MIRABEL

Micro-Request-Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution

Specific Targeted Research Project: 248195

D2.3 Final data model, specification of request and negotiation messages and contracts

Work package 2

Leading partner: TNO

July, 2011

Version 1.0

DOCUMENT INFORMATION	
ID	D2.3 Data model, specification of request and negotiation messages and contracts
Work Package(s)	Work package 2 Data Specification
Type	Report
Dissemination	Public
Version	1.0
Date	July 29th 2011
Author(s)	M.J Konsman (TNO), F.J. Rumph (TNO)
Reviewer(s)	Gregor Černe (INEA)

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. Copyright 2011 by TNO.

Table of Contents

1 Summary

The aim of the MIRABEL project is to exploit potential flexibility in both demand for and generation of electricity to compensate for the intermittent nature of Renewable Energy Sources (RES). For instance by postponing certain electricity demand until energy from a RES (e.g. solar panel) becomes available.

In order to leverage flexibility in demand and generation means for expressing and communicating this flexibility are needed. For this purpose, this MIRABEL deliverable provides a data model. This document contains the final version of the data model, a previous version of the data model was presented in deliverable D2.2 which was published in December 2010.

The aim of the data model is twofold. Firstly it serves as a means of alignment between the various work packages ensuring proper co-operation of all the components involved in the MIRABEL Electricity Data Management Systems (EDMS). Secondly there is an aim for standardization; the data model (and messages specifications derived from this model) is input for an active standardization process that is the subject of WP7.

The central concept of this data model is `FlexEnergy`; which is used to describe flexibility in consumption or production of energy. It is crucial for the project to have a common view on `FlexEnergy` as it plays an important role in the work packages on aggregation and scheduling. The model that is specified in this deliverable is the result of various discussions within the project as well as feedback from the first version of the model.

Apart from the update of the existing model this deliverable also contains an XML schema that was derived from the data model. The purpose of the XML schema is to facilitate the implementation of the data model.

2 Introduction

The aim of the data model that is being developed as part of the MIRABEL project is twofold. First it serves as a means of alignment between the various work packages ensuring proper co-operation of all the components involved in the MIRABEL Electricity Data Management Systems (EDMS).

Second there is the standardization aim; the data model (and the messages that are derived from it) is input for an active standardization process. For standardization purposes it is important that the model is as generic as possible making it suitable for a wide range of applications.

In December 2010 a first version of the data model was presented in deliverable D2.2. This deliverable D2.3 contains the final version of the model. In the past months the concepts of the initial version of the data model were extensively used by several other work packages such as WP1 (Architecture & Process Model) [MČB+11], WP3 (Data Collection & Analysis) [SKT+11] and WP5 (Scheduling & Negotiation) [TDF+11]. Their experiences with the data model have been fed back to the Data Specification WP and were incorporated in this final version.

Apart from the update of the existing model this deliverable also contains an XML schema that was derived from the data model. The purpose of the XML schema is to facilitate the implementation of the data model.

2.1 Relation to objectives in Description of Work

The terminology in this deliverable differs from the terminology that was originally used in the Description of Work. The table clarifies the relation between the terms used in the DoW and those in this deliverable.

Terminology in DoW	Terminology in deliverable D2.3
Micro request	FlexOffer
Assignment	FlexOfferAssignment
Exceptions	Exceptions were not modeled as they were not deemed necessary
Message protocol	FlexOffer business process
Contracts	FlexOfferAcceptance in combination with FlexOfferAssignment

2.2 Modeling approach

Deliverable D2.1 [KonRum10] of this project elaborated on various data modeling techniques relevant for the MIRABEL project. It also investigated existing data models in the energy sector to see which models could be reused. The conclusions was to use the Common Information Model [CIM09] by the International Electrotechnical Commission and ebIX models [EMD09][ebiXCuS] as a basis for the MIRABEL model and to adopt UN/CEFACT's Modeling Methodology (UMM) as the modeling technique of choice.

2.3 Document structure

Chapter 3 describes the data model. It starts with explaining the modeling conventions followed by the data model itself. The model itself is centered on the exploitation of flexibility in energy consumption and supply; the `FlexEnergy` concept. The concepts are formulated as generic as possible because of the focus on standardization. However for the MIRABEL demonstration implementation specific constraints will be added to limit the degrees of freedom. These constraints can be found in paragraph 3.2.3. The chapter also features examples of `FlexEnergy` usage.

Chapter 5 discusses the messages which use the data model for conveying information. It describes the business process that is associated with offering, accepting and assigning flexibility in consumption and production of energy.

In chapter 5 XML Schema Definitions are provided based on the data model and messages specified in sections 3 and 4 respectively for implementation purposes.

The Common Information Model (CIM) [CIM09] by the International Electrotechnical Committee (IEC) is being reused where possible in our data model. In Appendix A the elements adopted from CIM are listed.

3 Data model

This section describes the data model used within the MIRABEL project. First in section 3.1 the conventions which are used in this model are described. Second in section 3.2 an introduction to the model is provided, indicating the scope of the model. Sections 3.3 through 3.5 provide specification of the model. Section 3.6 then provides initial limitations to the use of the model in the software developed in the project. Finally section 3.7 provides a number of examples for use of the model. Chapter 4 builds upon the data model for the description of messages.

3.1 Modeling conventions

This section describes a number of conventions applied in the description of the MIRABEL data model.

3.1.1 Diagramming technique

UML Class diagramming is used for the description of the data model. Although a data/information model is not part of UN/CEFACT’s Modeling Methodology (UMM, the methodology of choice for this deliverable) it is a requirement (see “Task T2.1: Data modeling” in the DoW) for MIRABEL to develop this first.

The data model serves as a solid basis from which UMM artifacts such as Business Entity Views can be derived. By having the data model as common ground the consistency between the various Business Entity Views (see chapter 4) is ensured.

In some cases it was not possible to describe certain aspects of the data models with diagramming techniques alone. This is especially true for constraints. These were expressed using Object Constraint Language (OCL), which is part of UML.

Please note that within the data model specified, the derived attribute construct is used. An attribute name preceded by a slash denotes that this a derived attribute that is determined by combining other fixed attributes. The choice for a derived attribute is made when it is deemed convenient for the user of the data model to have the information explicitly available.

In order to keep the data model readable associations between different classes have been labeled. The UML navigability construct was used to indicate the direction of the labels; it does not impose any limits on the structure of the data model.

3.1.2 Naming conventions

The naming conventions used in the data model are as followed:

- package names must be in lower case;
- class names must be in camel case[1], first letter in upper case;
- field and method names must be in camel case, first letter in lower case;

[1] Camel case is the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound and the first letter is either upper or lower case (source: Wikipedia).

- when names from the data model are used, they are represented as follows: `EnergyAmount`.

3.2 Model introduction

The data model that is presented here describes different possibilities of shifting energy demand or supply. The model is centered on the `FlexEnergy` concept (see section 3.4). It features various options to describe such flexibility both in an energetic and a financial way. It should be noted that it is highly unlikely for specific instances to use all of the available options. In most cases only one way for expressing flexibility is used.

Throughout the model several data types are used that are derived from the Common Information Model by the IEC, such as `RealEnergy`. See Appendix A for a description of the CIM classes used.

3.3 Time series related utility classes

This paragraph defines a number of utility classes which are used throughout the data model specified in this deliverable for expressing time series related concepts.

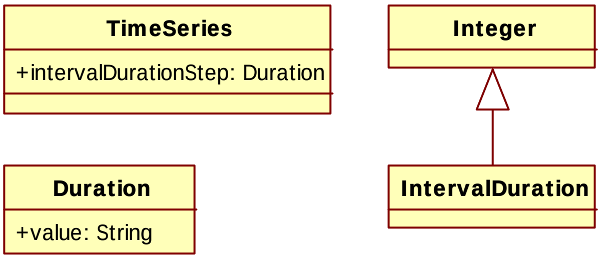


Figure 1: Utility classes

`TimeSeries`		
Description:	This is a parent class from which more specific `TimeSeries` classes can be derived.	
Attributes:	`interval DurationStep`	A `TimeSeries` consists of a series of intervals. The `intervalDurationStep` attribute indicates the step size of the length of such an interval. The length of an `interval` should be a multiple of `intervalDurationStep` (see the `EnergyConstraintInterval` explanation on page 11 for more details).

`IntervalDuration`	
Description:	Expresses a multitude (as an integer) of the `intervalDurationStep` attribute. This interval duration must be referred to when using this class; e.g. elements in a `TimeSeries` refer to the `intervalDurationStep` attribute of that class.

`Duration`		
Description:	Duration represents a period of time. The value space of duration is a six-dimensional space where the coordinates designate the Gregorian year, month, day, hour, minute, and second components defined in § 5.5.3.2 of [ISO8601].	
Attributes:	`Value`	The lexical representation for duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where `nY` represents the number of years, `nM` the number of months, `nD` the number of days, 'T' is the date/time separator, `nH` the number of hours, `nM` the number of minutes and `nS` the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

3.4 Classes for negotiating flexibility in energy supply and demand

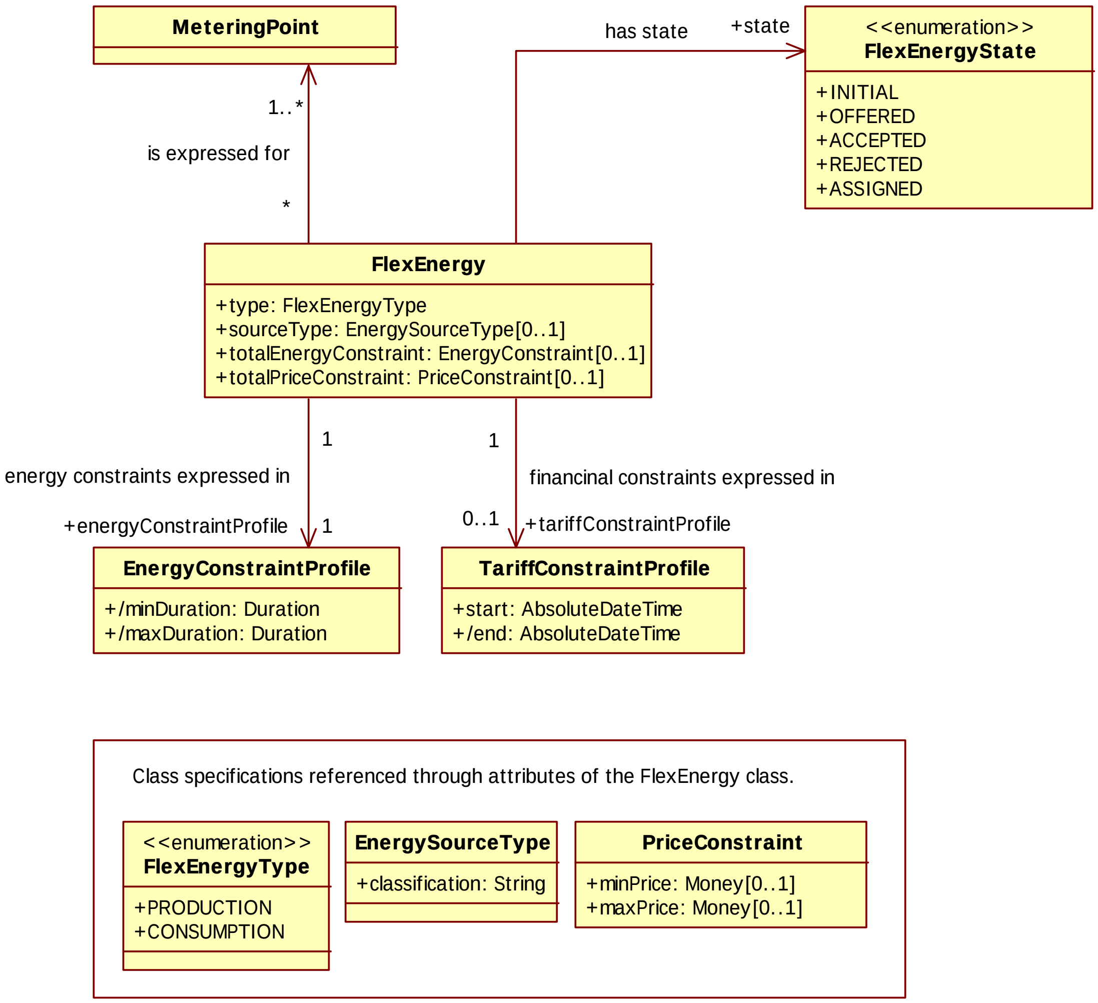


Figure 2: `FlexEnergy`

FlexEnergy		
Description:	This class is the central class to describe the flexibility in energy. It is associated with other classes that express constraints in terms of time, energy and costs. Time shifts can be expressed through the `EnergyConstraintProfile` class. This class also serves as the basis for a flexible energy profile. Financial aspects can be expressed with the `TariffConstraintProfile` and the `PriceConstraint` classes. The flexibility expressed in `FlexEnergy` objects refers to one `MeteringPoint` instance or multiple instances if the `FlexEnergy` object reflects aggregated flexibility of these metering points.	
Attributes:	`type`	Indicates whether the `FlexEnergy` object relates to either production or consumption of electricity.
	`sourceType`	This attribute indicates, in case of production, the classification of the electricity source. E.g. it may indicate that photovoltaics is used or combined heat and power. The scheme for classification is beyond the scope of this document.
	`totalEnergy Constraint`	With this attribute limits can be put on the overall amount of energy that is expressed in a `FlexEnergy` object. The attribute is an array of the type `EnergyConstraint`, its definition can be found on page 12.
	`totalPrice Constraint`	This attribute limits the total monetary amount to be paid or earned for the electricity consumed or produced related to this `FlexEnergy` object. Note a maximum price is used in case of consumption and a minimum price is used in case of production.

FlexEnergyState		
Description:	This enumeration contains the valid states a `FlexEnergy` object can be in. Figure 3 provides a graphical view of the states of a `FlexEnergy` object.	
Enumeration literals:	`INITIAL`	The initial state of a `FlexEnergy` object. The object has been constructed but not yet offered.
	`OFFERED`	The `FlexEnergy` object is offered; i.e. a `FlexOffer` (c.f. section 4.2) object is constructed and associated with the `FlexEnergy` object and is communicated.
	`ACCEPTED`	The `FlexEnergy` object is accepted; the flexibilities expressed in this `FlexEnergy` object will be used.
	`REJECTED`	The `FlexEnergy` object is rejected; the flexibilities expressed in this `FlexEnergy` object will not be used.
	`ASSIGNED`	An assignment is expressed for the `FlexEnergy` object, i.e. a ‘choice’ is made for all the flexibilities expressed in the `FlexEnergy` object and this ‘choice’ (a `FlexEnergySchedule` object) is communicated.

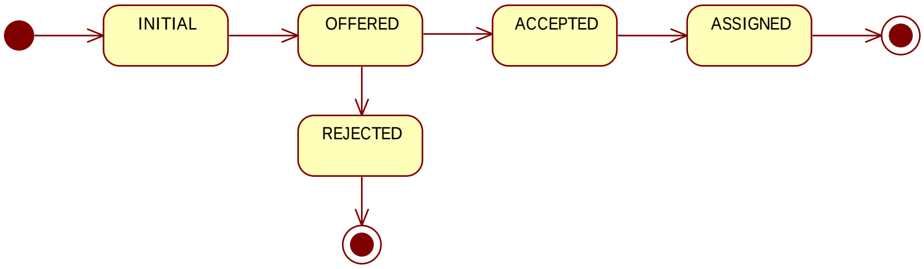


Figure 3: `FlexEnergyState`

`FlexEnergyType`		
Description:	This class is used to indicate whether a FlexOffer is about producing or consuming electricity	
Enumeration literals:	`PRODUCTION`	This literal indicates a FlexOffer that describes flexibilities with regard to the production of electricity.
	`CONSUMPTION`	This literal indicates a FlexOffer that describes flexibilities with regard to the consumption of electricity.

`EnergySourceType`		
Description:	The EnergySourceType class can be used to indicate the source that produces electricity, e.g. photovoltaic or wind. This is useful when regulations or policies are in place that treat particular energy sources differently.	
Attributes:	`classification`	This attribute is used to indicate the energy source. It is a free format String. An enumeration was also considered but since it not possible to come up with an exhaustive list of energy sources for the coming years a String was chosen.

`PriceConstraint`		
Description:	This class is optionally associated with the `FlexEnergy` class. It can be used to express an overall constraint to a `FlexEnergy` object on the price paid or earned for the energy exchanged in association to the `FlexEnergy` object.	
Attributes:	`minPrice`	The minimum price for the exchanged energy associated with the `FlexEnergy` object; only applicable in case of production.
	`maxPrice`	The maximum price for the exchanged energy associated with the `FlexEnergy` object; only applicable in case of consumption.

`MeteringPoint`

Description:	A point in the grid to which production and/or consumption is connected. C.f. the modeling and description of `MeteringPoint` in [ENTS09].

`EnergyConstraintProfile`		
Description:	The `EnergyConstraintProfile` can be used to describe an energy profile. It is associated with an ordered sequence of `EnergyConstraintIntervals`. `EnergyConstraintProfile` is derived from the `TimeSeries` class as can be seen in the diagram in Figure 4.	

Attributes:	`/minDuration`	A derived attribute indicating the minimum duration of an `EnergyConstraintProfile`. This minimum duration is based on the minimum duration of the `EnergyConstraintInterval` objects referred to by the `EnergyConstraintProfile`. This duration can be calculated with the following algorithm:
		`minDuration = 0`
		`for i in energyConstraintProfile.intervals {`
		`      if i.minDuration->empty() == false then`
		`            minDuration += i.minDuration *`
		`                  energyConstraintProfile`
		`                  .intervalDurationStep`
		`      endif`
		`}`

	`/maxDuration`	A derived attribute indicating the maximum duration of an `EnergyConstraintProfile`. This maximum duration is based on the `startAfter` attribute of the first `EnergyConstraintInterval` and the `endBefore` of the last interval:
		`maxDuration =`
		`  self.intervals->last().endBefore –`
		` self.intervals->first().startAfter`

Constraints:	`self.intervals->first().startAfter->notEmpty()`
	`and self.intervals->last().endBefore->notEmpty()`
	This constraint entails that the first interval must specify `startAfter` and the last interval must specify `endBefore`.	

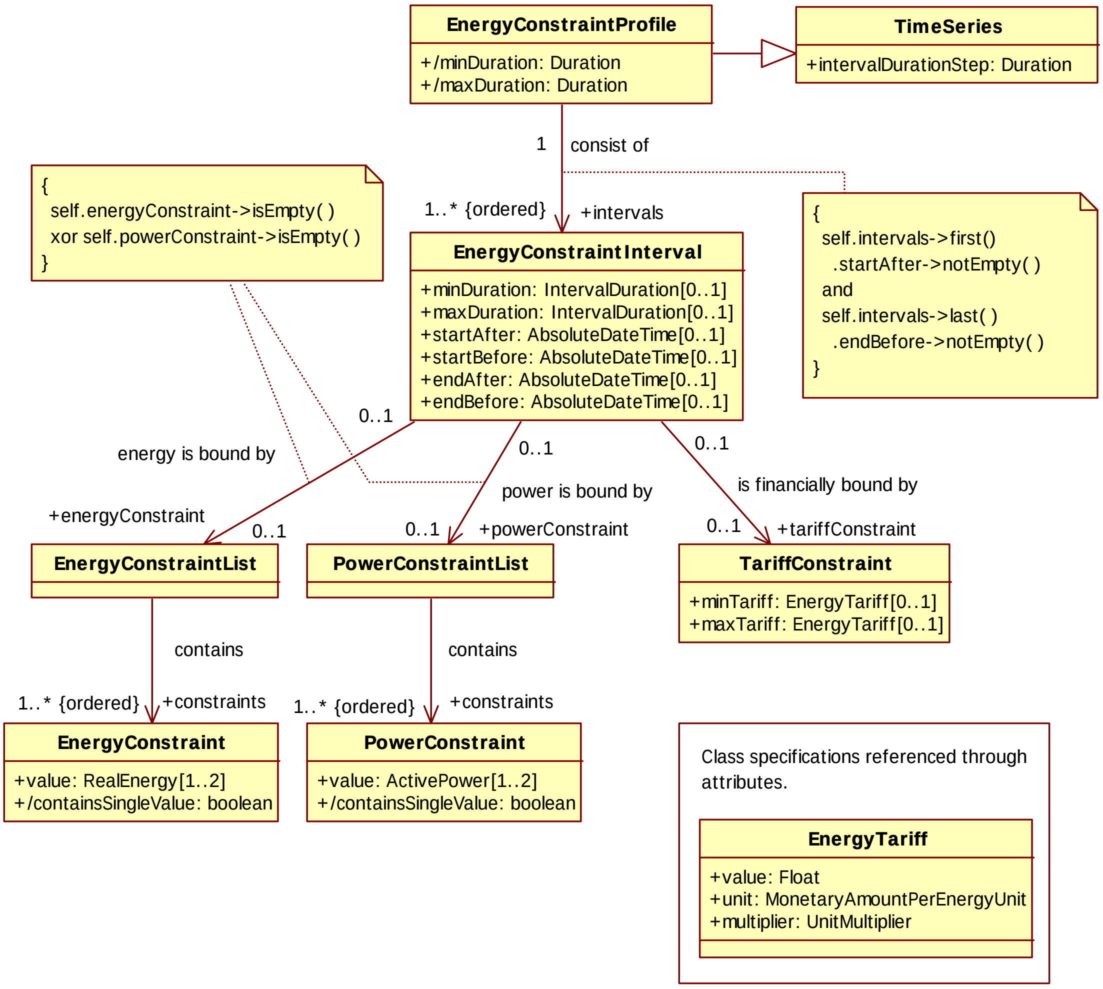


Figure 4: `EnergyConstraintProfile`

`EnergyConstraintInterval`		
Description:	The `EnergyConstraintInterval` represents an interval of time that is part of an `EnergyConstraintProfile`. The duration of an `EnergyConstraintInterval` may be flexible and is expressed through the `minDuration` and `maxDuration` attributes.	
Attributes:	`minDuration`	This optional attribute represents the shortest possible duration for this `EnergyConstraintInterval`. The attribute is of the `IntervalDuration` type, which is based on an integer, and expresses a multitude of the `intervalDurationStep` attribute that is defined in the `TimeSeries` class.
	`maxDuration`	This optional attribute represents the longest possible duration for this `EnergyConstraintInterval`. The attribute is of the `IntervalDuration` type, which is based on

		an integer, and expresses a multitude of the `intervalDurationStep` attribute that is defined in the `TimeSeries` class.
	`startAfter`	Optionally defines the *earliest* time (inclusive) for the *start* of this `EnergyConstraintInterval`.
	`startBefore`	Optionally defines the *latest* time (inclusive) for the *start* of this `EnergyConstraintInterval`.
	`endAfter`	Optionally defines the *earliest* time (inclusive) for the *end* of this `EnergyConstraintInterval`.
	`endBefore`	Optionally defines the *latest* time (inclusive) for the *end* of this `EnergyConstraintInterval`.

Constraints:	`self.energyConstraint->isEmpty( )`
	`xor self.powerConstraint->isEmpty( )`
	This constraint entails that for this interval either power or energy consumed or produced in this interval is to be constrained.	

`EnergyConstraintList`	
Description:	Instances of this class aggregate a list of `EnergyConstraint` instances. An `EnergyConstraint` is either a fixed value or an upper and lower bound. With this list structure combinations can be made, e.g.: *1 kWh* *1 kWh* or *2 kWh* or *3 kWh* between *1* and *3 kWh* *1 kWh* or between *2* and *3 kWh*

`EnergyConstraint`		
Description:	Describes a constraint on an amount of energy consumed or provided. If one value is provided this value is the only allowable value (unless contained in an `EnergyConstraintList` instance with multiple elements). If two values are provided, this indicates a lower and upper bound.	
Attributes:	`value`	The values as described above.
	`/containsSingle Value`	This boolean is a derived attribute that indicates whether a single value is provided (true) or whether 2 values have been provided (false) indicating a lower and upper bound.
Constraints:	`self.value[0] < self.value[1]` In case two values have been provided (referring to a lower and an upper bound) the above mentioned constraint applies. It specifies that the first value provided should be smaller (lower bound) than the second value (upper bound).	

`PowerConstraintList`	
Description:	Instances of this class aggregate a list of `PowerConstraint` instances. A `PowerConstraint` is either a fixed value or an upper and lower bound. With this list structure combinations can be made, e.g.: *1 kW* *1 kW* or *2 kW* or *3 kW* between *1* and *3 kW* *1 kW* or *2* between *3 kW*

`PowerConstraint`		
Description:	Describes a constraint on an amount of power consumed or provided. If one value is provided this value is the only allowable value (unless contained in a `PowerConstraintList` instance with multiple elements). If two values are provided, this indicates an upper and lower bound.	
Attributes:	`value`	The values as described above.
	`/containsSingleValue`	This boolean is a derived attribute that indicates whether a single value is provided (true) or whether 2 values have been provided (false) indicating a lower and upper bound.
Constraints:	`self.value[0] < self.value[1]` In case two values have been provided (referring to a lower and an upper bound) the above mentioned constraint applies. It specifies that the first value provided should be smaller (lower bound) than the second value (upper bound).	

`TariffConstraint`		
Description:	This class expresses tariff constraints that are related to the `EnergyConstraintInterval` it is associated with. Either a minimum or maximum tariff may be given; a minimum in case of production, a maximum in case of production.	
Attributes:	`minTariff`	This is the minimum tariff (price per unit of energy) that one is willing to receive when selling energy during the associated `EnergyConstraintInterval`.
	`maxTariff`	This is the maximum tariff (price per unit of energy) that one is willing to pay when buying energy during the associated `EnergyConstraintInterval`.

EnergyTariff		
Description:	EnergyTariff expresses a monetary amount in units of currency per quantity of electrical energy to be paid in case of consumption or is earned in case of production. This class exactly follows the CIM class CostPerEnergyUnit, but is renamed for readability.	
Attributes:	value	A float value that expresses the cost.
	unit	This refers to the currency that is being used and is of the type MonetaryAmountPerEnergyUnit; it expresses e.g. Euros, dollars, etc.
	multiplier	This attribute is of the type UnitMultiplier. An example would be k Euros (1000 Euros) or using the none multiplier indicating that the unit should not be multiplied.

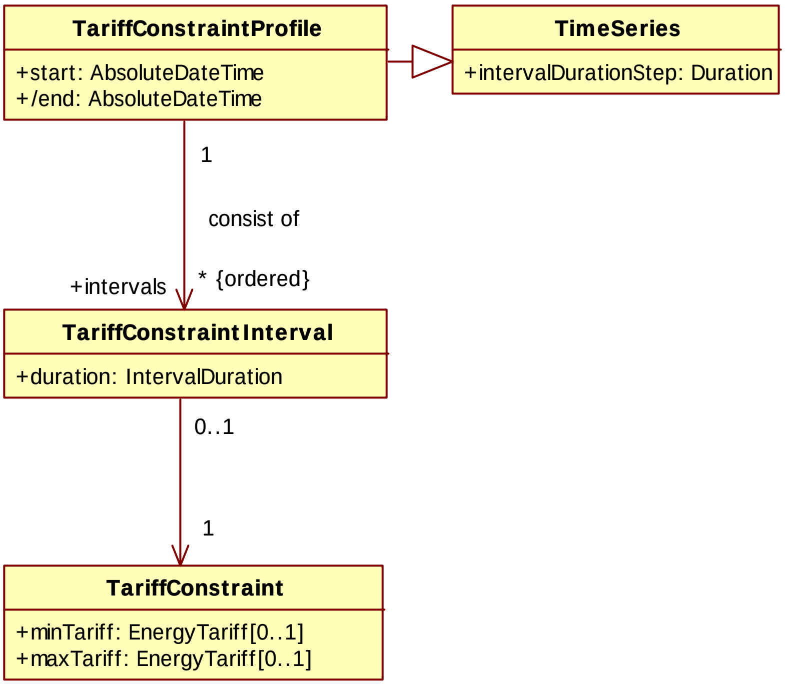
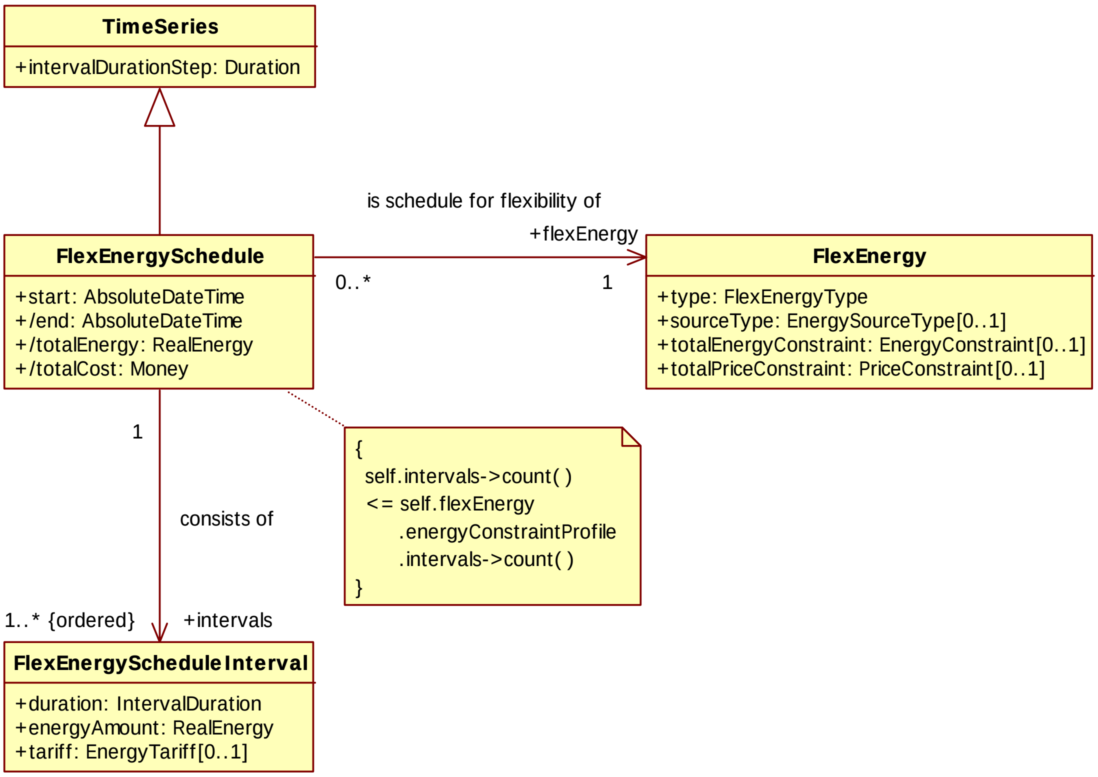


Figure 5: TariffConstraintProfile

TariffConstraintProfile		
Description:	The `TariffConstraintProfile` is a way of expressing financial constraints for the `FlexEnergy` structure. Here the financial constraints have their own `TimeSeries` independent from the `EnergyConstraintProfile`. This way one can express the willingness to shift an instance of an `EnergyConstraintProfile` to a particular point in time. Considering the case of consumption; by specifying a low maximum price for buying energy in a certain interval it will become very unlikely that energy consumption is shifted to that interval. The duration of a `TariffConstraintProfile` instance must be equal to the maximum length of the `EnergyConstraintProfile` or longer; c.f. section 3.5. *Please note that the `TariffConstraintProfile` can never be used in combination with a `TariffConstraint` on an `EnergyConstraintInterval` (see Figure 4) as this may lead to conflicting constraints. Only one of these options is allowed in a specific instance.*	
Attributes:	`Start`	This is the starting time of the `TariffConstraintProfile`

	`/end`	This is the end time of the `TariffConstraintProfile`; this attribute is derived from the durations of the `TariffConstraintInterval` objects referred to.
		`end = 0;`
		`foreach(interval in this.intervals) {`
		`      end += interval.duration;`
		`}`

TariffConstraintInterval		
Description:	The `TariffConstraintInterval` represents an interval of time that is part of a `TariffConstraintProfile`. The duration of these intervals is of fixed length (in contrast to the `EnergyConstraintProfile`).	
Attributes:	`Duration`	This attribute expresses the duration for this `TariffConstraintInterval`. The attribute is of the `IntervalDuration` type, which is based on an integer, and expresses a multitude of the `intervalDurationStep` attribute that is defined in the `TimeSeries` class.



TimeSeries
+intervalDurationStep: Duration
is schedule for flexibility of
+flexEnergy
FlexEnergySchedule
+start: AbsoluteDateTime
+/end: AbsoluteDateTime
+/totalEnergy: RealEnergy
+/totalCost: Money
0..*
1
FlexEnergy
+type: FlexEnergyType
+sourceType: EnergySourceType[0..1]
+totalEnergyConstraint: EnergyConstraint[0..1]
+totalPriceConstraint: PriceConstraint[0..1]
1
consists of
{
self.intervals->count( )
<= self.flexEnergy
.energyConstraintProfile
.intervals->count( )
}
1..* {ordered}
+intervals
FlexEnergyScheduleInterval
+duration: IntervalDuration
+energyAmount: RealEnergy
+tariff: EnergyTariff[0..1]

Figure 6: `FlexEnergySchedule`

`FlexEnergySchedule`		
Description:	The `FlexEnergySchedule` class is derived from `TimeSeries` and is associated with 1 or more `ScheduleInterval` classes. This structure is used to express a fixed energy profile. Please note that the constraint in Figure 6 indicates that a `FlexEnergySchedule` must consist of equal or less intervals than in the intervals in the `EnergyConstraintInterval` of a `FlexEnergy` object.	
Attributes:	`start`	The point in time at which the flex-offer schedule starts.

	`/end`	This calculated attribute represents the end point of this flex-offer schedule.
		`end = self.start;`
		`foreach(interval in this.intervals) {`
		`        end += interval.duration;`
		`}`

	`/totalEnergy`	This is total amount of energy that is expressed through this flex-offer schedule.
		
		`totalEnergy = 0;`
		`foreach(interval in this.intervals) {`
		`      totalEnergy += interval.energyAmount;`
		`}`

	`/totalCost`	The total amount of money that is expressed through this flex-offer schedule.
		
		`totalCost = 0;`
		`foreach(interval in this.intervals) {`
		`      totalCost += (interval.energyAmount *`
		`                   interval.tariff);`
		`}`

Constraints:	`self.intervals->count( )`
	`<= self.flexEnergy.energyConstraintProfile.intervals->count( )`
	
	The number of intervals for `FlexEnergySchedule` is equal or less than the number of intervals in `FlexEnergy`. The number of intervals is less in case the period of one or more intervals that originally were in `FlexEnergy` is set to a length of zero. The zero length intervals will not be present in the `FlexEnergySchedule` since they have no meaning.	

`ScheduleInterval`		
Description:	The `ScheduleInterval` represents an interval of time that is part of a `FlexEnergySchedule`.	
Attributes:	`Duration`	This attribute expresses the duration for this `ScheduleInterval`. The attribute is of the `IntervalDuration` type, which is based on an integer, and expresses a multitude of the `intervalDurationStep` attribute that is defined in the `TimeSeries` class. Please note that the `intervalDurationStep` of a `ScheduleInterval` object must have the same value as `intervalDurationStep` that is used in the corresponding `EnergyConstraintProfile`.
	`energyAmount`	The amount of energy that is associated with this `ScheduleInterval`.
	`tariff`	The tariff that is associated with this `ScheduleInterval`.

3.5 Temporal constraints

The data structure that describes all the concepts that are related to `FlexEnergy` contains several distinct points in time. These points in time are interdependent. The temporal relation between `TariffConstraintProfile` and `EnergyConstraintProfile` is expressed in the formula below where `self` is an instance of `FlexEnergy`.

$$
\begin{aligned}
& \mathit{self.tariffConstraintProfile.start} \\
= \; & \mathit{self.energyConstraintProfile.intervals\text{->}first(\,).startAfter} \\
\leq \; & \mathit{self.energyConstraintProfile.intervals\text{->}first(\,).startBefore} \\
\leq \; & \mathit{self.energyConstraintProfile.intervals\text{->}last(\,).endAfter} \\
\leq \; & \mathit{self.energyConstraintProfile.intervals\text{->}last(\,).endBefore} \\
= \; & \mathit{self.tariffConstraintProfile.end(\,)}
\end{aligned}
$$

Note: this timeline expresses the fact that the `TariffConstraintProfile` should completely envelop the interval that is the maximal interval potentially covered by the `EnergyConstraintProfile`.

Note: `startBefore` of the first interval of the `energyConstraintProfile` and `endAfter` of the last interval are optional (see also Figure 4).

The `FlexEnergySchedules` expressed relate to the temporal bounds expressed in the related `EnergyConstraintInterval` as specified in the formula below where `self` is an instance of `FlexEnergySchedule`.

$$
\begin{aligned}
& \mathit{self.flexEnergy.energyConstraintProfile.intervals\text{->}first(\,).startAfter} \\
\leq \; & \mathit{self.intervals\text{->}first(\,).start} \\
\leq \; & \mathit{self.intervals\text{->}last(\,).end} \\
\leq \; & \mathit{self.flexEnergy.energyConstraintProfile.intervals\text{->}last(\,).endBefore}
\end{aligned}
$$

The intervals in a `FlexEnergySchedule` and an `EnergyConstraintProfile` are consecutive which means that the end time of one interval is the start time of the next interval.

3.6 Model usage constraints considered within MIRABEL

This section describes constraints (at least initially) applied within the MIRABEL project for further specification of e.g. algorithms and implementation purposes.

3.6.1 MIRABEL Financial constraints

The data model provides various mechanisms to express financial constraints. The use of these constraints within MIRABEL will be restricted in the following way:

- `TotalPriceConstraint` and `TariffConstraintProfile` will not be used
- The usage of `TariffConstraint` and `ScheduleInterval.cost` is obligatory.

3.6.2 MIRABEL Energy constraints

The following restrictions are used when expressing energy profiles.

- `EnergyConstraintInterval.minDuration` equals `EnergyConstraintInterval.maxDuration`
- `PowerConstraintList` will initially not be used

3.7 Examples usage of expressions of flexibility in `FlexEnergy`

This section describes some concrete examples of `FlexEnergy` for various devices. It shows how flexibility can be expressed for heat pumps, combined heat power systems and electrical vehicle charging.

3.7.1 Heat Pump

The power consumption of a heat pump is usually controlled by a thermostat. By intervening between the thermostat and the heat pump it is possible to shift the heat pump's demand. Postponing the operation of a heat pump for a small amount of time (0-15 minutes) can already create a considerable amount of flexibility given the large energy consumption. Figure 7 shows an example of a flex-offer that makes this flexibility explicit.

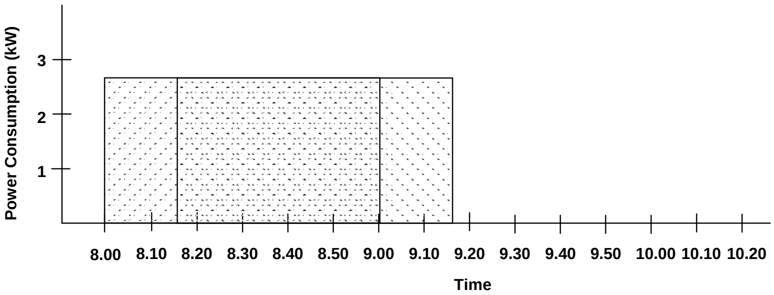


Figure 7: Example of the time shifting potential of a heat pump

This figure shows the two extremes; an energy block that starts at 8.00 and ends at 9.00 and the same block but now it runs from 8.15 till 9.15. All other options that start between 8.00 and 8.15 are also valid. The example is a simplification of a real situation in that all options for shifting this block have the same duration. In reality a later start time means that the heat pump has to operate for a slightly longer period of time because the start temperature will have dropped a bit further.

The constraints can also expressed in formulas. For this we use *a* to denote the block of energy, subscripts *d*, *p* and *s* are used to express duration, power and start time respectively (all of which can be expressed through `EnergyConstraintInterval`).

$$60\ min. \leq a_d \leq 60\ min. \qquad (1)$$
$$2.5\ kW \leq a_p \leq 2.5\ kW \qquad (2)$$
$$8.00\ hr \leq a_s \leq 8.15\ hr \qquad (3)$$

3.7.2 Combined Heat Power System

A Combined Heat Power (CHP) system consumes gas and produces heat and electricity. Like the heat pump the CHP is also temperature driven. The operation of a CHP can be postponed or advanced in the same manner as the heat pump. In addition to the shifting operation in time it is also possible for certain CHP's to operate at partial (70%) or full power (100%). This can be exploited by offering two flexibility options; one to run at 100% (producing 1kW) and one that produces 700W of power but that runs for a longer period of time (otherwise the target temperature will not be reached). This is depicted in Figure 8.

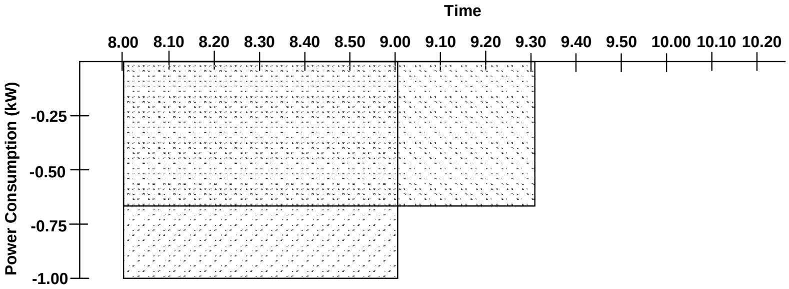


Figure 8: Example of the flexibility of a CHP

The CHP produces electricity hence the negative energy consumption. The CHP can also be shifted in time but this figure focuses only on the different power output levels. The surface of the two options for this block are more or less equal depicting the fact that the same amount of energy is required to raise the temperature to the required level.

Again we will use *a* to denote the energy block and subscripts *d* for duration, *p* for power and s for start time.

$$a_p = 1\ kW\ xor\ a_p = 700\ W \qquad (4)$$
$$a_s = 8.00\ hr \qquad (5)$$
$$a_p \times a_d = 1\ kWh \qquad (6)$$

3.7.3 Electric Vehicle charging

The `FlexEnergy.totalEnergyBounds` constraint can be used to express the upper and lower bound of the energy in a `FlexEnergy` object. E.g. a `FlexEnergy` object expressing a profile with several `EnergyConstraintIntervals` with flexibility in the amount of energy per interval results in a variable total amount of energy for the `FlexEnergy` object.

Such a construct can be used to model the bounds for charging an electric vehicle. Figure 9 shows such a profile consisting of a interval *a* of fixed length and fixed power, a interval *b* of fixed length but variable power, and a interval *c* of variable length (and thus variable amount of energy) and variable power (thus also of variable amount of energy).

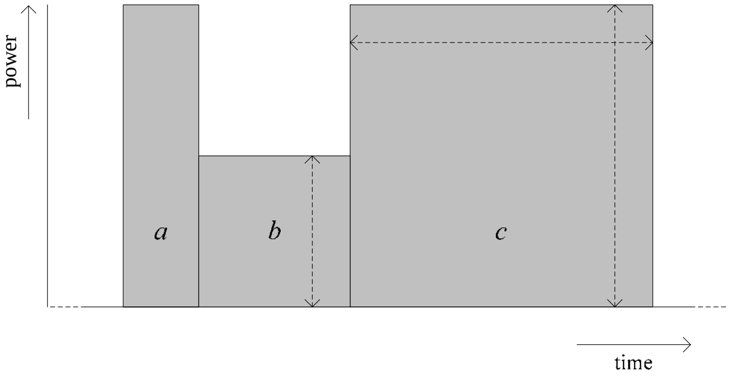


Figure 9 Example constraints in charging pattern for an electric vehicle

The first interval ensures that a certain minimal state of charge is reached in an initial interval to e.g. ensure a certain level of comfort. The second interval in this example can be caused by a priority for other consumption. For the final interval this restriction is no longer of influence and the maximum amount of power can be used, and this interval has a variable duration.

Formulas 1 through 6 are example constraints on the charging pattern for an electric vehicle. They express the upper and lower bounds of the duration and power consumption, indicated with a *d* and *p* respectively, per interval. The total energy which can be consumed according to these constraints lies between 1.5 kWh and 10.5 kWh. These constraints can be expressed with `EnergyConstraintInterval .energyConstraint`.

$$15\ \text{min.} \le a_d \le 15\ \text{min.} \qquad (7)$$

$$6\ \text{kW} \le a_p \le 6\ \text{kW} \qquad (8)$$

$$60\ \text{min.} \le b_d \le 60\ \text{min.} \qquad (9)$$

$$0\ \text{kW} \le b_p \le 3\ \text{kW} \qquad (10)$$

$$0\ \text{min.} \le c_d \le 60\ \text{min.} \qquad (11)$$

$$0\ \text{kW} \le c_p \le 6\ \text{kW} \qquad (12)$$

Equation 7 presents a constraint on the total amount of energy consumed. This is an additional constraint with respect to those formulated in expressions 1 through 6, which ensures that the desired state of charge will finally be reached. This constraint can be expressed with `FlexEnergy.totalEnergyBounds`.

$$6\ \text{kWh} \le a_d \times a_p + b_d \times b_p + c_d \times c_p \le 6\ \text{kWh} \qquad (7)$$

4 Message model

This chapter builds upon the data model that is specified in chapter 3. It focuses on the data that needs to be exchanged by players that want to offer and acquire FlexEnergy.

4.1 Modeling conventions

UMM is used as the methodology to describe the message model. This entails describing a business process first followed by a description of the messages that are exchanged. These messages are called “BusinessEntity”.

4.2 Role level message model introduction

There is a relation between the processes that are specified in WP1 and the business process specified here. The latter only focus on the exchange of messages between swimming lanes. The internal processes of a swimming lane are not specified; these are the subject of the WP1 processes. This also means that the process that is the subject of this chapter can be mapped on multiple WP1 processes as long as these WP1 processes use the same messages (FlexOffer, FlexOfferAcceptance and FlexOfferAssignment). To reflect this the roles that play a part in this business process have been labeled with generic names; ”FlexEnergy Issuer” and ”FlexEnergy Acquirer”.

4.3 FlexOffer Business Process

The main idea behind the business process that is specified here is that FlexEnergy can be offered to another party by means of a FlexOffer. This process covers the steps from issuing a FlexOffer to receiving an assignment. It focuses solely on the interaction of information between the FlexEnergy Issuer and the FlexEnergy Acquirer, e.g. how FlexOffers are scheduled is not part of this process description.

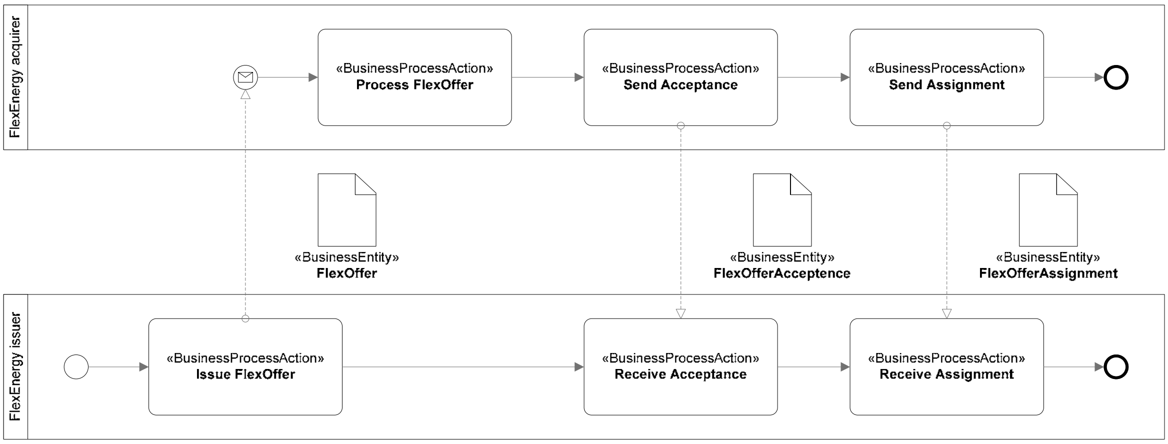


Figure 10: FlexOffer business process

The terms FlexEnergy Issuer and FlexEnergy Acquirer where chosen to support multiple levels of aggregation. Whether the interaction takes place between a Party Connected to the grid and a BRP or between two BRP’s the process remains the same.

The first step in the business process is undertaken by the Issuer by issuing the `FlexOffer`. It is then received by the Acquirer which in turn processes the `FlexOffer`. The nature of this processing is out of scope for this business process.

During this processing a decision will be made whether the FlexOffer can be fit in or not. The outcome of this decision is communicated to the Issuer by the Acquirer in the form of a `FlexOfferAcceptance BusinessEntity`. This `BusinessEntity` only contains a confirmation or a rejection of the FlexOffer.

In the case that the FlexOffer is confirmed a `FlexOfferAssignment BusinessEntity` will follow later. This `BusinessEntity` contains the choices that were made by the Acquirer within the boundaries that were stated in the original FlexOffer. The `FlexOfferAcceptance` and `FlexOfferAssignment` entities may also be combined into a single message.

4.4 FlexOffer

This section specifies the message for offering flexibility as indicated in Figure 11.

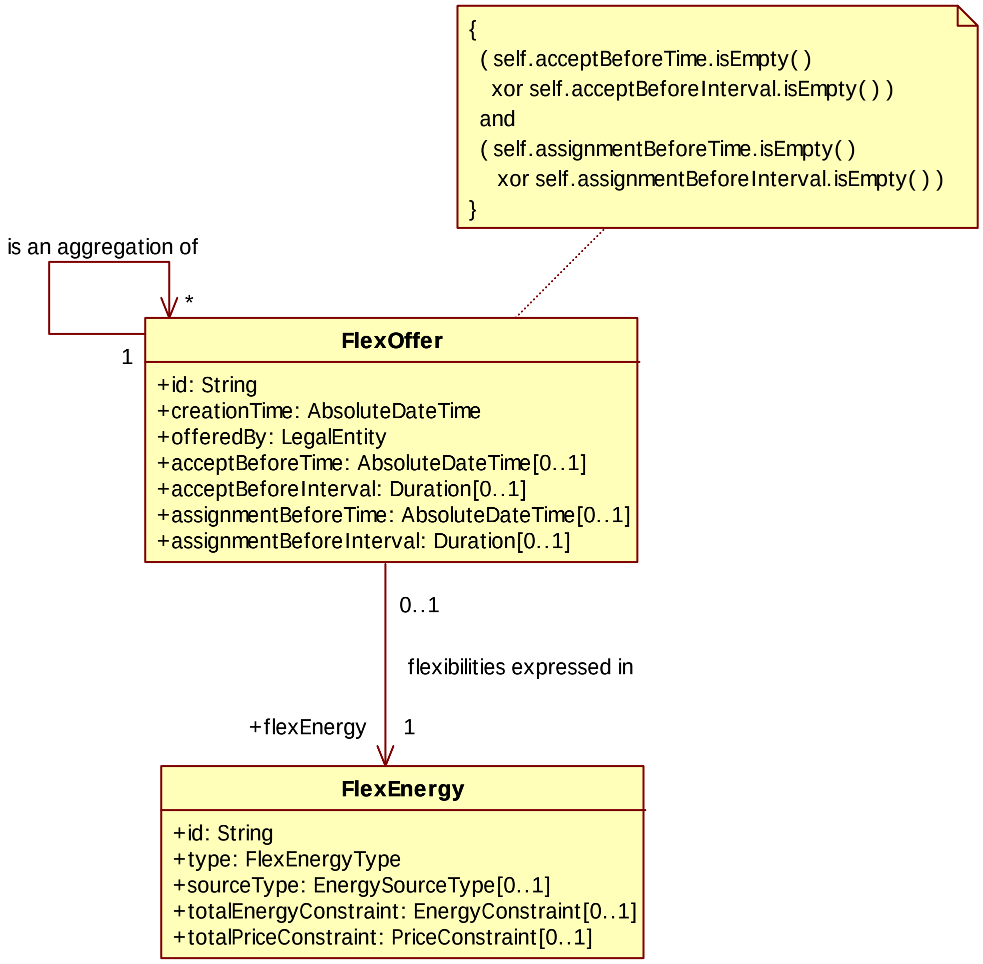

{
(self.acceptBeforeTime.isEmpty( )
xor self.acceptBeforeInterval.isEmpty( ))
and
(self.assignmentBeforeTime.isEmpty( )
xor self.assignmentBeforeInterval.isEmpty( ))
}
is an aggregation of
*
1
FlexOffer
+id: String
+creationTime: AbsoluteDateTime
+offeredBy: LegalEntity
+acceptBeforeTime: AbsoluteDateTime[0..1]
+acceptBeforeInterval: Duration[0..1]
+assignmentBeforeTime: AbsoluteDateTime[0..1]
+assignmentBeforeInterval: Duration[0..1]
0..1
flexibilities expressed in
+flexEnergy
1
FlexEnergy
+id: String
+type: FlexEnergyType
+sourceType: EnergySourceType[0..1]
+totalEnergyConstraint: EnergyConstraint[0..1]
+totalPriceConstraint: PriceConstraint[0..1]

Figure 11: `FlexOffer BusinessEntityView`

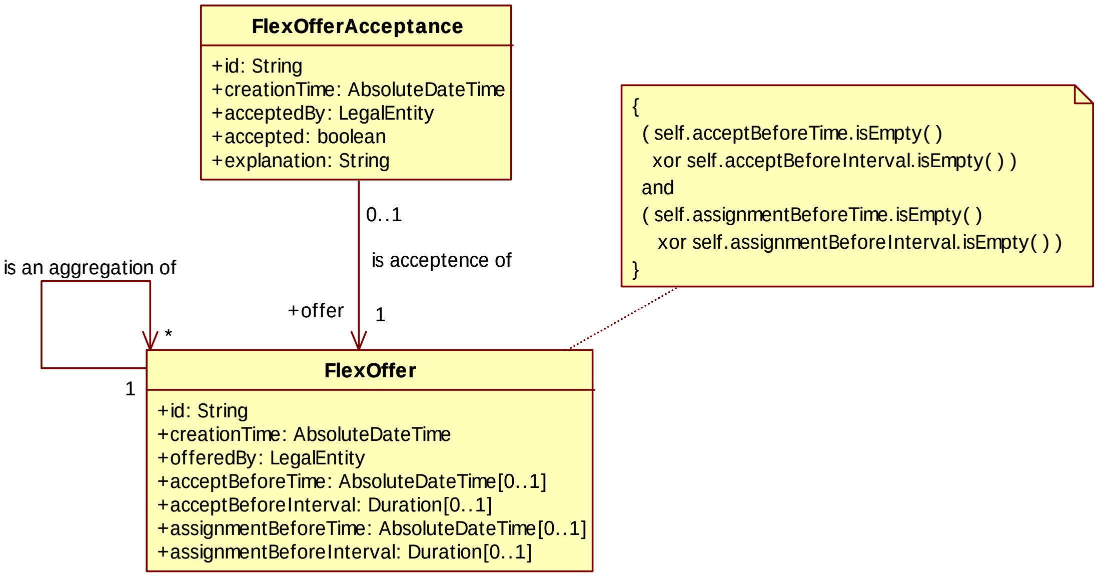
FlexOffer		
Description:	Through a FlexOffer `FlexEnergy` can be offered on a marketplace. A `FlexOffer` is always associated with a `LegalEntity` that is responsible for it. `FlexOffers` can also be aggregated into a new `FlexOffer` as is shown by the self association.	
Attributes:	`id`	A unique identifier for the `FlexOffer`. The selection of an identifier scheme is beyond the scope of this document as it is up to the user of this model to make these choices when implementing the model.
	`offeredBy`	This attribute represents the `LegalEntity` that is responsible for issuing this `FlexOffer`.
	`acceptBeforeTime`	An absolute moment in time by which the `FlexOffer` is to be accepted, i.e. the Issuer needs to know whether the `FlexOffer` is going to be exerted or not. The Acquirer notifies the Issuer by sending a `FlexOfferAcceptance` message (see 4.6).
	`acceptBefore Interval`	This attribute expresses the same deadline as the `acceptBeforeTime` attribute, but instead of specifying an absolute point in time, an interval is expressed. This interval is the amount of time before the start of operations (as expressed in `FlexEnergySchedule.start`) before which acceptance of the FlexOffer is to be communicated.
	`assignmentBefore Time`	A moment in time by which the Issuer needs to know what the actual assignment is. This assignment should always respect the boundaries that were specified in the `FlexOffer`. The `acceptBefore` and `assignmentBefore` moment may coincide. The assignment cannot be changed or revoked after the `assignmentBefore` moment has passed. As with `acceptBeforeTime`, here an absolute point in time is specified.
	`assignmentBefore Interval`	This attribute defines the same deadline as `assignmentBeforeTime` but as a deadline relative to the start of operations (as expressed in `FlexEnergySchedule.start`) instead of as an absolute point in time.

Constraints	This constraint expresses that accept and assignment deadlines are specified as absolute or relative points in time.
	`( self.acceptBeforeTime.isEmpty( )`
	`  xor self.acceptBeforeInterval.isEmpty( ) )`
	`and`
	`( self.assignmentBeforeTime.isEmpty( )`
	`  xor self.assignmentBeforeInterval.isEmpty( ) )`	

LegalEntity	
Description:	The LegalEntity class represents the entity that can issue a FlexOffer and accepts all resulting legal responsibilities. The same goes for FlexOfferAcceptance and FlexOfferAssignment.

4.5 FlexOfferAcceptance

The message for accepting offered flexibility is specified in this section.



FlexOfferAcceptance
+id: String
+creationTime: AbsoluteDateTime
+acceptedBy: LegalEntity
+accepted: boolean
+explanation: String
0..1
is acceptence of
+offer
1
is an aggregation of
*
1
{
(self.acceptBeforeTime.isEmpty( )
xor self.acceptBeforeInterval.isEmpty( ))
and
(self.assignmentBeforeTime.isEmpty( )
xor self.assignmentBeforeInterval.isEmpty( ))
}
FlexOffer
+id: String
+creationTime: AbsoluteDateTime
+offeredBy: LegalEntity
+acceptBeforeTime: AbsoluteDateTime[0..1]
+acceptBeforeInterval: Duration[0..1]
+assignmentBeforeTime: AbsoluteDateTime[0..1]
+assignmentBeforeInterval: Duration[0..1]

Figure 12: FlexOfferAcceptance BusinessEntityView

FlexOfferAcceptance		
Description:	This class contains information about whether a FlexOffer has been accepted by the FlexOffer Acquirer or not. Therefore it can only be associated with a single FlexOffer. Acceptance of a FlexOffer only signifies that the Acquirer of the FlexOffer is going to make use of it at some point in time. The actual assignment may be postponed to a later point in time.	
Attributes:	id	A unique identifier for the FlexOfferAcceptance. The selection of an identifier scheme is beyond the scope of this document as it is up to the user of this model to make these choices when implementing the model.
	acceptedBy	This attribute represents the LegalEntity that is responsible for issuing this FlexOfferAcceptance.
	accepted	A Boolean that is true in case the FlexOffer has been accepted and false in case the FlexOffer has been rejected.

	explanation	This attribute might be used to provide some background on the acceptance or rejection of the FlexOffer.

4.6 FlexOfferAssignment

The assignment of `FlexEnergySchedules` for accepted `FlexOffers` is specified in this section.

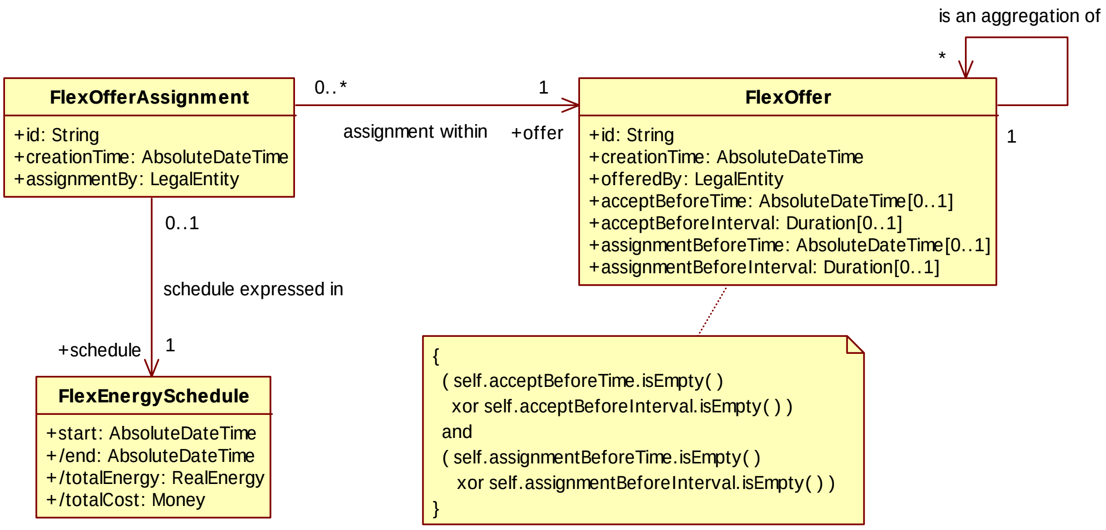

is an aggregation of
*
1
FlexOfferAssignment
+id: String
+creationTime: AbsoluteDateTime
+assignmentBy: LegalEntity
0..*
1
assignment within
+offer
FlexOffer
+id: String
+creationTime: AbsoluteDateTime
+offeredBy: LegalEntity
+acceptBeforeTime: AbsoluteDateTime[0..1]
+acceptBeforeInterval: Duration[0..1]
+assignmentBeforeTime: AbsoluteDateTime[0..1]
+assignmentBeforeInterval: Duration[0..1]
0..1
schedule expressed in
+schedule
1
FlexEnergySchedule
+start: AbsoluteDateTime
+/end: AbsoluteDateTime
+/totalEnergy: RealEnergy
+/totalCost: Money
{
(self.acceptBeforeTime.isEmpty( )
xor self.acceptBeforeInterval.isEmpty( ))
and
(self.assignmentBeforeTime.isEmpty( )
xor self.assignmentBeforeInterval.isEmpty( ))
}

Figure 13: `FlexOfferAssignment BusinessEntityView`

FlexOfferAssignment		
Description:	The `FlexOfferAssignment` is related to a FlexOffer. For each level of flexibility (time, energy, costs) that is offered through the FlexOffer a fixed choice is made (expressed via an instance of the associated `FlexEnergySchedule`) so that the Issuer of the FlexOffer knows what to do. An assignment is always associated with a single `FlexOffer`.	
Attributes:	Id	A unique identifier for the `FlexOfferAssignment`. The selection of an identifier scheme is beyond the scope of this document as it is up to the user of this model to make these choices when implementing the model.
	creationTime	The moment in time the `FlexOfferAssignment` was created.
	assignedBy	This attribute represents the `LegalEntity` that is responsible for issuing this `FlexOfferAssignment`.

4.7 Temporal constraints

This section indicates the temporal constraints which apply to `FlexOffer`, `FlexOfferAcceptance`, `FlexOfferAssignment and FlexEnergySchedule`. They are specified below; the constraints are preceded by the definitions of `absoluteAcceptBeforeTime` and `absoluteAssignmentBeforeTime` as these concepts play an important role in the constraints.

$$\begin{aligned}
&\mathit{absoluteAcceptBeforeTime} = \mathit{FlexOffer.acceptBeforeTime} \\
&\qquad \mathit{xor} \\
&\mathit{absoluteAcceptBeforeTime} = \mathit{FlexEnergySchedule.start} - \\
&\mathit{FlexOffer.acceptBeforeInterval}
\end{aligned}$$

$$\begin{aligned}
&\mathit{absoluteAssignmentBeforeTime} = \mathit{FlexOffer.assignmentBeforeTime} \\
&\qquad \mathit{xor} \\
&\mathit{absoluteAssignmentBeforeTime} = \mathit{FlexEnergySchedule.start} - \\
&\mathit{FlexOffer.assignmentBeforeInterval}
\end{aligned}$$

$$\begin{aligned}
& && \mathit{FlexOffer.creationTime} \\
& < && \mathit{FlexOfferAcceptance.creationTime} \\
& \leq && \mathit{absoluteAcceptBeforeTime} \\
& \leq && \mathit{FlexOfferAssignment.creationTime} \\
& \leq && \mathit{absoluteAssignmentBeforeTime} \\
& \leq && \mathit{FlexEnergySchedule.start} \\
& < && \mathit{FlexEnergySchedule.end}
\end{aligned}$$

This timeline describes the order which all the events related to issuing, accepting and assigning a FlexOffer should adhere to.

5 XML Schema mapping

This section provides a mapping of the model specified in section 3 and the messages specified in section 4 into an XML schema. First the strategy for derivation of the XML schema is discussed; in the subsequent sections the schemas are presented.

5.1 Schema derivation strategy

The XML schema mapping consists of two separate schemas, one for the model and one for the messages which are based on the model. Each schema has http://mirabel-project.eu/schemas/ as a prefix.

Each class in the model and message specifications is translated into a XML schema type with exactly the same name. The attributes of the classes are translated into elements of sequences in complex types. In case of class attributes where exactly one of two attributes should have a value, the XML schema choice construct is applied. For ordered relations, the sequence construct is used.

Independent element specifications are only provided in the messages schema for the three messages defined in section 4.

5.2 Model schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://mirabel-project.eu/schemas/model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cim="http://iec.ch/TC57/2001/CIM-schema-cim10"
  attributeFormDefault="qualified"
  elementFormDefault="qualified"
  targetNamespace="http://mirabel-project.eu/schemas/model"
>

  <!-- Schema imports -->

  <import
    namespace="http://iec.ch/TC57/2001/CIM-schema-cim10"
    schemaLocation="cim-schema-cim10.xsd"
  />


  <!-- Type definitions -->

  <complexType name="FlexEnergy">
    <sequence>
      <element name="meteringPointID" type="string" />
      <element name="type" type="tns:FlexEnergyType" />
      <element name="sourceType" type="tns:EnergySourceType" minOccurs="0" />
      <element name="totalEnergyConstraint" type="tns:EnergyConstraint" minOccurs="0" />
      <element name="totalPriceConstraint" type="tns:PriceConstraint" minOccurs="0" />
      <element name="energyConstraintProfile" type="tns:EnergyConstraintProfile" />
      <element name="tariffConstraintProfile" type="tns:TariffConstraintProfile" minOccurs="0" />
    </sequence>
  </complexType>

  <simpleType name="FlexEnergyType">
    <restriction base="string">
```

```
      <enumeration value="PRODUCTION" />
      <enumeration value="CONSUMPTION" />
    </restriction>
  </simpleType>

  <complexType name="EnergySourceType">
    <sequence>
      <element name="classification" type="string" />
    </sequence>
  </complexType>

  <complexType name="PriceConstraint">
    <choice>
      <element name="minPrice" type="cim:Money" />
      <element name="maxPrice" type="cim:Money" />
    </choice>
  </complexType>

  <complexType name="EnergyConstraintProfile">
    <complexContent>
      <extension base="tns:TimeSeries">
        <sequence>
          <element
            name="energyConstraintInterval"
            type="tns:EnergyConstraintInterval"
            maxOccurs="unbounded"
          />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="EnergyConstraintInterval">
    <sequence>
      <element name="minDuration" type="tns:IntervalDuration" minOccurs="0" />
      <element name="maxDuration" type="tns:IntervalDuration" minOccurs="0" />
      <element name="startAfter" type="dateTime" minOccurs="0" />
      <element name="startBefore" type="dateTime" minOccurs="0" />
      <element name="endAfter" type="dateTime" minOccurs="0" />
      <element name="endBefore" type="dateTime" minOccurs="0" />
      <choice>
        <element name="energyConstraintList" type="tns:EnergyConstraintList" />
        <element name="powerConstraintList" type="tns:PowerConstraintList" />
      </choice>
      <element name="tariffConstraint" type="tns:TariffConstraint" />
    </sequence>
  </complexType>

  <complexType name="EnergyConstraintList">
    <sequence>
      <element name="energyConstraint" type="tns:EnergyConstraint" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="EnergyConstraint">
    <choice>
      <element name="value" type="cim:RealEnergy" />
      <sequence>
        <element name="lowerBound" type="cim:RealEnergy" />
        <element name="upperBound" type="cim:RealEnergy" />
      </sequence>
    </choice>
  </complexType>

  <complexType name="PowerConstraintList">
    <sequence>
      <element name="powerConstraint" type="tns:PowerConstraint" maxOccurs="unbounded" />
    </sequence>
```

```
</complexType>

<complexType name="PowerConstraint">
  <choice>
    <element name="value" type="cim:ActivePower" />
    <sequence>
      <element name="lowerBound" type="cim:ActivePower" />
      <element name="upperBound" type="cim:ActivePower" />
    </sequence>
  </choice>
</complexType>

<complexType name="TariffConstraint">
  <sequence>
    <element name="minTariff" type="tns:EnergyTariff" minOccurs="0" />
    <element name="maxTariff" type="tns:EnergyTariff" minOccurs="0" />
  </sequence>
</complexType>

<complexType name="EnergyTariff">
  <sequence>
    <element name="value" type="float" />
    <element name="unit" type="cim:MonetaryAmountPerEnergyUnit" />
    <element name="multiplier" type="cim:UnitMultiplier" />
  </sequence>
</complexType>

<complexType name="TariffConstraintProfile">
  <complexContent>
    <extension base="tns:TimeSeries">
      <sequence>
        <element name="start" type="dateTime" />
        <element name="tariffConstraintInterval" type="tns:TariffConstraintInterval"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="TariffConstraintInterval">
  <sequence>
    <element name="duration" type="tns:IntervalDuration" />
    <element name="tariffConstraint" type="tns:TariffConstraint" />
  </sequence>
</complexType>

<complexType name="FlexEnergySchedule">
  <complexContent>
    <extension base="tns:TimeSeries">
      <sequence>
        <element name="start" type="dateTime" />
        <element name="interval" type="tns:FlexEnergyScheduleInterval" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="FlexEnergyScheduleInterval">
  <sequence>
    <element name="duration" type="tns:IntervalDuration" />
    <element name="energyAmount" type="cim:RealEnergy" />
    <element name="tariff" type="tns:EnergyTariff" />
  </sequence>
</complexType>

<complexType name="TimeSeries">
  <sequence>
    <element name="intervalDurationStep" type="duration" />
    </sequence>
  </complexType>

  <simpleType name="IntervalDuration">
    <restriction base="int">
      <minInclusive value="0"></minInclusive>
    </restriction>
  </simpleType>

</schema>
```

5.3 Messages schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://mirabel-project.eu/schemas/messages"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:model="http://mirabel-project.eu/schemas/model"
  attributeFormDefault="qualified"
  elementFormDefault="qualified"
  targetNamespace="http://mirabel-project.eu/schemas/messages"
>

  <!-- Schema imports -->

  <import namespace="http://mirabel-project.eu/schemas/model" schemaLocation="model.xsd" />

  <!-- Element definitions -->

  <element name="flexOffer" type="tns:FlexOffer" />
  <element name="flexOfferAcceptance" type="tns:FlexOfferAcceptance" />
  <element name="flexOfferAssignment" type="tns:FlexOfferAssignment" />

  <!-- Type definitions -->

  <complexType name="FlexOffer">
    <sequence>
      <element name="id" type="string" />
      <element name="creationTime" type="dateTime" />
      <element name="offeredById" type="string" />
      <choice>
        <element name="acceptBeforeTime" type="dateTime" />
        <element name="acceptBeforeInterval" type="duration" />
      </choice>
      <choice>
        <element name="assignmentBeforeTime" type="dateTime" />
        <element name="assignmentBeforeInterval" type="duration" />
      </choice>
      <element name="flexEnergy" type="model:FlexEnergy" />
    </sequence>
  </complexType>

  <complexType name="FlexOfferAcceptance">
    <sequence>
      <element name="id" type="string" />
      <element name="creationTime" type="dateTime" />
      <element name="flexOfferId" type="string" />
      <element name="acceptedById" type="string" />
      <element name="accepted" type="boolean" />
      <element name="explanation" type="string" />
    </sequence>
  </complexType>
  <complexType name="FlexOfferAssignment">
    <sequence>
      <element name="id" type="string" />
      <element name="creationTime" type="dateTime" />
      <element name="flexOfferId" type="string" />
      <element name="acceptedById" type="string" />
      <element name="schedule" type="model:FlexEnergySchedule" />
    </sequence>
  </complexType>

</schema>
```

Appendix A IEC CIM derived classes

This appendix contains descriptions from [CIM09] of the classes of this Common Information Model which are used in this deliverable. The descriptions of the classes, their attributes, enumerations and their constants are literal quotes of the Common Information Model standard document.

Figure 14 shows the currently adopted classes from [CIM09]. Please note that the relationships in this diagram are duplicates of the attributes defined (i.e. of `unit` and `multiplier` attributes).

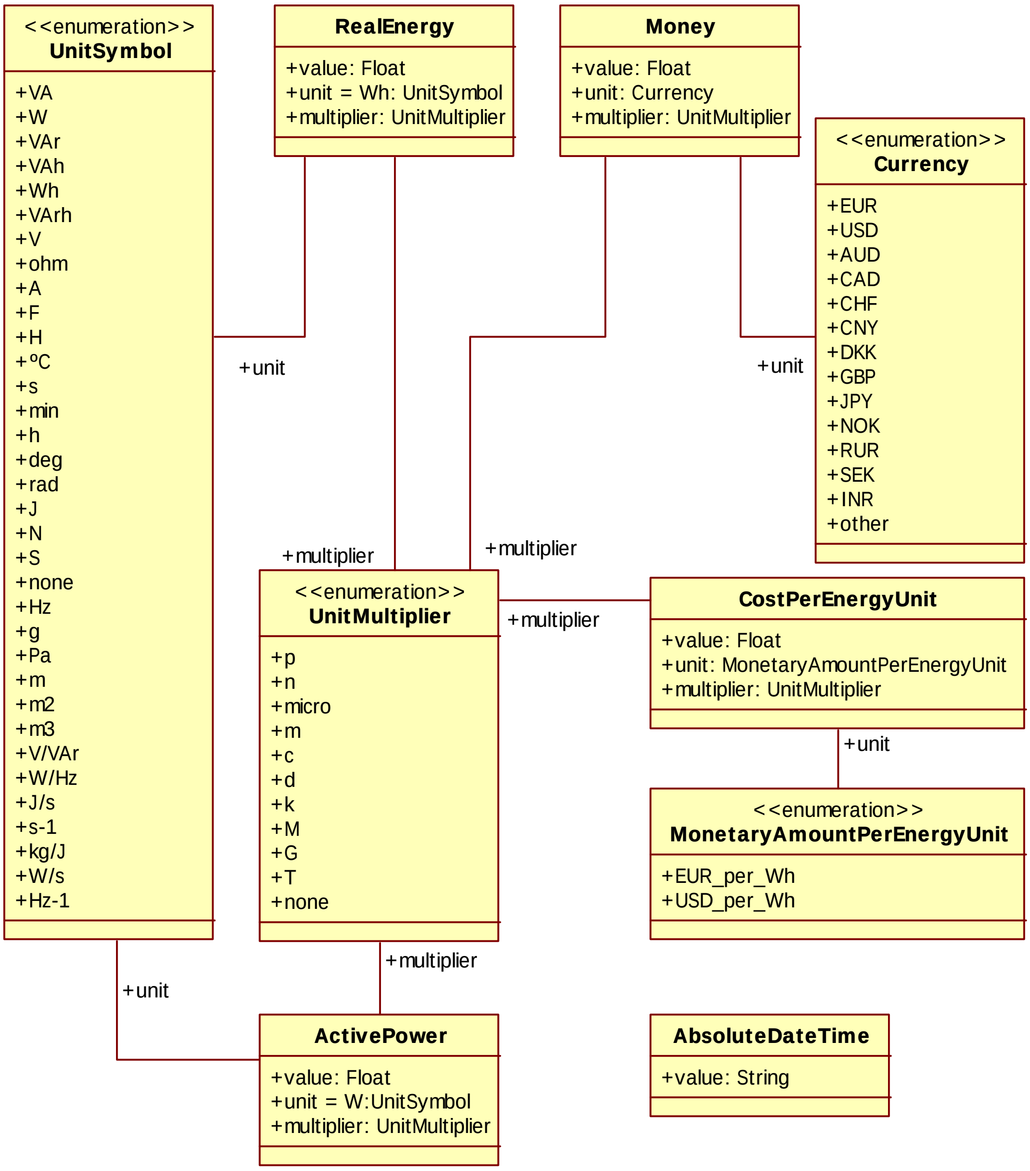


Figure 14: IEC CIM derived classes in the common package

`AbsoluteDateTime`		
Description:	Date and time as "`yyyy-mm-ddThh:mm:ss.sss`", which conforms to ISO 8601. UTC time zone is specified as "`yyyy-mm-ddThh:mm:ss.sssZ`". A local time zone relative UTC is specified as "`yyyy-mm-ddThh:mm:ss.sss-hh:mm`". `AbsoluteDateTime` can be used both for calendar time, e.g. 2007-02-07T10:30, and for relative time, e.g. 10:30.	
Attributes:	`value`	String representation of date and time; refer to description of the class.
Source:	[CIM09]	

`ActivePower`		
Description:	Product of root mean square (RMS) value of the voltage and the RMS value of the in-phase component of the current.	
Attributes:	`value`	The amount of Watts.
	`unit`	The unit of the `ActivePower` class, which is W within [CIM09].
	`multiplier`	A multiplier, e.g. kilo or mega.
Source:	[CIM09]	

`CostPerEnergyUnit`		
Description:	Cost, in units of currency, per quantity of electrical energy produced or consumed.	
Attributes:	`value`	A float value that expresses the cost.
	`unit`	This refers to the currency that is being used and is of the type `MonetaryAmountPerEnergyUnit`; it expresses e.g. Euros, dollars, etc.
	`multiplier`	This attribute is of the type `UnitMultiplier`. An example would be k Euros (1000 Euros) or using the none multiplier indicating that the unit should not be multiplied.
Source:	[CIM09]	

Currency		
Description:	Monetary currencies. Apologies for this list not being exhaustive.	
Enum. constants:	USD	US dollar
	EUR	European Euro
	AUD	Australian dollar
	CAD	Canadian dollar
	CHF	Swiss francs
	CNY	Chinese yuan renminbi
	DKK	Danish crown
	GBP	British pound
	JPY	Japanese yen
	NOK	Norwegian crown
	RUR	Russian ruble
	SEK	Swedish crown
	INR	India rupees
	other	Another type of currency.
Source:	[CIM09]	

MonetaryAmountPerEnergyUnit		
Description:	Monetary amount per energy unit.	
Enum. constants:	USD_per_Wh	A number of USD per Watt-hour.
	EUR_per_Wh	A number of EUR per Watt-hour.
Source:	[CIM09]	

Money		
Description:	Amount of money	
Attributes:	value	The basic monetary amount, e.g. 20 or 0,15.
	unit	The monetary unit.
	multiplier	A multiplier of the basic value, e.g. to express k€.
Source:	[CIM09]	

RealEnergy		
Description:	Real electrical energy	
Attributes:	value	The amount of Watt-hours.
	Unit	The unit of the RealEnergy class, which is Wh within [CIM09].
	multiplier	A multiplier, e.g. kilo or mega.
Source:	[CIM09]	

`UnitMultiplier`		
Description:	The unit multipliers defined for the CIM.	
Enum. constants:	`p`	Pico 10^{-12}
	`n`	Nano 10^{-9}
	`micro`	Micro 10^{-6}
	`m`	Milli 10^{-3}
	`c`	Centi 10^{-2}
	`d`	Deci 10^{-1}
	`k`	Kilo 10^{3}
	`M`	Mega 10^{6}
	`G`	Giga 10^{9}
	`T`	Tera 10^{12}
	`none`	
Source:	[CIM09]	

`UnitSymbol`		
Description:	The units defined for usage in the CIM.	
Attributes:	`VA`	Apparent power in volt ampere
	`W`	Active power in watt
	`VAr`	Reactive power in volt ampere reactive
	`VAh`	Apparent energy in volt ampere hours
	`Wh`	Real energy in watt hours
	`VArh`	Reactive energy in volt ampere reactive hours
	`V`	Voltage in volt
	`ohm`	Resistance in ohm
	`A`	Current in ampere
	`F`	Capacitance in farad
	`H`	Inductance in Henry
	`ºC`	Relative temperature in degrees Celsius
	`s`	Time in seconds
	`min`	Time in minutes
	`h`	Time in hours
	`deg`	Plane angle in degrees
	`rad`	Plane angle in radians
	`J`	Energy in joule
	`N`	Force in Newton
	`S`	Conductance in Siemens
	`none`	Dimension less quantity, e.g. count, per unit, etc.
	`Hz`	Frequency in hertz
	`g`	Mass in gram
	`Pa`	Pressure in Pascal (n/m2)

	m	Length in meter
	m2	Area in square meters
	m3	Volume in cubic meters
	V/VAr	Volt per volt ampere reactive
	W/Hz	Watt per hertz
	J/s	Joule per second
	s-1	per second
	kg/J	Mass per energy
	W/s	Watt per second
	Hz-1	per Hertz
Source:	[CIM09]	

References

[EMD09] ebIX, ‘Business information model for the exchange of metered data in the energy domain – Measure’, version 1.0, revision D, April 2009

[CIM09] IEC61970-301 Ed. 2, ‘Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base’, International Electrotechnical Commission, 2009

[ISO8601] ISO (International Organization for Standardization), representations of dates and times, 1988-06-15

[KonRum10] D2.1 State of the art on data specifications, June 2010, MIRABEL project

[ENTS09] ENTSO-E: The Harmonized Electricity Market Role Model, version 2009-01, 2009.

[ebiXCuS] ebiX CuS Model, http://www.ebix.org/Documents/CuS%20Draft%20v2r1A%2020100409.pdf, retreived on June 21 2010

[MČB+11] Zoran Marinsek, Gregor Černe, Henrike Berthold, Alexander Savinov, Laurynas Siksnys, Tea Tušar, Bogdan Filipic, Matthias Böhm: D1.3 Conceptual draft of the request management architecture and its design. Technical report, MIRABEL Consortium, July 2011.

[SKT+11] Laurynas Šikšnys, Dalia Kaulakienė, Torben B. Pedersen, Ralf Rantzau, Matthias Böhm: D3.2 Initial Specification of Data Collection and Analysis System. Technical report, MIRABEL Consortium, November 2010.

[TDF+11] Tea Tušar, Erik Dovgan, Bogdan Filipič, Alexandr Savinov: D5.3 Initial draft of the scheduling and negotiation framework. Technical report, MIRABEL Consortium, May 2011.