MIRACLE energy

Balancing energy supply and demand

MIRACLE

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

Specific Targeted Research Project: 248195

D2.2 Data model, specification of request and negotiation messages and contracts

Work package 2

Leading partner: TNO

December, 2010

Version 1.0

DOCUMENT INFORMATION	
ID	D2.2 Data model, specification of request and negotiation messages and contracts
Work Package(s)	Work package 2 Data Specification
Type	Report
Dissemination	Public
Version	Final
Date	December 22nd 2010
Author(s)	M.J Konsman (TNO), F.J. Rumph (TNO)
Reviewer(s)	D. Kaulakienė (Aalborg Universitet)

Table of Contents

1 Management summary

The aim of the MIRACLE 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 MIRACLE deliverable provides a data model. This document contains the first version of the data model; a subsequent deliverable D2.3 will follow in M18 of the MIRACLE project and will provide further enhancements and elaborations.

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 MIRACLE 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 first 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 will serve as input for the work packages on aggregation and scheduling. The model that is described in this deliverable is the result of various discussions within the project.

The final version of the data model will be the subject of deliverable D2.3. It will contain refinements on the `FlexEnergy` structure based on the feedback from other work packages and a wider focus that includes all major data concepts that are used by each EDMS component. Not only will it focus on a data model but there will also be much emphasis on the actual messages that are exchanged between the various players of the energy market.

2 Introduction

The aim of the data model that is being developed as part of the MIRACLE 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 MIRACLE 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.

This deliverable is an initial version of the data model; a final version of the data model and the specification of messages will be available in M18 of the MIRACLE project. This aligns the timelines of this work package with other work packages depending on the data model. Also it allows feedback from these work packages to be processed, thereby strengthening the final version of the model.

The focus of this initial version is on the `FlexEnergy` concept which is a central concept for the MIRACLE project. With this concept flexibility in both the production and consumption of electricity can be expressed. The model presented in this document is the result of various discussions within the project. The `FlexEnergy` concept will be directly used by the work packages 3 (aggregation) and 5 (scheduling) and it is also the focal point of the standardization activities.

The final version of the data model will have a broader focus than this one. It will describe all major data concepts that are used by each EDMS component. Not only will it focus on a data model but there will also be much emphasis on the actual messages that are exchanged between the various players of the energy market. The processes described in D1.2 serve as important input for this part of the deliverable.

2.1 Modeling approach

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

2.2 Document structure

Chapter 3 describes the data model. It starts with explaining the modeling conventions followed by the data model itself. As mentioned earlier the classes that can be found in this initial version of the model center on the `FlexEnergy` concept. The concepts are formulated as generic as possible because of the focus on standardization. However for the MIRACLE 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 4 discusses the messages that are derived from the data model itself. It describes the business process that is associated with offering, accepting and assigning flexibility in consumption and production of energy.

The Common Information Model (CIM) 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. Appendix B contains a list of classes that are potentially relevant for MIRACLE but are not being used yet.

3 Data model

This section describes the data model used within the MIRACLE project. First in section 3.1 the conventions which are used in this model are described. Second in section 3.2 an overview of the model is provided, indicating the scope 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 MIRACLE 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 for MIRACLE 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 (that will be developed later) 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 described, the derived attribute construct is used. An attribute name preceded by a slash denotes that this a derived attribute that can be determined by combining other fixed attributes.

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

[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).

3.2 Model

The data model that is presented here describes different possibilities of shifting energy demand or supply. The model is centered around the `FlexEnergy` concept. 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.

3.2.1 Classes for negotiating flexibility in energy supply and demand

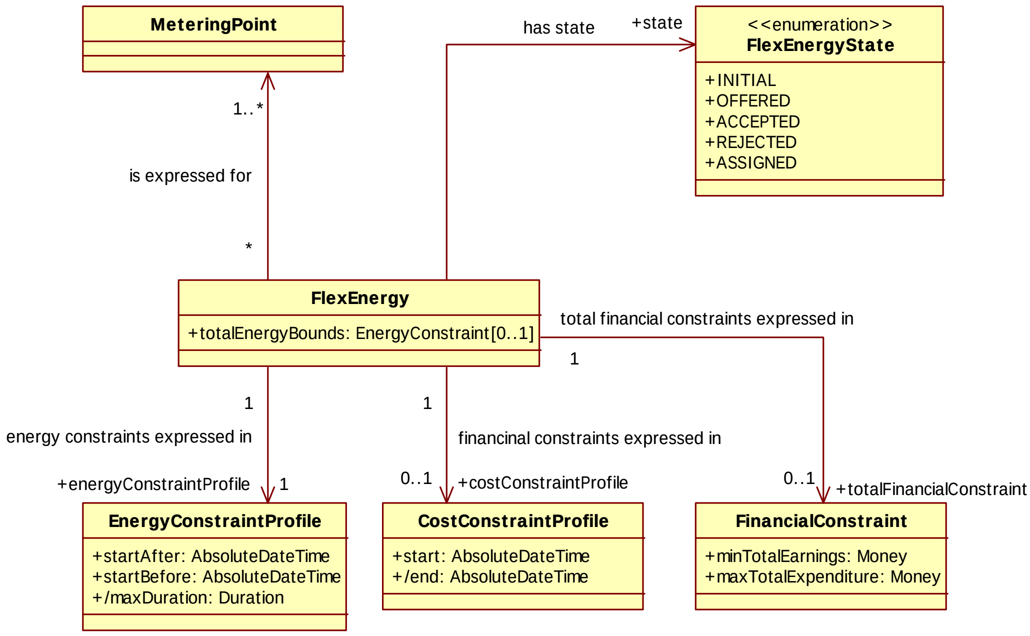

Figure 1: `FlexEnergy`

`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 `CostConstraintProfile` and the `FinancialConstraint` 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:	`totalEnergyBounds`	With this attribute limits can be put on the overall amount of energy that is expressed in a `FlexEnergy` object.

FlexEnergyState		
Description:	This enumeration contains the valid states a FlexEnergy object can be in. Figure 2 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 Schedule object) is communicated.

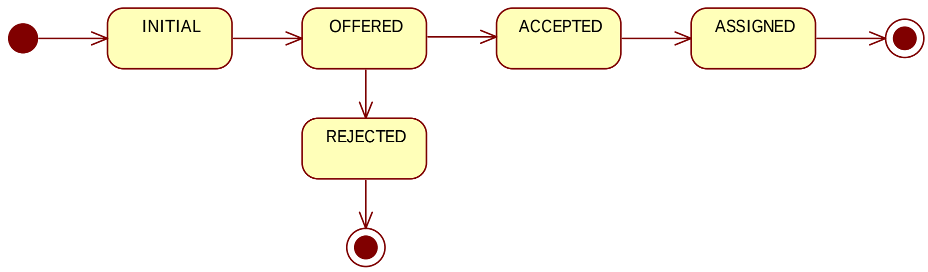

Figure 2: FlexEnergyState

FinancialConstraint		
Description:	This class is optionally associated with the FlexEnergy class. It can be used to express an overall financial constraint to a FlexEnergy object	
Attributes:	minTotalEarnings	The minimum total amount that has to be earned with the associated FlexEnergy object in case of production.
	maxTotalExpenditure	The maximum total amount of money that one is willing to pay in relation to the FlexEnergy object 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].

<table>
<tr><th colspan="3">EnergyConstraintProfile</th></tr>
<tr><td>Description:</td><td colspan="2">The <code>EnergyConstraintProfile</code> can be used to describe an energy profile. It is associated with an ordered sequence of <code>EnergyConstraintIntervals</code>. <code>EnergyConstraintProfile</code> is derived from the <code>TimeSeries</code> class as can be seen in the diagram in Figure 3.</td></tr>
<tr><td>Attributes:</td><td><code>startAfter</code></td><td>The earliest possible moment in time the energy profile can be started.</td></tr>
<tr><td></td><td><code>startBefore</code></td><td>The last possible moment in time the energy profile can be started.</td></tr>
<tr><td></td><td><code>/maxDuration</code></td><td>A derived attribute indicating the maximum duration of an <code>EnergyConstraintProfile</code>. This maximum duration is based on the maximum duration of the <code>EnergyConstraintInterval</code> objects referred to by the <code>EnergyConstraintProfile</code>. This duration can be calculated with the following algorithm:

<pre>maxDuration = 0
for p in energyConstraintProfile.intervals {
      maxDuration += p.maxDuration *
      energyConstraintProfile
      .intervalDurationStep
}</pre></td></tr>
</table>

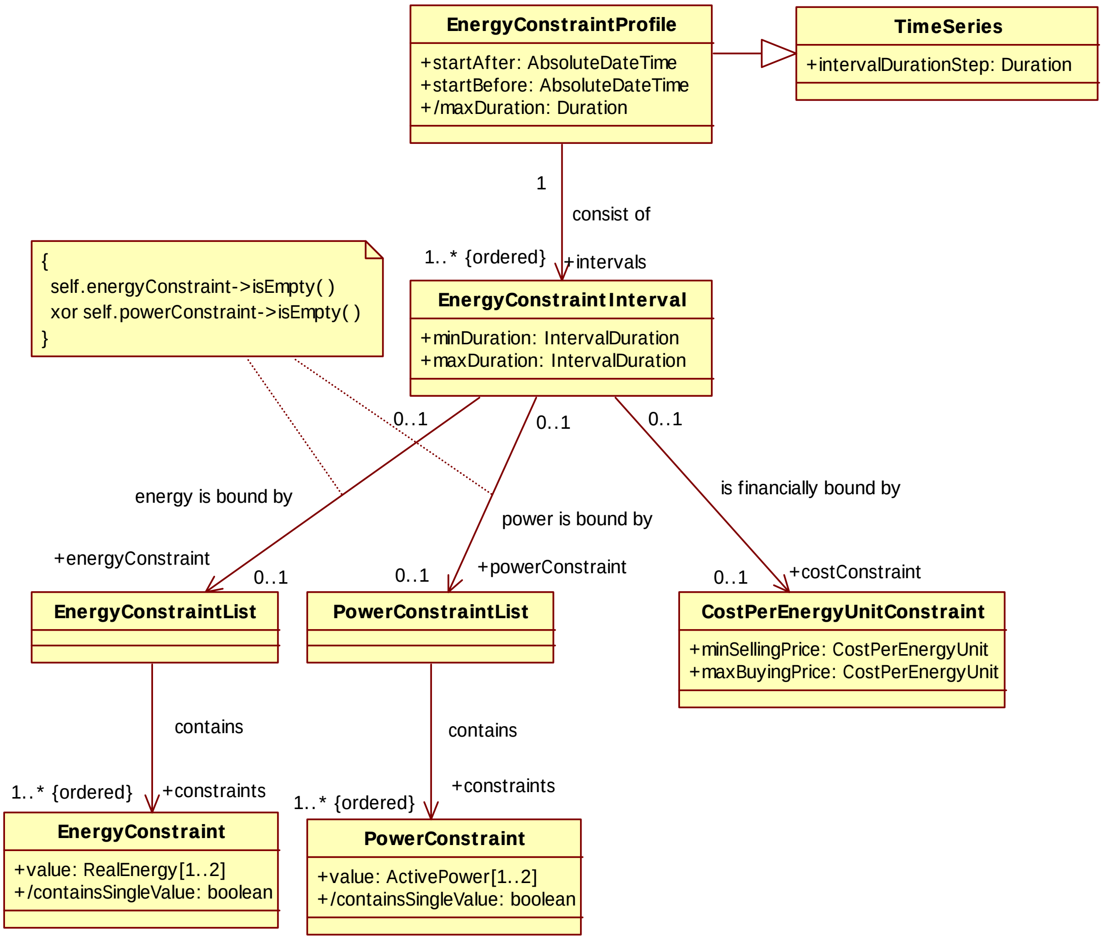

Figure 3: `EnergyConstraintProfile`

`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`.

`EnergyConstraintInterval`		
Description:	The `EnergyConstraintInterval` represents an interval of time that is part of an `EnergyConstraintProfile`. The duration of an `EnergyConstraintInterval` may be flexible.	
Attributes:	`minDuration`	This 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 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.

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.
	`/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).	

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).	

CostPerEnergyUnitConstraint		
Description:	This class expresses financial constraints that are related to the `EnergyConstraintInterval` it is associated with.	
Attributes:	`minSellingPrice`	This is the minimum price per unit of energy that one is willing to receive when selling energy during the associated `EnergyConstraintInterval`.
	`maxBuyingPrice`	This is the maximum price per unit of energy that one is willing to pay when buying energy during the associated `EnergyConstraintInterval`.

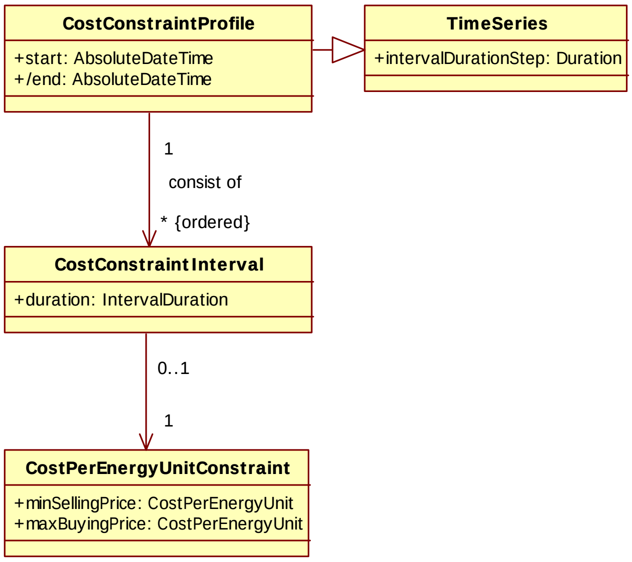

Figure 4: `CostConstraintProfile`

`CostConstraintProfile`		
Description:	The `CostConstraintProfile` 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 `CostConstraintProfile` instance should be equal to the maximum length of the `EnergyConstraintProfile`; c.f. section 3.2.2.	
Attributes:	`start`	This is the starting time of the `CostConstraintProfile`
	`end`	This is the end time of the `CostConstraintProfile`; this attribute is derived from the durations of the `CostConstraintInterval` objects referred to.

`CostConstraintInterval`		
Description:	The `CostConstraintInterval` represents an interval of time that is part of a `CostConstraintProfile`. The duration of these intervals is fixed.	
Attributes:	`duration`	This attribute expresses the duration for this `CostConstraintInterval`. 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.

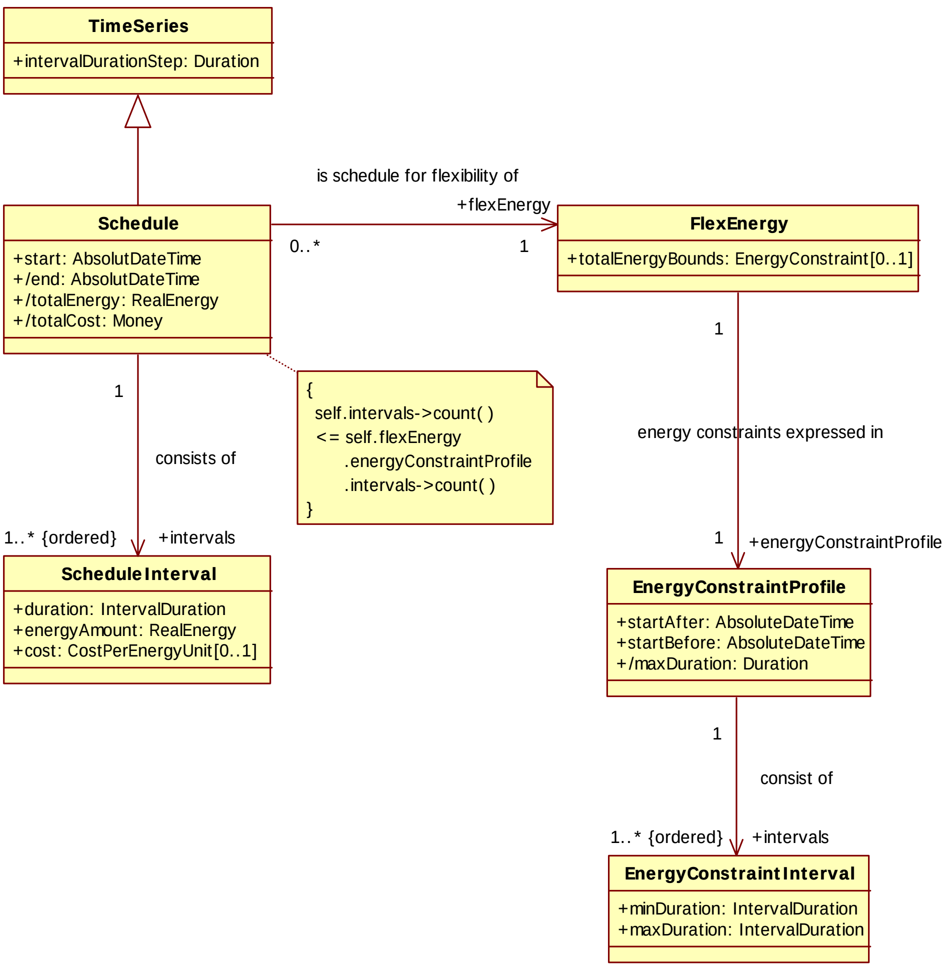

Figure 5: Schedule

Schedule	
Description:	The Schedule 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 5 indicates that a schedule 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 schedule starts.

	`/end`	This calculated attribute represents the end point of this schedule.
		
		`end = 0;`
		`foreach(interval in this.intervals) {`
		`      end += interval.duration;`
		`}`

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

	`/totalCost`	The total amount of money that is expressed through this schedule.
		
		`totalCost = 0;`
		`foreach(interval in this.intervals) {`
		`      totalCost += interval.cost;`
		`}`

Constraints:	`self.intervals->count( )`
	`<= self.flexEnergy.energyConstraintProfile.intervals->count( )`
	
	This constraint entails that for this interval either power or energy consumed or produced in this interval is to be constrained.	

`ScheduleInterval`		
Description:	The `ScheduleInterval` represents a interval of time that is part of a Schedule	
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.
	`energyAmount`	The amount of energy that is associated with this `ScheduleInterval`.
	`cost`	The costs per unit of energy that is associated with this `ScheduleInterval`.

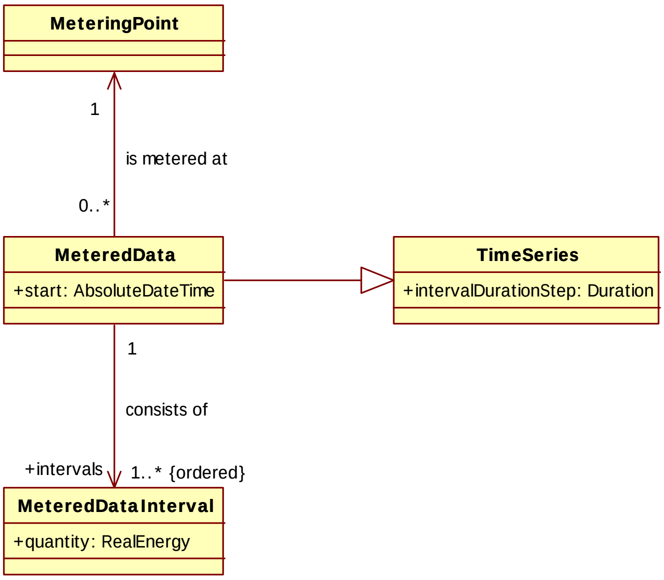

Figure 6: `MeteredData`

`MeteredData`		
Description:	The `MeteringData` class is used to see to which extent the Schedule has been adhered to. It is based on Exchange Metered Data from ebIX (see [EMD09].	
Attributes:	`start`	The starting point in time of the `MeteredDataInterval` array that is associated with the `MeteringData` class.

`MeteredDataInterval`		
Description:	The `MeteredDataInterval` class contains an actual measurement.	
Attributes:	`quantity`	The amount of energy that is associated with this metered data interval.

3.2.2 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. Their relations are described in formula (1) below; it describes the relation between `CostConstraintProfile` and `EnergyConstraintProfile`.

$$\begin{aligned} & \mathit{CostConstraintProfile.start} \\ = \; & \mathit{EnergyConstraintProfile.startAfter} \\ \leq \; & \mathit{Schedule.start} \\ \leq \; & \mathit{EnergyConstraintProfile.startBefore} \\ < \; & (\mathit{EnergyConstraintProfile.startBefore} \\ & + \mathit{EnergyConstraintProfile.maxDuration}) \\ = \; & \mathit{CostConstraintProfile.end} \end{aligned} \qquad (1)$$

Please note that this timeline expresses the fact that the `CostConstraintProfile` should completely envelop the interval that is the maximal interval potentially covered by the `EnergyConstraintProfile`.

3.2.3 Model usage constraints considered within MIRACLE

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

3.2.3.1 MIRACLE Financial constraints

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

- `FinancialConstraint` and `CostConstraintProfile` will not be used
- The usage of `CostPerEnergyUnitConstraint` and `ScheduleInterval.cost` is obligatory.

3.2.3.2 MIRACLE Energy constraints

The following restrictions are used when expressing energy profiles.

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

3.2.4 Examples usage of expressions of flexibility in `FlexEnergy`

This section describes a number of examples of the usage of the `FlexEnergy` construct. First the shifting of a simple block is described, and then a more complex scenario within which flexibilities in charging electric vehicles is provided.

3.2.4.1 Shiftable device example using `EnergyConstraintProfile.startBefore` and `startAfter`

Figure 7 graphically represents an example `FlexEnergy` with one `EnergyConstraintInterval` (which is fixed in duration as well as energy). The `startAfter` and `startBefore` are unequal to each other, thus the profile as described with the single `EnergyConstraintInterval` can start at any point between `startAfter` and `startBefore` (e.g. can start between 2pm and 7pm).

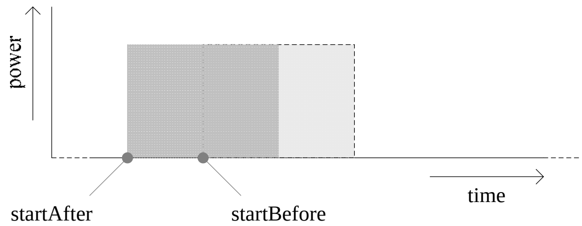

Figure 7 Example of simple shiftable device

The block with the solid line represents the profile at its earliest starting point; the lighter block with the dashed line represents the profile at its latest starting point. It must be noted that more complex modeling than a single block profile might be required.

3.2.4.2 Electric Vehicle charging example with `FlexEnergy.totalEnergyBounds` in combination with `EnergyConstraintInterval.energyConstraint`

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 8 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).

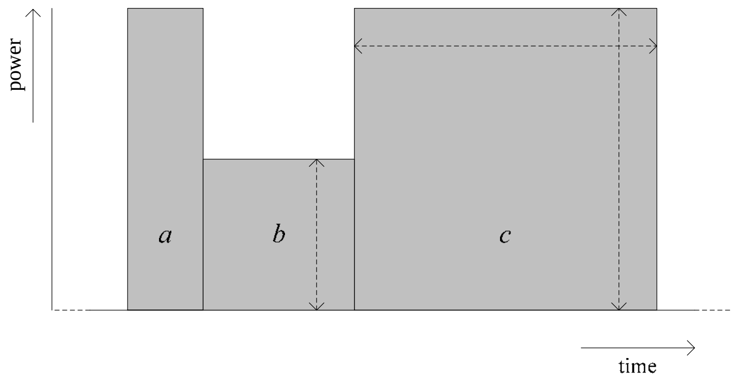

Figure 8 Example constraints in charging pattern for an electric vehicle

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`.

$$15\ min. \leq a_d \leq 15\ min. \quad (1)$$
$$6\ kW \leq a_p \leq 6\ kW \quad (2)$$

$$60\ min. \leq b_d \leq 60\ min. \quad (3)$$
$$0\ kW \leq b_p \leq 3\ kW \quad (4)$$

$$0\ min. \leq c_d \leq 60\ min. \quad (5)$$
$$0\ kW \leq c_p \leq 6\ kW \quad (6)$$

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`.

$$6\ kWh \leq a_d \times a_p + b_d \times b_p + c_d \times c_p \leq 6\ kWh \quad (7)$$

3.2.5 General utility classes

This paragraph defines a number of general utility classes which are used throughout the data model described in this deliverable.

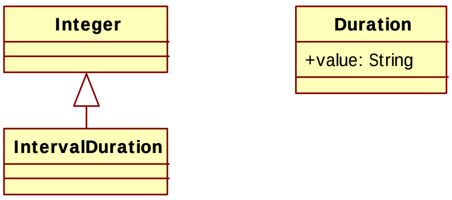

Figure 9: Utility classes

`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.

4 Message model

This chapter builds upon the data model that is described 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 “BusinessEntityState”.

4.2 Role level message model

There is a relation between the processes that are described in WP1 and the business process described here. The latter only focus on the exchange of messages between swimming lanes. The internal processes of a swimming lane are not described; 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). The business process in this chapter could be seen as a template process for WP1 processes.

4.2.1 FlexOffer Business Process

The main idea behind the business process that is described 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 matched is not part of this process description.

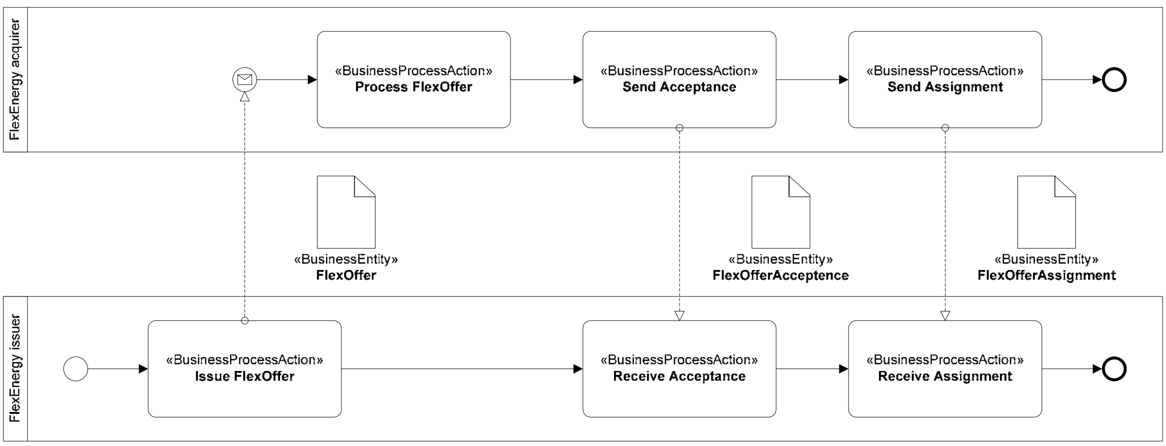

Figure 10: FlexOffer business process

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 Receiver 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 Receiver within the boundaries that were stated in the original FlexOffer. The `FlexOfferAcceptance` and `FlexOfferAssignment` entities may also be combined into a single message.

4.2.1.1 FlexOffer

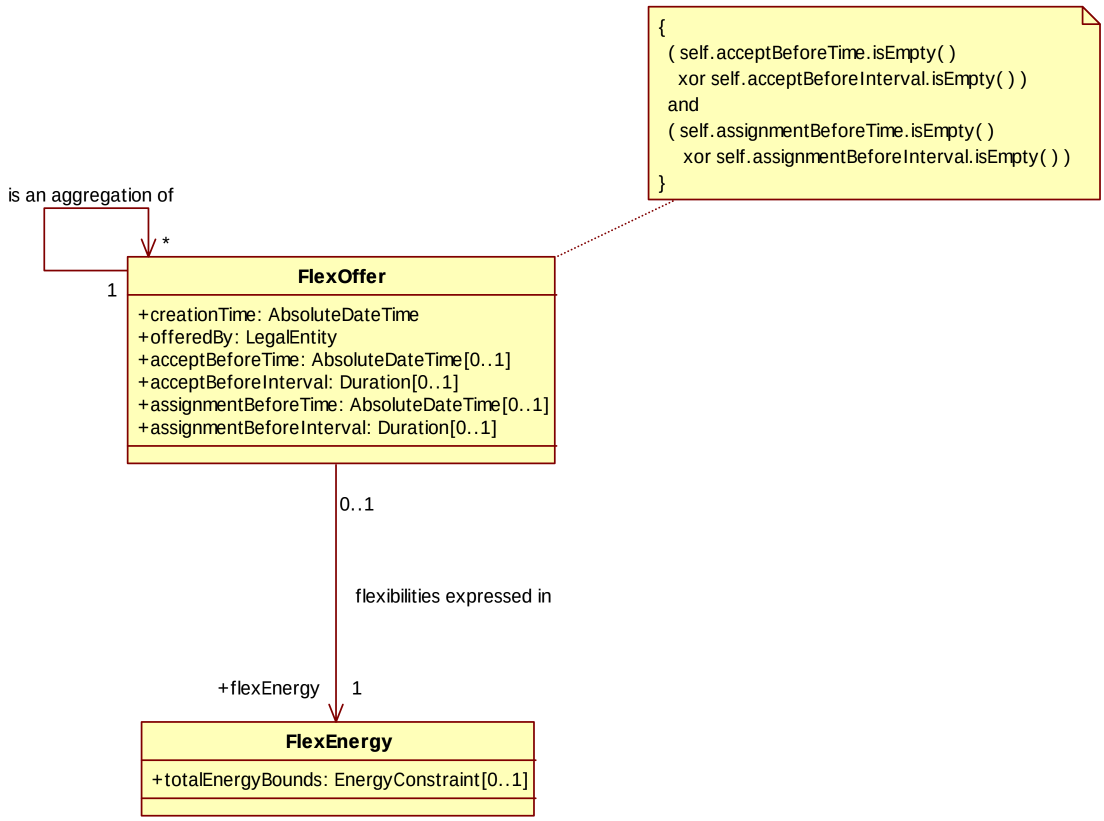

Figure 11: FlexOffer BusinessEntityView

`FlexOffer`	
Description:	Through a FlexOffer `FlexEnergy` can be offered on a marketplace. This can either be a bilateral market or auctioning. 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:	`creationTime`	The moment in time this `FlexOffer` was created.
	`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 sender needs to know whether the `FlexOffer` is going to be exerted or not. The receiver notifies the sender by sending a `FlexOfferAcceptance` message (see **Error! Reference source not found.**).
	`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 `Schedule.start`) before which acceptance of the FlexOffer is to be communicated.
	`assignmentBefore Time`	A moment in time by which the sender 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 `Schedule.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 responsibilities. The same goes for `FlexOfferAcceptance` and `FlexOfferAssignment`.

4.2.1.2 FlexOfferAcceptance

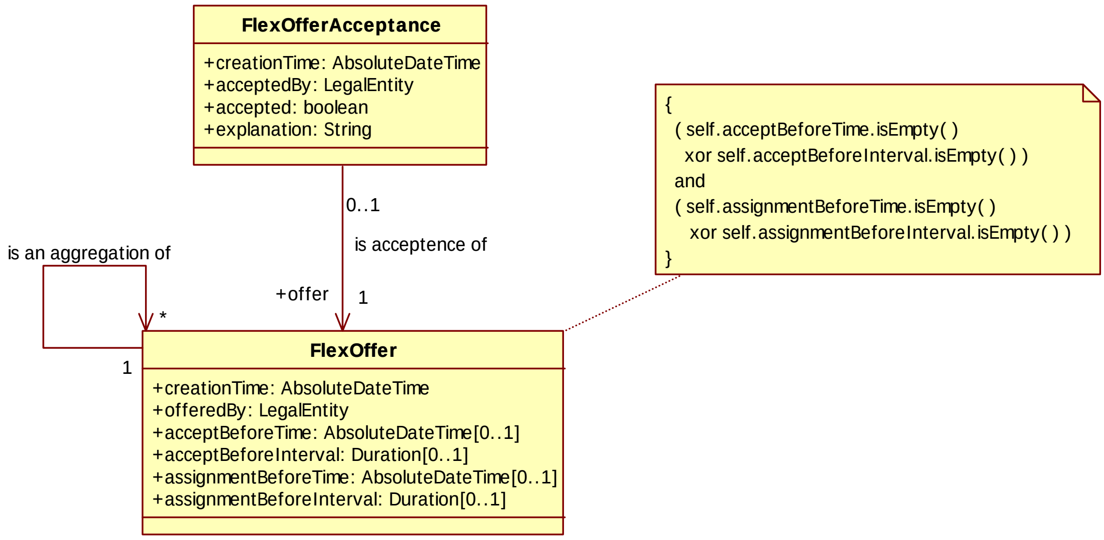

Figure 12: FlexOfferAcceptance BusinessEntityView

`FlexOfferAcceptance`		
Description:	This class contains information about whether a `FlexOffer` has been accepted by the `FlexOffer` receiver or not. Therefore it can only be associated with a single `FlexOffer`. Acceptance of a `FlexOffer` only signifies that the receiver 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:	`creationTime`	The moment in time the `FlexOfferAcceptance` was created.
	`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.

4.2.1.3 FlexOfferAssignment

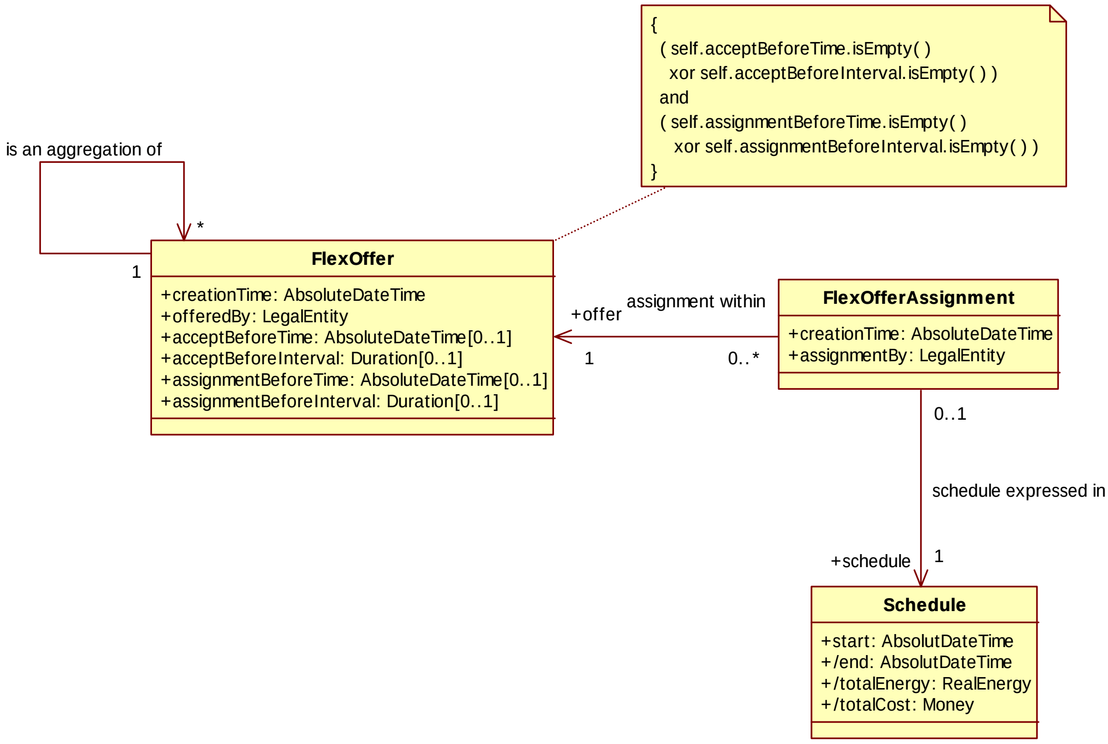

Figure 13: FlexOfferAssignment BusinessEntityView

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 Schedule) so that the sender of the FlexOffer knows what to do. An assignment is always associated with a single FlexOffer.	
Attributes:	creationTime	The moment in time the FlexOfferAssignment was created.
	assignedBy	This attribute represents the LegalEntity that is responsible for issuing this FlexOfferAssignment.

4.2.2 Temporal constraints

This section indicates the temporal constraints which apply to `FlexOffer`, `FlexOfferAcceptance`, `FlexOfferAssignment and Schedule`. They are described in formula (1) below.

$$absoluteAcceptBeforeTime = FlexOffer.acceptBeforeTime$$
$$xor$$
$$absoluteAcceptBeforeTime = Schedule.start - FlexOffer.acceptBeforeInterval$$

$$absoluteAssignmentBeforeTime = FlexOffer.assignmentBeforeTime$$
$$xor$$
$$absoluteAssignmentBeforeTime = Schedule.start - FlexOffer.assignmentBeforeInterval$$

$$\begin{aligned} & FlexOffer.creationTime \\ < \; & FlexOfferAcceptance.creationTime \\ \leq \; & absoluteAcceptBeforeTime \\ \leq \; & FlexOfferAssignment.creationTime \\ \leq \; & absoluteAssignmentBeforeTime \\ \leq \; & Schedule.start \\ < \; & Schedule.end \end{aligned} \qquad (1)$$

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

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, MIRACLE project

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

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).

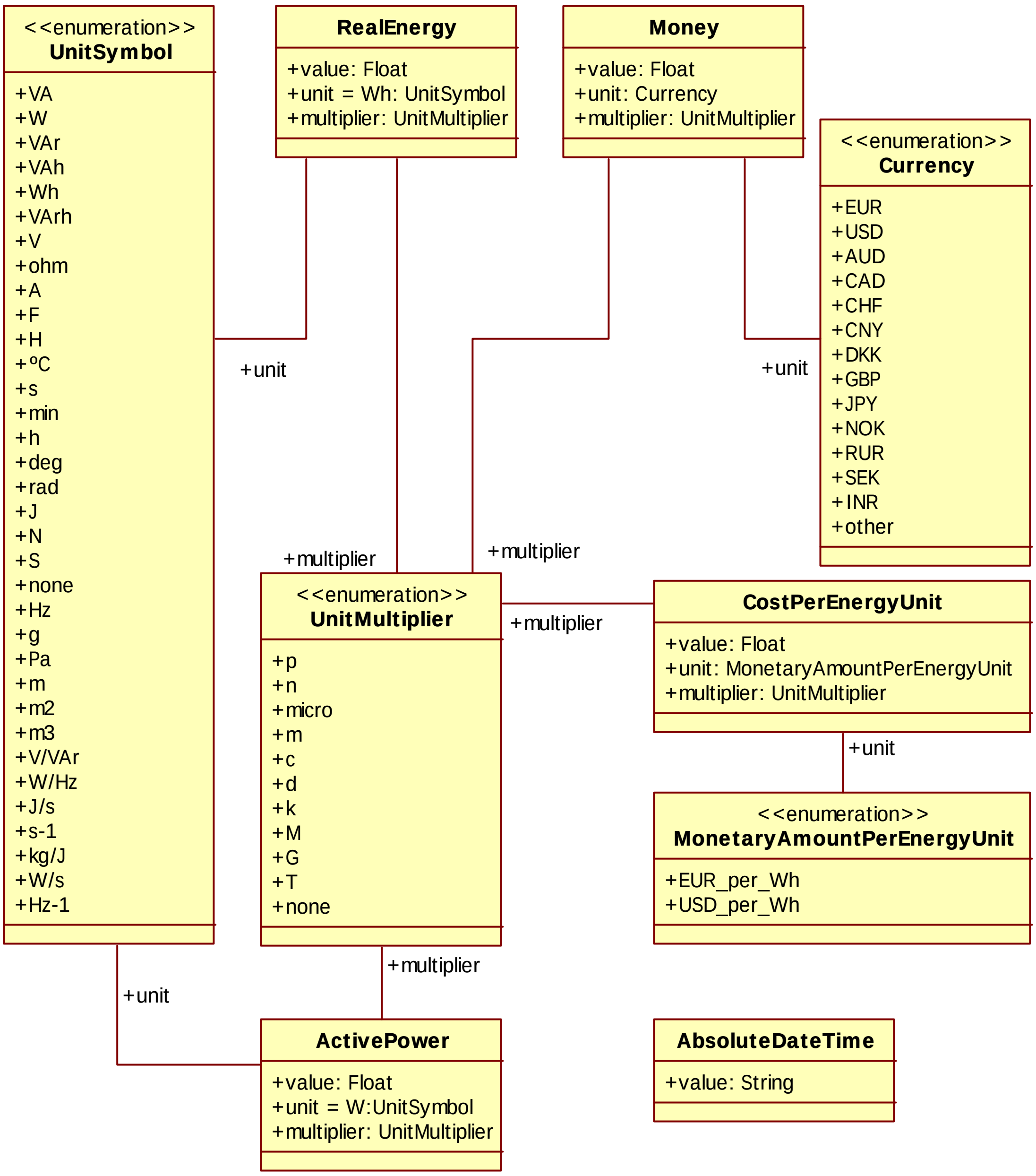

Figure 14: IEC CIM derived classes in the common package

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]	

B Potentially relevant CIM classes

This appendix contains a list of classes from [CIM09] which are identified as potentially relevant for the MIRACLE project, but which are not yet used.

`AbsoluteDate`	
Description:	Date and time as "yyyy-mm-dd", this conforms to ISO 8601. UTC time zone is specified as "yyyy-mm-dd".
Source:	[CIM09]

`BasicIntervalSchedule`	
Description:	Schedule of values at points in time.
Source:	[CIM09]

`CostRate`	
Description:	Cost, in units of currency, per hour of elapsed time.
Source:	[CIM09]

`Curve`	
Description:	Relationship between an independent variable (X-axis) and one or two dependent variables (Y1-axis and Y2-axis). Curves can also serve as schedules.
Source:	[CIM09]

`CurveData`	
Description:	Data point values for defining a curve or schedule.
Source:	[CIM09]

`CurveStyle`	
Description:	Style or shape of curve.
Source:	[CIM09]

`IrregularIntervalSchedule`	
Description:	The schedule has `TimePoints` where the time between them varies.
Source:	[CIM09]

`IrregularTimePoint`	
Description:	`TimePoints` for a schedule where the time between the points varies.
Source:	[CIM09]

MonetaryAmountRate	
Description:	Monetary amount per second.
Source:	[CIM09]

RegularIntervalSchedule	
Description:	The schedule has `TimePoints` where the time between them is constant.
Source:	[CIM09]

RegularTimePoint	
Description:	`TimePoints` for a schedule where the time between the points is constant.
Source:	[CIM09]

Temperature	
Description:	Value of temperature in degrees Celsius.
Source:	[CIM09]

Voltage	
Description:	Electrical voltage.
Source:	[CIM09]