follows a system theory and business science approach. The Pricing module covers the range of concepts which are needed to adequately describe price structures in the service industry. Given the increasing complexity and variety of this aspect of service provision a modular representation of price charges is allowed, together with the elements necessary to specify common segmentation strategies.

Table of Contents

1 Introduction 3

1.1 About this document 4

1.2 Acknowledgements 4

2 Pricing Module 4

2.1 Module Info 5

2.2 Module Dependencies 6

2.3 PricePlan 7

2.4 PriceComponent 8

2.5 PriceLevel 9

2.6 AbsolutePriceLevel 10

2.7 ProportionalPriceLevel 11

2.8 ExternalBase 12

2.9 PriceAdjustment 12

2.10 PriceMetric 13

2.11 Metric 14

2.12 Quantity 14

2.13 PaymentMethod 15

2.14 PaymentTerms 15

2.15 PriceFence 16

2.16 CustomLiteral 17

2.17 Tax 18

2.18 PaymentInstrument 18

2.19 PaymentModality 19

2.20 PriceAdjustmentType 19

1 Introduction

As outlined in the central document of this series *“USDL Technical Overview Paper”*, services are becoming the backbone for electronic commerce. Especially the trend to provision IT-based services outside company “firewalls” with the help of intermediaries is on the increase, as it allows organizations to take new opportunities relatively quickly. In this context services are seen as tradable entities that constitute a well defined, encapsulated, reusable and business-aligned set of capabilities. The term business service is used for such services, in order to distinguish them from other types, e.g., those that are provided in a service-oriented IT infrastructure within an organization.

The Unified Service Description Language (USDL) defines a way to describe services from a business and operational point of view and align this with the technical perspective. While the latter is captured quite well by existing service description languages, USDL explicitly enables to express business characteristics set by an organization for the purpose of providing means for consumers to invoke and use business services and for intermediaries to (re)use and repurpose services. A detailed explanation of the scope and objectives of USDL is given in *“USDL Technical Overview Paper”*.

USDL on a whole is made up of a set of modules, each addressing different aspects of the overall service description. Modularization was introduced to improve readability of the model, which drastically grew in size compared to its predecessor. The modules have dependencies among each other (shown in Figure 1), as they may reuse concepts from other modules. Currently, there are 8 modules in the set that constitutes USDL version 3.0.

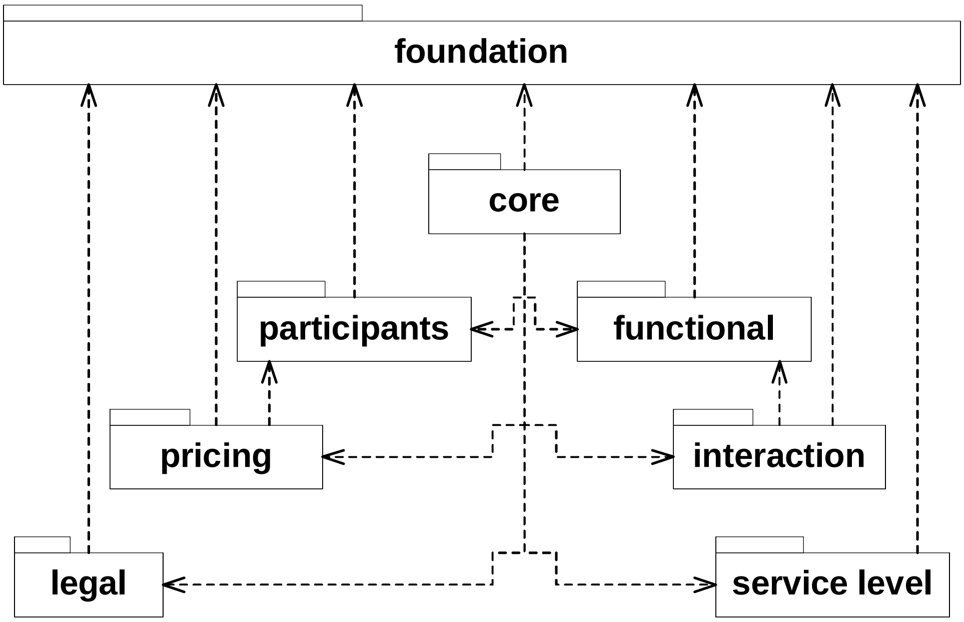

Figure 1 Packages comprising the USDL model and their dependencies (represented as arrows)

1.1 About this document

The USDL meta-model is formally defined in Ecore (the meta-modeling language of EMF), with each USDL module being captured in a separate package. This document is one in a series of USDL documents and covers the Pricing module defined in package "pricing". The series also includes:

- *USDL Technical Overview Paper*
- Module-specific documentation of the modules *Core*, *Foundation*, *Participants*, *Functional*, *Interaction*, *Service Level* (includes geographical and temporal availability) and *Legal*

The document only provides insights into the concepts of the Pricing module. For a complete overview of USDL it is necessary to go through all documents of the series.

1.2 Acknowledgements

Work on USDL has been mainly carried out in the context of the THESEUS TEXO– a project in the frame of the THESEUS Lighthouse research program initiated by the German Ministry of Information and Technology.

Several European, German, and Australian research projects also contributed to the development of USDL. Naturally, there is an extensive number of people that have contributed to conceptualization and documentation of USDL either directly or through feedback. Rather than giving the full list of individuals, it shall suffice to name the institutions they work for. The full list is available on http://www.internet-of-services.com.

SAP Research

European Research Center for Information Systems (ERCIS), University of Muenster

Queensland University of Technology, Brisbane, Australia

University of New South Wales, Sydney, Australia

2 Pricing Module

The peculiar characteristics of service transactions and the economic properties exhibited by major segments of the service industry make the pricing of services both an increasingly significant and complex matter. Proper care must be taken in the modeling process in order to address its requirements and successfully adhere to real-world marketing practices.

First of all the distinctive nature of services must be considered with regard to the pricing issues it raises. Since a service is perishable by definition, i.e. it cannot be stored or inventoried by either providers or consumers, the time of purchase[1] always precedes the time of production[2]. That's a key difference with regard to the selling of goods which can instead occur either before (e.g. in a make-to-order scenario) or after production. This is equally true for simple and complex services alike: calling a plumber to request a job, we accept his hourly rate, likewise a subscription is needed first to use a SaaS application. This is to say, the pricing necessarily takes place *in advance* of service production.

If the pricing is to precede production – and thus consumption – an important consequence must be underlined: in some instances the total price of the service will remain uncertain until after its performance has actually taken place, e.g. the fee for a legal support in a litigation whose length and

[1] Defined as the instant when the consumer (contractually) commits to pay a certain sum in exchange of the provisioned service, not to be confused with the time of payment, which could occur at a later date.

[2] Ng (2007), "*The Pricing and Revenue Management of Services: A Strategic Approach*", Chapter 3.

outcomes are unknown. The same uncertainty hampers the predictability of the costs incurred providing the service, for they might depend on the customer's specific usage or on external factors. Nowadays, advanced price structures encompass mechanisms to lower this uncertainty: for example metered proxies that link charges to usage, aligning prices with costs[3], or price caps and flat rates to hedge the consumer's risk[4].

In addition to that, increasingly significant segments of the service industry – namely those related to new media, telecommunication and software – have a particular cost structure with high initial fixed costs and very low marginal ones, so that traditional pricing wisdom is turned obsolete. Price strategies need then embed discrimination techniques such as multi-part-tariffs, bundling and versioning, alongside more common volume- or time-based discounting. Given service perishability, the crucial problem for a service provider is matching demand and supply so as to avoid losing revenues from unused or insufficient capacity. The price structure is once again a key tool to manipulate demand appropriately.

Hence, the fundamental challenge for the pricing module is modeling the segmentation rules within the price structure, i.e. those rules determining when and how different consumers are charged different prices. Reviewing common segmenting practices[5] to assess how they impact the price structure design, we see that, given the variety of offerings in the service industry targeted by USDL, all of them continue to be relevant for our purpose. However, some of them do gain importance in the light of the increasing range of IT services available. For example, while the customization of a product or physical service may require re-deployment of resources, it's instead a relatively easy step for an IT provider to design a set of service variants responding to different user needs, and price them accordingly[6]. Moreover, while a manufacturing company is only rarely involved in the consumption phase, the contrary is true for a service company[7], and consequently a number of price drivers may relate to the consumer's usage.

For the model to be flexible and comprehensive enough to deal with the above-mentioned pricing complexity of today's service market, the cascading backbone of the pricing module is made up of three basic elements in a strict hierarchical structure: **PricePlans**, **PriceComponents**, and **PriceLevels**. This allows to readily model scenarios where alternative price plans may be assigned to an offered service or bundle, each plan possibly made up of multiple components and each component possibly varying its charges, either by specifying different levels or by adjusting them by means of premiums and discounts. All elements may then be constrained by segmenting conditions detailed in the so-called price fences[8], i.e. the criteria a customer must meet or the service limitations he/she needs to accept to qualify for a certain price.

2.1 Module Info

Parameters of the package that captures the module

- Namespace: *http://internet-of-services.com/usdl/modules/pricing*
- Name: *pricing*

[3] Stern (1986), "*The Strategic Value of Price Structure*", pg 28-30.
[4] Faruqui and Wood (2008), "*Quantifying the Benefits of Dynamic Pricing in the Mass Market*", pg 29.
[5] Nagle & Holden (2006), "*Segmented pricing using price fences to segment markets and capture value*"
[6] Shapiro and Varian (1998), "*Versioning: the smart way to sell information*", pg 106-110.
[7] For our purpose, we accept the assumption of inseparability of production and consumption as a distinctive characteristic of a service, commonly found in the literature. For a discussion about the validity of this point of view see "*Whither Services Marketing? In Search of a New Paradigm and Fresh Perspectives*" (Lovelock and Gummesson, 2004), pg. 21-23
[8] This terminology comes from "*The Strategy and tactics of Pricing*" (Nagle & Holden, 2006).

Note: Example fragments are provided for some of the classes. In order to improve readability they are presented in XML-based pseudo syntax. This is **NOT** the official USDL syntax, which is still under development. However, there currently exists a serialization format that is XMI-based and supported through a USDL editor developed by SAP Research.

The remainder of this section describes the classes and enumerations that are part of the package.

Figure 2 depicts a class diagram of the package.

2.2 Module Dependencies

In order to understand concepts from referenced modules in detail, it is recommended to go through the following documents describing other USDL modules:

- Core
- Foundation
- Participants

A quick overview of the concepts used in the Pricing module is given below to avoid jumping between documents.

Name	Type	Module	Description
NetworkProvisionedEntity	EClass	Core	The central concept of the USDL meta model that represents all entities provisioned into a service network, i.e. the service or service bundle to be priced
IdentifiableElement	Interface EClass	Foundation	Serves as the super type of all elements of USDL that can be uniquely identified, either globally or within a certain namespace
ElementDescription	EClass	Foundation	A generic concept that provides various information elements to describe USDL concepts
ExpressionElement	EClass	Foundation	A generic concept to model expression in an arbitrary expression language
Time	EClass	Foundation	Serves as the super type of the different concrete time concepts
Location	EClass	Foundation	Serves as the super type of the different concrete location concepts
TargetConsumer	EClass	Participants	Set of user groups at which the NetworkProvisionedEntity is targeted

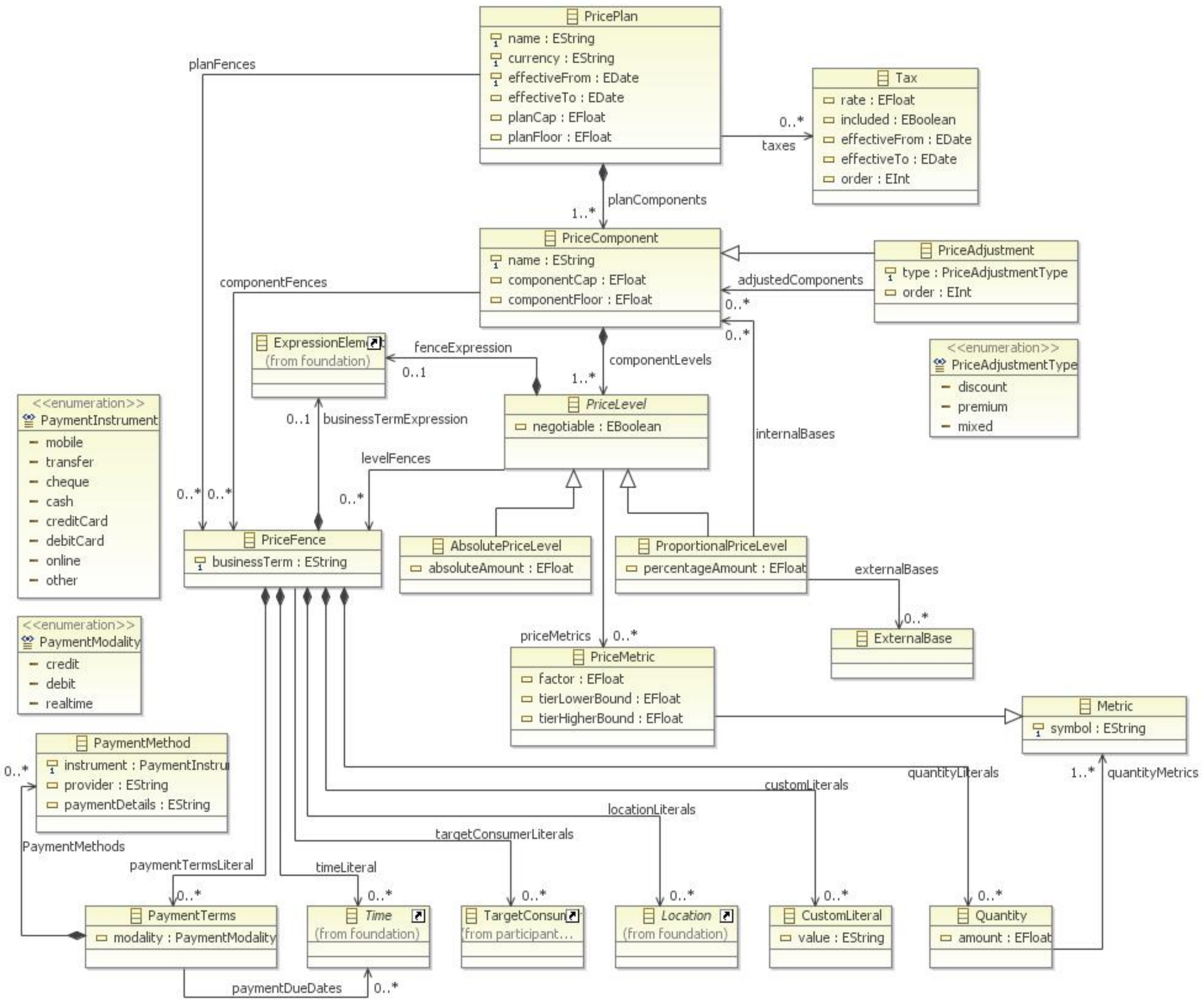

Figure 2 Class diagram of the package that captures the Pricing module

2.3 PricePlan

A **PricePlan** is a set of charges associated with the **NetworkProvisionedEntity**. Alternative sets of fees (i.e. alternative **PricePlans**) to 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 one (a common practice in the telecom 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.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

PricePlan			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	A unique, meaningful name for the PricePlan.
currency	EString	1	The currency for all price amounts within this PricePlan. Permitted values are the ISO 4217 currency codes.
planCap	EFloat	0..1	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. For example, it may be used to set an upper limit in a strictly usage-based plan.
planFloor	EFloat	0..1	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. For example, it may be used to set a lower limit to discounts that may result in an excessively low price.
effectiveFrom	EDate	1	The date and time when the PricePlan becomes effective.
effectiveTo	EDate	0..1	The date and time when the PricePlan ceases to be effective.
Relations			
Name	Target	Cardinality	Description
planComponents	PriceComponent	1..*	The set of price components that constitute the PricePlan.
fenceExpression	ExpressionElement	0..1	An expression language must be used to define the logical relation between multiple price fences related to the same PricePlan. This relation is not shown in the diagram for better readability.
description	ElementDescription	0..1	A description of the PricePlan.
taxes	Tax	0..*	The set of taxes the PricePlan is subjected to.
planFences	PriceFence	0..*	The set of conditions that must be evaluated before the PricePlan is granted.
Examples (in pseudo concrete syntax)			
<PricePlan name="*Educational*" currency=*”EUR” planCap=”200.00”*>...</PricePlan> <PricePlan name="*Professional*" currency=*” EUR” planFloor=”400.00”*>...</PricePlan> <PricePlan name="*Enterprise*" currency=*” EUR” planFloor=”1000.00”*>...</PricePlan>			
<PricePlan name="*updatedPlan*" effectiveFrom=”07-01-2010” effectiveTo=”06-31-2011” currency=*”USD”*>...</PricePlan> <PricePlan name="*currentPlan*" effectiveFrom=”07-01-2009” effectiveFrom=”06-31-2010” currency=*”USD”*>...</PricePlan>			

2.4 PriceComponent

PriceComponents are fees a **PricePlan** may encompass. Components within the same plan are summed together in order to get the total 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), option 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).

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

PriceComponent			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	A unique, meaningful name for the PriceComponent.
componentCap	EFloat	0..1	Providing this maximum PriceComponent value prevents the component final price from exceeding it, regardless of its levels and the parameters they are indexed to. For example, it may be used to set an upper limit for a component whose levels vary with usage.
componentFloor	EFloat	0..1	Providing this minimum PriceComponent value prevents the component final price from falling below it, regardless of its levels and the parameters they are indexed to. For example, it may be used to set a lower limit for a component whose levels vary with usage.
Relations			
Name	Target	Cardinality	Description
componentLevels	PriceLevel	1..*	The set of PriceLevels the PriceComponent may assume.
componentFences	PriceFence	0..*	The set of conditions that must be evaluated before the PriceComponent is considered active and increment the total price for the service.
fenceExpression	ExpressionElement	0..1	An expression language must be used to define the logical relation between multiple price fences related to the same PriceComponent. This relation is not shown in the diagram for better readability.
description	ElementDescription	0..1	A description of the PriceComponent.
Examples (in pseudo concrete syntax)			
<PricePlan name="*Subscription*"> **<PriceComponent name="*PeriodicalFee*">...</PriceComponent>** </PricePlan> <PricePlan name="*PayAsYouGo*"> **<PriceComponent name="*UsageBasedFee*">...</PriceComponent>** </PricePlan>			
<PricePlan name="*CompositePlan*"> **<PriceComponent name="*PeriodicalFee*">...</PriceComponent>** **<PriceComponent name="*UsageBasedFee*">...</PriceComponent>** </PricePlan>			

2.5 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 the customers' behavior and aligning usage with capacity or incurred costs (just like utilities do by offering different electricity rates for different times of day).

- Ecore Type: Abstract EClass
- Interfaces: N/A
- Superclass: N/A

PriceLevel			
Attributes			
Name	Type	Cardinality	Description
negotiable	EBoolean	0..1	This attribute specifies if negotiation of this PriceLevel is allowed.
Relations			
Name	Target	Cardinality	Description
levelFences	PriceFence	0..*	The set of conditions that must be evaluated to decide which PriceLevel applies.
fenceExpression	ExpressionElement	0..1	An expression language must be used to define the logical relation between multiple price fences related to the same PriceLevel.
description	ElementDescription	0..1	A description of the PriceLevel.
priceMetrics	PriceMetric	0..*	The set of price metrics by which the usage of the service is measured.
Examples (in pseudo concrete syntax)			

```
<PricePlan name="Subscription" currency="EUR">
    <PriceComponent name="PeriodicalFee">
        <AbsolutePriceLevel absoluteAmount="30.00">...</ AbsolutePriceLevel >
    </PriceComponent>
</PricePlan>
<PricePlan name="PayAsYouGo" currency="EUR”>
    <PriceComponent name="UsageBasedFee">
        <AbsolutePriceLevel absoluteAmount="0.50">...</ AbsolutePriceLevel >
    </PriceComponent>
</PricePlan>
```

```
<PricePlan name="CompositePlan" currency="EUR">
    <PriceComponent name="PeriodicalFee">
        <AbsolutePriceLevel absoluteAmount="20.00">...</ AbsolutePriceLevel >
    </PriceComponent>
    <PriceComponent name="UsageBasedFee">
        <AbsolutePriceLevel absoluteAmount="0.25">...</ AbsolutePriceLevel >
    </PriceComponent>
</PricePlan>
```

2.6 AbsolutePriceLevel

An **AbsolutePriceLevel** specifies a price amount as an absolute monetary quantity.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: PriceLevel

AbsolutePriceLevel

Attributes

Name	Type	Cardinality	Description
absoluteAmount	EFloat	0..1	Price amount expressed as a monetary quantity.

Examples (in pseudo concrete syntax)

```
<PricePlan name="PlumberCharges" currency="GBP">
   <PriceComponent name=”CallFee">
      <AbsolutePriceLevel absoluteAmount="80.00" negotiable=”FALSE”>...</ AbsolutePriceLevel >
  </PriceComponent>
   <PriceComponent name="HourlyFee">
      <AbsolutePriceLevel absoluteAmount="30.00" negotiable=”FALSE”>...</ AbsolutePriceLevel >
  </PriceComponent>
</PricePlan>
```

2.7 ProportionalPriceLevel

A **ProportionalPriceLevel** specifies a price amount as a percentage of another monetary quantity referenced by an internal or external base. In case more than one base is referenced, the **PriceLevel** is meant as percentage of the sum of all referenced bases. At least one base (either internal or external) needs to be specified.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: PriceLevel

ProportionalPriceLevel

Attributes

Name	Type	Cardinality	Description
percentageAmount	EFloat	0..1	Price amount expressed as percentage of a monetary quantity.

Relations

Name	Target	Cardinality	Description
internalBases	PriceComponent	0..*	The PriceComponent (or set of PriceComponents) the ProportionalPriceLevel is a percentage of. In case multiple PriceComponents are referenced, the total sum of those referenced PriceComponents is considered as the percentage base.
externalBases	ExternalBase	0..*	A monetary quantity which is not represented by a price element within the model. In case multiple ExternalBases are referenced, the total sum of those referenced ExternalBases is considered as the percentage base.

Examples (in pseudo concrete syntax)

```
<PricePlan name="TransactionProcessing" currency="USD">
   <PriceComponent name="TrasactionValueFee">
      <ProportionalPriceLevel percentageAmount="0.03"> <!-- 3% -->
         <ExternalBase>Transaction Amount</ExternalBase> <!—of the transaction value -->
      </ProportionalPriceLevel>
   </PriceComponent>
</PricePlan>
```

```
<PricePlan name="PlanWithDiscount" currency="EUR">

    <PriceComponent name="BasicPrice">
        <AbsolutePriceLevel percentageAmount="100.00">...</AbsolutePriceLevel>
    </PriceComponent>

    <PriceAdjustment type="Discount">
        <ProportionalPriceLevel percentageAmount="0.10"> <!-- 10% discount -->
                <internalBases>BasicPrice</internalBases>
        </ProportionalPriceLevel>
    </PriceAdjustment>

</PricePlan>
```

2.8 ExternalBase

ExternalBase captures a monetary quantity not represented by an element within any of the USDL modules. Examples are pricing schemes based on the value of an atomic service transaction (i.e. the price is a percentage of the value of the processed transaction), financial data of the requesting business partner (e.g. the price is a percentage of the requestor's overall revenues) or profits obtained from third parties by exploiting the underlying service (e.g. the price is a percentage of the added value).

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

ExternalBase			
Relations			
Name	Target	Cardinality	Description
description	Element Description	0..1	A description of the external base.
Examples (in pseudo concrete syntax)			

```
<PricePlan name="TransactionProcessing" currency="USD">
    <PriceComponent name="TransactionValueFee">
        <ProportionalPriceLevel percentageAmount="0.03"> <!-- 3% -->
            <ExternalBase>Transaction Amount</ExternalBase> <!--of the transaction value -->
        </ProportionalPriceLevel>
    </PriceComponent>
</PricePlan>
```

2.9 PriceAdjustment

PriceAdjustment is used to express a charge adjusting the final price for the service: for example a promotional discount, an area surcharge or a premium for a more comprehensive support service.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: PriceComponent

PriceAdjustment			
Attributes			
Name	Type	Cardinality	Description
type	Price AdjustmentType	1	The type of PriceAdjustment. It determines if the PriceAdjustment amounts are to be summed to the final price or subtracted from it.
order	EInt	0..1	The processing order of adjustments, to be used when more than one price adjustment are defined and the final outcome depends on the calculation sequence (e.g. when one adjustment is expressed in percentage and one in absolute monetary terms).
Relations			
Name	Target	Cardinality	Description
adjustedComponents	PriceComponent	0..*	The PriceComponents affected by the PriceAdjustment.
Examples (in pseudo concrete syntax)			

```
<PricePlan name="PlanWithDiscount" currency="EUR">

    <PriceComponent name="BasicPrice">
        <AbsolutePriceLevel percentageAmount="100.00">...</AbsolutePriceLevel>
    </PriceComponent>

    <PriceAdjustment type="Discount">
        <ProportionalPriceLevel percentageAmount="0.10"> <!-- 10% discount -->
            <internalBases> BasicPrice </internalBases>
        </ProportionalPriceLevel>
    </PriceAdjustment>

</PricePlan>
```

2.10 PriceMetric

PriceMetric represents the unit of measurement by which the customer is charged for the consumption of the service or bundle.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: Metric

PriceMetric			
Attributes			
Name	Type	Cardinality	Description
factor	EFloat	0..1	The minimum block of units that is priced, i.e. the step increase the price metric may take. E.g. a Gigabyte metric could be equivalently expressed as Megabyte with a factor of 1024. It may also be a fraction, e.g. a professional service priced with hourly rates but charged in 15 minutes increments (factor would then be 0.25).
tierLowerBound	EFloat	0..1	In case of tiered pricing it specifies the (inclusive) lower limit of the range of units the current PriceLevel applies to. This means that only the consumption units falling within this

			range will be priced according to this PriceLevel. For example, with a tiered pricing of 1€ within a range of 1 to 10 hits and 0.5€ for a usage range higher than 10 hits, a customer consuming 12 hits would get a total price of 11€ (10 x 1€ + 2 x 0.5€).
tierHigherBound	EFloat	0..1	In case of tiered pricing it specifies the (inclusive) upper limit of the range of units the current PriceLevel applies to. This means that only the consumption units falling within this range will be priced according to this PriceLevel. For example, with a tiered pricing of 1€ within a range of 1 to 10 hits and 0.5€ for a usage range higher than 10 hits, a customer consuming 12 hits would get a total price of 11€ (10 x 1€ + 2 x 0.5€).

Examples (in pseudo concrete syntax)

```
<PricePlan name="tieredPricing" currency="EUR">
    <PriceComponent name="PerHitFee-tier1">
        <PriceLevel amount="1.00">
            <PriceMetric tierLowerBound="1" tierHigherBound="10" >hit</PriceMetric>
            <!--  i.e. only the first 10 hits will be priced 1€ each -->
        </PriceLevel>
    </PriceComponent>
    <PriceComponent name="PerHitFee-tier2">
        <PriceLevel amount="0.50">
            <PriceMetric tierLowerBound="11" >hit</PriceMetric>
            <!-- i.e. all the hits except the first 10 will be priced 0,50€ each -->
        </PriceLevel>
        <PriceFence businessTerm="hit">
            <Quantity>more than 10</Quantity>
        </PriceFence>
    </PriceComponent>
</PricePlan>
```

2.11 Metric

Metric represents a measuring unit used to meter the service usage.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

Metric			
Attributes			
Name	Type	Cardinality	Description
symbol	EString	1	The symbol or code identifying the unit of measurement.
Relations			
Name	Target	Cardinality	Description
description	Element Description	0..1	A description of the metric.

2.12 Quantity

Quantity represents a generic numerical value used as a literal in a **PriceFence** evaluation.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

Quantity			
Attributes			
Name	Type	Cardinality	Description
amount	EFloat	0..1	The quantity value to be used for volume/usage comparison within a price fence.
Relations			
Name	Target	Cardinality	Description
quantityMetrics	Metric	1..*	The unit of measurement used to meter a quantity condition.
description	Element Description	0..1	A description of the quantity.

2.13 PaymentMethod

PaymentMethod captures the means by which the customer may pay the specified price.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

PaymentMethod			
Attributes			
Name	Type	Cardinality	Description
instrument	Payment Instrument	1	The payment instrument required to initiate the payment.
provider	EString	0..1	The provider of the payment infrastructure and electronic clearing methods.
paymentDetails	EString	0..1	Other details of the payment method.

2.14 PaymentTerms

PaymentTerms is used to specify the terms the customers must adhere to when paying the specified price.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

PaymentTerms			
Attributes			
Name	Type	Cardinality	Description
modality	Payment Modality	0..1	This attribute specifies when the payment step takes place in the process flow, i.e. before consumption, afterwards or in real-time.

Relations			
Name	Target	Cardinality	Description
paymentMethods	PaymentMethod	0..*	The set of allowed payment methods.
paymentDueDates	Time	0..*	The set of dates when the payment is due.

2.15 PriceFence

PriceFence represents a conditional expression evaluated to determine if the price element (whether a **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). An expression language (e.g. XPath[9], OCL[10]) is required to fully specify the semantics of the condition statement.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

PriceFence			
Attributes			
Name	Type	Cardinality	Description
businessTerm	EString	1	A business relevant entity that's under scrutiny in the price fence. It can refer to different dimensions of the service consumption process: time (e.g. time of delivery, time of request), volume (e.g. purchased quantity, consumed quantity), space (e.g. delivery location, location size), consumer identity (e.g. member of a certain group or class, new or with an established and privileged relationship to the provider), payment (when and how is the payment supposed to be performed).
Relations			
Name	Target	Cardinality	Description
businessTermExpression	Expression Element	0..1	An expression language must be used to define the syntax of the price fence (how the business term is evaluated against the literals).
description	Element Description	0..1	A description of the price fence.
paymentTermsLiteral	PaymentTerms	0..*	Values that account for the payment terms options and constraints.
timeLiteral	Time	0..*	Values that account for the temporal aspect of service consumption or production (e.g. time of request, time of delivery)
targetConsumerLiterals	TargetConsumer	0..*	Values that account for the consumer identity (e.g. new or VIP customer).
locationLiterals	Location	0..*	Values that account for the spatial aspect of service consumption or production (e.g. delivery location, request location).
customLiterals	CustomLiteral	0..*	Generic container for custom values.
quantityLiterals	Quantity	0..*	Values that account for the quantity aspect of service consumption or production (e.g. volume of

[9] World Wide Web Consortium (W3C), XML Path Language (XPath) 2.0, W3C Recommendation, 23 January 2007.
[10] Object Management Group (OMG), Object Constraint Language OMG Available Specification Version 2.0, May 2006.

			usage units consumed or ordered).

Examples (in pseudo concrete syntax)

```
<!-- price fences determine which of the two levels applies -->
<PricePlan name="dynamicPricing" currency="EUR">
    <PriceComponent name="messageFee">
        <PriceLevel amount="0.45">
            <PriceMetric>sent message</PriceMetric>
            <PriceFence businessTerm="timeOfRequest">
                < timeLiteral >9:01-18:00</ timeLiteral >
            </PriceFence>
        </PriceLevel>
        <PriceLevel amount="0.15">
            <PriceMetric>sent message</PriceMetric>
            <PriceFence businessTerm="timeOfRequest">
                <timeLiteral>18:01-9:00</ timeLiteral >
            </PriceFence>
        </PriceLevel>
    </PriceComponent>
</PricePlan>
```

```
<!-- price fences determine when the discount applies -->
<PricePlan name="PlanWithDiscount" currency="EUR">

    <PriceComponent name="BasicPrice">
        <AbsolutePriceLevel percentageAmount="100.00">...</AbsolutePriceLevel>
    </PriceComponent>

    <PriceAdjustment type="Discount">
        <ProportionalPriceLevel percentageAmount="0.10"> <!-- 10% discount -->
                <internalBases>BasicPrice</internalBases>
        </ProportionalPriceLevel>
        <PriceFence businessTerm="purchaseVolume">
            <Quantity>more than 100</Quantity>
        </PriceFence>
    </PriceAdjustment>

</PricePlan>
```

2.16 CustomLiteral

CustomLiteral captures a custom value to be used in conditional evaluation within **PriceFences**.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

CustomLiteral			
Attributes			
Name	Type	Cardinality	Description
value	EString	0..1	Custom value to be used in a conditional evaluation.

Relations			
Name	Target	Cardinality	Description
description	Element Description	0..1	A description of the custom literal.

2.17 Tax

Tax represents a tax the **PricePlan** is subjected to.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

Tax			
Attributes			
Name	Type	Cardinality	Description
rate	EFloat	0..1	The tax percentage rate.
included	EBoolean	0..1	This attribute explicitly states whether the tax rate specified in this tax item is already included in the price amounts of the price plan (i.e. the given prices are gross) or need to be separately added (i.e. the given prices are net).
effectiveFrom	EDate	0..1	The date and time when the tax becomes effective.
effectiveTo	EDate	0..1	The date and time when the tax ceases to be effective.
order	EInt	0..1	Determines the taxation sequence in case multiple concurrent taxes are defined.
Relations			
Name	Target	Cardinality	Description
description	Element Description	0..1	A description of the tax.

2.18 PaymentInstrument

PaymentInstrument is used to indicate available payment instruments.

- Ecore Type: EEnum

PaymentInstrument	
Items	
Name	Description
creditCard	A card granting a line of credit to the holder
debitCard	A card enabling the holder to have his purchases directly charged to his account
mobile	Mobile handset based payments
transfer	Movement of funds from one bank account to another
cheque	A written order from one party to another requiring to pay a specified sum
cash	Direct exchange of money in the form of coins or notes
online	Payment through the Internet managed by an on-line payment service provider.
other	Other diverse payment instruments

2.19 PaymentModality

PaymentModality indicates the payment policy with regard to the relative position in the process flow of the consumption and payment phases.

- Ecore Type: EEnum

PaymentModality	
Items	
Name	Description
credit	Payment takes place after consumption. This policy may entail the collection of usage information over a certain time period in order to charge the consumer accordingly.
debit	Payment takes place before consumption. This policy may entail the collection of usage information in order to allow the consumer to make use of resources up to the paid amount and avoid system abuse.
realtime	Payment takes place at the time of resource usage. This policy entails real-time metering and accounting, plus possibly the availability of micropayment method.

2.20 PriceAdjustmentType

PriceAdjustmentType indicates the type of **PriceAdjustment**. It determines if the **PriceAdjustment** amounts are to be summed to the final price or subtracted from it.

- Ecore Type: EEnum

PriceAdjustmentType	
Items	
Name	Description
premium	An adjustment of type premium will increase the final price, i.e. it adds a positive amount to the final price.
discount	An adjustment of type discount will decrease the final price, i.e. it subtracts a positive amount from the final price.
mixed	An adjustment of type mixed may increase or decrease the final price depending on the sign of its PriceLevel amounts, i.e. a negative amount will decrease the final price while a positive one will increase it.