university of groningen

faculty of mathematics and natural sciences

A STANDARDISED MULTI-AGENT SYSTEM FOR ENERGY MARKET SIMULATION

RALPH KIERS

Master's Thesis

Distributed Systems Group
Faculty of Mathematics and Natural Sciences
University of Groningen

May 2015 – version 1.1

SUPERVISORS:
Prof.dr.ir. Marco Aiello
Prof.dr.ir. Paris Avgeriou
Dr. Giuliano Andrea Pagani

ABSTRACT

In recent years the interest in smart grids, a modernised electrical grid that uses information and communication technology to gather and act on information in an automated manner, has grown. In previous work, the distributed systems group at the University of Groningen has created a multi-agent system through which agents can trade energy, simulating the behaviour of a smart grid. The agent communication however was done in a non standardised way. In this thesis the Energy Market Information Exchange, Open Automated Demand Response Communications Specification and Energy Interoperation standards are evaluated based on the requirements of the current implementation and several software quality attributes. Based on this evaluation the Energy Interoperation standard was chosen as a basis for a standardised implementation of the multi-agent system. This standardised implementation is compared to the original in terms of data overhead. The results show an increase in the order of magnitude of seven more bytes of data exchanged compared to the original. Further calculations show a $70 - 80\%$ overhead compared to the data exchanged. Finally, the thesis concludes with a calculation of needed bandwidth using a real world example and remarks on the Energy Interoperation standard.

We live in a society exquisitely dependent on science and technology, in which hardly anyone knows anything about science and technology.

— Carl Sagan

ACKNOWLEDGEMENTS

Firstly, I would like to thank Dr. Giuliano Andrea Pagani for the guidance and feedback he provided during my thesis. I am grateful for the time he took to read my draft versions. I would also like to thank Prof.dr.ir. Marco Aiello for his support and insight. Finally, I wish to thank my parents for their support throughout the entirety my study.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

GLOSSARY

L

load shedding

Reducing the electrical load on something, in order to prevent excessive load/current from damaging electrical components.

load shifting

Moving electrical energy from one part of the electrical grid to an other part of the grid.

O

option

As defined in eMIX[39]: "An option is an instrument that gives the buyer the right, but not the obligation, to buy or sell a product at a set price during given time windows. Many typical energy agreements, including demand response and reserves, include elements that would give them the name option in any other market.".

P

prosumer

A person who combines the economic roles of producer and consumer.

T

Transmission System Operator

Entity that is responsible for transporting electrical power in a region using fixed infrastructure, for more info [see 30].

ACRONYMS

Numbers

3DES

Triple Data Encryption Standard

A

ACL

Agent Communication Language

AES

Advanced Encryption Standard

D

dr

demand response

DRAS

Demand Response Automation Server

E

EI

Energy Interoperation

eMIX

Energy Market Information Exchange

F

FIPA

Foundation of Intelligent Physical Agents

G

gebp

general event-based program

Genco

Any company doing electricity generation

GML

Geography Markup Language

guid

globally unique identifier

H

HMAC

Key-based Message Authentication Code

I

iso

independent system operator

J

JADE

Java Agent Development Framework

M

MAC

Message Authentication Code

mas

multi-agent system

MIC

Message Integrity Code

O

OASIS

Organisation for the Advancement of Structured Information Standards

OpenADR

Open Automated Demand Response Communications Specification

R

RSA

Rivest, Shamir, & Adleman

RUG

University of Groningen

S

SHA

Secure Hash Algorithm

SI

International System of Units (abbreviated SI from French: Le Système International d'Unités)

SIP

Session Initiation Protocol

SOAP

Simple Object Access Protocol

T

tc

technical committee

TCP

Transmission Control Protocol

TLS

Transport Layer Security

TSO

Transmission System Operator

U

uri

uniform resource identifier

UTC

Coordinated Universal Time

V

VEN

Virtual End Node

VTN

Virtual Top Node

W

WS-Calendar

Web Services Calendar

WSDL

Web Services Description Language

X

XML

Extensible Markup Language

XSD

XML Schema Definition

1 INTRODUCTION

Due to the continuous growth in capacity of renewable energy sources, such as wind- and solar power more and more of the generated amount of energy becomes irregular. Due to this irregular nature excess amounts of energy have to be stored when the production exceeds the demand. Storage costs space and energy inevitably partly converts to heat during transport, all of this costs money, and might lead to the burning of more fossil fuel than needed. An other option is to shape the demand is such a way that it can deal with the irregularity of these renewable energy sources. So the need for a more sophisticated energy grid is apparent.

The smart grid is a modernised electrical grid that uses information and communication technology to gather and act on information in an automated manner. The goal of smart grids is to maximise load shifting while the energy production/demand constantly changes. Through such a smart grid it is possible for consumers, prosumers and suppliers to be involved directly with energy production and consumption. For example, a prosumer who has solar panels, but is currently producing more energy than he is using could sell his excess energy through the smart grid. An other case could be lowering the price during overproduction, so people can run big energy consuming devices while the price is low, thus avoiding storage and energy loss. Smart grids will facilitate a deregulated open energy market, opposite to the situation where there are only a few power producing companies, an open energy market will increase competition and give more power to the consumer. This leads to increased innovation, more optimised production methods and lower energy prices. A deregulated market will also allow new start up companies to enter the market without being fiercely undercut by the competition. It also allows smaller energy producers, such as prosumers to enter the market. Finally customers will be able to easily compare prices and can directly infer how there energy was generated.

In order to operate local or global smart grids it is vital that each individual component can communicate with the other components. Moreover each component should be able to accurately and reliably communicate with other components so that it can determine a subset of production, pricing, consumption and other information that suit its needs. In order to facilitate this, standardisation is needed. Without standardisation an open market in which consumers, prosumers and producers participate will not work. Since expectations and obligations cannot be verified, effectively meaning deals cannot be made.

In previous work, the distributed systems group at the University of Groningen (RUG) has created a multi-agent system (mas)[5, 15] through which agents can trade energy, simulating the behaviour of a smart grid. The aim of this thesis is to:

- Evaluate several state of the art standards based on:
  - Security
  - Reliability
  - Extensibility
  - Data efficiency
- Implement a standard chosen based on the evaluation, on top of the current platform
- Analyse resulting impact on the current implementation

2 STATE OF THE ART

2.1 REFERENCE MAS IMPLEMENTATION

As mentioned in chapter 1 the distributed systems group at the RUG has created a mas through which agents can trade energy, simulating the behaviour of a smart grid. This chapter explains the agents and how they interact with each other in the mas.

There are two versions of the mas available, namely the original[5] and an extended[15] version. The main difference between the two is, that the original version runs the agent simulation in a single local agent container whereas the extended version allows simulations to be run distributed over multiple agent containers. In the latter version, containers can be situated locally as well as remotely. Although the extended version does implement an additional agent for exception handling and extra features such as replication, the original agents still interact largely in the same manner on the energy exchange front.

2.1.1 *Agents*

The mas implementations have three main roles among which the agents are divided. The roles and agents are as follows:

- Supplier
  - Genco agent
  - Prosumer agent
- Consumer
  - Consumer agent
- Top-level intermediaries
  - Time agent
  - Weather agent
  - Balancer agent
  - Exception handling agent[1]

2.1.1.1 *Genco*

Each company doing electricity generation (Genco) agent represents a power producing company that can produce non-renewable energy

1 The exception handling agent is only present in the extended mas.

at a maximum capacity $E_{g_{max}}$. The actual generated energy E_g is suggested by the balancer agent, based upon a number of variables, see section 2.1.1.6. Once a Genco has its E_g it can respond to requests from consumers, with a proposed price, or with a message that it has reached its energy limit. The proposed price is calculated by equation (2.1).

$$C = \begin{cases} T(\|B_{loc} - S_{loc}\| + 1) + C_p & \text{if } \tau \leqslant E_g \\ C_e \Delta\tau (T(\|B_{loc} - S_{loc}\| + 1) + C_p) & \text{if } \tau > E_g \end{cases} \tag{2.1}$$

where C denotes the proposed price, T denotes the cost per distance; as per contract with the Transmission System Operator (TSO), B_{loc} and S_{loc} denote the location of the buyer/seller, C_p denotes the cost of energy production, $\tau = T + E_r$; where T denotes the amount of energy already supplied and E_r denotes the requested amount of energy, C_e denotes an external cost constant; with $C_e > 1$, and $\Delta\tau = \tau - E_g$ denotes how much threshold is exceeded. The energy production cost C_p term depends upon the weather, market prediction, and balancing forecasting.

2.1.1.2 *Prosumer*

Prosumer agents represent user of the electrical power grid that are also capable of generating their own energy. They start off by determining their location and requesting the temperature and solar- and wind-power values. Using this data they calculate the amount of energy they produce, and the amount of energy E_p they can supply. Once E_p is calculated a prosumer can negotiate an energy contract with multiple consumers and supply them with energy.

2.1.1.3 *Consumer*

The consumers start off by determining their energy demand and location. After their demand is determined they can start contacting prosumers in order to get the amount of energy they produce, as well as their proposed price. Next the consumers contact all Gencos and get their position. Based on this information, the consumer sets its preferred Genco as the one closest to itself. Consumers now try to negotiate a contract with the cheapest suitable prosumer, if they fail they move on to the next cheapest prosumer. If the consumer fails to make a deal with any of the prosumers, then it contacts all Gencos to get their prices and negotiates a deal with the cheapest Genco.

2.1.1.4 *Time*

The time agent listens to requests from other agents. It responds to them with the global time. In the current implementation the time ranges from $[0, 5] \in \mathbb{N}$.

2.1.1.5 *Weather*

When the weather agent is started it first requests the time of the day from the time agent. Depending on the time of the day, the weather agent randomly generates a weather forecast. The forecast consists of three components, namely the temperature in °C, a value representing the solar power and a value representing the wind power. No units are given for the solar- and wind-power. Once the random weather is generated, prosumers can request the current weather from this agent.

2.1.1.6 *Balancer*

The balancer agent is at the heart of the simulation. As its first step the balancer agent searches for all Genco- and prosumer-agents and requests their positions as well as energy production. As the next step the balancer agent searches for all consumer agents and requests their positions as well as their energy demands. The agent also queries the weather agent in order to get the current weather, which will be displayed by the balancer later.

After acquiring the initial information, the agent calculates the exact amount of energy E_g each Genco should provide. E_g is calculated from the demand/supply equation given in equation (2.2).

$$\sum_{i=1}^{N_g} S_{G_i} + \sum_{j=1}^{N_p} S_{P_j} = \sum_{k=1}^{N_c} D_{C_k} \tag{2.2}$$

where N_g is the number of Gencos, N_p is the number of prosumers, N_c is the number of consumers, S_{G_i} is the amount of energy Genco i provides, S_{P_j} is the amount of energy prosumer j provides and D_{C_k} is the energy demand of consumer k. E_g is defined as $\frac{\sum_{i=1}^{N_g} S_{G_i}}{N_g}$, using equation (2.2), E_g can be expressed as shown in equation (2.3).

$$\begin{aligned}
\sum_{i=1}^{N_g} S_{G_i} + \sum_{j=1}^{N_p} S_{P_j} &= \sum_{k=1}^{N_c} D_{C_k} \\
\sum_{i=1}^{N_g} S_{G_i} &= \sum_{k=1}^{N_c} D_{C_k} - \sum_{j=1}^{N_p} S_{P_j} \\
E_g = \frac{\sum_{i=1}^{N_g} S_{G_i}}{N_g} &= \frac{\sum_{k=1}^{N_c} D_{C_k} - \sum_{j=1}^{N_p} S_{P_j}}{N_g}
\end{aligned} \tag{2.3}$$

E_g is then send to all the Gencos.

The balancer agent also listens to messages from consumers that confirm they have successfully made a contract with a supplier. This information is also used later in order to show which consumer- and supplier-agents engaged in a contract with each other.

2.1.1.7 *Communication*

Communication of the agents is done through the Java Agent Development Framework (JADE) platform explained in section 2.1.2. The messages sent in the current implementation of the mas do not adhere to any standard. They are simple control messages containing simple strings and integers.

2.1.2 *JADE*

JADE[1] is a Java based framework that can be used to develop agent based systems and forms the basis of the mas. JADE allows agents to be spawned in containers. JADE agents can communicate with other agents in the same container, agents in a different container on the same platform and with agents in a different container on an other platform. For agents in the same platform no messages are sent through external interfaces, but the passing of objects is used instead. For the local mas simulation this is the case and thus does not induce extra performance overhead. The content of messages between JADE agents follow the Foundation of Intelligent Physical Agents (FIPA) Agent Communication Language (ACL)[31] specification. This works fine for the current mas implementation, but might be superfluous when standardised messages are used. Each of the agents executes behaviours in its own thread. Coupled with the fact that message passing is asynchronous, this allows agents to execute their behaviours simultaneously much like one would expect from real world agents handling energy tenders and transactions.

2.2 RELATED WORK

Smart grids are a relatively new emerging phenomenon in the energy business. As such there is not one single standard that is used or widely accepted as "the" standard to use, yet. A lot of different parties have an interest in a standard for smart grids and energy exchange. Coupled with the fact that energy markets are local and likely to differ from country to country, this leads to a lot of research and proposals on the subject.

Various different mas platforms have been proposed and tested. Lamparter, Becher and Fischer[17] define a mas platform that balances demand and supply through a central market agent. The central market agent receives all the bids, production and consumption info from the consumers, prosumers and Gencos. It then uses a linear problem solver to do the balancing and decide who wins bids. On the energy agent's side it allows for the applying of constraints and policies; to energy use and device parameters. In other work, Dutch researchers have developed a mas platform called PowerMatcher[13,

14]. PowerMatcher has distributed control and aims to balance supply and demand and as such supports demand response (dr). It is different from the mas implementation by Capodieci[5] as it does not have a centralised balancing algorithm. It has been tested on the field by connecting six different energy producing sites with the Amsterdam Power Exchange to enable the trading of energy. In yet other research Garcia, Oliver and Gosch[9] define a layered mas architecture that can be used to manage agents in the smart grid. The architecture consists of a network mediation layer, management applications layer and an optional intermediate management layer. It supports a lot of features such as: task scheduling, network status monitoring, data collection, alarms, configuration management and more. There are no doubt many more mas platforms and architectures to be found in other literature. However, the aim of this thesis is not to implement a different mas system. Rather the current implementation is to be used and the messaging is to be standardised.

Research done by Wang and Leung[26] examines the possibility of application layer protocols for smart meter node communication on top of the Internet Protocol. More specifically it examines the ANSI C12.22[32] standard and the Session Initiation Protocol (SIP)[23]. Both are evaluated based upon a set or criteria; such as interoperability, scalability, security, extensibiliy, etc. Both of the protocols define message formats that facilitate communication between nodes and a meter data management system. The SIP however does not specify the format for messages relating to energy, bidding or pricing. It merely specifies how communication sessions are handled. And while the ANSI C12.22 standard supports schedules and time intervals, it does not specify interactions for bidding or energy exchange from prosumers to consumers. The paper[12] by Gungor et al. provides an explanation on a number of technologies and standards related to smart grids, including the aforementioned ANSI C12.22 standard. What is interesting is that it only lists one standard out of twenty-two which is capable of dynamic pricing, namely Open Automated Demand Response Communications Specification (OpenADR); a standard which is discussed in section 3.2.

3 STANDARDS

In this chapter three standards relating to energy exchange are discussed. First a general overview of the standards and how they work is given. Then they are evaluated based on how well they fit the current mas implementation. Based upon the evaluation one of these standards is chosen to be implemented.

3.1 ENERGY MARKET INFORMATION EXCHANGE

Energy Market Information Exchange (eMIX)[39] is a standard developed by the Organisation for the Advancement of Structured Information Standards (OASIS) eMIX technical committee (tc). EMIX defines an information model for exchanging price and product information for power and energy markets. The product information includes quantity and quality of supply as well as schedules to enable effective exchange of information. The supply and the use of energy are time dependent, thus the market value of energy is time dependent as well. Therefore the time of delivery is a significant component of the product definition in eMIX. In order to incorporate this information in the product definition eMIX makes use of the Web Services Calendar (WS-Calendar)[40] standard.

EMIX specifies its information model in an Extensible Markup Language (XML)[3, 4] format. The XML Schema Definition (XSD)[2, 25] files provided by the standard provide core eMIX abstract types, as well as several extensions, most notably for electrical power. The standard itself does not specify communication protocols or business processes, it solely defines the information model. As a result there is nothing in the standard regarding security. To paraphrase the eMIX standard[39, line 186], "eMIX relies on the business processes and communication protocols using the standard to ensure secure exchange of price and product information."

In the following sections the eMIX information model and its types are explained.

3.1.1 *Namespaces*

Types and elements discussed throughout section 3.1 will be referred to with their namespace prefix attached were appropriate. As such, as a convenience to the reader all the namespaces pertaining to the eMIX and standards it makes use of can be found in table 3.1.

Table 3.1: EMIX namespaces, [as defined in 39, line 154].

Prefix	Namespace
emix	http://docs.oasis-open.org/ns/emix/2011/06
scale	http://docs.oasis-open.org/ns/emix/2011/06/siscale
power	http://docs.oasis-open.org/ns/emix/2011/06/power
resource	http://docs.oasis-open.org/ns/emix/2011/06/power/resource
xs	http://www.w3.org/2001/XMLSchema
gml	http://www.opengis.net/gml/3.2
xcal	urn:ietf:params:xml:ns:icalendar-2.0

3.1.2 *Core types*

In this section an overview of the most important core types for the product information as specified in the XSD is given. Figures 3.1 to 3.2 show a class diagram of all the types and attributes present in the emix namespace, with the exception of classes derived from `emix:BaseTermType`.

DeliveryType Derived from `emix:EmixBaseType`, reports the delivery of a product.

EmixBaseType A type derived from the `xcal:VcalendarType`, this means that an `emix:EmixBaseType` is an actual calendar type and as such conveys the duration and time intervals for the energy delivery. All `emix:ProductType` types are derived from this type.

EmixInterfaceType Abstract base type that describes products delivery, measurements and optionally pricing. For power this can be a node, geographical area, etc.

EmixOptionType An elaboration of the `emix:ProductType`. The option gives a buyer the right to buy or sell a product at a set price during the times provided by the schedule. It also contains information about availability and performance.

EnvelopeContentsType A type used to deliver eMIX warrants. Warrants are used to provide additional information about a product, such as information about the source of the energy or about its environmental characteristics.

ItemBaseType Abstract base type for eMIX product items. Items derived from `emix:ItemBaseType` should have at least a name, a unit of measure, and a scale factor; the scale factor is an International System of Units (SI) metric prefix[27] which denotes the scale factor

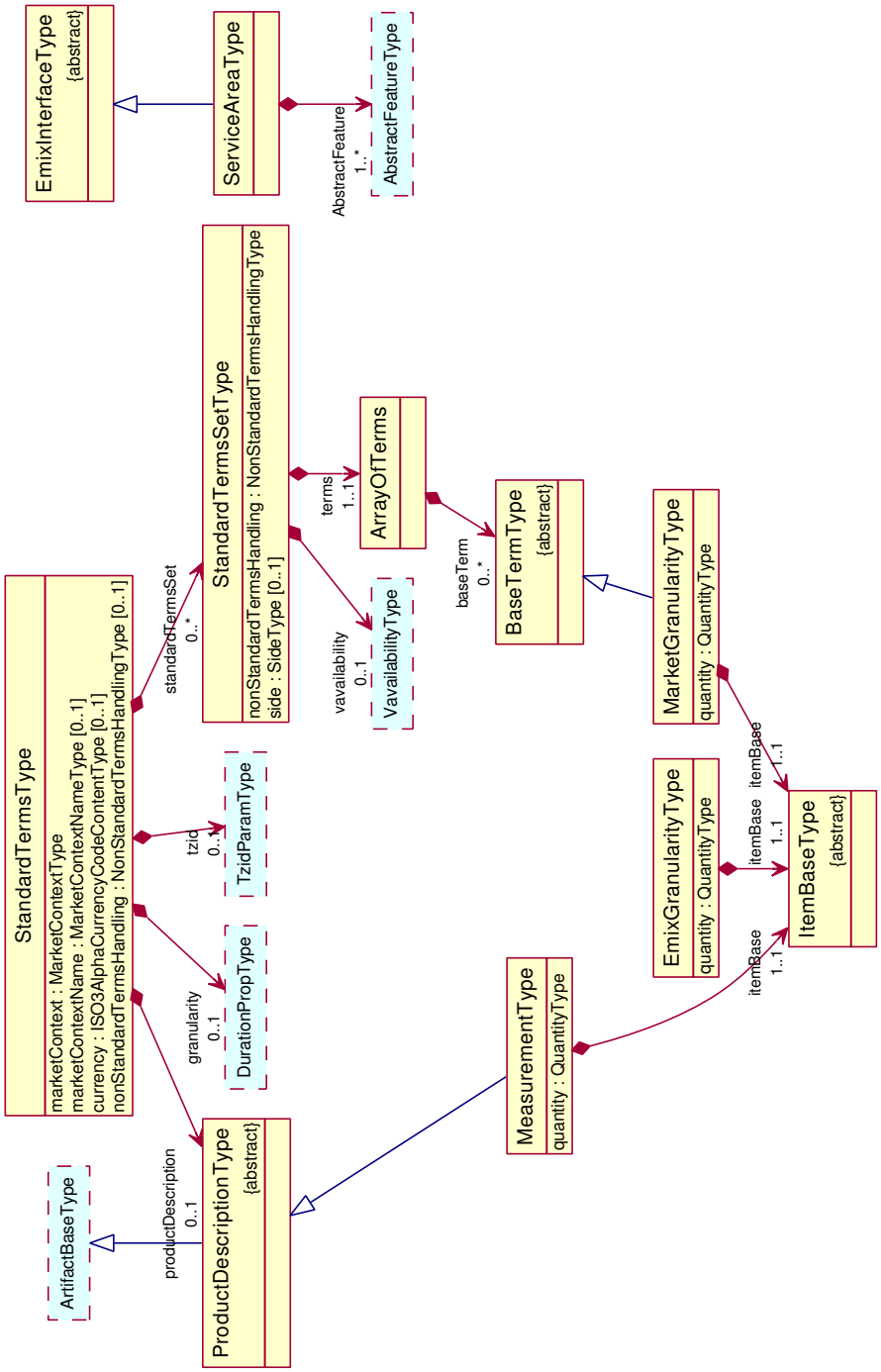


Figure 3.1: Class diagram view of classes related to `emix:StandardTermsType` and `emix:EmixInterfaceType`.

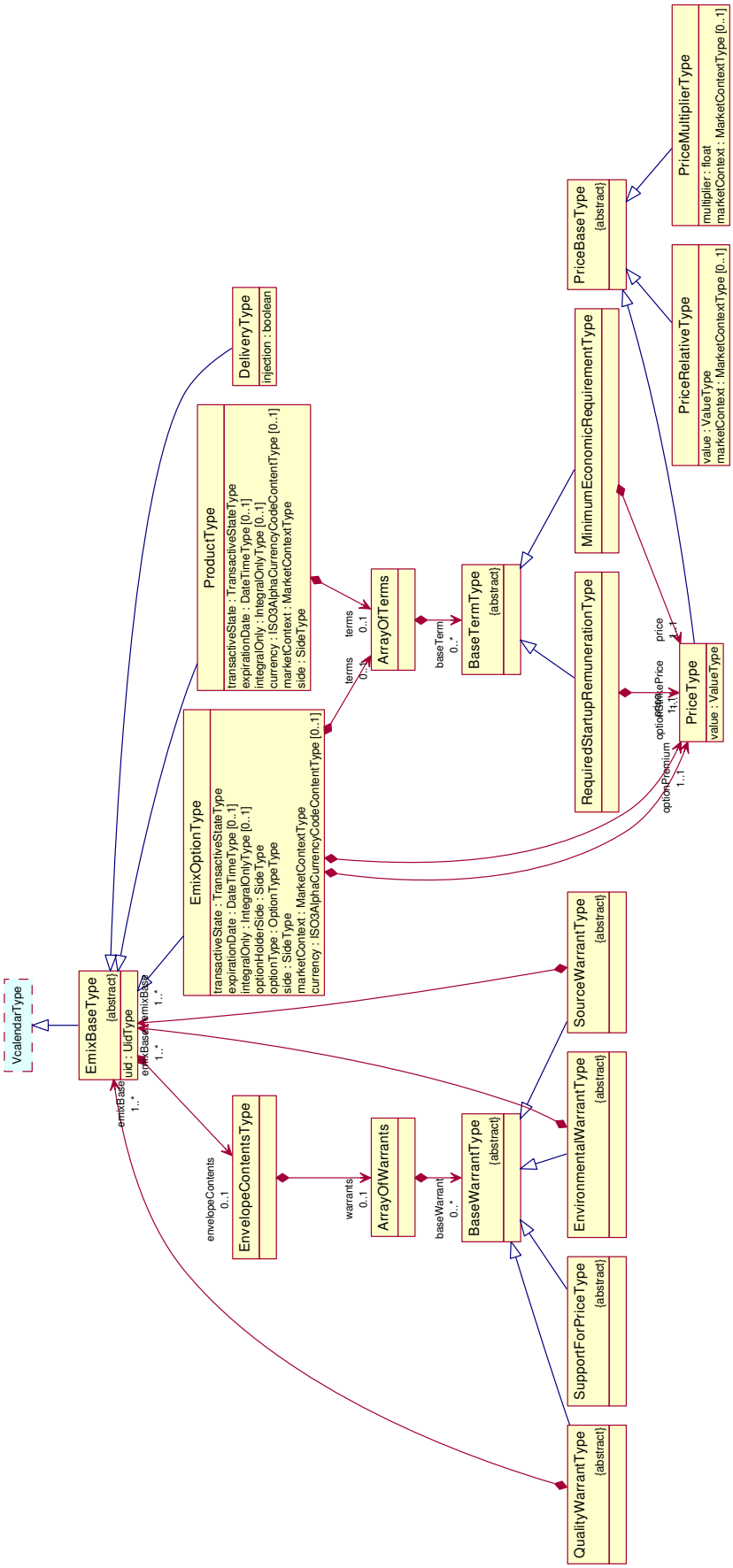


Figure 3.2: Class diagram view of classes related to `emix:EmixBaseType`.

in multiples of ten e. g. $p = 10^{-12}$ for pico or $\mu = 10^{-6}$ for micro. Quantity and price attributes are not included, because a single item can have different values for quantity and price over an interval.

PriceBaseType Abstract base type from which all other price types are inherited. Unless otherwise specified, wherever an `emix:PriceBaseType` is required, any of its derived types are allowed as well.

PriceMultiplierType A multiplier that is applied to a reference price to get the actual price per unit of a product. The reference price used is either the `emix:PriceType` from the optional `emix:MarketContextType`, or if the optional `emix:MarketContextType` is not defined the containing element's `emix:MarketContextType` is used instead. An `emix:PriceMultiplierType` can optionally contain an `emix:MarketContextType` for the reference price.

PriceRelativeType A price that is added to a reference price to get the actual price per unit of a product. The relative price value can be either positive or negative. Works similar to `emix:PriceMultiplierType`.

PriceType This type contains an absolute price which represents the price per unit of a product.

ProductDescriptionType Derived from the abstract `xcal:ArtifactBaseType`. An `xcal:ArtifactBaseType` can be placed inside a `xcal:VcalendarContainedComponentType`, this allows for `emix:ProductDescriptionType` types to be placed inside components which are part of the `xcal:VCalendarType`.

ProductType A type derived from `emix:EmixBaseType` representing a product which can be exchanged. It describes the schedule, location and source of the product.

ServiceAreaType A type that represents locations or geographical regions relevant to price communication. The `emix:ServiceAreaType` is defined using the simplest Geography Markup Language (GML)[35] profile.

3.1.2.1 *Simple types*

Apart from the more complex types eMIX also has a few simple types that represent semantics. These are the types of the attributes shown in figures 3.1 to 3.2 that are not part of an other class through an aggregation relationship. An explanation of those types is given in this section.

AutonomousType Derived from `xs:boolean`, describes if the resource of service is autonomous. If true, then the resource or service is able to respond or maintain service independently, otherwise it must await dispatch.

DateTimeType The `xcal:DateTimeType` listed for completeness.

EmixExtensionType Derived from `xs:string`, allows extending of string enumeration. The allowed values must match the regular expression "x-\S.*".

IntegralOnlyType Derived from `xs:boolean`, this type is present if the element is all(true) or nothing(false).

ISO3AlphaCurrencyCodeContentType Used to as the type for currency, all currencies are of this type. Currencies are always formatted according to the ISO 4217[33] standard.

MarketContextType Derived from `xs:anyURI`, the eMIX standard describes the `emix:MarketContextType` as: "A uniform resource identifier (uri) uniquely identifying a source for market terms, market rules, market prices, etc. The uri may or may not resolve.". For a standardisation of the `emix:MarketContextType` terms [refer to 39, chapter 7].

OptionTypeType An enumerated list of options. The possible values it can contain are a union of the `emix:OptionTypeEnumeratedType`[1] and `emix:EmixExtensionType`.

QuantityType Derived from `xs:float`, base type for all quantities.

SideType Derived from `xs:string`, an enumeration representing the interest of the party sending a derived `emix:EmixBaseType` type. The possible enumeration values are: Buy and Sell.

TransactiveStateType Derived from `xs:string`, This type is used as an enumeration to help with the processing of types derived from `emix:EmixBaseType`. The enumeration values are: Indication Of Interest, Tender, Transaction, Exercise, Delivery, Transport Commitment and Publication.

3.1.2.2 *Terms*

EMIX defines terms related to performance, scheduling and market requirements. As mentioned in section 3.1.2, not all the classes derived from `emix:BaseTermType` are shown in the class diagrams and thus not all terms are shown. For completeness a list of all the terms

1 `emix:OptionTypeEnumeratedType` is derived from `xs:string`, but the enumeration values are not defined.

is given in this section. The class names provide enough information to be able to understand their functionality. For a more detailed explanation [refer to 39, chapter 5].

- Performance terms

  `MinimumResponseDurationType`

  `MaximumResponseDurationType`

  `MinimumRecoveryDurationType`

  `MinimumDurationBetweenInvocationsType`

  `MinimumNotificationDurationType`

  `MaximumNotificationDurationType`

  `ResponseTimeType`

  `MaximumInvocationsPerDurationType`

  `MaximumConsecutiveDurationsType`

  `MaximumRunDurationType`

  `MinimumRunDurationType`

- Schedule terms

  `AvailabilityScheduleType`

  `UnavailabilityScheduleType`

  `NotificationScheduleType`

- Market requirement terms

  `MarketGranularityType`

  `MinimumEconomicRequirementType`

  `RequiredStartupRemunerationType`

  `MinimumStartsPerDurationType`

  `MinimumRemunerationRateType`

3.1.3 *Power extensions*

Next to the types described in the previous section, eMIX also defines extensions for electrical power. The number of defined extensions is quite impressive, there are extensions for transport charges, congestion charges, transport loss, maximum load and resource ramp up/-down to name a few. A lot of these extended types are not relevant to the current implementation of the mas described in chapter 2. Furthermore, the extensions for power derive from the core types described in section 3.1.2. The explanation of the core types should be sufficient to have an understanding of the eMIX standard and its capabilities, as such the eMIX power extensions will not be elaborated on in this section. For more information on the power extensions [see 39, chapters 8-17].

3.1.4 *Evaluation*

In order to evaluate the eMIX standard software quality attributes are used.

3.1.4.1 *Security*

As previously mentioned in section 3.1, eMIX relies on business processes and communication protocols for security. Nothing inherent in the standard provides any mechanics for encrypted data exchange, or user/device identity validation.

3.1.4.2 *Reliability*

EMIX only defines the structure of the data to be exchanged. As such there are no mechanisms to ensure reliability. There are no checks for possible wrong data, nor are there any integrity checks to unsure the data received is not altered along the way, either through error or malicious intent. As with security, eMIX has to rely on business processes or communication protocols for reliability.

3.1.4.3 *Efficiency*

In terms of efficiency the main focus is on data overhead and computation time needed for parsing.

As mentioned earlier, eMIX makes use of XML to send its messages. An example of an XML message conforming with the eMIX standard is shown in listing 3.1. The message clearly shows a lot more XML tags than actual message data. Granted the tags do convey information, namely the classification of data in the message. This example message is $\|\mathtt{full\_msg}\| = 1279$ bytes in length, including newlines. Note that although XML documents are specified in Unicode, the size of this example message corresponds to that of an ASCII document. This is the case because the example only contains characters that are in the ASCII set, which is a subset of Unicode that pertains the same binary format and thus size.

The newlines do not affect the parsing of the XML and thus can be removed in order to send a condensed message. The $\mathtt{full\_msg}$ contains 45 newlines, the condensed message length is the given by $\|\mathtt{condensed\_msg}\| = \|\mathtt{full\_msg}\| - 45 = 1234$ bytes. Stripping all the XML tags and removing all white space from $\mathtt{condensed\_msg}$ results in the actual data that the message conveys. In the case of the message in listing 3.1 this is equivalent to $\|\mathtt{data}\| = 155$ bytes. Without the tags all the meta data about the message is lost. However since each XML tag pair conveys classification information, they can also be replaced by five bytes of information. The data representation in listing 3.2 shows what the replacement bytes would represent. It should be noted that it is assumed no more than 256 different data

types are used. It should also be noted that this representation is not an absolute minimum, for example varints could be used to encode the length field, similar to Google's protocol buffers. The message in listing 3.1 has 32 tag pairs, using this information the minimal message which still conveys the same amount of information is equal to $\|minimal\_msg\| = \|data\| + 5 * 32 = 315$ bytes. The overhead of this message is thus $\|overhead\| = \|condensed\_msg\| - \|data\| = 1234 - 315 = 919$ bytes, which is 74% of the message. The examined message only represents a single case. It does however show that there is quite a lot of data overhead, even for a minimalistic message that does not include terms or warrants. In research conducted by Lawrence[18] it was found that data-centric XML documents contain 20–30% data, the rest is overhead and schema. Thus the amount of actual data in the message(26%) is inline with what is to be expected.

Listing 3.1: Example of a simple eMIX message. The data conveyed in the message is a tender to sell 100*W* of 240V/50Hz AC power. The power is being offered in a single interval with a duration of two hours, starting at 2014-08-28 09:00:00 Netherlands/Amsterdam time.

```
<emix:product>
<xcal:components>
<xcal:interval>
<xcal:properties>
<xcal:uid>
<xcal:text>0x00</xcal:text>
</xcal:uid>
<xcal:duration>
<xcal:duration>PT2H</xcal:duration>
</xcal:duration>
<xcal:dtstart>
<xcal:parameters>
<xcal:tzid>
<xcal:text>Netherlands/Amsterdam</xcal:text>
</xcal:tzid>
</xcal:parameters>
<xcal:date-time>2014-07-28T09:00:00</xcal:date-time>
</xcal:dtstart>
<xcal:x-wsCalendar-attach>
<power:fullRequirementsPower>
<power:productType>energy</power:productType>
<power:meterAsset>
<power:mrid>meter-identifier</power:mrid>
</power:meterAsset>
<power:powerReal>
<power:itemDescription>RealPower</power:itemDescription>
<power:itemUnits>W</power:itemUnits>
<scale:siScaleCode>none</scale:siScaleCode>
<power:powerAttributes>
<power:hertz>50</power:hertz>
```

```
<power:voltage>240</power:voltage>
<power:ac>true</power:ac>
</power:powerAttributes>
</power:powerReal>
<power:maximumPower>100</power:maximumPower>
</power:fullRequirementsPower>
</xcal:x-wsCalendar-attach>
</xcal:properties>
</xcal:interval>
</xcal:components>
<emix:uid>0x01</emix:uid>
<emix:transactiveState>tender</emix:transactiveState>
<emix:currency>EUR</emix:currency>
<emix:marketContext>http://docs.oasis-open.org/ns/emix/2011/06</
    emix:marketContext>
<emix:side>sell</emix:side>
</emix:product>
```

Listing 3.2: Alternative to XML data representation, note: the number of data types are assumed to not exceed 256.

```
struct {
    byte id;
    byte[4] length;
    byte[length] data;
}
```

Parsing of the eMIX message also induces a computational overhead. Using xmllint[2] to parse the message in listing 3.1 took 20ms for 100 iterations, thus parsing the message once took 0.2ms = 200µs. This timing was performed on a computer with a Intel(R) Core(TM)2 Duo CPU T7250 @ 2.00GHz.

3.1.4.4 *Extensibility*

EMIX supports extending of the standard in various ways. Some of the enumerations defined by eMIX can be extended. Enumerations that supported extensions do so through the `emix:EmixExtensionType` discussed in section 3.1.2.1, and thus except strings that begin with "x-". Enumerations that support extension all use the `xs:union` construct with the `emix:EmixExtensionType` as the second member type, an example of an extensible type can be found in listing 3.3.

Listing 3.3: EMIX extensible enumerated type.

```
<xs:simpleType name="MeasurementProtocolType">
<xs:union memberTypes="power:MeasurementProtocolEnumeratedType
    emix:EmixExtensionType"/>
</xs:simpleType>
```

2 A command line utility for XML parsing that is included in libxml, see http://man.cx/xmllint

EMIX also allows the extension of more complex information structures. These extensions require additions to the XSD schema. In order to accommodate easy extension, most of the eMIX types and elements that exchange information are abstract. These abstract types and elements can be extended. This type of extension can be found in the `emix:PriceBaseType`. Both the type `emix:PriceBaseType` and the element `emix:priceBase` are defined as abstract. As shown in section 3.1.2 figure 3.2 `emix:PriceBaseType` is extended by three different price types. One of these extension types is the `emix:PriceType` and its corresponding element `emix:price`. In listing 3.4 the definition of both types and elements is shown. In line 8 `emix:PriceType` extends the `emix:PriceBaseType`, and as such it inherits all its data fields. However just extending the type is not sufficient. The elements that are of the same type as the extended type need to explicitly define that they can be used in place of an other element. The `emix:price` element definition in line 5 shows the use of the `substitutionGroup` attribute which allows an `emix:price` to replace any instance of an `emix:priceBase` element.

Listing 3.4: EMIX extensible abstract type.

```
1  <xs:element name="priceBase" type="emix:PriceBaseType" abstract="
       true">
2  </xs:element>
3  <xs:complexType name="PriceBaseType" abstract="true">
4  </xs:complexType>
5  <xs:element name="price" type="emix:PriceType" substitutionGroup
       ="emix:priceBase"/>
6  <xs:complexType name="PriceType" mixed="false">
7  <xs:complexContent>
8  <xs:extension base="emix:PriceBaseType">
9  <xs:sequence>
10 <xs:element ref="emix:value" minOccurs="1" maxOccurs="1"/>
11 </xs:sequence>
12 </xs:extension>
13 </xs:complexContent>
14 </xs:complexType>
```

The eMIX types that are designed with this type of extension in mind are: `emix:EmixBaseType`, `emix:ProductDescriptionType`, `emix:ItemBaseType`, `emix:EmixInterfaceType`, `power:BasePowerQualityIndicatorType`, `emix:BaseTermType` and `emix:BaseWarrantType`.

3.2 OPEN AUTOMATED DEMAND RESPONSE COMMUNICATIONS SPECIFICATION

OpenADR[19] is a standard funded by the California Energy Commission and created by the Demand Response Research Center which is managed by the Lawrence Berkeley National Laboratory. Open-

ADR is an open standards-based data model for communications that facilitates information exchange and between utilities/independent system operators (isos) and customers based on events and signals. OpenADR only specifies a communications data model to facilitate dr and dynamic pricing. As such the actual implementation of how to act upon dr and how to deal with dynamic pricing is not specified.

OpenADR defines the interface and data model of a Demand Response Automation Server (DRAS) and a DRAS client. Customers run their own DRAS client which communicates with the DRAS of the utility/iso. The DRAS is used to facilitate automation of a customers response to dr programs and dynamic pricing. Supported operations include bidding on a specific dr event and opting out of dr events that adhere to conditions specified by the customer.

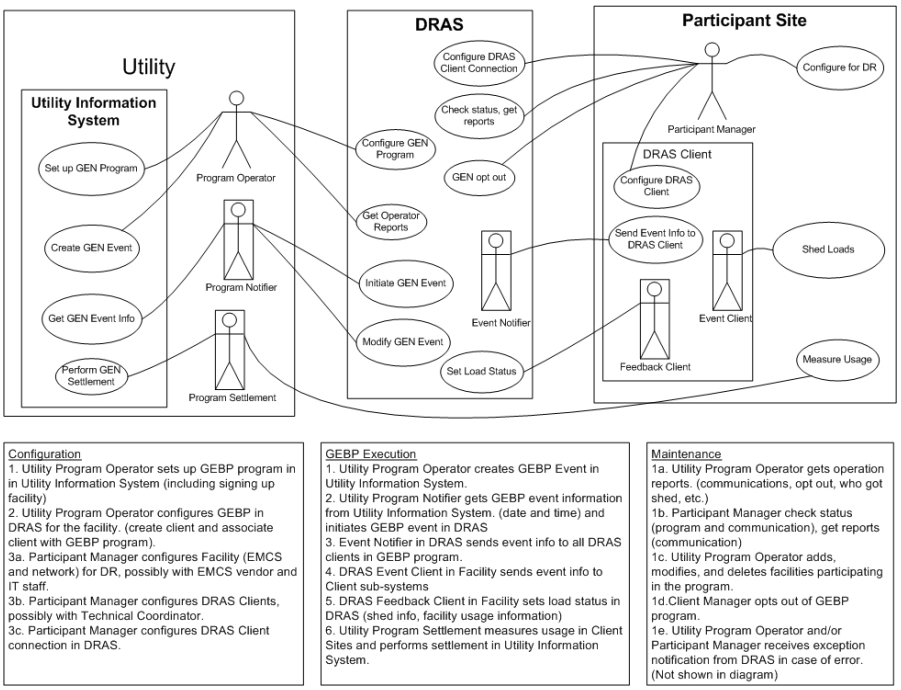


Figure 3.3: Use case diagram for a gebp. Roles can either be human operators or computer sub-systems. It should be noted that the `Event Notifier` role is always a sub-system of the DRAS.
Source: Open Automated Demand Response Communications Specification[19]

An example of how OpenADR can be used for a general event-based program (gebp) can be found in figure 3.3. Figure 3.3 shows that the responsibilities of each party involved do not necessarily have to belong to one role. Of course implementations can differ depending on the needs of the utility/iso, which is fine as long as the required interfaces specified by OpenADR are implemented.

The next sections will elaborate on the interfaces, data model and requirements specified by OpenADR.

3.2.1 *Interfaces*

OpenADR specifies three interface groups, one for each entity that communicates with the DRAS. The interface groups are as follows:

1. Utility operator and iso interfaces
2. Participant(customer) operator interfaces
3. DRAS client interfaces

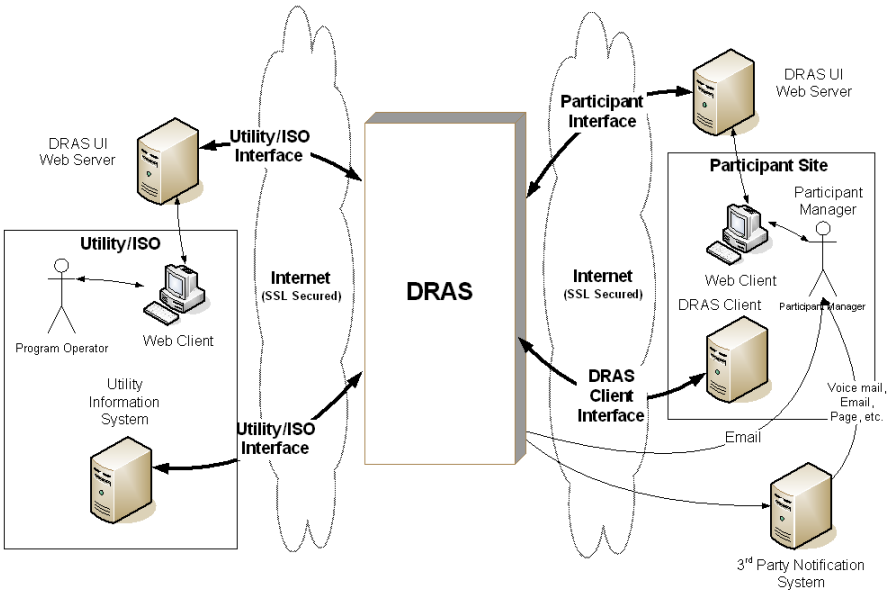


Figure 3.4: DRAS interfaces groups and their interaction with the systems belonging to the various entities.
Source: Open Automated Demand Response Communications Specification[19]

Figure 3.4 shows an overview of which entities and systems correspond to an interface group. It is not always needed to implement all interface groups e. g., if the DRAS is owned and fully integrated into the utility's/iso's infrastructure interface group (1) is not needed. Note that while interface group (1) and (2) might not be needed, interface group (3) is always required.

The interfaces specified by OpenADR are specified using Web Services Description Language (WSDL)[6, 7]. By making use of WSDL programs that need to communicate with the DRAS and DRAS client can read the WSDL interface description and determine which operations are available. WSDL also allows the specification of different bindings(protocol/data format), but in the case of OpenADR only the Simple Object Access Protocol (SOAP)[10, 11] binding is used. A graphical representation of the interfaces can be found in the appendix, see figures A.13 to A.15. For the full list of parameters, return types and authorised users [see 19, section 9].

3.2.2 *Data model*

OpenADR specifies its data model in an XSD format, therefore the model data that is in the XML format. Data is exchanged through the interfaces touched upon in section 3.2.1. Since the data is in the XML format is can be wrapped inside the SOAP message body which is used to make that actual function calls.

The data model contains several namespaces each corresponding to the major interface functions. A graphical representation of the types defined in the namespaces can be found in the appendix, see figures A.1 to A.12. For the full specification [see 19, section 8].

3.2.3 *Evaluation*

In order to evaluate the OpenADR standard software quality attributes are used, just like was done in the eMIX evaluation.

3.2.3.1 *Security*

The OpenADR standard specifies the following requirement regarding security: "All functions must restrict access based upon a well documented set of security roles.", [see 19, section 6.3]. Table 3.2 shows an overview of all the roles and their access to the defined interfaces. The definition of the interfaces [in 19, section 9] also includes on a per function basis which roles have access to them, including restrictions.

Table 3.2: Security rules of Interfaces, [as defined in 19, table 5].

	Utility Interface	**Participant Interface**	**DRAS Client Interface**	**DRAS Interface on DRAS Client**
DRAS	N/A	N/A	N/A	Full access
DRAS Operator	Full access	Full access	Full access	None
Participant Operator	None	Access to all methods, but limited scope in what can be done, viewed, etc. with each method.	None	None
Utility Program Operator	Full access	Full access	None	None

Table 3.2: Security rules of Interfaces, [as defined in 19, table 5] – continued from previous page.

	Utility Interface	**Participant Interface**	**DRAS Client Interface**	**DRAS Interface on DRAS Client**
DRAS Client	None	None	Full access	N/A
DRAS Client Installer	None	Limited to a limited number of methods used for testing.	None	None

On top of that all public communication interfaces have the following requirements; note terms are as defined in RFC4949[24]:

- Confidentiality
- Integrity
- Authentication
- Non-repudiation

The interfaces described in the standard do not provide any functions or implementation details regarding the above mentioned requirements. Instead OpenADR opts three methods to fulfil the requirements, namely:

1. A secure tunnel with server-side certificates, as well as user name and password authentication.
2. A secure tunnel with server-side and client-side certificates.
3. Web Services Security.[3]

The secure tunnel mentioned should use Transport Layer Security (TLS)[8] version 1.0 or higher with a Rivest, Shamir, & Adleman (RSA)[22] extension. Further requirements are posed on the algorithms and standards used for the certain parts of the tunnelling.

- Key exchange – 1024-bit RSA keys.
- Data encryption – Triple Data Encryption Standard (3DES)[34] or 128-bit Advanced Encryption Standard (AES)[36].
- Message Integrity Code (MIC)[4] – SHA-1.

Secure Hash Algorithm (SHA)[37]

3 Not explained in OpenADR.
4 MIC is deprecated according to the *Internet Security Glossary*[24] in this context hash or checksum is meant.

- Message Authentication Code (MAC) – Key-based Message Authentication Code (HMAC)[16] using SHA-1 for hashing.

Note that stronger algorithms and standards are encouraged as long as they are part of the TLS specification.

3.2.3.2 *Reliability*

OpenADR also specifies the following paraphrased reliability constraint: "All DRAS functions must be implemented using some type of reliable communications, meaning that it must be possible to determine if information exchanged as part of executing the function was received correctly.". This requirement is not part of the data model, since the data structures do not have any extra embedded information that allows for checking through e. g. a checksum; nor is it part of the interfaces specified. However checking if the data exchanged is correct can be accomplished using the tunnelling mechanism described in section 3.2.3.1. In order to verify if the data has arrived, simply using the Transmission Control Protocol (TCP)[20], which is the most used protocol for these type of communications should suffice.

3.2.3.3 *Efficiency*

As mentioned in sections 3.2.1 and 3.2.2 at its core the data model and interface specification[5] of OpenADR use XML. As shown in section 3.1.4.3 the XML format can be quite verbose, for more information see the aforementioned section.

3.2.3.4 *Extensibility*

OpenADR does not specify anything about extensibility in its standard. However this does not mean that there is no room for extension. Most of the data types can easily be extended by inheritance, similar to how eMIX handles inheritance of its data types. Adding new interface functions should also be relatively easy, since the interface description is done in the WSDL format. New functions can simply be added to the WSDL description files. These files will be queried from the DRAS and thus new functions can easily be detected, and used by applications that know how to utilise them.

3.3 ENERGY INTEROPERATION

Energy Interoperation (EI)[38] is a standard developed by OASIS. The standard describes an information- and communication model for energy interoperation, which supports:

5 WSDL is XML based.

- Transactive Energy
- Distribution of dynamic and contract prices
- Demand response approaches ranging from dispatch of load resources to price levels embedded in an event
- Measurement and confirmation of response
- Projected price, demand, and energy

The standard once again only defines the information- and communication model, the actual implementation of the aforementioned points is not described. It does however define three profiles which relate to energy interoperation standards. The profiles defined are: OpenADR, TeMIX and Price Distribution. These profiles specify which services of the EI standard should be implemented.

3.3.1 *Interactions*

EI provides service descriptions and data models for the interaction between two parties. Only pair-wise interactions are described, but chaining is possible.

3.3.1.1 *Transactive energy interactions*

Transactive energy interaction is the communications of energy transactions and options. In EI tenders are used for these type of interactions. Figure 3.5 shows what a typical energy transaction interaction

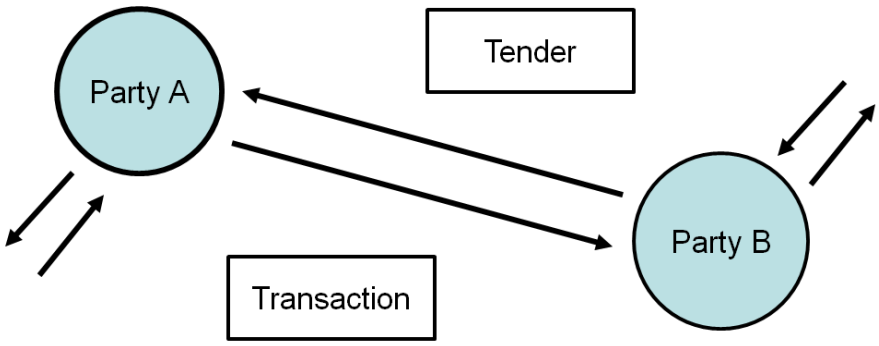


Figure 3.5: EI transactive energy interaction. Party B takes on the role of the buyer and initiates a tender to Party A. Thus Party A is on the sell side of the tender. If Party A accepts the tender then Party A is on the sell side and Party B is on the buy side of the following transaction.
Source: Energy Interoperation[38]

looks like. In EI interactions concerning transport prices follow the same interaction pattern as the energy interactions.

3.3.1.2 *Event interactions*

Event interactions are aimed at demand and generation resources. The interaction model allows generation and dr resources[6] . During an interaction a party is either a Virtual Top Node (VTN) or Virtual End Node (VEN). The role of the party that is the VTN/VEN depends on the context of the interaction. Interactions between a VTN and VEN are similar to the interactions between parties using a tender. However, a VEN is associated with one VTN, while a VTN is associated with 1...n VENs. The interaction between a VTN and VEN is

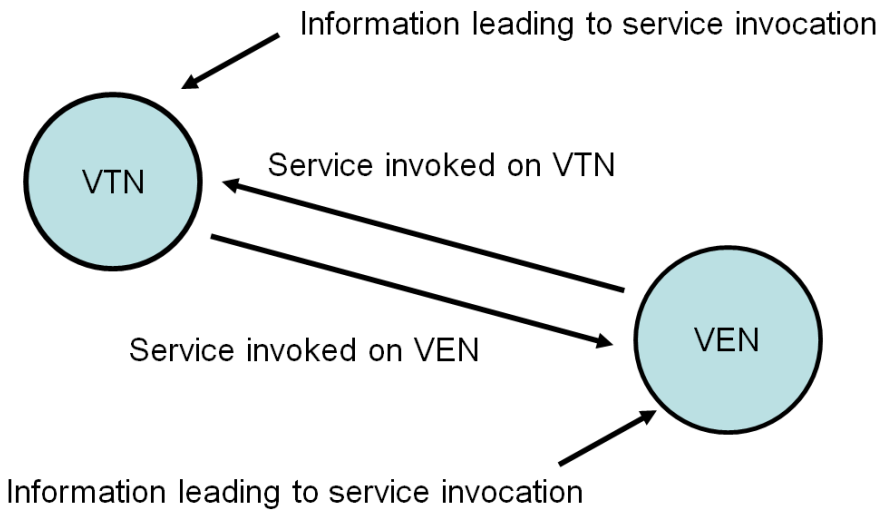


Figure 3.6: EI VTN and VEN interaction.
Source: Energy Interoperation[38]

shown in figure 3.6. As can be seen interaction works both ways allowing both push and pull based interactions. Parties can also act as both a VTN and VEN at the same time, but to different parties. Figure 3.7 shows a chain of possible VTN and VEN interactions. In

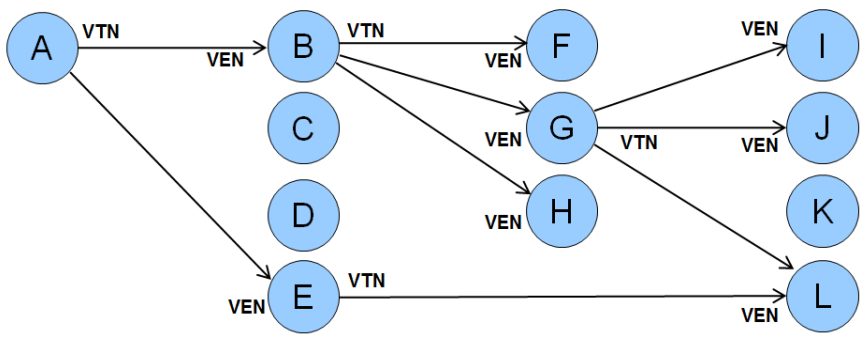


Figure 3.7: EI example of a VTN and VEN interaction chain.
Source: Energy Interoperation[38]

this example party A could be a system operator pushing an event

6 Dr resources are called curtailment resources in *Energy Interoperation*.

to a building(party B), which in turn could push an event to a machine(parties F–H).

3.3.2 *Interfaces*

EI defines multiple services, which each have their own interface. The services are divided into a couple of categories, as per EI:

- Transactive Services – For energy transactions, registration, and tenders.
  - `EiRegisterParty` – Allows parties to register(establish an identify), necessary for actors in order to interact with other parties.
  - `EiTender` – Offers prices which allows transactions to be made, they are actionable.
  - `EiQuote` – Indicates a possible tender price. Not actionable.
  - `EiTransaction` – Manages the agreement of transactions based on the market context.
  - `EiDelivery` – Provides measured information of a specified interval.
- Event Services – For implementing events and linked reports.
  - `EiEvent` – Used to create, send, cancel, modify and reply to events.
- Report Services – for exchanging remote sensing and feedback
  - `EiReport` – Allows requesting of reports independently of any event. Reports can be sent one time or periodically. `EiReport` also facilitates creation of a `EiHistorian` that will start recording the requested information.
- Enrolment Services – for identifying and qualifying service providers, resources, and more
  - `EiEnroll` – Establishes a relationship between two parties to allow further interaction.
- Support Services – for additional capabilities
  - `EiAvail` – Used to set constraints on when an event may or may not be accepted.
  - `EiOpt` – Used to create and send opt-in and opt-out schedules from the VEN to the VTN.
  - `EiMarketContext` – Used to communicate market information that rarely changes.

For the a more detailed explanation [see 38, sections 7–12]. The interfaces are specified using WSDL, the full description can be found at `http://docs.oasis-open.org/energyinterop/ei/v1.0/os/wsdl/`[7].

3.3.3 *Data model*

EI makes use of the eMIX standard for part of its data model, as such the data model of EI is specified using XSD, just like eMIX. Because of the usage of eMIX, the namespaces listed in table 3.1 also apply to this standard. EI does use some other namespaces that eMIX does not, however since most types are not discussed only knowing that the `ei` namespace belongs to the EI standard is relevant. Some eMIX types are extended e. g. `emix:MarketContextType` is extended to `ei:EiMarketContextType`, but eMIX is still used to communicate product definitions, quantities, and prices. Since eMIX is used as a basis for the data model, the schedules and time intervals are communicated using WS-Calendar; similar to eMIX but with an extension for streams. The data types needed by the interfaces previously mentioned in section 3.3.2 are specified using XSD similar to eMIX. More detailed explanations about the types in the data model can be found [in 38, sections 4–12]. The full XSD description of the data model can be found at `http://docs.oasis-open.org/energyinterop/ei/v1.0/os/xsd/`[8].

3.3.4 *Evaluation*

As with the previous standards, several software quality attributes of EI are examined in this section.

3.3.4.1 *Security & Reliability*

As previously mentioned in section 3.3 EI only defines an information- and communication model. As such the standard does not provide any mechanics for encrypted data exchange, or user/device identity validation or reliability. To quote the EI[38, lines 1939–1944] standard, "The approach for enterprise software has evolved to defining key services and information to be exchanged, without definitively specifying how to communicate with services and how to exchange information–there are many requirements for distributed applications in many environments that cannot be taken into account in a service and information standard. To make such choices is the realm of other standards for specific areas of practice, and even there due care must be taken to avoid creating a monoculture of security.".

7 Visited on 24/09/2014.
8 Visited on 24/09/2014.

3.3.4.2 *Efficiency*

EI makes use of XSD to specify its data model, more specifically it uses eMIX as part of its data model. Thus the same XML overhead as previously discussed in section 3.1.4.3 applies.

3.3.4.3 *Extensibility*

The part of the data model used by EI that is eMIX retains its extensibility that was described in section 3.1.4.4. The rest of the data model and interfaces are described in XSD and WSDL similar to OpenADR and as such the same extensibility as described in section 3.2.3.4 applies.

3.3.4.4 *Adaptability*

EI is designed to be non monolithic. The standard does not consist of one or two services but of eleven services, as described in section 3.3.2. This allows for adaptation of EI to the needs of a system. A good example of this are the profiles defined by EI, they specify which services should be implemented to achieve similar functionality to OpenADR and TeMIX. The non monolithic design also facilitates the replacement of services with functionally equivalent services. Allowing for easier integration with existing and new technology.

3.4 IMPLEMENTATION CHOICE

One of the standards explained in sections 3.1 to 3.3 will be implemented in the current mas.

3.4.1 *Requirements*

Chapter 2 explains how the current implementation of the mas functions. The next sections will determine the requirements that a standard needs to fulfil based upon the agents in the current mas implementation.

In the current implementation a Genco agent represents a non-renewable energy source which generates a certain amount of energy based on the demand. Equation (2.1) shows what kind of information and parties are involved with the pricing of the energy delivered by a Genco. The type of interactions that are needed require the standard to have support for TSO charges, as well as support for location, and energy pricing data exchange. Like the Genco agent, the prosumer agent also has the ability to produce energy. As such it has the same requirements as a Genco. However, a prosumer and consumer agent can both negotiate on the energy prices. Therefore, the prosumer and consumer agent require a way to exchange price negotiation data.

There is also a need for exchanging of time and weather forecast data for the interactions with the time and weather agent respectively. The balancer agent does not add any further requirements, since all the messages it sends/receives are already covered.

The messages to the balancer agent about who successfully negotiated a contract with an other party are also required. While these messages are used in the mas for data aggregation and then visualisation, which could be done differently; the information (in a real life scenario) is also required by the TSO in order to know how much and where the power is needed. With this information they can then solve imbalances and prevent possible overloading of the grid.

So far a standard which can be used to implement the current mas must support the following data exchanges:

- TSO charges
- Energy pricing
- Price negotiation
- Time
- Location
- Weather
- Successful contract notification to TSO

On top of this, agents also need to know with which other agent they are communicating. So there also needs to be a way to distinguish agents from each other. Table 3.3 gives an overview of which requirements are fulfilled by the previously mentioned standards in sections 3.1 to 3.3. The table shows that the standard which fulfils the most requirements is EI.

Table 3.3: Requirements fulfilled by standards.

Requirement	**Standards**		
	eMIX	**OpenADR**	**EI**
TSO charges	Yes[9]	No	Yes
Energy pricing	Yes	Yes	Yes
Price negotiation	No	Yes	No[10]
Time[11]	No	No	No

9 Defined in power extensions.

10 Tenders can be accepted or rejected by a party, but there is no such thing as a real bidding process.

11 All standards support time and schedule information for events or energy products. However there is no support for parties to acquire a time from an other party similar to how interaction with the time agent works.

Table 3.3: Requirements fulfilled by standards – continued from previous page.

Requirement	Standards		
	eMIX	**OpenADR**	**EI**
Location[12]	Partial	Partial	Partial
Weather	No	No	No
Identification	No	Yes[13]	Yes
Successful contract notification to TSO	No	No	No

3.4.2 *Software quality attributes*

Sections 3.1.4, 3.2.3 and 3.3.4 evaluate some of the software quality attributes of the standards. This section assigns scores to each of the quality attributes discussed in the previous sections. Scores can have a value of $[-2, 2] \in \mathbb{Z}$, where a negative number denotes a negative impact and a positive number denotes a positive impact. Table 3.4 shows an overview of all the scores. The following paragraph will explain the reasoning behind the assigned scores.

Both eMIX and EI do not specify anything about security and reliability; neither do they negatively impact it, as such their security and reliability score is 0. OpenADR on the other hand does specify security features. Interface interactions are restricted to certain roles and requirements are placed on communication interfaces. The requirements placed on the communication interfaces apply to both security and reliability, see section 3.2.3.1. Due to the aforementioned restrictions and requirements OpenADR scores a 2 in both security and reliability. All standards have a negative impact on efficiency. Since the overhead they induce by using XML is pretty significant, they all score a -2 for efficiency. All of the standards are extensible by inheriting from types in their XSD definitions. The OpenADR and EI interfaces are even extensible by adding functions to their WSDL definitions. However eMIX and consequently EI are designed with extensibility in mind, see section 3.1.4.4. Therefore both eMIX and EI score a 2 while OpenADR scores a 1.

12 EMIX and EI only support locations associated with power product descriptions. OpenADR only supports location information for the DRAS clients.

13 Participants are added to an account list on the utility/iso side and can thus be identified. The utility itself is identified by its uri where the WSDL interface description can be found.

Table 3.4: Software quality attribute scores of the standards.

Quality attribute	Standards		
	eMIX	OpenADR	EI
Security	0	2	0
Reliability	0	2	0
Efficiency	−2	−2	−2
Extensibility	2	1	2

3.4.3 *Other features*

Each of the standards described in sections 3.1 to 3.3 also has other features not mentioned in the previous sections. The features listed in Table 3.5 are not strictly necessary for the implementation of the mas. They do however provide a good basis for enhancement and extension of the current mas. Some of the features such as scheduling will be implemented regardless, since for all standards they are required when exchanging energy load shedding/load shifting information.

Table 3.5: Other standard features.

Feature	Standards		
	eMIX	OpenADR	EI
Schedules	Yes	Yes	Yes
Intervals	Yes	Yes	Yes
Warrants	Yes	No	Yes
Logs	No	Yes	No
Dr events	No	Yes	Yes
Opt-in/Opt-out	No	Yes	Yes
Market context	Yes	No	Yes
Constraints	Yes	Yes	Yes

3.4.4 *Conclusion*

In this section a standard that is going to be implemented will be chosen. The previous sections show that eMIX satisfies the least requirements, is tied for last on quality attributes, and has the least additional features. Furthermore eMIX only specifies a data model and no interfaces or services. This means that if eMIX is used for the new mas implementation, the interface communication still has to be thought out. The previous points are enough reason not to use the eMIX standard.

Leaving eMIX behind, the choice is between OpenADR and EI. OpenADR scores higher on the quality attributes. The quality attrib-

utes OpenADR scores better on are security and reliability. However, the security and reliability of OpenADR mostly comes from outside of the standard specification. The standard specifies requirements on the public communication interfaces, however the actual security and reliability comes from a transport layer underneath the actual OpenADR implementation. As such the same method can be used when implementing EI. An other thing to note is that; the mas implementation handles security by using a JADE add-on, as well as services [see 5, section 5]. A similar approach could also be taken when using EI. This leaves only the interface security roles in favour of OpenADR. These security roles are however intended for the structure of OpenADR.

EI has better extensibility, does support TSO charges, price negotiation is possible with the current messages, and supports more additional features[14] than OpenADR. Given the above and the fact that EI can use the same security and reliability approach as detailed in OpenADR; EI is chosen as the standard used for implementation.

14 See table 3.5.

4 IMPLEMENTATION

In this chapter the implementation of the new mas is explained. There are some modifications to the original mas process flow in order to have a better compliance with the EI standard. The differences between the original and the new implementation are highlighted. Also not every feature defined by the standard is implemented, some things are left out and some data structures are modified slightly. These differences are also explained in this chapter.

4.1 OVERVIEW

Before going into the details an overview of the mas implementation is given. The first stage of the simulation is the registration stage. During this stage all the consumer, prosumer and Genco agents register with the balancer agent, the process is visualised in figure 4.1. Note that the registration of agents is done in parallel, thus the order of which agent registers first is not set and depends upon scheduling. Prosumer agents also send a query to the weather agent for their local weather during this stage, but only after their registration succeeds. After all the agent have registered; or the registration reaches its

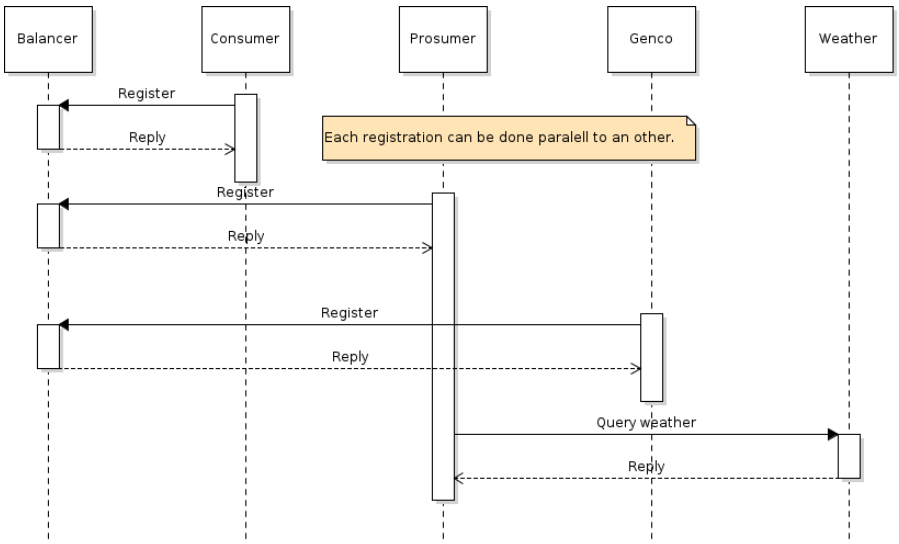


Figure 4.1: Registration sequence diagram.

timeout value, the next stage begins. The balancer agent sends a message to all the registered agents notifying them that the registration period is over. In response the registered agents send a query to the balancer requesting a list of all the registered agents. The interaction

between the agents is shown in figure 4.2. Once all the agents have

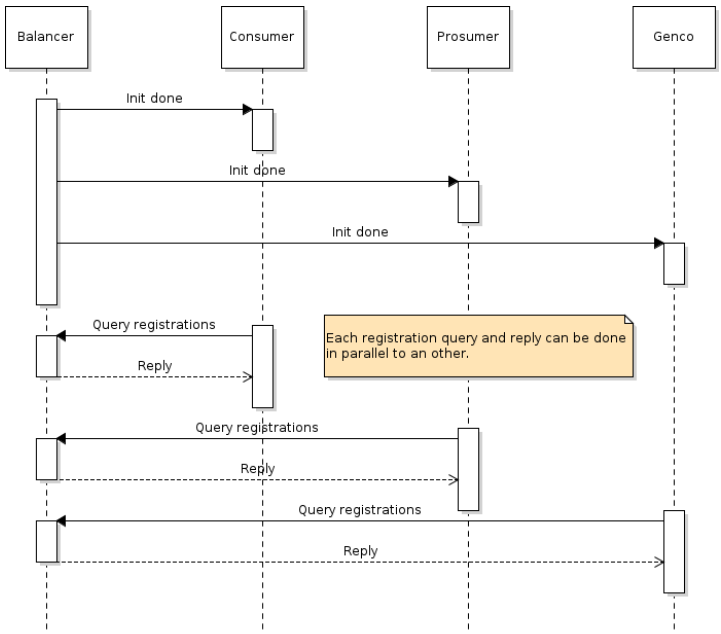


Figure 4.2: Registration query sequence diagram.

received a registration query reply; or a timeout value is reached, the balancer sends out a request to consumers and prosumers for their consumption/production, as shown in figure 4.3. Using the total consumption and production values, the balancer calculates the amount of energy each Genco needs to produce according to equation (2.2). This value is then sent to every Genco as shown in figure 4.4. Once

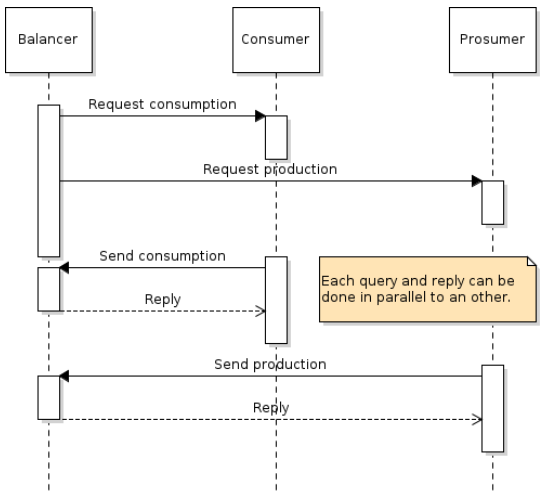


Figure 4.3: Consumption/Production query sequence diagram.

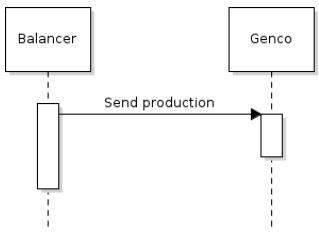


Figure 4.4: Genco production sequence diagram.

the balancer has sent the production to the Genco agents, it sends a message telling all the agents that the negotiation of energy can be-

gin. Upon receiving this information the prosumer and Genco agents send quotes with their prices and/or production to the consumer agents, as shown in figure 4.5. Next the consumer agents try to pur-

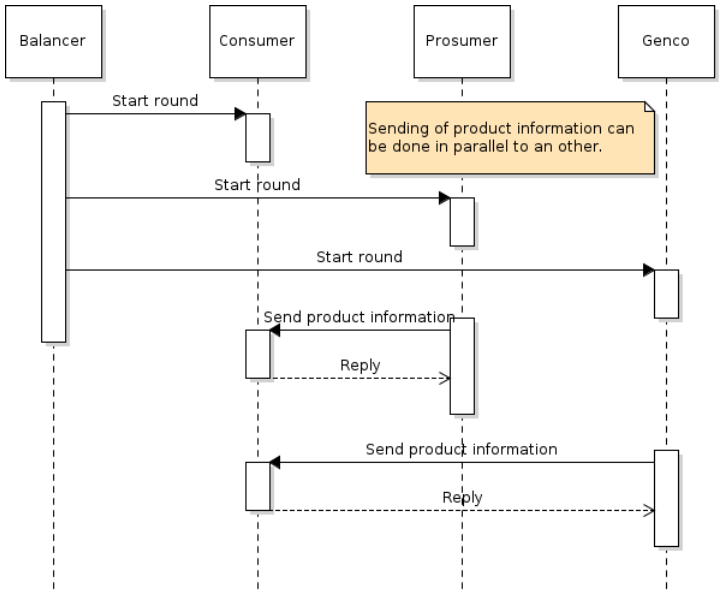


Figure 4.5: Negotiation start sequence diagram.

chase energy from a prosumer/Genco that meets their demands, at the lowest available price. If their offer is accepted a transaction occurs, if it does not the consumers tries again with a different price, or even a different agent. Finally, once a consumer reaches an agreement with an other agent; it sends the details of the contract to the balancer agent. The sequence of events for these last to interactions are shown in figures 4.6 and 4.7.

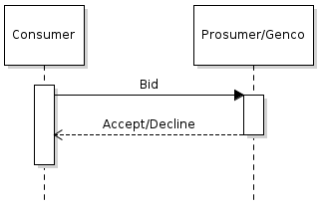


Figure 4.6: Bid sequence diagram.

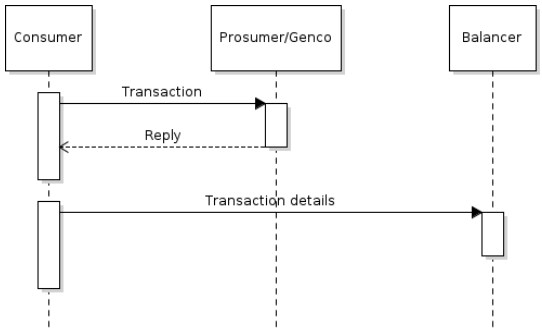


Figure 4.7: Transaction sequence diagram.

4.2 DIFFERENCES

As mentioned earlier, the new mas implementation is not exactly identical to the original implementation. Some changes have been

made to the messaging in the simulation, in order to comply with the standard as much as possible. In some cases message ordering has changed, in others, messages were added or the contents altered. There are also some changes to how the JADE agent behaviours are executed, these mainly relate to thread blocking and sleeping. However the non blocking nature of some of the original behaviours does results in redundant messages, leading to a difference in how many messages are sent. More on this will be explained in section 5.1.1

Numbers in the implementation have also changed. The original implementation uses random values for energy and pricing; initiated randomly in the range $[0, 5]$ for consumption and values $0 \leqslant x \leqslant 1$ for prices. Both of these do not seem to have any units assigned. The EI standard uses watt hour and watt as units, the standard also supports ISO 4217[33] currency codes, but they are not enforced. To keep the amount of extra information in messages to a minimum, the currency is just assumed to be euro. In order to comply with the watt hour and watt values, the calculations for consumption and production have been changed to work with them. Pricing is also adjusted slightly as a result. Although these changes affect the balance equation and the calculation of who the cheapest prosumer is, they do not affect the amount of messages sent any more than the random nature of these values does. The logic of these function is still the same, they just have different values.

4.2.1 *Messages*

Since not all messages used in the mas are part of the EI standard, there are two types of messages. Normal messages that adhere to the EI standard, and control messages; which are used to control parts of the simulation which are not defined in the standard, or have no real world counterpart. Messages such as the weather data query, and indication of registration period completion are examples of control messages.

A big difference in messaging are the registration messages. These are not present in the original mas, instead JADE functionality is used to query for agents. However the EI standard requires registration with a registrar before other interactions can happen. As such the registration messages, registration period done and registration queries are added messages that are newly introduced. Of these messages the registration period done message is a control message. It is merely used as a tool to let the agents know they can move along to the next step of the simulation and as such has no counterpart in the EI standard. An other minor difference concerning registration messages is the exchange of positional information. The original mas sends this along with the consumption/production analogous to the `send production information` messages in the sequence diagram depicted in

figure 4.5. The EI standard has no innate support for location of the agents, as such it was decided it was best suited as an extension to the registration messages. This makes sure the data will only be sent once, plus the standard explicitly demands extending the registration info to suit the implementation's needs.

An other difference in the messaging can be found in the consumer interaction with the prosumer and Genco agents. In the original mas the consumer agents contact the prosumers with a query for their starting price and energy production. After which they contact the Gencos with a query for their position, in order to determine the closest and thus cheapest Genco. In the new implementation the messaging is reversed, i. e. the prosumers send a quote stating their production and price to the consumers, and the Gencos send their TSO charge to the consumers. The reasoning behind this is that the goal is to use the messages of the standard for as much of the simulation interaction as possible. However EI does not support asking an other party for a quote, unless it is an already existing quote. Thus to still be able to use quote messages, the message direction was inverted. For the purpose of calculating overhead this has no impact as the amount of messages does not change.

4.3 STANDARD ADAPTATION

This EI standard is not implemented fully. Only the messages and types used by the mas are implemented. Some types and elements were also slightly modified in order to make them usable. There have also been some type extensions in order to facilitate the communication of necessary data. The next sections will explain the aforementioned changes and their reasoning.

4.3.1 *Deviations*

The following types have some non required fields that are not implemented, because they do not convey information used by the mas simulation.

- `ei:eiResponseType` does not contain `ei:responseTermsViolated`.
- `emix:ProductType` does not contain `emix:ArrayOfTerms`.
- `emix:EmixBaseType` does not contain `emix:EnvelopeContentsType`.
- `power:ProductTypeType` only implements the enumerations in `power:ProductTypeEnumeratedType`.
- `power:PowerProductDescriptionType` does not contain the **mandatory** `emix:EmixInterfaceType`.

- `ei:EIRequestPartyRegistrationType` does not contain the **mandatory** `ei:registreePartyID`[1].

Apart from removal of some fields from certain types, there were also some other changes. The `emix:product` elements can substitute `emix:emixBase` elements, otherwise there is no way for quotes and tenders to contain any products. The missing substitution rule is likely an oversight in the eMIX standard documentation, since the type of `emix:product` elements(`emix:ProductType`) does inherit from the type(`emix:EmixBaseType`) which is the type of the `emix:emixBase` element. The last deviation is that `xcal:wsCalendarAttachType` only accepts `xcal:artifact` types, since these are the only `xcal` types used in the implementation.

Only the EI transactive energy interaction services are implemented. The VEN/VTN interactions are not used. The original mas implementation does not use the event specified by the standard, nor does it use reports for the that allow retrieval of logged information. The enrolment services are not implemented either, since there is no need for an agreement of which market contexts are used.

4.3.2 *Extensions*

The extensions defined for use with the mas all reside in the `urn:rug:mas` namespace denoted as `mas`. The new element extensions are listed in table 4.1 and the types and their descriptions are listed in table 4.2. All of the extensions only pertain to the registration process. The registration types defined in EI made abstract and the standard specifically states they should be extended to fit the registration process in the current market context, rather than using the abstract types directly.

Table 4.1: Mas element extensions.

Element	Type
`mas:x`	`xs:decimal`
`mas:y`	`xs:decimal`
`mas:location`	`mas:location_type`
`mas:agentid`	`mas:agent_type`
`mas:masRegistrationInfo`[2]	`mas:mas_registration_info_type`

1 The purpose of sending this field along with a request for the registered parties is unclear from the documentation. If it is assumed to query the registration information from the specified party, then the party sending the query should somehow already know the globally unique identifier (guid) of a registered agent. If it is assumed that this field should contain the guid of the party doing the query, then it is totally redundant since the registrar already know the connection mapping, in the case of the mas. Either way the field is just removed, as it serves no purpose in the current implementation.

Table 4.1: Mas element extensions – continued from previous page.

Element	Type
`mas:address`	`mas:address_type`
`mas:masRegistration`[3]	`mas:mas_registration_type`

Table 4.2: Mas type extensions.

Type	Description
`mas:location_type`	Contains the `x` and `y` elements and indicates the position of an agent.
`mas:agent_type`	Maps from an integer to an agent type.
`mas:-mas_registration_info_type`	Inherits from `ei:EiRegistrationInfoType` and adds the `location` and `agentid` elements.
`mas:address_type`	A simple string value indicating the communication address of the agent.
`mas:mas_registration_type`	Inherits from `ei:EiRegistrationType` and adds the `masRegistrationInfo` and `address` elements.

2 Allowed to substitute `ei:eiRegistrationInfo`.
3 Allowed to substitute `ei:eiRegistration`.

5 RESULTS

Using the original and new mas implementations, the amount of messages, data sent and the differences between the two implementations can be closely analysed. The following sections show the difference between the two implementations. The new messages are also looked at more closely and their overhead is determined.

5.1 AMOUNT OF MESSAGES

In order to determine the overhead of the communication the amount of messages exchanged is needed. The minimum number of messages needed can be calculated based on the number of agents that are active in the simulation. Since some messages are not present in the standard and are mainly used to control the simulation as discussed in section 4.2.1, the types of messages are calculated separately. Note, all the calculations below assume that no errors occur.

At the start all consumer, prosumer and Genco agents need to register with the balancer agent. In turn the balancer agent needs to reply to each registration party indicating if it is successful or not. Thus the number of messages required for registration can be calculated as shown in equation (5.1).

$$N_{registration\_messages} = 2\,(N_c + N_p + N_g) \tag{5.1}$$

After registration the prosumer agents query the weather agent in order to retrieve the weather information for their location. The messaging for exchanging weather information is done through control messages. The calculation of the number of weather control messages $N_{C_{weather}}$ is then given by equation (5.2).

$$N_{C_{weather}} = 2N_p \tag{5.2}$$

Once all the consumer, prosumer and Genco agents are registered the balancer agent sends a control message to all registered agents that the initialisation period is finished. When the registered agents receive this message they need to query the balancer in order to know which agents, other than themselves have registered. The number of these messages can then be calculated by equations (5.3) and (5.4).

$$N_{C_{init\_done}} = N_c + N_p + N_g \tag{5.3}$$

$$N_{registration\_queries} = 2\,(N_c + N_p + N_g) \tag{5.4}$$

After replying to the registration queries the balancer sends a control message requesting the consumption/production of each consumer

and prosumer agent. The consumer and prosumer agents reply using a quote message from the standard, which in turn also requires a reply from the balancer. The calculation for these messages is shown in equations (5.5) and (5.6).

$$N_{C_{request\_power}} = N_c + N_p \tag{5.5}$$

$$N_{request\_power\_reply} = 2\left(N_c + N_p\right) \tag{5.6}$$

After receiving the consumption/production the balancer sends the amount of energy each Genco needs to produce in a control message. Right after sending the Genco production, the balancer sends a control message to all consumer, prosumer and Genco agents stating the bidding can start.

$$N_{C_{genco\_production}} = N_g \tag{5.7}$$

$$N_{C_{round\_start}} = N_c + N_p + N_g \tag{5.8}$$

At the start of a bidding round each prosumer and Genco agent sends a quote to all consumer agents. The Gencos send its price per Wh, while the prosumers send their maximum production as well as price. This leads to the amount of N_{start_quotes} quote messages sent, as calculated in equation (5.9).

$$N_{start\_quotes} = 2N_c\left(N_p + N_g\right) \tag{5.9}$$

After a consumer receives all the quotes from the registered prosumer and Genco agents, it tries to buy the energy from a prosumer or Genco. Assuming the consumer sends a tender to an other agent and it gets accepted at the first try, then the messages that get sent are one tender and one transaction including their replies. Giving the $N_{transaction\_messages}$ calculated in equation (5.10).

$$N_{transaction\_messages} = 4N_c \tag{5.10}$$

Finally, once a consumer agent negotiates a successful contract it sends the information to the balancer in a control message. Giving the $N_{C_{contracts}}$ calculated in equation (5.11).

$$N_{C_{contracts}} = N_c \tag{5.11}$$

Given equations (5.1) to (5.11) the minimum amount of messages required in the simulation can be calculated as follows.

$$N = 10N_c + 6N_p + 4N_g + 2N_c\left(N_p + N_g\right) \tag{5.12}$$

$$N_C = 4N_c + 5N_p + 3N_g \tag{5.13}$$

Using equations (5.12) and (5.13) for a simulation with one consumer, prosumer and Genco the amount of normal messages is 24 and the amount of control messages is 12. Figure 5.1 shows an overview of

messages exchanged during a simulation using one of each agent. As can be seen, the total amount of messages exchanged is greater than $24 + 12 = 36$. This is due to the fact that the prosumer did not accept the first offer made by the consumer, but accepted the second offer instead. This introduces two extra messages, namely one extra tender and one reply to the said tender. When using a more realistic preset of 30 consumers, 7 prosumers and 3 Gencos; such as the one used in the original mas[5], the minimum amount of messages is $917 + 164 = 1081$. When actually running the simulation the total amount of messages exchanged is 1386[1].

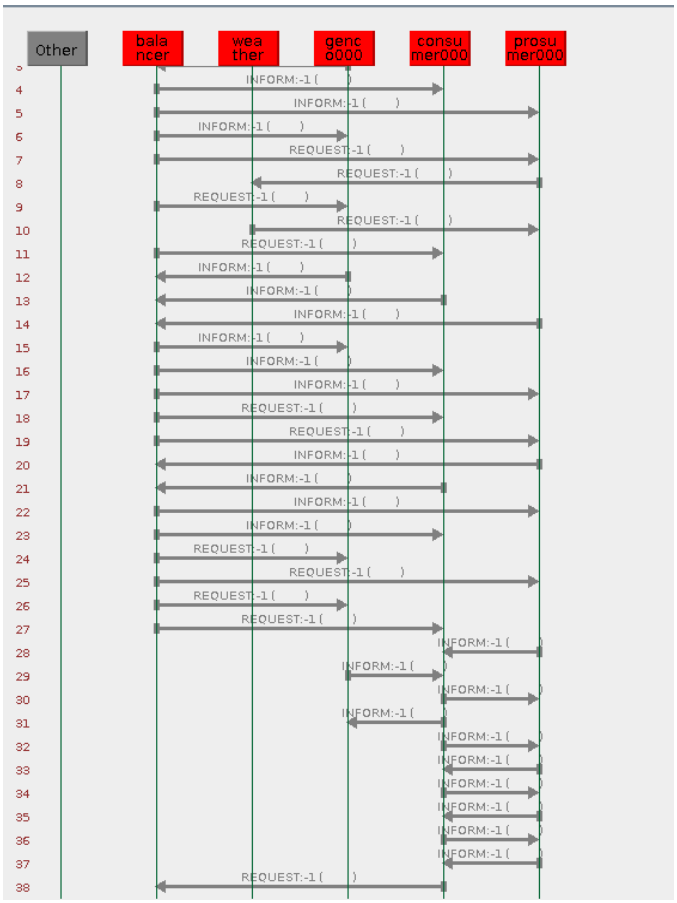


Figure 5.1: Overview of the messages sent between agents. Using one consumer, prosumer and Genco agent.

5.1.1 *Measured amount of messages.*

The amount of actual messages sent during a simulation can be measured. This was done for both the original and new mas implement-

1 This number will vary each run, due to random initialisation of production and consumption values, how the prosumers respond to bids and the nature of multithreading.

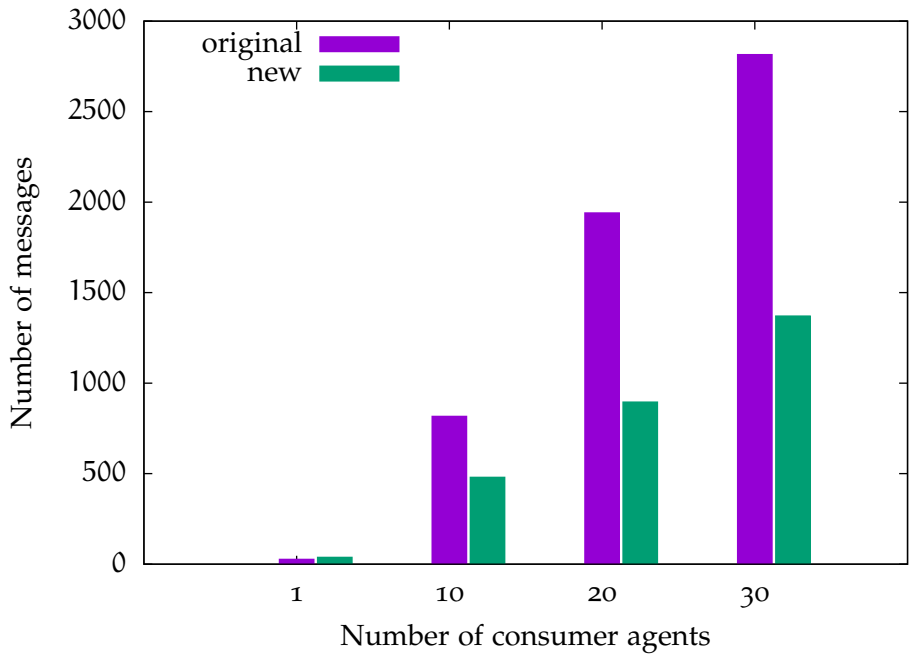


Figure 5.2: The actual amount of measured messages of the new and old mas implementation, averaged over 10 simulations. The number of consumers varies, while the number of prosumers is fixed at 7 and the amount of Gencos is fixed at 3.

ations. For the measurements[2] a setup of $\{1, 10, 20, 30\}$ consumers, 7 prosumers and 3 Gencos is used. The results were accumulated over ten simulations for each setup and then averaged. Figure 5.2 shows a graph of the number of messages measured. One would expect the new mas implementation to exchange roughly the same amount of messages as the old implementation, however this is not the case. The original implementation has some inefficiencies, instead of blocking and waiting for certain events to happen, the behaviour just keeps looping through the same state. Since some states involve the sending of messages without waiting, this means that the messages in these states will be sent multiple times. In this case it leads to the fact that the original implementation sends a lot more messages than the new one.

2 The original mas implementation sends a lot of queries to the Directory Facilitator agent, in order to query for information about the other agents present in the platform. This happens a lot and the results are not cached, as a result these type of messages eclipse the amount of actual simulation related messages that are sent. Therefore, these messages are not counted towards the total messages sent, in order to get at least some reasonable comparison of the number of messages sent. For the new implementation control messages are included, since these control messages are considered normal messages in case of the original implementation.

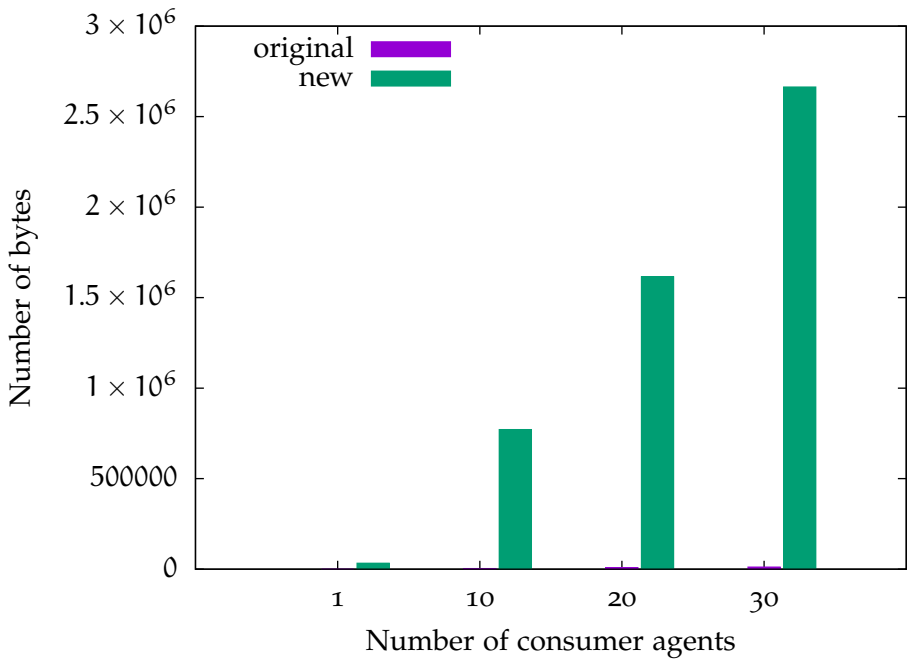


Figure 5.3: The amount of measured bytes sent by the new and old mas implementation, averaged over 10 simulations. The number of consumers varies, while the number of prosumers is fixed at 7 and the amount of Gencos is fixed at 3.

5.2 AMOUNT OF DATA

In this section the amount of bytes of data that is sent during a simulation is analysed. The amount of data sent by both the original and new mas simulations can be found in figure 5.3. The data is generated using the same setup as the one in section 5.1.1, but this time the number of bytes is measured, instead of the amount of messages. As touched upon briefly in section 2.1.2, the JADE platform sends messages compliant to the FIPA ACL specification. For the purpose of measuring the data that gets sent FIPA overhead is ignored and cut, as the implementation of the the mas does not use any of the FIPA headers for sending data. From the graph it can clearly be seen that the new implementation requires the sending of a lot more data than the original, about an order of magnitude 7 more for the simulations with $10 - 30$ consumer agents. Even though the total number of messages sent by the new implementation is less than that of the original. The reason for the massive overhead was briefly touched upon in section 3.3.4.2, the next sections analyse the overhead of the messages of the EI standard in more detail.

5.2.1 *Overhead per message*

In this section the overhead of each message from the EI standard that is used is analysed. Messages from actual simulation runs will be used as a basis.

Overhead includes XML tags and instances where data can be represented more compactly, e.g. using the string "100" for a number that will never be greater than 256. Such a number can be represented by one byte instead of a string made up of three bytes. The overhead calculation does not take into account the possibility to encode integers using variable lengths. Note that the overhead calculation does not care for extensibility or other features, it purely compares relevant sent data vs actual sent data. Also note that the calculation of the overhead is performed on the messages listed, the actual overhead may be off by a few bytes, because floating point values represented as a string can vary in size. E.g. the field can have a value of 0.0 in one message and 0.001 in an other. The same can be said for some other data time such as booleans, e.g. the string "true" is one less byte than the string "false".

Since EI makes use of XML, every message starts with a header indicating it is XML formatted and which character encoding is used. The header in the mas implementation is as follows: `<?xml version="1.0" encoding="UTF-8"?>`. This is a pure overhead of 39 bytes. Furthermore, EI makes use of the SOAP standard. As such each message is wrapped in SOAP XML tags, the combined opening and closing tags are shown in listing 5.1.

Listing 5.1: SOAP tags.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<soap:Body>
</soap:Body>
</soap:Envelope>
```

This SOAP tags amount to a total of 108 bytes excluding the newlines[3] In the context of the mas this can be seen as purely overhead, since it does not need an object access protocol to facilitate web services. Given that the above is true for every message in the standard, the following sections display the message without the aforementioned tags instead the overhead of $108 + 39 = 147$ bytes should be accounted for afterwards.

5.2.1.1 *Registration messages*

All agents send the same registration message to the balancer. An example of such a message can be found in listing 5.2.

3 Newlines are not counted, since they are only shown here to allow for better formatting. The newlines are not present in the messages exchanged within the mas.

Listing 5.2: Mas registration message.

```
<pyld:eiCreatePartyRegistration xmlns:pyld="http://docs.oasis-open.org/ns/energyinterop/201110/payloads">
<pyld:requestID>f43788d5-30e0-4e7f-81d9-7a56ffd7884c</pyld:requestID>
<ei:registreePartyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</ei:registreePartyID>
<mas:masRegistrationInfo xmlns:mas="urn:rug:mas">
<mas:agentid>3</mas:agentid>
<mas:location>
<mas:x>55</mas:x>
<mas:y>88</mas:y>
</mas:location>
</mas:masRegistrationInfo>
</pyld:eiCreatePartyRegistration>
```

Table 5.1: Data of the registration message.

Bytes	Name
1	id
16	request_id
16	registree_party_id
1	agent_id
1	x
1	y
36	**total**

Looking at listing 5.2 the data fields in table 5.1 can be identified. The `id` field is one byte, since there are far fewer message types than 256. There are only 5 agents, of which 3 register which also easily fits in one byte. The `x` and `y` coordinates are constraint to the interval $[0, 100]$ and thus also fit into one byte. Both the `request_id` and `registree_party_id` are hyphened strings containing only hexadecimal numbers, thus all hyphens can be stripped and every two characters make up one byte. This gives both the length of $\frac{36-4}{2} = 16$ bytes. The actual registration message as shown in listing 5.2 consists of 515 bytes, giving this message an overhead of $515 - 36 = 479$ bytes.

Each registration request is met with a reply to indicate if the registration was successful. A typical registration reply is depicted in listing 5.3. Identifiable data fields for the registration reply message can be found in table 5.2. The `id`, `ref_id` and `registrar_party_id` field sizes are based on the same reasoning as mentioned earlier. The size of the `response_code` is based on the fact that response codes are in the range $[100, 600)$, as such they do not fit in a single byte and

thus require two. Since the message in listing 5.3 consists of 470 bytes the total overhead for the reply is $470 - 35 = 435$ bytes.

Listing 5.3: Mas registration reply message.

```
<pyld:eiCreatedPartyRegistration xmlns:pyld="http://docs.oasis–open.org/ns/energyinterop/201110/payloads">
<ei:eiResponse xmlns:ei="http://docs.oasis–open.org/ns/energyinterop/201110">
<ei:responseCode>200</ei:responseCode>
<ei:refID>f43788d5-30e0-4e7f-81d9-7a56ffd7884c</ei:refID>
</ei:eiResponse>
<ei:registrarPartyID xmlns:ei="http://docs.oasis–open.org/ns/energyinterop/201110">56f9854a-590f-4f5c-a6e8-aa01002d7dd7</ei:registrarPartyID>
</pyld:eiCreatedPartyRegistration>
```

Table 5.2: Data of the registration reply message.

Bytes	Name
1	id
2	response_code
16	ref_id
16	registrar_party_id
35	**total**

5.2.1.2 *Registration query messages*

The consumer, prosumer and Genco agents all query the balancer agent in order to obtain all the registered agents. A typical registration query message can be found in listing 5.4. As can be seen, there are only three information fields, one message identifier and two guids. The derivation for the size of these fields is exactly the same as described in section 5.2.1.1, an overview can be found in table 5.3. The size of the registration query message is 351 bytes, giving an overhead of $351 - 33 = 318$ bytes.

Listing 5.4: Mas registration query message.

```
<pyld:eiRequestPartyRegistration xmlns:pyld="http://docs.oasis–open.org/ns/energyinterop/201110/payloads">
<pyld:requestID>30fd438a-7557-411e-aaed-e1c3d7ff66eb</pyld:requestID>
<ei:requestorPartyID xmlns:ei="http://docs.oasis–open.org/ns/energyinterop/201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</ei:requestorPartyID>
</pyld:eiRequestPartyRegistration>
```

Table 5.3: Data of the registration query message.

Bytes	Name
1	id
16	request_id
16	requestor_party_id
33	**total**

The balancer replies to registration queries with a list of all the registered agents, excluding the agents that makes the query. A registration query replies message for a simulation with one consumer, prosumer and Genco is shown in listing 5.5. Analysing the overhead for this message is a bit more complicated than it was for the previous messages. Firstly, the amount of data sent depends on the total amount of registered agents, this is expressed through $N_{agents} - 1$. Secondly, there is redundant data in the message. The redundant data in this case is the `registrar_party_id`. It is sent once for each agent, per the standard. However redundant data is still overhead and thus this field is considered as being sent only once. Lastly the `address` field is a string, which is of variable size. In the case of the mas this field can be one of possibly three different types, namely {consumerXXX, prosumerXXX, gencoXXX}; where a X denotes a digit$[0-9]$. The actual value thus depends on how many of each agent are present and on which agent does the query. In order to simplify things the highest possible value(11) is chosen, which will give the lowest possible overhead. The `id`, `response_code` and guid field sizes are again derived the same way as in section 5.2.1.1. An overview of the fields can be found in table 5.4. The size of the registration message itself is also calculated based on the N_{agents} that have registered. The headers and response code listed in listing 5.5 lines $1, 26-30$ are always present and constitute 325 bytes. The number of bytes per agent is also based the maximum size for the `address` field as shown above. This leaves the size of one registration block at 520 bytes and thus the total size of a registration reply is given as $520N_{agents} + 325$. Calculation of the overhead then follows as $(520N_{agents} + 325) - (30\,(N_{agents} - 1) + 35) = 490N_{agents} + 320$ bytes.

Listing 5.5: Mas registration query reply message. For a simulation containing one consumer, prosumer and Genco agent.

```
1 <pyld:eiReplyPartyRegistration xmlns:pyld="http://docs.oasis-open
      .org/ns/energyinterop/201110/payloads">
2 <mas:masRegistration xmlns:mas="urn:rug:mas">
3 <ei:registreePartyID xmlns:ei="http://docs.oasis-open.org/ns/
      energyinterop/201110">14420c0d-2694-45ac-acf1-7849a6250b40</
      ei:registreePartyID>
```

```
<ei:registrarPartyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">56f9854a-590f-4f5c-a6e8-aa01002d7dd7</ei:registrarPartyID>
<mas:masRegistrationInfo>
<mas:agentid>1</mas:agentid>
<mas:location>
<mas:x>42</mas:x>
<mas:y>47</mas:y>
</mas:location>
</mas:masRegistrationInfo>
<mas:address>genco000</mas:address>
</mas:masRegistration>
<mas:masRegistration xmlns:mas="urn:rug:mas">
<ei:registreePartyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">dd1e82d5-a9a7-475d-bd4e-a8947baa3575</ei:registreePartyID>
<ei:registrarPartyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">56f9854a-590f-4f5c-a6e8-aa01002d7dd7</ei:registrarPartyID>
<mas:masRegistrationInfo>
<mas:agentid>2</mas:agentid>
<mas:location>
<mas:x>76</mas:x>
<mas:y>24</mas:y>
</mas:location>
</mas:masRegistrationInfo>
<mas:address>consumer000</mas:address>
</mas:masRegistration>
<ei:eiResponse xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">
<ei:responseCode>200</ei:responseCode>
<ei:refID>30fd438a-7557-411e-aaed-e1c3d7ff66eb</ei:refID>
</ei:eiResponse>
</pyld:eiReplyPartyRegistration>
```

Table 5.4: Data of the registration query reply message.

Bytes	Name
1	id
$16\,(N_{agents} - 1)$	registree_party_id
16	registrar_party_id
$N_{agents} - 1$	agent_id
$N_{agents} - 1$	x
$N_{agents} - 1$	y
$11\,(N_{agents} - 1)$	address
2	response_code
16	ref_id
$30\,(N_{agents} - 1) + 35$	**total**

5.2.1.3 *Quote messages*

A quote message, like the previous messages has a field of one byte for its `id`. Quote messages also contain a lot of guid fields, namely: `request_id`, `publisher_party_id`, `subscriber_party_id`, `quote_id`, `uid_text` and `uid`. These are all 16 bytes in size, as explained in section 5.2.1.1. The `duration` field represents a time duration. In java the `System.currentTimeMillis()` which can be used to calculate a duration or indicate a time, returns a long which in java is a 64-bit value. Therefore the `duration` field can be seen as having a size of 8 bytes. The timezone identifier field(`tzid`) is two bytes of information. Most time zones are offsets from Coordinated Universal Time (UTC) by a whole hour, between -12 and $+14$. However some countries also have offsets specified in minutes, thus time zones offset from UTC require two bytes. The `date_time` field has a size of 8 bytes for exactly the same reason as the `duration` field. Fields regarding energy `product_type`, `energy_description`, `item_description`, `energy_units` and `item_units` convey information about the energy that is required/offered. This information does not have any more than a couple of variations(in the EI standard) and thus all these fields can be represented by one byte. Next, the `energy_scale_code` and `item_scale_code` give information about the scale of the `energy_units` and `item_units` fields. The standard defines eleven possible values for the scale, thus these fields convey one byte of information. Proceeding with the fields that contain floating point values `unit_price`, `hertz`, `voltage`, `charge` and `maximum_power`. Code wise these are represented as double precision floating point numbers, which in java are 8 bytes. The `ac` field is a boolean, which technically can be represented as one bit. However, because the smallest data size on most processors is one byte, and in order to keep the calculation in whole bytes, the field is defined as having the size of one byte. The two fields with enumerable types `transactive_state` and `side` both contain less than 256 types and thus can both be represented by one byte. Lastly, the `market_context` field can hold a string of variable length. In the mas implementation the only value assigned to this field is `urn:rug:mas`, giving it a size of 11 bytes. A full overview of all the fields can be found in table 5.5. The message overhead is calculated to be $2531 - 176 = 2355$ bytes. In the quote messages to the balancer agent, where some data is omitted, the overhead is calculated as $2027 - 157 = 1870$ bytes, see listing B.1 in appendix B for the reference message.

Listing 5.6: Mas quote message.

```
<pyld:eiCreateQuote xmlns:pyld="http://docs.oasis-open.org/ns/
    energyinterop/201110/payloads">
<pyld:requestID>fbcd65b5-2695-43eb-a67c-717cf53a597b</
    pyld:requestID>
```

```
3  <ei:publisherPartyID xmlns:ei="http://docs.oasis-open.org/ns/
       energyinterop/201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</
       ei:publisherPartyID>
4  <ei:subscriberPartyID xmlns:ei="http://docs.oasis-open.org/ns/
       energyinterop/201110">dd1e82d5-a9a7-475d-bd4e-a8947baa3575</
       ei:subscriberPartyID>
5  <ei:eiQuote xmlns:ei="http://docs.oasis-open.org/ns/energyinterop
       /201110">
6  <ei:quoteID>ed5b1cd5-69cb-4ff2-8878-d98113519f7b</ei:quoteID>
7  <emix:product xmlns:emix="http://docs.oasis-open.org/ns/emix
       /2011/06">
8  <xcal:components xmlns:xcal="urn:ietf:params:xml:ns:icalendar-2.0
       ">
9  <xcal:interval>
10 <xcal:properties>
11 <xcal:uid>
12 <xcal:text>ebd61bf6-2ee4-4de3-be08-972a7190d8c4</xcal:text>
13 </xcal:uid>
14 <xcal:duration>
15 <xcal:duration>PT15M</xcal:duration>
16 </xcal:duration>
17 <xcal:dtstart>
18 <xcal:parameters>
19 <xcal:tzid>
20 <xcal:text>Netherlands/Amsterdam</xcal:text>
21 </xcal:tzid>
22 </xcal:parameters>
23 <xcal:date-time>2014-07-28T09:00:00</xcal:date-time>
24 </xcal:dtstart>
25 <xcal:x-wsCalendar-attach>
26 <power:fullRequirementsPower xmlns:power="http://docs.oasis-open.
       org/ns/emix/2011/06/power">
27 <power:productType>energy</power:productType>
28 <power:unitEnergyPrice>
29 <emix:price>
30 <emix:value>9.00267499080293E-5</emix:value>
31 </emix:price>
32 <power:energyReal>
33 <power:itemDescription>RealEnergy</power:itemDescription>
34 <power:itemUnits>Wh</power:itemUnits>
35 <scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/
       emix/2011/06/siscale">none</scale:siScaleCode>
36 </power:energyReal>
37 </power:unitEnergyPrice>
38 <power:powerReal>
39 <power:itemDescription>RealPower</power:itemDescription>
40 <power:itemUnits>W</power:itemUnits>
41 <scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/
       emix/2011/06/siscale">none</scale:siScaleCode>
42 <power:powerAttributes>
43 <power:hertz>50.0</power:hertz>
44 <power:voltage>230.0</power:voltage>
```

```
<power:ac>true</power:ac>
</power:powerAttributes>
</power:powerReal>
<power:charges>
<mas:masTSOCharge xmlns:mas="urn:rug:mas">
<emix:price>
<emix:value>0.0085</emix:value>
</emix:price>
</mas:masTSOCharge>
</power:charges>
<power:maximumPower>509.1401460079817</power:maximumPower>
</power:fullRequirementsPower>
</xcal:x-wsCalendar-attach>
</xcal:properties>
</xcal:interval>
</xcal:components>
<emix:uid>1c5e7d8f-138c-4a5f-bf4c-b2b111c40bfb</emix:uid>
<emix:transactiveState>publication</emix:transactiveState>
<emix:marketContext>urn:rug:mas</emix:marketContext>
<emix:side>sell</emix:side>
</emix:product>
</ei:eiQuote>
</pyld:eiCreateQuote>
```

Table 5.5: Data of the quote message.

Bytes	Name
1	id
16	request_id
16	publisher_party_id
16	subscriber_party_id
16	quote_id
16	uid_text
8	duration
2	tzid
8	date_time
1	product_type
8	unit_price[4]
1	energy_description[4]
1	energy_units[4]
1	energy_scale_code[4]
1	item_description
1	item_units
1	item_scale_code
8	hertz

4 This value is not present in the initial quote messages sent to the balancer.

Table 5.5: Data of the quote message – continued from previous page.

Bytes	Name
8	voltage
1	ac
8	charge[4]
8	maximum_power
16	uid
1	transactive_state
11	market_context
1	side
176	**total**

The reply to quote messages as shown in listing 5.7 is similar to the replies seen so far. The derivation for the fields shown in table 5.6 is the same as done in the previous sections. Calculation of the overhead then follows as $571 - 51 = 520$ bytes.

Listing 5.7: Mas quote reply message.

```
<pyld:eiCreatedQuote xmlns:pyld="http://docs.oasis-open.org/ns/
    energyinterop/201110/payloads">
<ei:eiResponse xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">
<ei:responseCode>200</ei:responseCode>
<ei:refID>fbcd65b5-2695-43eb-a67c-717cf53a597b</ei:refID>
</ei:eiResponse>
<ei:subscriberPartyID xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">dd1e82d5-a9a7-475d-bd4e-a8947baa3575</
    ei:subscriberPartyID>
<ei:quoteID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop
    /201110">ed5b1cd5-69cb-4ff2-8878-d98113519f7b</ei:quoteID>
</pyld:eiCreatedQuote>
```

Table 5.6: Data of the quote reply message.

Bytes	Name
1	id
2	response_code
16	ref_id
16	subscriber_party_id
16	quote_id
51	**total**

5.2.1.4 *Tender/Transaction messages*

The tender/transaction messages and their replies are similar to the quote messages. They only differ in the field names, as such the amount of data sent is equal to what is listed in tables 5.5 and 5.6. For completeness the overhead for these four messages is still calculated in this section. The reference messages used for the calculations can be found in appendix B, listings B.2 to B.5. Starting with the tender message, the calculated overhead is $2507 - 176 = 2331$ bytes. Overhead for the reply to a tender message is $507 - 51 = 456$ bytes. Similarly the transaction message overhead is $2544 - 176 = 2368$ bytes. Finally the overhead for replies to a transaction message is $577 - 51 = 526$ bytes.

5.2.2 *Minimum amount of data overhead*

Using equations (5.1), (5.4), (5.6), (5.9) and (5.10) and the results from sections 5.1 and 5.2.1 the minimum amount of overhead generated by the messages from the EI standard can be calculated. Each calculation below is done in bytes. Starting with the registration gives:

$$\begin{aligned} O_r &= 479\,(N_c + N_p + N_g) + 435\,(N_c + N_p + N_g) \\ &= 914\,(N_c + N_p + N_g) \end{aligned}$$

Calculating the registration queries and their replies:

$$\begin{aligned} O_{rq} &= (490N_{agents} + 320)\,(N_c + N_p + N_g) \\ &\quad + 318\,(N_c + N_p + N_g) \\ &= (490\,(N_c + N_p + N_g) + 320)\,(N_c + N_p + N_g) \\ &\quad + 318\,(N_c + N_p + N_g) \\ &= 490\,(N_c + N_p + N_g)^2 + 638\,(N_c + N_p + N_g) \end{aligned}$$

Calculating the replies to power supply requests:

$$\begin{aligned} O_{rpr} &= 1870\,(N_c + N_p) + 520\,(N_c + N_p) \\ &= 2390\,(N_c + N_p) \end{aligned}$$

Calculating the start quotes overhead:

$$\begin{aligned} O_{sq} &= 2355N_c\,(N_p + N_g) + 520N_c\,(N_p + N_g) \\ &= 2875N_c\,(N_p + N_g) \end{aligned}$$

Calculating the overhead of the full transaction process:

$$\begin{aligned} O_t &= 2331N_c + 456N_c + 2368N_c + 526N_c \\ &= 5681N_c \end{aligned}$$

The total overhead O is then given by $O = O_r + O_{rq} + O_{rpr} + O_{sq} + O_t$.

$$
\begin{aligned}
O &= 914\,(N_c + N_p + N_g) + 490\,(N_c + N_p + N_g)^2 \\
&\quad + 638\,(N_c + N_p + N_g) + 2390\,(N_c + N_p) \\
&\quad + 2875 N_c\,(N_p + N_g) + 5681 N_c \\
&= 9223 N_c + 3942 N_p + 1552 N_g + 490\,(N_c + N_p + N_g)^2 \\
&\quad + 2875\,(N_c N_p + N_c N_g) \\
&= 9223 N_c + 3942 N_p + 1552 N_g + 2875\,(N_c N_p + N_c N_g) \\
&\quad + 490\left(N_c^2 + N_p^2 + N_g^2 + 2N_c N_p + 2N_c N_g + 2N_p N_g\right) \\
&= 9223 N_c + 3942 N_p + 1552 N_g + 3855\,(N_c N_p + N_c N_g) \\
&\quad + 490\left(N_c^2 + N_p^2 + N_g^2 + 2N_p N_g\right) \qquad (5.14)
\end{aligned}
$$

Equation (5.14) shows that the overhead is quite large. Furthermore it contains the quadratic terms N_c^2, N_p^2 and N_g^2, meaning the overhead scales quadratically with the number of agents. Using equation (5.14) with the number of agents equal to those in the simulation, the results in table 5.7 are obtained. Combining the overhead data with the data in figure 5.3, it is evident that the overhead is at least between $70 - 80\%$. This coincides with the amount of overhead for data-centric XML documents found by Lawrence[18], as discussed earlier.

Table 5.7: Minimum overhead numbers in bytes.

N_c	N_p	N_g	**Overhead**	**Rounded**
1	1	1	24877	$2.5e+04$
10	7	3	607980	$6.1e+05$
20	7	3	1232710	$1.2e+06$
30	7	3	1955440	$2.0e+06$

6 CONCLUSION

In this thesis several standards that define messages and/or interaction sequences that allow for dr pricing of energy in wholesale markets are evaluated. The evaluation takes into account software quality attributes as well as how well it would fit into the mas developed by Capodieci[5]. One standard is chosen, namely Energy Interoperation[38]. The amount of data sent when using the standard is then compared with the amount of data sent by the original implementation. Following this the overhead introduced by the standard is then calculated.

6.1 OVERHEAD

The resulting overhead of $70 - 80\%$ that results from standardising the messaging process is significant. To elaborate, according to the "Centeraal Bureau voor de Statistiek"[28] the province of Groningen in the Netherlands had 286645 private households in 2014. Unfortunately there are no similar numbers available for how many private households in the province of Groningen are able to produce energy. So lets assume only 1% is able to function as a prosumer. This means there would be 2866 prosumer households and the remaining 283779 will be consumer households. When looking at Gencos in the Groningen area upwards of 20 can be found. Probably not all of the Gencos have their own energy production sites, as such the number chosen for the amount of Gencos is 10. Plugging these numbers into equation (5.14) returns a minimum overhead of $42612888890759\text{B} \approx 39\text{TiB}$ for one round of negotiation, provided all agents participate. Successive rounds[1] do not require need to go through a registration phase again. Each round does require the starting quotes to be sent out to each consumer. The overhead that this produces for one prosumer or Genco is 2875N_c bytes, which using the numbers stated earlier results into $815864625\text{B} \approx 778\text{MiB}$. Field tests[21] using the G3-PLC[29] standard show that data rates over 100 kilo bit per second are achievable for sending data over the U.S. power distribution grid. The calculating how long it would take a single prosumer or Genco to send all the required quote overhead for the number above can be found in equation (6.1).

$$\frac{815864625 * 8\text{b}}{100 * 10^3 \frac{\text{b}}{\text{s}}} \approx 65269\text{s} \geqslant 18 \text{ hours} \tag{6.1}$$

1 Not implemented.

$$\frac{815864625 * 8b}{100 * 10^6 \frac{b}{s}} \approx 65s \tag{6.2}$$

As can be seen sending of the overhead(which makes up most of the message) alone takes 18 hours. Would a mas be implemented according to the implementation in chapter 4, then using the power distribution grid for communication would perhaps be feasible in a market that uses price and power calculations a day to a couple of days ahead of time. For a more dynamic market it would be more realistic to use an Ethernet connection. Equation (6.2) shows that sending of the overhead would merely take about 65s under the same circumstances, using and Ethernet connection with a speed of 100 Mbps.

6.2 STANDARD

The EI standard does not fulfil all of the needs of the mas created by Capodieci leading to the need for control messages in the implementation. Messages or interaction patterns concerning weather data are not present in any of standards. Weather data is also part as an external service in an other mas[17] that was discussed in section 2.2. Upcoming standard and revisions of current ones might want to include a mechanism for the data exchange of such information.

While EI[38, section 6.4] does seem to specifically support a push and pull patterns for quotes. Other parts such as EI[38, section 6.3, section 7.2] imply that only already created quotes can be requested. The documentation in the XSD specification also enforces this. The message ordering change as mentioned in section 4.2 where prosumer and Genco agents send their quotes to the consumers instead of the consumers making a query themselves is a direct effect of this. All in all the documentation for the interactions consists of a table stating the functions and a sequence diagram accompanied with the XSD specification. The EI standard would certainly benefit a bit of disambiguation in this area.

6.3 FUTURE WORK

As mentioned in chapter 4 not the whole EI standard is implemented in the mas. Improvements in this area can be made by extending the mas with support for terms and warrants. Adding support for the VEN/VTN and the dr events to the current mas is also a possibility. Other missing features, such as support for different currencies, different time schedules or power attributes could also be implemented.

Evaluation of other standards than the ones examined in this thesis might provide better messaging support, and less overhead. Other standards could be implemented and than compared to the original

mas as well as the mas implementation described in this thesis. A good candidate for examining would be the OpenADR standard version 2. At the time the standards in this thesis were examined, several parts of the second OpenADR 2 standard were already finalised, but it was not completely finished; and as such was not examined.

BIBLIOGRAPHY

[1] Fabio Bellifemine, Giovanni Caire, Agostino Poggi and Giovanni Rimassa. *JADE. A White Paper*. White Paper. Telecom Italia Lab, September 2003.

[2] Paul V. Biron and Ashok Malhotra, editors. *XML Schema Part 2: Datatypes*. 2nd edition. W3C Recommendation. 28th October 2004. URL: http://www.w3.org/TR/xmlschema-2/ (visited on 01/08/2014).

[3] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and François Yergeau, editors. *Extensible Markup Language*. 5th edition. W3C Recommendation. Version 1.0. 26th November 2008. URL: http://www.w3.org/TR/2008/REC-xml-20081126/ (visited on 01/08/2014).

[4] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau and John Cowan, editors. *Extensible Markup Language*. 2nd edition. W3C Recommendation. Version 1.1. 16th August 2006. URL: http://www.w3.org/TR/2006/REC-xml11-20060816/ (visited on 01/08/2014).

[5] Nicola Capodieci. "P2P energy exchange agent platform featuring a game theory related learning negotiation algorithm". Master's thesis. University of Modena and Reggio Emilia, 2011.

[6] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard and Sanjiva Weerawarana, editors. *Web Services Description Language Part 2: Adjuncts*. W3C Recommendation. Version 2.0. 26th June 2007. URL: http://www.w3.org/TR/wsdl20-adjuncts/ (visited on 13/08/2014).

[7] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman and Sanjiva Weerawarana, editors. *Web Services Description Language Part 1: Core Language*. W3C Recommendation. Version 2.0. 26th June 2007. URL: http://www.w3.org/TR/wsdl20/ (visited on 13/08/2014).

[8] Tim Dierks. *The Transport Layer Security (TLS) Protocol*. RFC 2246. Version 1.0. Internet Engineering Task Force. January 1999. URL: https://www.ietf.org/rfc/rfc2246.txt (visited on 10/09/2014).

[9] A.P. Garcia, J. Oliver and D. Gosch. "An intelligent agent-based distributed architecture for Smart-Grid integrated network management". In: *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. October 2010, pages 1013–1018. DOI: 10.1109/LCN.2010.5735673.

[10] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar and Yves Lafon, editors. *SOAP Part 1: Messaging Framework*. 2nd edition. W3C Recommendation. Version 1.2. 27th April 2007. URL: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (visited on 13/08/2014).

[11] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar and Yves Lafon, editors. *SOAP Part 2: Adjuncts*. 2nd edition. W3C Recommendation. Version 1.2. 27th April 2007. URL: http://www.w3.org/TR/2007/REC-soap12-part2-20070427/ (visited on 13/08/2014).

[12] V.C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati and G.P. Hancke. "Smart Grid Technologies: Communication Technologies and Standards". In: *Industrial Informatics, IEEE Transactions on* 7.4 (November 2011), pages 529–539. ISSN: 1551-3203. DOI: 10.1109/TII.2011.2166794.

[13] M.P.F. Hommelberg, C.J. Warmer, I.G. Kamphuis, J.K. Kok and G.J. Schaeffer. "Distributed Control Concepts using Multi-Agent technology and Automatic Markets: An indispensable feature of smart power grids". In: *Power Engineering Society General Meeting, 2007. IEEE*. June 2007, pages 1–7. DOI: 10.1109/PES.2007.385969.

[14] J. K. Kok, C. J. Warmer and I. G. Kamphuis. "PowerMatcher: Multiagent Control in the Electricity Infrastructure". In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '05. The Netherlands: ACM, 2005, pages 75–82. ISBN: 1-59593-093-0. DOI: 10.1145/1082473.1082807.

[15] Marcel Koster. "Reliable Multi-agent System for a large scale distributed energy trading network". Master's thesis. University of Groningen, 2011.

[16] Hugo Krawczyk, Mihir Bellare and Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. February 1997. URL: https://www.ietf.org/rfc/rfc2104.txt (visited on 11/09/2014).

[17] Steffen Lamparter, Silvio Becher and Jan-Gregor Fischer. "An Agent-based Market Platform for Smart Grids". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Industry Track*. AAMAS '10. Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pages 1689–1696. ISBN: 978-0-9826571-4-0.

[18] Ramon Lawrence. "The space efficiency of XML". In: *Information and Software Technology* 46.11 (2004), pages 753–759. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2004.02.003.

[19] Mary Ann Piette, Girish Ghatikar, Sila Kiliccote, Ed Koch, Dan Hennage, Peter Palensky and Charles McParland. *Open Automated Demand Response Communications Specification*. Version 1.0. April 2009. California Energy Commission, PIER Program. CEC-500-2009-063.

[20] J. Postel. *Transmission Control Protocol*. RFC 793. September 1981. URL: https://www.ietf.org/rfc/rfc793.txt (visited on 11/09/2014).

[21] K. Razazian, A. Kamalizad, M. Umari, Qi Qu, V. Loginov and M. Navid. "G3-PLC field trials in U.S. distribution grid: Initial results and requirements". In: *Power Line Communications and Its Applications (ISPLC), 2011 IEEE International Symposium on*. April 2011, pages 153–158. DOI: 10.1109/ISPLC.2011.5764382.

[22] R. L. Rivest, A. Shamir and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 26.1 (January 1983), pages 96–99. ISSN: 0001-0782. DOI: 10.1145/357980.358017.

[23] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler. *SIP: Session Initiation Protocol*. RFC 3261. Version 2. June 2002. URL: https://www.ietf.org/rfc/rfc3261.txt (visited on 07/04/2015).

[24] R. Shirey. *Internet Security Glossary*. RFC 4949. Version 2. August 2007. URL: https://www.ietf.org/rfc/rfc4949.txt (visited on 10/09/2014).

[25] Henry S. Thompson, David Beech, Murray Maloney and Noah Mendelsohn, editors. *XML Schema Part 1: Structures*. 2nd edition. W3C Recommendation. 28th October 2004. URL: http://www.w3.org/TR/xmlschema-1/ (visited on 01/08/2014).

[26] Jun Wang and V.C.M. Leung. "A survey of technical requirements and consumer application standards for IP-based smart grid AMI network". In: *Information Networking (ICOIN), 2011 International Conference on*. January 2011, pages 114–119. DOI: 10.1109/ICOIN.2011.5723144.

[27] Bureau International des Poids et Mesures. *The International System of Units (SI)*. 8th edition. 1, Boulevard Ney, 75018 Paris: STEDI MEDIA, May 2006. ISBN: 92-822-2213-6.

[28] Centraal Bureau voor de Statistiek, Den Haag/Heerlen. *Huishoudens; samenstelling, grootte, regio, 1 januari*. April 2015. URL: http://statline.cbs.nl/StatWeb/publication/?VW=T&DM=SLNL&PA=71486NED&D1=0-2,23-26&D2=0&D3=0,5-16&D4=(l-

1)-l&HD=090402-0910&HDR=T,G3&STB=G1,G2 (visited on 08/04/2015).

[29] Electricite Reseau Distribution. *G3-PLC physical layer specification*. August 2009.

[30] European Parliament and Council of the European Union. "concerning common rules for the internal market in electricity and repealing Directive 2003/54/EC". In: *Official Journal of the European Union*. L 52.211 (14th August 2009), pages 55–93. ISSN: 1725-2555. DOI: doi:10.3000/17252555.L_2009.211.eng.

[31] FIPA TC Communication. *FIPA ACL Message Structure Specification*. 3rd December 2002.

[32] "IEEE Standard for Local Area Network/Wide Area Network (LAN/WAN) Node Communication Protocol to Complement the Utility Industry End Device Data Tables". In: *IEEE Std 1703-2012* (June 2012), pages 1–239. DOI: 10.1109/IEEESTD.2012.6226334.

[33] ISO/TC 68. *ISO 4217. Codes for the representation of currencies and funds*. International Organization for Standardization. 2008.

[34] ISO/IEC JTC 1/SC 27. *ISO 18033-3. Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers*. International Organization for Standardization. 2005.

[35] ISO/TC 211. *ISO 19136. Geographic information – Geography Markup Language*. International Organization for Standardization. 2007.

[36] National Institute of Standards and Technology. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. Federal Information Processing Standards Publication 197. Department of Commerce, November 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (visited on 11/09/2014).

[37] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication 180–4. Department of Commerce, March 2012. URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf (visited on 11/09/2014).

[38] OASIS Energy Interoperation TC. *Energy Interoperation*. Edited by Toby Considine. OASIS Standard. Version 1.0. OASIS. 11th June 2014. URL: http://docs.oasis-open.org/energyinterop/ei/v1.0/energyinterop-v1.0.html (visited on 19/09/2014).

[39] OASIS Energy Market Information Exchange TC. *Energy Market Information Exchange*. OASIS Committee Specification 02. Version 1.0. OASIS. 11th January 2012. URL: http://docs.oasis-open.org/emix/emix/v1.0/cs02/emix-v1.0-cs02.html (visited on 06/06/2014).

[40] OASIS WS-Calendar TC. *WS-Calendar*. OASIS Committee Specification 01. Version 1.0. OASIS. 30th July 2011. URL: http://docs.oasis-open.org/ws-calendar/ws-calendar-spec/v1.0/cs01/ws-calendar-spec-v1.0-cs01.html (visited on 06/06/2014).

APPENDIX

A

OPENADR

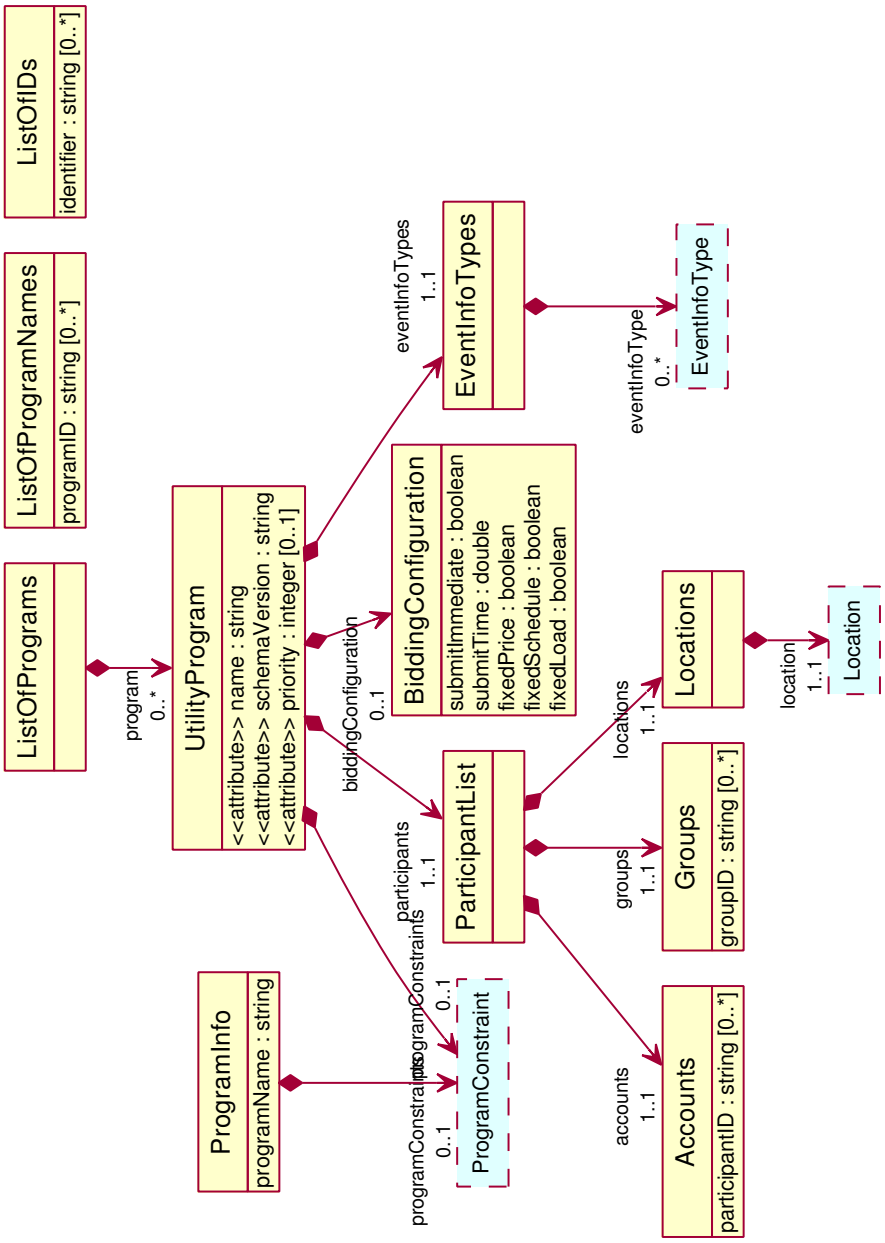


Figure A.1: Data types defined in the UtilityProgram namespace. Yellow rectangles denote types that are in the namespace and green rectangles denote types that are defined in an other namespace.

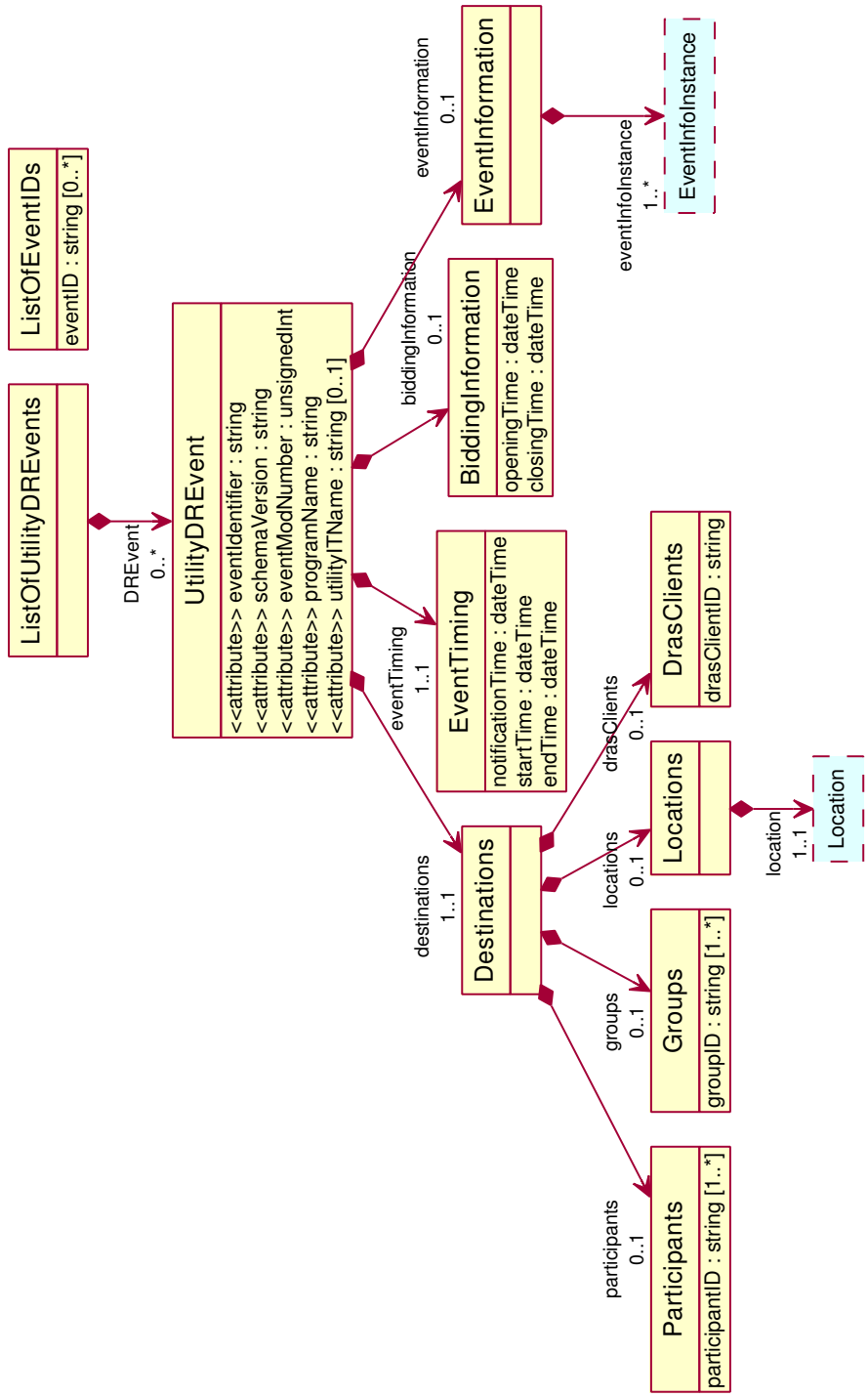


Figure A.2: Data types defined in the UtilityDREvent namespace. Yellow rectangles denote types that are in the namespace and green rectangles denote types that are defined in an other namespace.

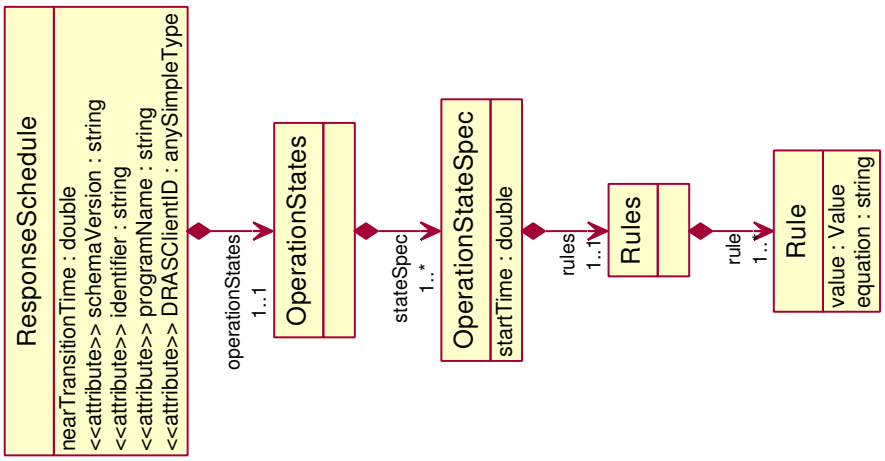


Figure A.3: Data types defined in the ResponseSchedule namespace.

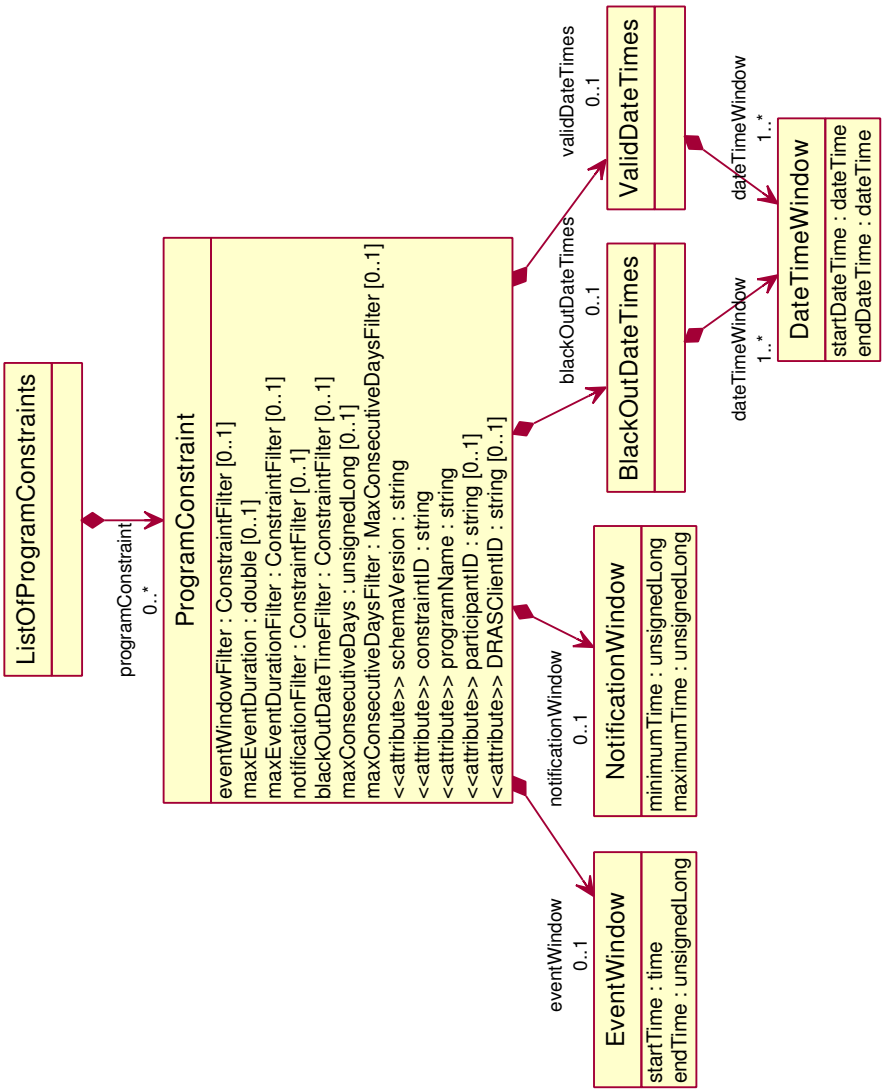


Figure A.4: Data types defined in the ProgramConstraint namespace.

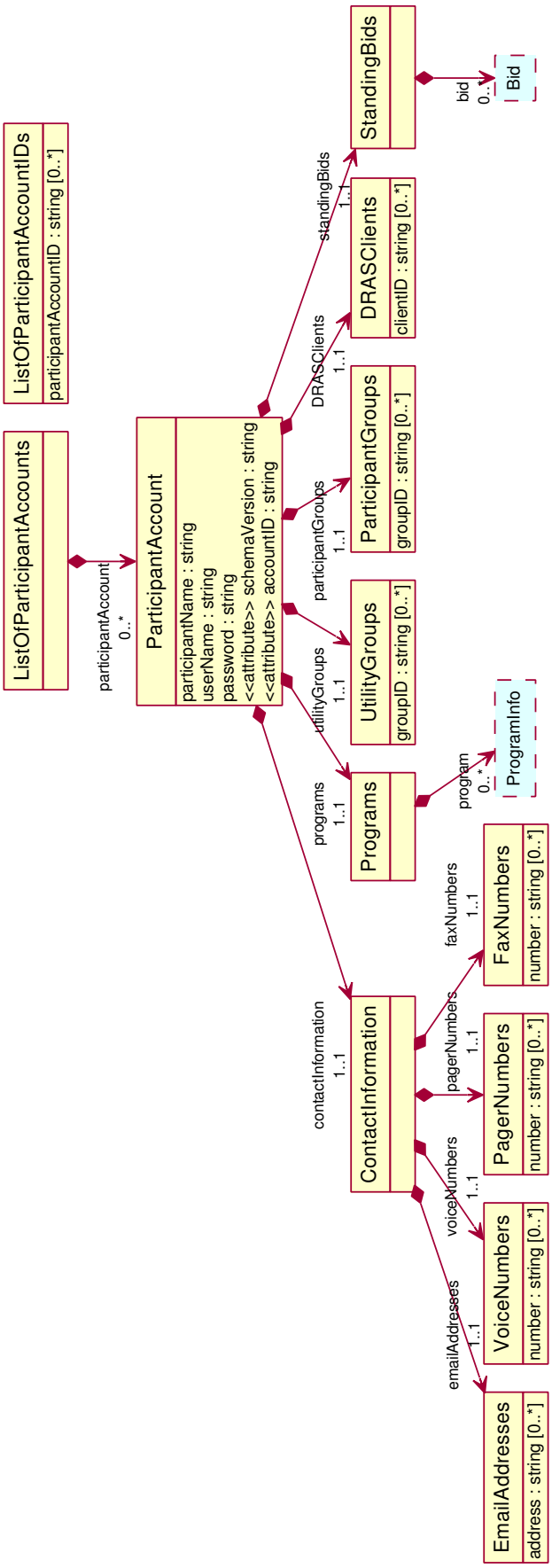


Figure A.5: Data types defined in the ParticipantAccount namespace. Yellow rectangles denote types that are in the namespace and green rectangles denote types that are defined in an other namespace.

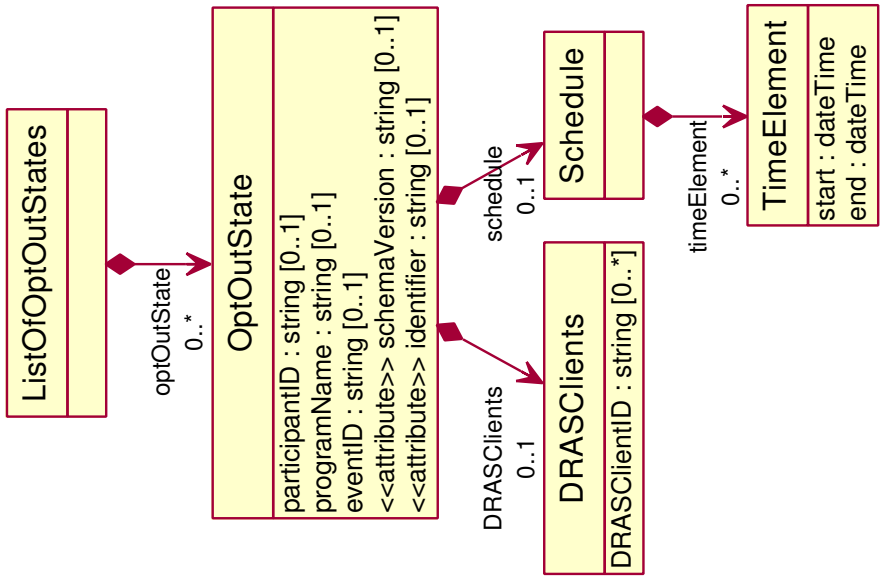


Figure A.6: Data types defined in the OptOutState namespace.

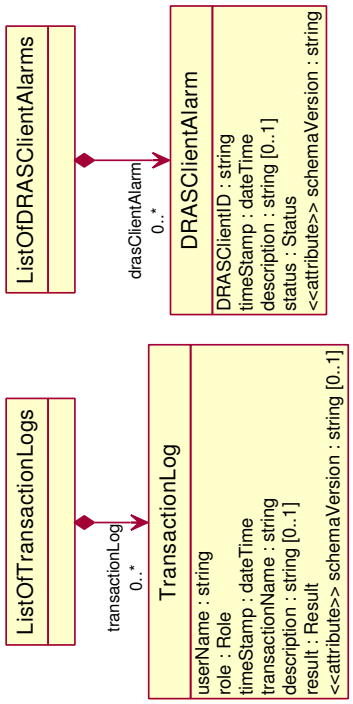


Figure A.7: Data types defined in the Logs namespace.

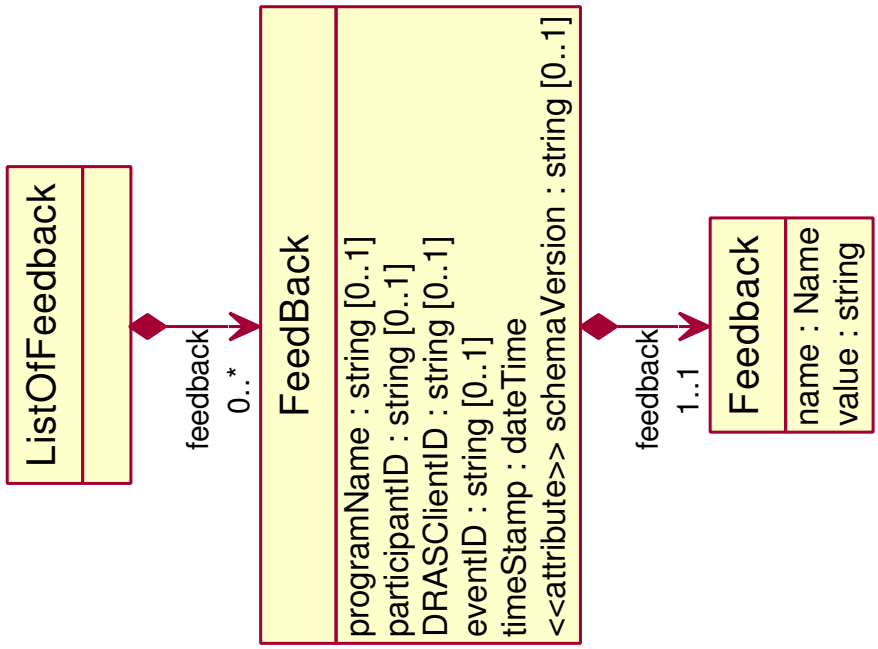


Figure A.8: Data types defined in the Feedback namespace.

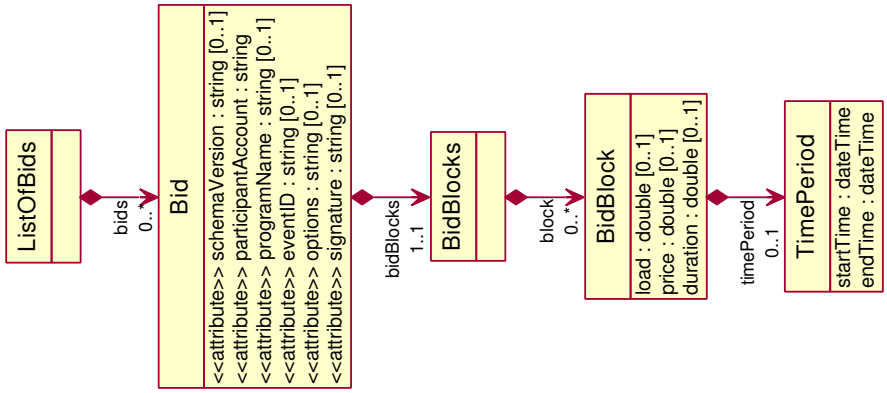


Figure A.9: Data types defined in the Bid namespace.

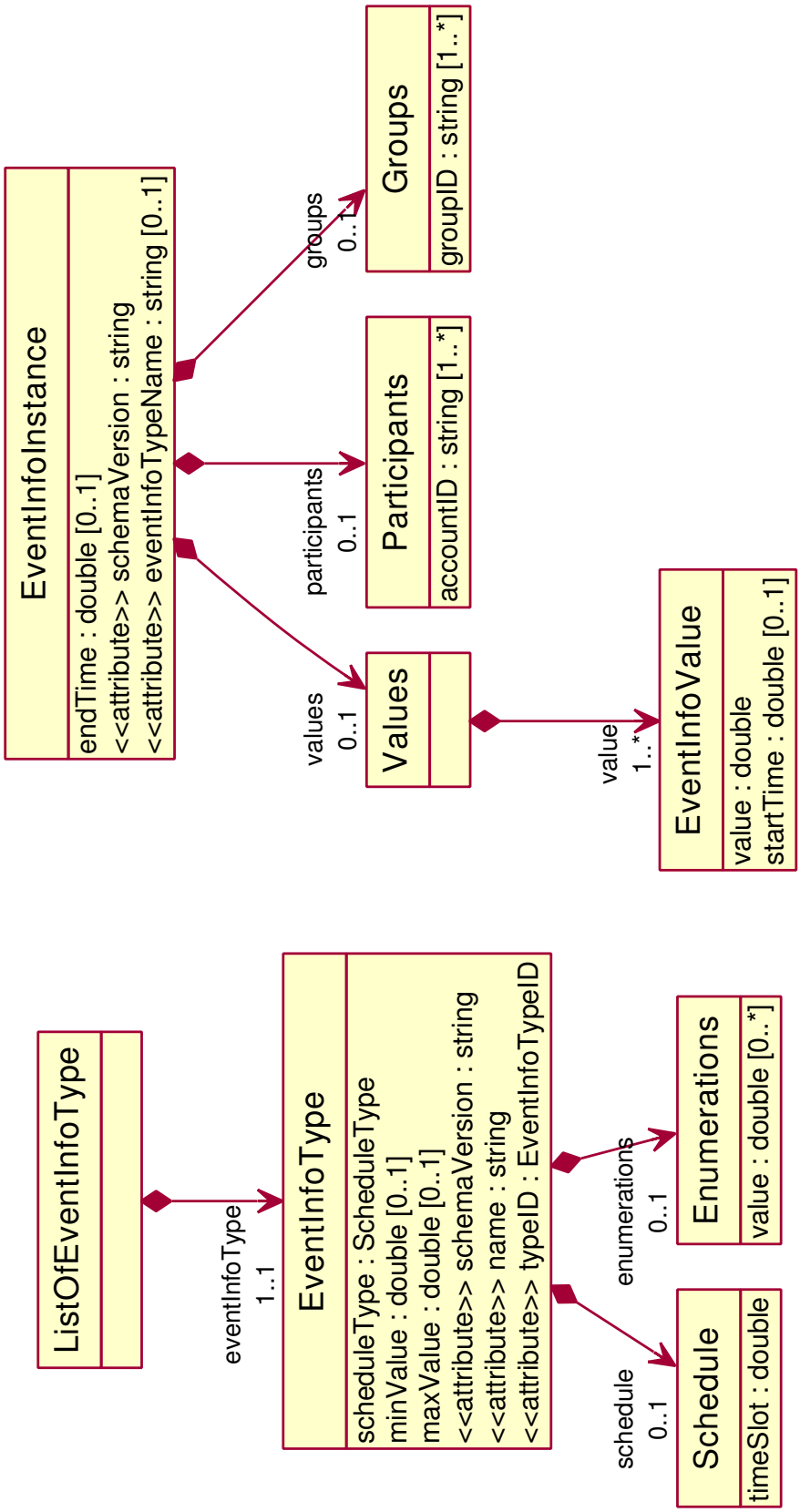


Figure A.10: Data types defined in the EventInfo namespace.

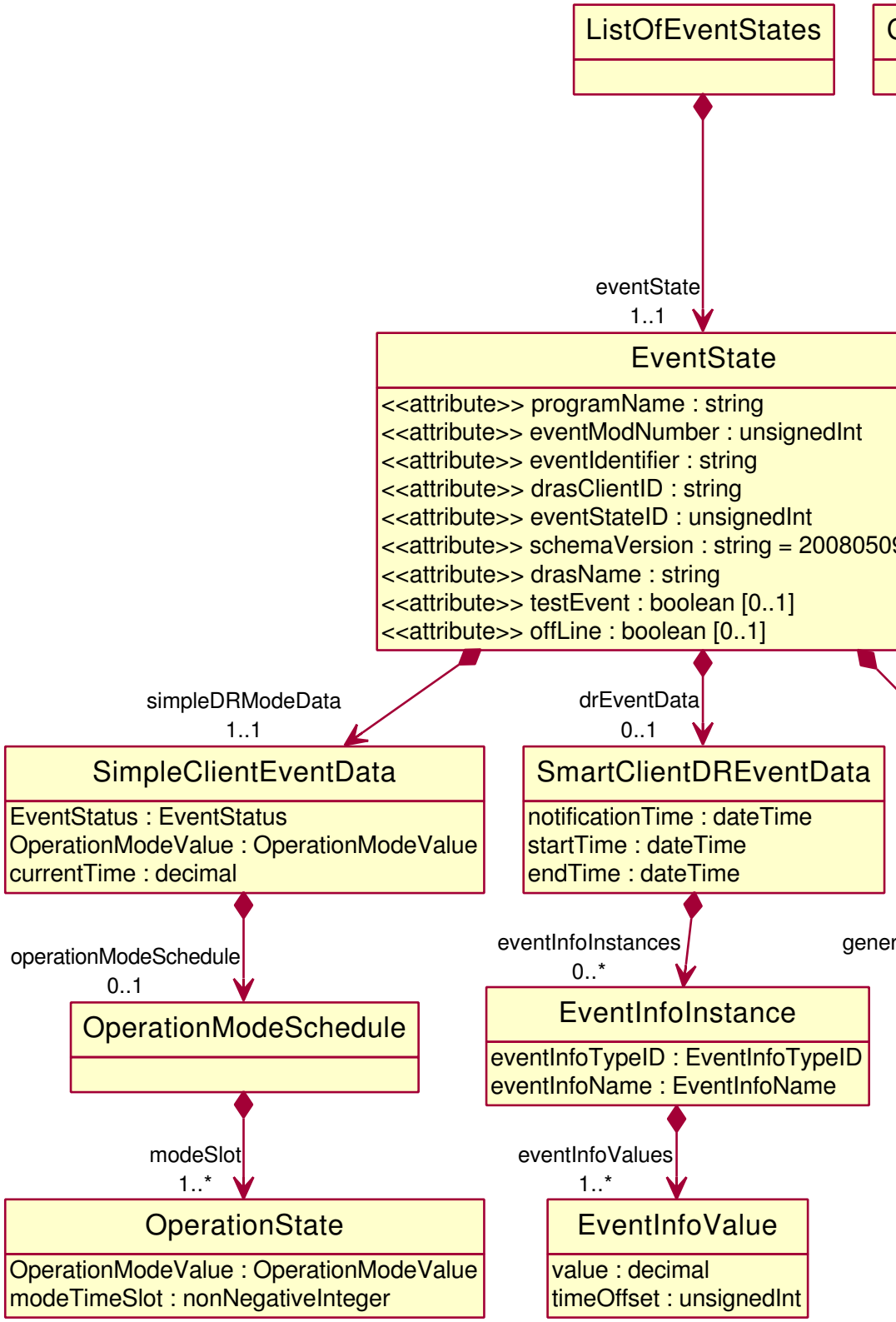
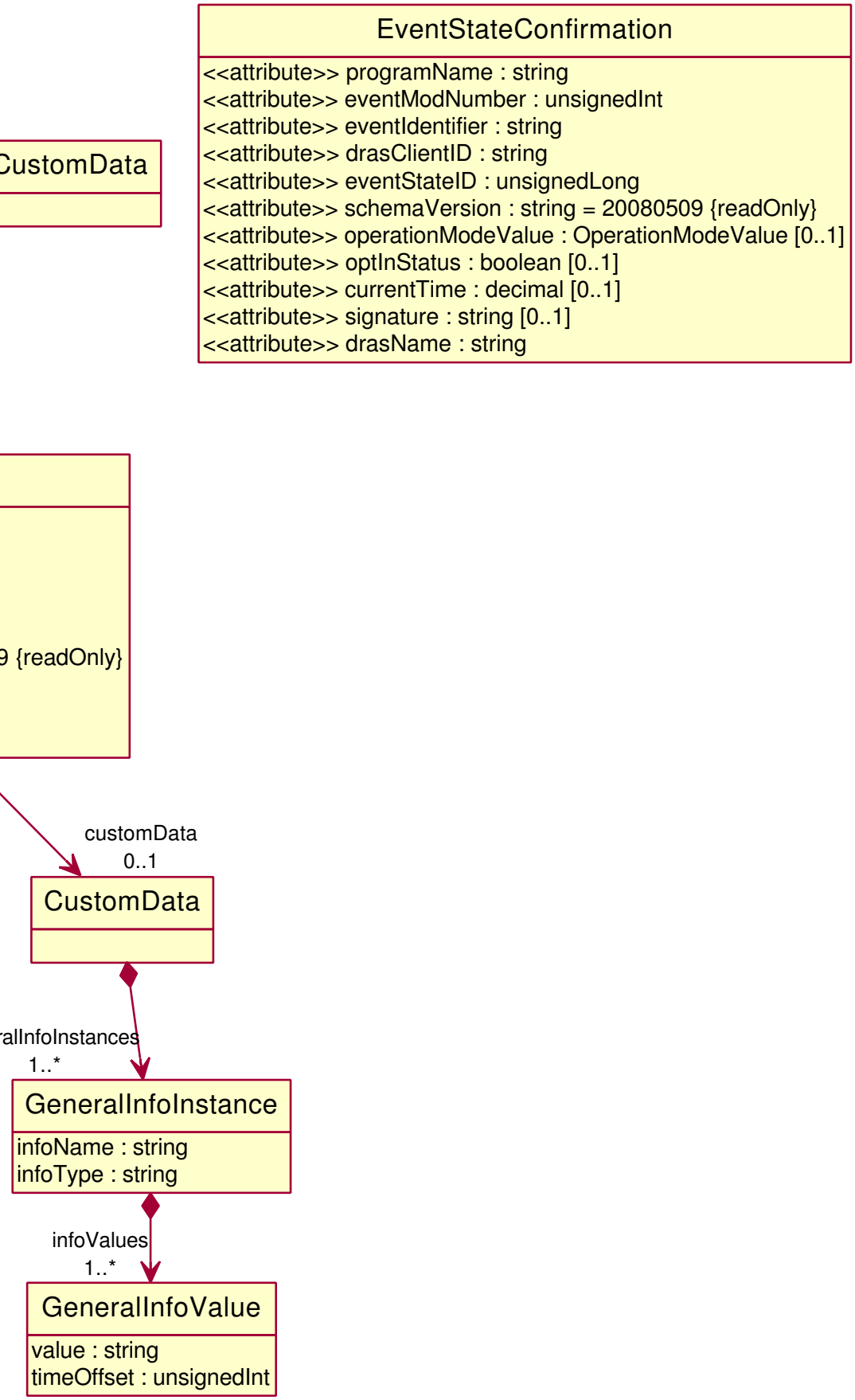


Figure A.11: Data types defined in the EventState namespace.


EventStateConfirmation
<<attribute>> programName : string
<<attribute>> eventModNumber : unsignedInt
<<attribute>> eventIdentifier : string
<<attribute>> drasClientID : string
<<attribute>> eventStateID : unsignedLong
<<attribute>> schemaVersion : string = 20080509 {readOnly}
<<attribute>> operationModeValue : OperationModeValue [0..1]
<<attribute>> optInStatus : boolean [0..1]
<<attribute>> currentTime : decimal [0..1]
<<attribute>> signature : string [0..1]
<<attribute>> drasName : string
CustomData
9 {readOnly}
customData
0..1
CustomData
alInfoInstances
1..*
GeneralInfoInstance
infoName : string
infoType : string
infoValues
1..*
GeneralInfoValue
value : string
timeOffset : unsignedInt

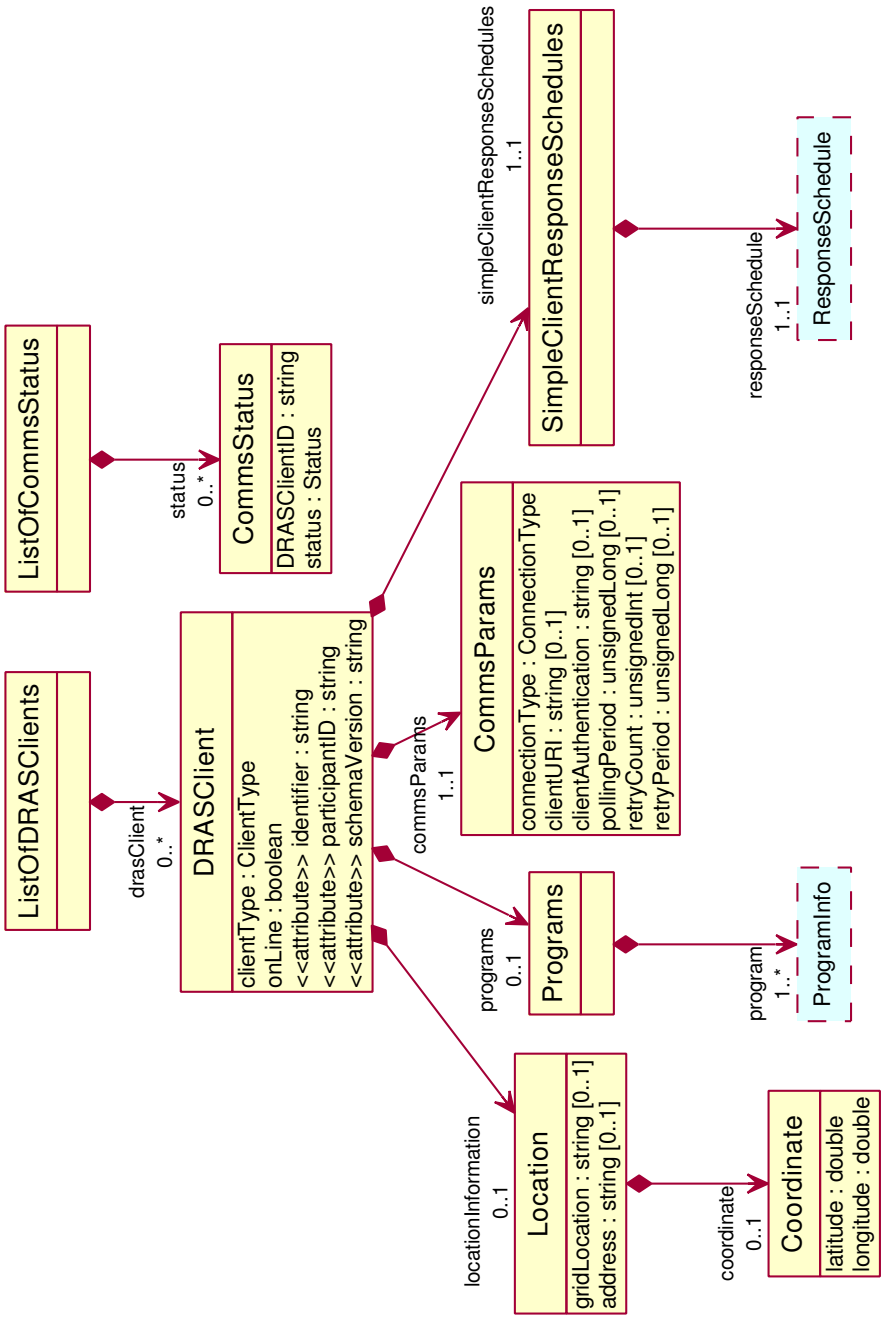


Figure A.12: Data types defined in the DRASClient namespace. Yellow rectangles denote types that are in the namespace and green rectangles denote types that are defined in an other namespace.

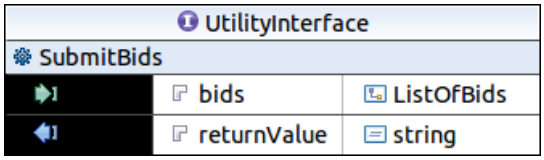


Figure A.13: Utility interface as defined by OpenADR.

UtilityOperator		
InitiateDREvent		
	DR_Event	UtilityDREvent
	returnValue	ConstraintFilter
ModifyDREvent		
	eventID	string
	modificationType	string
	DR_Event	UtilityDREvent
	returnValue	ConstraintFilter
AdjustDREventParticipants		
	eventID	string
	addParticipants	boolean
	participantList	ParticipantList
	returnValue	string
GetDREventInformation		
	eventID	ListOfEventIDs
	programNames	ListOfProgramNames
	participants	ParticipantList
	DREvents	ListOfUtilityDREvents
CreateProgram		
	program	UtilityProgram
	returnValue	string
ModifyProgram		
	program	UtilityProgram
	returnValue	string
DeleteProgram		
	programName	string
	returnValue	string
GetPrograms		
	programNames	ListOfProgramNames
	participants	ParticipantList
	programList	ListOfPrograms
CreateParticipantAccounts		
	participantAccounts	ListOfParticipantAccounts
	returnValue	string
ModifyParticipantAccounts		
	participantAccounts	ListOfParticipantAccounts
	returnValue	string
DeleteParticipantAccounts		
	participantAccountIDs	ListOfParticipantAccountIDs
	returnValue	string
GetParticipantAccounts		
	participants	ParticipantList
	programName	string
	participantAccounts	ListOfParticipantAccounts
AdjustProgramParticipants		
	programName	string
	addParticipants	boolean
	participantList	ParticipantList
	returnValue	string

Figure A.14: Utility operator/Iso interface as defined by OpenADR.

GetCurrentBids		
	eventID	string
	programName	string
	bids	ListOfBids
	biddingOpen	boolean
CloseBidding		
	eventID	string
	returnValue	string
SetBidStatus		
	eventID	string
	acceptedBids	ListOfUtilityDREvents
	rejectedBids	ListOfParticipantAccountIDs
	returnValue	string
GetDRASClientCommsStatus		
	participants	ParticipantList
	programNames	ListOfProgramNames
	DRASClientStatus	ListOfCommsStatus
GetDRASTransactions		
	userNames	ListOfIDs
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfIDs
	methodName	string
	timeInterval	DateTimeWindow
	resultCode	string
	transactions	ListOfTransactionLogs
GetDRASClientAlarms		
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfIDs
	timeInterval	DateTimeWindow
	alarms	ListOfDRASClientAlarms
GetParticipantFeedback		
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfIDs
	programNames	ListOfIDs
	timeInterval	DateTimeWindow
	name	string
	feedback	ListOfFeedback
GetGroups		
	no part specified	
	groups	ListOfIDs
SetEventConstraint		
	eventID	string
	constraint	ProgramConstraint
	returnValue	string
GetEventConstraint		
	eventID	string
	constraint	ProgramConstraint

Figure A.14: Utility operator/Iso interface as defined by OpenADR – continued.

ParticipantOperator		
GetParticipantAccounts		
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	participantAccounts	ListOfParticipantAccounts
ModifyParticipantAccounts		
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	participantAccount	ParticipantAccount
	operationSuccessStatus	string
CreateDRASClient		
	DRASClient	DRASClient
	operationSuccessStatus	string
ModifyDRASClient		
	DRASClient	DRASClient
	operationSuccessStatus	string
DeleteDRASClient		
	DRASClientID	string
	operationSuccessStatus	string
GetDRASClientInfo		
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	DRASClients	ListOfDRASClients
GetParticipantProgramConstraints		
	participantIDs	ListOfParticipantAccountIDs
	programName	string
	participantGroup	string
	programConstraints	ListOfProgramConstraints
SetParticipantProgramConstraints		
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	programConstraint	ProgramConstraint
	operationSuccessStatus	string
DeleteParticipantProgramConstraints		
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	programName	string
	operationSuccessStatus	string
GetDRASClientProgramConstraints		
	drasClientIDs	ListOfIDs
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	programConstraints	ListOfProgramConstraints
SetDRASClientProgramConstraints		
	drasClientIDs	ListOfIDs
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	programConstraint	ProgramConstraint
	operationSuccessStatus	string

Figure A.15: Participant operator interface as defined by OpenADR.

DeleteDRASClientProgramConstraints		
	drasClientIDs	ListOfIDs
	participantIDs	ListOfParticipantAccountIDs
	participantGroup	string
	programName	string
	operationSuccessStatus	string
GetProgramInformation		
	programName	string
	drasClientID	string
	programConstraints	ProgramConstraint
	eventInfo	ListOfEventInfoType
CreateResponseSchedule		
	responseSchedule	ResponseSchedule
	operationSuccessStatus	string
DeleteResponseSchedule		
	programName	string
	drasClientID	string
	operationSuccessStatus	string
GetResponseSchedule		
	programName	string
	drasClientID	string
	responseSchedule	ResponseSchedule
SetDREventFeedback		
	feedBack	FeedBack
	operationSuccessStatus	string
SubmitStandingBid		
	bid	Bid
	operationSuccessStatus	string
GetStandingBid		
	participantID	string
	programName	string
	bid	Bid
DeleteStandingBid		
	participantID	string
	programName	string
	operationSuccessStatus	string
SubmitBid		
	bid	Bid
	operationSuccessStatus	string
GetBid		
	eventID	string
	participantID	string
	bid	Bid
CreateOptOutState		
	optOutState	OptOutState
	operationSuccessStatus	string
DeleteOptOutState		
	optOutStateID	string
	operationSuccessStatus	string
GetOptOutStates		
	participantID	string
	programName	string
	drasClientID	string
	optOutID	string
	eventID	string
	optOutStates	ListOfOptOutStates

Figure A.15: Participant operator interface as defined by OpenADR – continued.

GetDRASClientCommsStatus		
	participants	ParticipantList
	programNames	ListOfProgramNames
	DRASClientStatus	ListOfCommsStatus
GetDRASTransactions		
	userNames	ListOfIDs
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfIDs
	methodName	string
	timeInterval	DateTimeWindow
	resultCode	string
	transactions	ListOfTransactionLogs
GetDRASClientAlarms		
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfIDs
	timeInterval	DateTimeWindow
	alarms	ListOfDRASClientAlarms
GetDREventFeedback		
	DRASClientIDs	ListOfIDs
	participantIDs	ListOfIDs
	programNames	ListOfIDs
	timeInterval	DateTimeWindow
	name	string
	feedback	ListOfFeedback
SetTestMode		
	drasClientID	string
	mode	string
	operationSuccessStatus	string
SetTestModeState		
	drasClientID	string
	opertaionModeValue	string
	eventStatus	string
	operationSuccessStatus	string
GetTestModeState		
	drasClientID	string
	GetTestModeStateResponse	string

Figure A.15: Participant operator interface as defined by OpenADR – continued.

B

RESULTS

Listing B.1: Mas quote message to the balancer.

```
<pyld:eiCreateQuote xmlns:pyld="http://docs.oasis-open.org/ns/energyinterop/201110/payloads">
<pyld:requestID>cab6f700-b6cd-4291-ac83-251037ae0764</pyld:requestID>
<ei:publisherPartyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">dd1e82d5-a9a7-475d-bd4e-a8947baa3575</ei:publisherPartyID>
<ei:subscriberPartyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">56f9854a-590f-4f5c-a6e8-aa01002d7dd7</ei:subscriberPartyID>
<ei:eiQuote xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110">
<ei:quoteID>317ba523-14ca-4738-a3f3-d7e8e4fa3ade</ei:quoteID>
<emix:product xmlns:emix="http://docs.oasis-open.org/ns/emix/2011/06">
<xcal:components xmlns:xcal="urn:ietf:params:xml:ns:icalendar-2.0">
<xcal:interval>
<xcal:properties>
<xcal:uid>
<xcal:text>17518a04-944c-4822-a1ce-562ebc8ac699</xcal:text>
</xcal:uid>
<xcal:duration>
<xcal:duration>PT15M</xcal:duration>
</xcal:duration>
<xcal:dtstart>
<xcal:parameters>
<xcal:tzid>
<xcal:text>Netherlands/Amsterdam</xcal:text>
</xcal:tzid>
</xcal:parameters>
<xcal:date-time>2014-07-28T09:00:00</xcal:date-time>
</xcal:dtstart>
<xcal:x-wsCalendar-attach>
<power:fullRequirementsPower xmlns:power="http://docs.oasis-open.org/ns/emix/2011/06/power">
<power:productType>energy</power:productType>
<power:powerReal>
<power:itemDescription>RealPower</power:itemDescription>
<power:itemUnits>W</power:itemUnits>
<scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/emix/2011/06/siscale">none</scale:siScaleCode>
<power:powerAttributes>
```

```
33 <power:hertz>50.0</power:hertz>
34 <power:voltage>230.0</power:voltage>
35 <power:ac>true</power:ac>
36 </power:powerAttributes>
37 </power:powerReal>
38 <power:maximumPower>310.5092862553328</power:maximumPower>
39 </power:fullRequirementsPower>
40 </xcal:x-wsCalendar-attach>
41 </xcal:properties>
42 </xcal:interval>
43 </xcal:components>
44 <emix:uid>70e06944-30dd-4e38-87e1-30a1cc00b2a7</emix:uid>
45 <emix:transactiveState>publication</emix:transactiveState>
46 <emix:marketContext>urn:rug:mas</emix:marketContext>
47 <emix:side>buy</emix:side>
48 </emix:product>
49 </ei:eiQuote>
50 </pyld:eiCreateQuote>
```

Listing B.2: Mas tender message.

```
1 <pyld:eiCreateTender xmlns:pyld="http://docs.oasis-open.org/ns/
    energyinterop/201110/payloads">
2 <pyld:requestID>d4e5da3d-c27f-4838-a1ba-b0cb27caafc6</
    pyld:requestID>
3 <ei:partyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop
    /201110">dd1e82d5-a9a7-475d-bd4e-a8947baa3575</ei:partyID>
4 <ei:counterPartyID xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</
    ei:counterPartyID>
5 <ei:eiTender xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">
6 <ei:tenderID>689c5134-9aba-47b0-88a7-2438614610e8</ei:tenderID>
7 <emix:product xmlns:emix="http://docs.oasis-open.org/ns/emix
    /2011/06">
8 <xcal:components xmlns:xcal="urn:ietf:params:xml:ns:icalendar-2.0
    ">
9 <xcal:interval>
10 <xcal:properties>
11 <xcal:uid>
12 <xcal:text>7ad96996-c472-4117-899e-66f59b1f3d28</xcal:text>
13 </xcal:uid>
14 <xcal:duration>
15 <xcal:duration>PT15M</xcal:duration>
16 </xcal:duration>
17 <xcal:dtstart>
18 <xcal:parameters>
19 <xcal:tzid>
20 <xcal:text>Netherlands/Amsterdam</xcal:text>
21 </xcal:tzid>
22 </xcal:parameters>
23 <xcal:date-time>2014-07-28T09:00:00</xcal:date-time>
```

```
24 </xcal:dtstart>
25 <xcal:x-wsCalendar-attach>
26 <power:fullRequirementsPower xmlns:power="http://docs.oasis-open.
       org/ns/emix/2011/06/power">
27 <power:productType>energy</power:productType>
28 <power:unitEnergyPrice>
29 <emix:price>
30 <emix:value>9.00267499080293E-5</emix:value>
31 </emix:price>
32 <power:energyReal>
33 <power:itemDescription>RealEnergy</power:itemDescription>
34 <power:itemUnits>Wh</power:itemUnits>
35 <scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/
       emix/2011/06/siscale">none</scale:siScaleCode>
36 </power:energyReal>
37 </power:unitEnergyPrice>
38 <power:powerReal>
39 <power:itemDescription>RealPower</power:itemDescription>
40 <power:itemUnits>W</power:itemUnits>
41 <scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/
       emix/2011/06/siscale">none</scale:siScaleCode>
42 <power:powerAttributes>
43 <power:hertz>50.0</power:hertz>
44 <power:voltage>230.0</power:voltage>
45 <power:ac>true</power:ac>
46 </power:powerAttributes>
47 </power:powerReal>
48 <power:charges>
49 <mas:masTSOCharge xmlns:mas="urn:rug:mas">
50 <emix:price>
51 <emix:value>0.0085</emix:value>
52 </emix:price>
53 </mas:masTSOCharge>
54 </power:charges>
55 <power:maximumPower>310.5092862553328</power:maximumPower>
56 </power:fullRequirementsPower>
57 </xcal:x-wsCalendar-attach>
58 </xcal:properties>
59 </xcal:interval>
60 </xcal:components>
61 <emix:uid>edcd2606-5e93-4c3e-bd6e-f4390a0f1943</emix:uid>
62 <emix:transactiveState>tender</emix:transactiveState>
63 <emix:marketContext>urn:rug:mas</emix:marketContext>
64 <emix:side>buy</emix:side>
65 </emix:product>
66 </ei:eiTender>
67 </pyld:eiCreateTender>
```

Listing B.3: Mas tender reply message.

```
1 <pyld:eiCreatedTender xmlns:pyld="http://docs.oasis-open.org/ns/
       energyinterop/201110/payloads">
```

```
2 <ei:eiResponse xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">
3 <ei:responseCode>502</ei:responseCode>
4 <ei:refID>d4e5da3d-c27f-4838-a1ba-b0cb27caafc6</ei:refID>
5 </ei:eiResponse>
6 <ei:responses xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110"/>
7 <ei:partyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop
    /201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</ei:partyID>
8 </pyld:eiCreatedTender>
```

Listing B.4: Mas transaction message.

```
1 <pyld:eiCreateTransaction xmlns:pyld="http://docs.oasis-open.org/
    ns/energyinterop/201110/payloads">
2 <pyld:requestID>77488aef-bcb8-4e54-928c-cdee770835d8</
    pyld:requestID>
3 <ei:partyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop
    /201110">dd1e82d5-a9a7-475d-bd4e-a8947baa3575</ei:partyID>
4 <ei:counterPartyID xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</
    ei:counterPartyID>
5 <ei:eiTransaction xmlns:ei="http://docs.oasis-open.org/ns/
    energyinterop/201110">
6 <ei:transactionID>7eef25fb-3f1b-4360-8173-4be387f38825</
    ei:transactionID>
7 <emix:product xmlns:emix="http://docs.oasis-open.org/ns/emix
    /2011/06">
8 <xcal:components xmlns:xcal="urn:ietf:params:xml:ns:icalendar-2.0
    ">
9 <xcal:interval>
10 <xcal:properties>
11 <xcal:uid>
12 <xcal:text>abcc8ffb-c40d-4665-9f6b-409fc095b3d5</xcal:text>
13 </xcal:uid>
14 <xcal:duration>
15 <xcal:duration>PT15M</xcal:duration>
16 </xcal:duration>
17 <xcal:dtstart>
18 <xcal:parameters>
19 <xcal:tzid>
20 <xcal:text>Netherlands/Amsterdam</xcal:text>
21 </xcal:tzid>
22 </xcal:parameters>
23 <xcal:date-time>2014-07-28T09:00:00</xcal:date-time>
24 </xcal:dtstart>
25 <xcal:x-wsCalendar-attach>
26 <power:fullRequirementsPower xmlns:power="http://docs.oasis-open.
    org/ns/emix/2011/06/power">
27 <power:productType>energy</power:productType>
28 <power:unitEnergyPrice>
29 <emix:price>
```

```
30 <emix:value>1.0057190380491614E-4</emix:value>
31 </emix:price>
32 <power:energyReal>
33 <power:itemDescription>RealEnergy</power:itemDescription>
34 <power:itemUnits>Wh</power:itemUnits>
35 <scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/
       emix/2011/06/siscale">none</scale:siScaleCode>
36 </power:energyReal>
37 </power:unitEnergyPrice>
38 <power:powerReal>
39 <power:itemDescription>RealPower</power:itemDescription>
40 <power:itemUnits>W</power:itemUnits>
41 <scale:siScaleCode xmlns:scale="http://docs.oasis-open.org/ns/
       emix/2011/06/siscale">none</scale:siScaleCode>
42 <power:powerAttributes>
43 <power:hertz>50.0</power:hertz>
44 <power:voltage>230.0</power:voltage>
45 <power:ac>true</power:ac>
46 </power:powerAttributes>
47 </power:powerReal>
48 <power:charges>
49 <mas:masTSOCharge xmlns:mas="urn:rug:mas">
50 <emix:price>
51 <emix:value>0.0085</emix:value>
52 </emix:price>
53 </mas:masTSOCharge>
54 </power:charges>
55 <power:maximumPower>310.5092862553328</power:maximumPower>
56 </power:fullRequirementsPower>
57 </xcal:x-wsCalendar-attach>
58 </xcal:properties>
59 </xcal:interval>
60 </xcal:components>
61 <emix:uid>9ca9803b-b550-42ee-a5ca-eb2a93e58308</emix:uid>
62 <emix:transactiveState>transaction</emix:transactiveState>
63 <emix:marketContext>urn:rug:mas</emix:marketContext>
64 <emix:side>buy</emix:side>
65 </emix:product>
66 </ei:eiTransaction>
67 </pyld:eiCreateTransaction>
```


Listing B.5: Mas transaction reply message.


```
1 <pyld:eiCreatedTransaction xmlns:pyld="http://docs.oasis-open.org
      /ns/energyinterop/201110/payloads">
2 <ei:counterPartyID xmlns:ei="http://docs.oasis-open.org/ns/
      energyinterop/201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</
      ei:counterPartyID>
3 <ei:partyID xmlns:ei="http://docs.oasis-open.org/ns/energyinterop
      /201110">2af9a0ac-1501-4bf8-93e5-82752deaf32c</ei:partyID>
4 <ei:eiResponse xmlns:ei="http://docs.oasis-open.org/ns/
      energyinterop/201110">
```

```
5 <ei:responseCode>200</ei:responseCode>
6 <ei:refID>77488aef-bcb8-4e54-928c-cdee770835d8</ei:refID>
7 </ei:eiResponse>
8 </pyld:eiCreatedTransaction>
```