The eBADGE Data Model Report – Final Version

Deliverable report

Document Information

Programme	**FP7 – Cooperation / ICT**
Project acronym	eBADGE
Project Full Title	Development of Novel ICT tools for integrated Balancing Market Enabling Aggregated Demand Response and Distributed Generation Capacity
Grant agreement number	318050
Number of the Deliverable	D3.1.3
WP/Task related	WP3 / T3.1
Type (distribution level)[1]	PU
Due date of deliverable	30.09.2015 (Month 36)
Date of delivery	30.09.2015
Status and Version	Final, v1.0
Number of pages	65 pages
Document Responsible	Marjan Šterk - XLAB
Author(s)	Marjan Šterk, Staš Strozak - XLAB, Radovan Sernec - TS, Mitja Kolenc, Peter Nemček, Andraž Andolšek - CG, Gianluigi Migliavacca, Alessandro Zani - RSE, Darko Kramar - ELES, Andraž Šavli - Borzen, Boris Turha - EL, Daniel Burnier de Castro - AIT, Ingo Sauer - EUDT
Reviewers	Tomaž Lovrenčič - TS

1

PU	Public
RP	Restricted to other programme participants (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Issue record

Version	Date	Author(s)	Notes	Status
1.0	30.09.2015	Marjan Šterk, XLAB	Integrated comments from internal review	Final
0.8	23.09.2015	Marjan Šterk, XLAB	Expanded message explanations in Section 7	Draft
0.7	14.09.2015	Andraž Andolšek, CG	Improvements throughout, esp. Section 7	Draft
0.6	11.09.2015	Marjan Šterk, XLAB	Updated glossary, summary, conclusions	Draft
0.5	10.09.2015	Marjan Šterk, XLAB	Added field comparisons to OpenADR	Draft
0.4	09.09.2015	Andraž Andolšek, CG	Added a schematic comparison to OpenADR	Draft
0.3	09.09.2015	Marjan Šterk, XLAB	Added comparison to OpenADR throughout Section 5	Draft
0.2	04.09.2015	Marjan Šterk, XLAB	Updated all the changed messages in Section 5	Draft
0.1	11.08.2015	Marjan Šterk, XLAB	Renamed D3.1.2 to D3.1.3, updated administrative parts	Draft

NOTICE
The research leading to the results presented in the document has received funding from the European Community's Seventh Framework Programme under grant agreement n. 318050.

Table of Contents

Table of Figures

Table of Tables

Table of Acronyms

Acronym	Meaning
ACER	Agency for the Cooperation of Energy Regulators
ACSI	Abstract Communication Service Interface
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BCHP	Block-type Combined Heat and Power plants
BSP	Balancing Service Provider
CIM	Common Information Model
CSV	Comma-separated values, a text-based tabular data format
DG	Distributed Generation
DR	Demand response
DSO	Distribution System Operator
ECAR	Electric car
ENTSO-E	European Network of Transmission System Operators for Electricity
EPRI	Electric Power Research Institute
EVPP	EDISON Virtual Power Plant
GME	Gestore Mercati Energetici, the Italian market operator
GOOSE	Generic Object Oriented Substation Events
GUI	Graphical user interface
HEH	Home Energy Hub
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
HVAC	heating, ventilating, and air conditioning
HP	Heat Pumps
ICT	Information and Communication Technologies
IEC	International Electrotechnical Commission
IED	Intelligent Electronic Device
IP	Internet Protocol; also used to denote the addresses in IP-based networks
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LED	light emitting diode
LD	Logical Device
LN	Logical Node
MAC	Media Access Control address, a unique address assigned to each network interface
MMS	Manufacturing Message Specification
NTP	Network Time Protocol
NYSE	New York Stock Exchange
NYSE MKT	NYSE MKT LLC, formerly known as the American Stock Exchange (AMEX)
OASIS	Organization for the Advancement of Structured Information Standards
OLE	Object Linking and Embedding
OPC	Open Platform Communications (initially OLE for Process Control)
OPC UA	OPC Unified Architecture
OpenADR	Open Automated Demand-Response

OSGP	Open Smart Grid Protocol
PLC	Programmable Logic Controller
PV	Photovoltaics
RES	Renewable Energy Source
REST	Representational State Transfer
RTU	Remote terminal Unit
SCL	Substation Configuration Language
SOAP	Simple Object Access Protocol
TSO	Transmission System Operator
URL	Universal Resource Locator, an “address” of a (typically) HTTP resource
UTF-8	UCS Transformation Format – 8-bit, a variable-width encoding of characters and strings
UTP	Unshielded Twisted Pair, the most common type of computer network cable
UUID	Universally Unique Identifier
VEN	Virtual End Node
VPP	Virtual Power Plant
VTN	Virtual Top Node
W3C	World Wide Web Consortium
WP	Work Package
XML	Extensible Mark-up Language
XMPP	Extensible Messaging and Presence Protocol

Glossary

Term	Meaning
.NET	a software framework and run-time environment for multiple programming languages (e.g. C#, Visual Basic.NET), developed by Microsoft
C	a low-level programming language
C++	an object-oriented programming language based on C
data attribute	a type of data that represents a specific feature or data type (e.g. voltage, etc.)
data class	represents meaningful information inside the nodes and are declared recursively.
data model	a specification of data types (objects) and the fields they contain; in this document refers to the eBADGE data model
gateway	synonymous to proxy
Protocol Buffers	a method for serializing structured data, developed by Google
gzip	a file compression/decompression tool
Java	a programming language and corresponding run-time environment developed by Oracle (originally by Sun Microsystems)
JavaDoc	an automatic documentation generator, creates HTML documentation from Java source
JavaScript	an interpreted programming language, mostly used in web applications
logical device	an entity that contains one or more logical nodes
logical node	represents a function, measurement or a visualization of a real object like a circuit breaker, diesel generator, battery, etc.
market model	a mathematical model of balancing energy market, being developed in eBADGE WP2
market simulator	a software simulator of the market model, being developed in eBADGE WP2
message acknowledgement	a return message that acknowledges that the previous message was received
message bus	a generic name for a messaging proxy, the required server and client software, typically also capable of multicast (one-to-many) communication
message header	the part of a message that contains the message metadata and typically precedes the payload
message payload	the part of a message that contains the actual data and typically follows the header
messaging proxy	an intermediary entity (typically, a server) through which one entity (end-point) can communicate with another without requiring a direct connection
multi-cast	a communication pattern with multiple recipients in a single transmission from the source; one-to-many
namespace	a container (software concept) for a set of identifiers (for example, field names), typically used to prevent naming conflicts
payload encoding	method of serializing structured data into the message payload
proxy	an intermediary entity through which one entity (end-point) can reach another entity in the absence of direct access
Python	a high-level programming language
semantic model	a formal description of a certain domain's semantics
semantics	the meaning of something (messages, commands, fields etc.)
serialization/deserialization	the process of transforming a software object to/from a stream of bytes
super-TSO	the central operator of the international balancing market that maintains the common merit order list
stanza	a message of any kind in XMPP protocol
X10	a protocol for communication among home automation devices
XML Schema	a formal description of the allowed syntax of XML files, also itself an XML file
ZigBee	a suite of high-level communication protocols for low-power, close-range wireless networks

eBADGE Project

The 3rd Energy Package clearly boosts the development of an Integrated European balancing mechanism. In this context, ACER has in 2011 started the development of the Framework Guidelines on Electricity Balancing. It is expected from the ACER statements that Demand Response will play significant role in the future integrated balancing market allowing Virtual Power Plants, comprising Demand Response and Distributed Generation resources to compete on equal ground.

The overall objective of the eBadge project is to propose an optimal pan-European Intelligent Balancing mechanism also able to integrate Virtual Power Plant Systems by means of an integrated communication infrastructure that can assist in the management of the electricity Transmission and Distribution grids in an optimized, controlled and secure manner.

In order to achieve the above overall objective the eBadge project will have four objectives focusing on:

1. Developing the components: simulation and modelling tool; message bus; VPP data analysis, optimisation and control strategies; home energy cloud; and business models between Energy, ICT and Residential Consumers sector;
2. Integrating the above components into a single system;
3. Validating these in lab and field trials;
4. Evaluating its impact.

Project Partners

Telekom Slovenije d.d. – Slovenia
cyberGRID GmbH – Austria
Ricerca sul sistema energetico – RSE Spa - Italy
XLAB Razvoj programske opreme in svetovanje d.o.o. – Slovenia
ELES d.o.o. – Slovenia
Austrian Power Grid – Austria
Borzen, Organizator trga z električno energijo, d.o.o. – Slovenia
Elektro Ljubljana, Podjetje za distribucijo električne energije d.d. – Slovenia
Technische Universitaet Wien – Austria
Austrian Institute of Technology GmbH – Austria
EUDT Energie- u. Umweltdaten Treuhand GmbH - Austria
SAP AG – Germany
Vaasaett Ltd Ab Oy – Finland

Project webpage

http://www.ebadge-fp7.eu/

Executive Summary

This document defines the final, third version of the eBADGE data model and encoding used for the communication between all the stakeholders in the pilot project, such as a home energy hub sending usage data to its VPP operator and receiving curtailment commands, or a TSO collecting balancing energy bids and sending dispatching signals when needed. Throughout the document this data model is compared to the OpenADR 2.0 standard.

We first list some general requirements of our pilot and then evaluate the suitability of the existing standards in the field, most notably OpenADR 2.0. We explain our original decision for a "green field" solution focused on achieving the eBADGE project objectives early on and the process of parallel integration of OpenADR 2.0 in the final year of the project, following the increasing uptake of this standard.

Some general conventions used in our solution are described next, such as avoidance of binary encodings for reasons of human-readability, ease of development, extensibility, and maintainability, the use of JSON, the encoding of standard data types, and extensibility conventions. All these choices are compared to the XML-based OpenADR 2.0, which makes use of many XML's advantages at the price of the messages being at least an order of magnitude longer.

A full, formal definition of our data model and encoding is given, containing the list of all the message types required to cover the currently foreseen eBADGE use cases, with complete details and JSON examples. This is accompanied by sequence diagrams for the more complex communication patterns. Also of note is that the related eBADGE public deliverable D3.2.3, "The eBADGE Message Bus", contains a reference open source implementation of the eBadge data model.

Finally, a semantic comparison of eBADGE data model to OpenADR 2.0 is given in the form of typical message sequence of the latter and their counterparts in the former. Fields of messages, their meaning, and gaps of each are given. While both of them support certain functionalities that the other does not, the most important semantical difference is that the eBADGE data model is much more clear and self-contained, not requiring the communicating parties to agree on any additional specifications, although at the cost of being less flexible than OpenADR.

1. Introduction

The aim of this deliverable and the corresponding project task is to analyse the existing ICT interfaces and standards used in TSOs (ENTSO-E), VPPs, DSOs, Resources (loads, DG, RES), Electricity and Balancing Markets. As each of these stakeholders and its ICT components has its own set of interfaces, the data attributes of all of them have to be harvested and a common data model and communication protocol defined with all the relevant attributes for information exchange between the entities. This is an updated version of last year's deliverable [1].

The originally proposed title of this document was "The eBADGE Data Standard Report – Final Version". However, the choice of that title was not appropriate because formal acceptance by a standardization body was not planned. Rather, the purpose of this data model is solely to enable and unify communication between the entities in the eBADGE pilot; that is, the data model itself is a pilot intended to show one possible approach to this communication.

This deliverable presents the third, final version of the proposed open data model. The model defines the types of messages, their contents, encoding, meaning, and the typical sequences of messages in certain non-trivial scenarios. Since the first version we have unified the approach to reporting load, generation, and detailed measurements such as voltage using general *report* type of messages. We added the capability for third parties to access measurements for multiple end-users stored in a central location. Changes from the second to final version are smaller, the most important being the option to combine signals from all the appliances and devices measured by a single HEH in a single *report* message.

The major change to this deliverable is the detailed comparison with the data model used in the OpenADR 2.0 standard and the description of how OpenADR was integrated into the eBADGE pilot alongside our own data model.

1.1 Approach

This first version of the eBADGE data model was derived by first defining the typical demand-response and balancing energy market scenarios and analysing the communication patterns. Next, existing standards were reviewed to find out whether eBADGE can benefit from using them, either as-is, or by defining our extensions. Unfortunately, as is argued in this report, this turned out not to be the case at the time of our first evaluation; nevertheless, we looked for the features and solutions implemented in these standards that could be re-used when defining our own data model.

All the required message types and contents were extracted from the defined scenarios. During this process, inconsistencies, corner cases and additional possibilities arose, which required refinement of the scenarios. This iterative process resulted in the full message types specification as given in this document. The scenarios themselves are briefly summarized at the beginning of the respective message specifications (Sections 5 and 6) and the more complex ones are provided in detail in the form of sequence diagrams.

In parallel, the non-functional requirements such as message volume, reliability, security, and expandability/scalability/extensibility were defined. Although these are mostly relevant for the message bus implementation [2], they were followed when defining some general approaches in the data model, such as how the message contents are encoded and how the message acknowledgements are formatted.

During the last year of the project the support for the OpenADR 2.0 standard has been added into the eBADGE pilot. In order to be able compare it to our original approach it was deployed in parallel, rather than by implementing some sort of a protocol gateway. For each scenario and its corresponding workflow of eBADGE data model messages its OpenADR counterpart was defined and implemented in software. Furthermore, the OpenADR functionalities not used by any such scenarios were enumerated and existence of their equivalents in the eBADGE data model was checked, thereby producing also the theoretical comparison provided in this report.

1.2 Relation to the Rest of the Project

This deliverable, together with the eBADGE message bus [2] developed in the same work package, provides the data exchange software for the eBADGE pilot. The requirements for the data model presented here are influenced by and result from the market model and simulator developed in WP2 (data exchange in international balancing energy market), the business use cases developed in WP4 (required data exchange with demand response resources and users) and the home energy hub and cloud prototype specifications, also from WP4 (e.g., performance and other limitations, network security model).

In the final year of the project we have also performed detailed tests of the reliability, performance, and security of the message bus, which is highly interdependent with the data model itself. The results are published in the public deliverable D3.3 [3]. Furthermore, that deliverable also contains detailed comparison of the performance, reliability, and security of OpenADR to our original approach.

1.3 Outline of This Report

The next section briefly summarizes the requirements and the corresponding division of the model into two communication levels. Section 3 analyses existing standards related to eBADGE communication. Section 4 describes the chosen approach in general, common to all message types, and compares our choices to the ones taken by OpenADR. Sections 5 and 6 contain detailed specifications of all the message types and their contents. Additionally, section 6 illustrates some of the typical market scenarios using sequence diagrams, which help understanding the communication patterns. Section 7 provides typical message workflows in OpenADR, their analogues in eBADGE data model, and analyses message contents of both. The last section provides the conclusions.

2. Requirements

The required communication in the eBADGE pilot can be divided into multiple communication domains where different entities communicate:

- energy resources, energy storage, smart meters, home energy hubs (gateways), VPPs, microgrids, DSOs, energy providers exchange messages containing home energy usage profiles, VPP activation commands and similar,
- in certain business models, where one stakeholder owns and operates the HEHs and provides to other stakeholders services such as measurements and the capability to activate demand response resources, this is a separate inter-stakeholder communication domain,
- VPPs and other BSPs (balancing service providers), e.g. traditional generators, send bids to their TSO, who send activation commands back,
- in an international balancing market based on the TSO-to-TSO model, TSOs forward bids to a common merit order list and coordinate balancing energy allocation from this list.

No messages pass the borders of these communication domains (for example, the home energy hub never communicates directly with a TSO). The first of the four domains has the largest message volume while the last one must be the most resistant to attacks and failures. The first two domains are semantically related, as are the last two. Consequently, we will refer in this document to the first two domains as the *home energy hub (HEH) communication level* and to the remaining two domains as the *market level*.

Ideally, all levels can be covered using the same technology. However, the actual communication channels must be isolated so that a potential attack on a home energy hub does not pose any threat on the market level.

In the eBADGE pilot the maintenance of the common merit order list and the coordination of balancing energy allocations from this list is simulated through a central market operator. The role of the latter also includes the process of selecting the optimal bids in any given moment not only with regard to its price but also its dispatchability to the target area given the capacities and flows in the network. To encompass this complex role, we refer to this central market operator as a *super-TSO*.

The main purpose of the market level and the simulator is to demonstrate the regulatory and economic feasibility of an integrated balancing market and the increased social welfare that such a market would provide. The implementation challenges beyond this pilot are mostly regulatory and information technology is secondary. On the other hand, the HEH level of the pilot also strives to demonstrate the technical feasibility of using a large number of households or small businesses as demand response resources. It is therefore conceivable that commercial projects following this pilot will strongly consider re-using the HEH-level technology developed here, while for the market level this is far less likely. Consequently, when one technical choice is optimal for the HEH level and another for the market level, we will most often choose the former over the latter.

The full list of required message types and contents that resulted from the requirements collection process was used directly to define this data model. For example, a scenario was envisaged where the VPP subscribes to periodic load reports from the home energy hub, thus the message types *get_periodic_report* and *report* were defined. As the resulting model is provided in this document, the full requirements would just duplicate the information and are thus not listed here; rather, an overview is given in this section.

2.1 Home Energy Hub Level Requirements

The basic business idea is a VPP aggregating the demand-response potential of many end-users (through HEHs) and selling it as balancing energy on the market. More elaborate scenarios were developed in WP4 [4] where, for example, telecommunication companies may offer data acquisition from HEHs and data storage, and DSOs and energy providers also take part in end-user-level demand-response. Thus, this level of the data model defines how the metering data is communicated from the HEH to the VPP or other relevant entity, how the activation commands are communicated back, and various status inquiries/reports.

We estimate that the demand-response potential of a single home is in the order of 1 kW at times when it is available for activation. Note that the homes with electrical heating/cooling (heat pumps, air conditioning, water boilers etc) or electrical vehicles may be available for activation a significant portion of the time while the homes that can only offer appliances such as washing machines will be unavailable most of the time.

Thousands of such currently available homes must be aggregated to provide megawatts to the balancing market. Thus, the data model must be scalable to at least thousands, preferably tens of thousands HEHs per VPP. Since two VPPs can (and most often should) use completely isolated channels to communicate with their HEHs, the total number of VPPs is not the limiting factor for the HEH-level communications.

The HEH has to be able to send the following reports:

- power usage and generation profile for the whole building,
- power usage and generation profile for each device (load, generator or storage) that is measured individually,
- anomalous events, such as power outages, overvoltages,
- detailed, high-resolution profile (U, I, P, Q, S, harmonics; or a subset of those if not all are being measured),
- HEH status,
- capabilities of the HEH and of individual devices.

These reports are sent either on request or periodically, where the period and other parameters must be remotely configurable.

The VPP must be able to control the HEH and the appliances through the following:

- activation commands for individual devices (i.e. increase/decrease load or generation on the specified device),
- activation commands for the whole household/business, where it is up to the HEH and/or the user how to achieve it,
- tariff (pricing) updates through which the VPP can motivate/boost/incentivize demand response.

The data model must allow the HEH to reject or modify activations. However, depending on the contract between e.g. the home user and the VPP, this may only be allowed in certain limited cases.

The data model must also allow for the business case where an intermediary entity (for example, a telco provider) acts as HEH owner, operator, and measurement data curator. Such a HEH operator then provides services such as access to measurement data, activation capabilities etc to VPPs and other parties.

2.2 Market Level Requirements

The market-level of the data model will be used to collect bids from BSPs at the TSOs and to send activation commands to the BSPs. On the international market, the model must also cover sharing bids of each TSO with the super-TSO and coordinating the cross-border balancing mechanism. Scalability to tens of TSOs and thousands of BSPs must be ensured.

This part of the model is not intended to replace the existing protocols used to allocate cross-border capacities, control the transmission grid, inform about failures in the energy grid etc.; rather, it should contain just the messages necessary for the implementation of the eBADGE pilot. In particular, the market simulator developed in WP2 will be the pilot entity representing the central market operator (so called super-TSO) and the pilot VPPs will act as the BSPs communicating with the simulator.

The BSPs must be able to, regardless of whether the balancing energy market is national or cross-border:

- send bids to their TSO,
- receive activation commands from their TSO and acknowledge or reject them.

In an international market operated by a super-TSO, the latter must also be able to:

- receive forwarded bids from all TSOs in order to build a common merit order list,
- receive a balancing request from a TSO that detects an imbalance in its area, i.e. a requests to activate bid(s) from the common merit order list,
- send a bid activation order to the TSO that the bid came from, who then forwards it to the BSP,
- inform the imbalanced TSO whether their request was granted or not,
- it must be possible to express imbalance netting.

How the super-TSO is informed about the network state (cross-border capacities, flows etc.) is beyond the scope of this data model. In the eBADGE pilot the network state will be simulated by the market simulator while in a production environment existing communication protocols would most probably be used to exchange network state data.

For an illustration please refer to the sequence diagrams in Sections 6.2 and 6.3.

3. Analysis of Existing Standards

Currently, the data standards related to the HEH level communications and those related to the market level are mostly different. The home level only started evolving with the Smart Grid initiatives, thus the standards are relatively recent and in line with modern software patterns and technologies. On the other hand, the market level is currently a mix of various legacy and more modern protocols, many of which are defined by individual software/system vendors rather than a standardization consortium.

The original standards analysis was performed in the Spring of 2013. The conclusion at that time was that no existing standard fits our requirements perfectly, thus any of them would have to be extended to the point of losing direct compatibility. Additionally, many of the existing standards are quite complex to implement, and it makes little sense to spend development time on implementation of large standards without the compatibility benefits. Finally, many smart grid approaches, data standards and middlewares were competing and it was far from certain which ones will prevail [5, 6], thus it was not advisable for this pilot project to invest a significant effort into adoption of standards whose future is uncertain unless there are other benefits.

Afterwards, within the last two years, two standards gained significantly more traction. OpenADR 2.0 is increasingly popular in the USA and IEC 61850 in Europe. Both advertise publicly available software implementations and OpenADR 2.0 also lists many certified compatible devices. Therefore, these two are described and evaluated in greater detail from the aspect of HEH-level communication in eBADGE, followed by our original evaluation of the others.

3.1 OpenADR 2.0

OpenADR 2.0 (Open Automated Demand-Response 2.0) originates from the USA and supersedes the original OpenADR. The latter focused on price-based demand-response, which is not the focus of eBADGE. In addition it covered utility-to-aggregator communication and bidding, which are not relevant to the HEH level of eBADGE [5, 7].

OpenADR 2.0 is much expanded and updated. It is increasingly popular in the USA, in particular in California, where the new building code demands that all new non-residential buildings have HVAC systems with demand-response capabilities [8]. This building code went into effect in January 2014 and one of the ways to meet its requirements is integrating an OpenADR-compatible HVAC control system.

The standard is available free of charge for evaluation and development purposes (registration required) from the web site [9] and needs to be tested against openADR Alliance test suit for production systems. In contrast to IEC 61850, it is a single, focused, self-contained specification that is not too complex to implement in newly designed ICT systems. OpenADR 2.0 (profile B) was also adopted by the IEC, thus it is also known as IEC PAS 62746-10-1.

3.1.1. Architecture of OpenADR 2.0

OpenADR 2.0 is an information exchange standard for fully automatic demand-response. It supports both price-based demand response, which is carried over from the original version and depends on the electricity consumers and generators to behave smartly when the prices are dynamic, as well as direct demand-response, which directly specifies the intended behaviour of the consumers and generators but usually permits them to opt out. Finally, it supports reporting of flexibility and measured signals, such as electricity consumption, as well as of various statuses and metadata.

The communication architecture follows a master-slave paradigm, where a Virtual Top Node (VTN) is a controlling entity communicating with one or more Virtual End Nodes (VENs). The Home Energy Hub corresponds to a VEN and a VPP controlling many HEHs would be a VTN. As shown in Figure 1, a single entity (aggregator in the case shown) can simultaneously be a VTN controlling multiple residential VENs and a VEN when participating in a market operated by the utility's VTN.

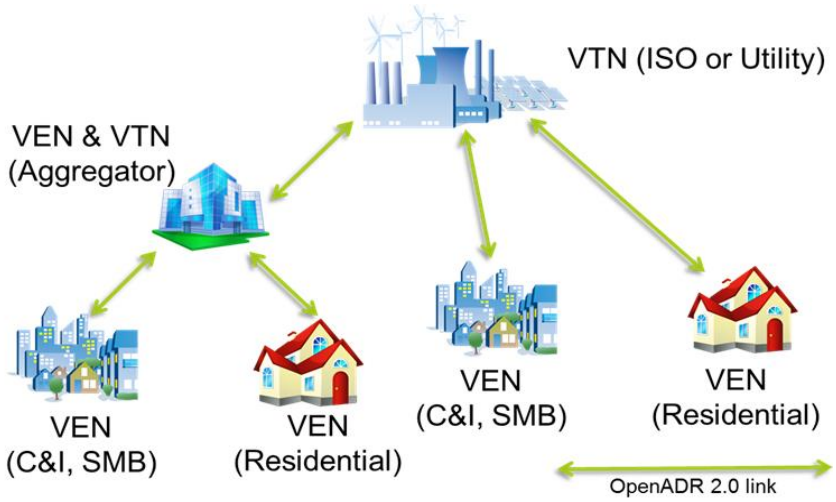

Figure 1: VEN and VTN roles in a large OpenADR 2.0 deployment[2]

The following main services and functions are defined in OpenADR 2.0:

- EiEvent, through which a VTN informs a VEN about the demand-response programmes. EiEvents cover tariff and pricing updates, controlling loads or storage resources etc,
- EiOpt, through which a VEN informs the VTN about when it is available for demand response,
- EiReport, through which a VEN sends periodic or one-time information on its state or a measured quantity,
- auxiliary services such as EiRegisterParty and OadrPoll.

All these services and functions are described through a rich data model based on XML.

OpenADR 2.0 defines three profiles differing in the required features and implementation complexity:

- Profile A is intended for relatively simple devices such as smart thermostats. A profile A VEN only supports a limited subset of the EiEvent service and standard security level.
- Profile B adds EiOpt, EiReport, and EiRegisterParty, which must all be supported in a profile B VEN. A profile B VEN must also support standard security level but can optionally also support the high security level.
- Profile B (energy reporting only) is a subset of profile B intended for smart meters and similar devices with no actual demand response capability. It leaves out the EiEvent and EiOpt services.
- Profile C will add aggregator-to-utility communications and other advanced operations, but the specification is not available yet.

A VTN must always support the full B profile, thus it must be compatible with all profile A and profile B VENs.

3.1.2. Communication Protocol

Although the communication mechanism used is a property orthogonal to the data model, it is important for use in eBADGE and is thus worth explaining here. The OpenADR 2.0 standard specifies two communication protocols, HTTPS and XMPP [9]. However, some implementations add additional possibilities, such as AMQP[3].

Both push and pull modes are in principle supported to, for example, communicate EiEvent messages. While XMPP naturally supports push regardless of firewall configuration, HTTPS push requires that the VEN have a public IP address and its OpenADR URL exposed publicly. The latter is often not possible or desired, thus it is envisaged that HTTPS deployments will primarily use the pull mode.

Profile A devices must support HTTPS transport in pull mode only. Profile B VENs can support HTTPS, XMPP, or both, and profile B VTNs must support both.

[2] Source: http://www.openadr.org/faq
[3] https://github.com/EnerNOC/oadr2-vtn-new

3.1.3. Existing Implementations

There is a formal certification process with a list of certified hardware devices and software artefacts listed in [10]. The certified hardware devices include various smart meters, thermostats, and DR management appliances. However, for most of those it is difficult to find out details about which version of which product is OpenADR-compatible and what functionalities it really offers. One of the more promising options is the Universal Devices ISY-994i series of home automation controllers [11], which is advertised as being able to act as a gateway between OpenADR demand-response programmes and smart home equipment such as X10 load controllers, ZigBee thermostats, etc.

A number of software implementations also exist, including open source ones. However, most of the implementations are stand-alone software libraries that have no actual devices to control and no “smartness”, thus they only implement the syntactical aspect of OpenADR[4,5,6,7]. This is rather trivial to do because the full XML schema is available anyway as part of the standard.

A software implementation that does stand out is the EPRI VTN[8], an open-source VTN implementation for which a corresponding VEN implementation is also available. They still do not actually do anything more than communicate; however, as they implement both ends, they are very useful for learning about OpenADR and for testing custom OpenADR software with one of them.

3.1.4. Adoption

We originally, in Spring 2013, decided against using OpenADR in our pilot, but its increasing popularity, ubiquity, and suitability for demand response prompted us to re-evaluate it. Although the existing eBADGE data model fully covers our requirements, to increase the value of the pilot we integrated OpenADR support as well.

We implemented OpenADR 2.0 as an alternative to the eBADGE data model and message bus. A subset of the deployed HEHs was then switched to use this alternative while the rest will remain using the data model specified in this deliverable. The coexistence allowed us to test in detail the functionality, performance, and security of both approaches.

The second option was to develop a gateway for message translation rather than replace our existing model entirely. Such a protocol gateway could run somewhere in the eBADGE cloud, allowing e.g. a VPP to control eBADGE-based HEHs through OpenADR messages. It could also run inside the HEH itself, thus providing a dual-standard HEH. However, it turned out that direct message translation cannot in general be implemented. For example, as is shown in Chapter 7, OpenADR semantics depends on the specifics of demand response programme that are not specified in the standard, thus such a gateway would be specific to an individual programme. Furthermore, the performance, security and reliability disadvantages of both would be obtained by such stacking of both protocols, which we obviously wanted to avoid.

That said, certain good practices from OpenADR are already used from the very first version of the eBADGE data model. For example, each activation order contains a *modification_count* field so that an issued order can be changed without the need for atomic "cancel-this-activation-and-activate-this-new-one" transactions.

3.2 IEC 61850

IEC 61850 [12] strives to not only replace 60870-5-104 but also to significantly expand the scope. It is a well-established IEC standard with first ratified documents from 2002 and covering aspects of communication networks and systems for substation, power utility and feeder equipment automation. Thus it supports integration and networked energy (power) systems built from multiple vendors to perform protection, monitoring, automation, metering, and control.

[4] https://github.com/EnerNOC/oadr2-ven-python/
[5] https://github.com/EnerNOC/oadr2-ven
[6] https://github.com/EnerNOC/oadr2-vtn-new
[7] http://open.enernoc.com/code/
[8] http://sourceforge.net/projects/openadr2vtn/

The scope of IEC 61850 was originally focused only on the “inside” the substation, later widened to inter substations and only recently there is activity to promote it also for distributed energy generators [13, 14]. Notable is the object-oriented, system-of-systems approach for substation automation where standardized device models are using names (and not object/register numbers), defined by Substation Configuration Language (SCL). From this aspect it is similar to OPC UA. On the other hand it is very focused on substation communication supporting even multi-cast messaging for inter-relays (used for protection). It is also possible to use Common Information Model (CIM) descriptions but mapping from SCL is required.

Considerable work has been ongoing the last three to five years to extend IEC 61850 [15] to new domains and make it applicable for new application areas like DR solution e.g. VPP. The use cases of different distribution automation concepts need to be considered in the information data models.

To identify the appropriate parts of the IEC 61850 that are relevant for VPP communication purposes the review of whole standard collection was performed. Based on this review the classification of most relevant parts can be made. From Figure 2 it is evident that for demand response purposes the IEC 61850-7 standard series are relevant.

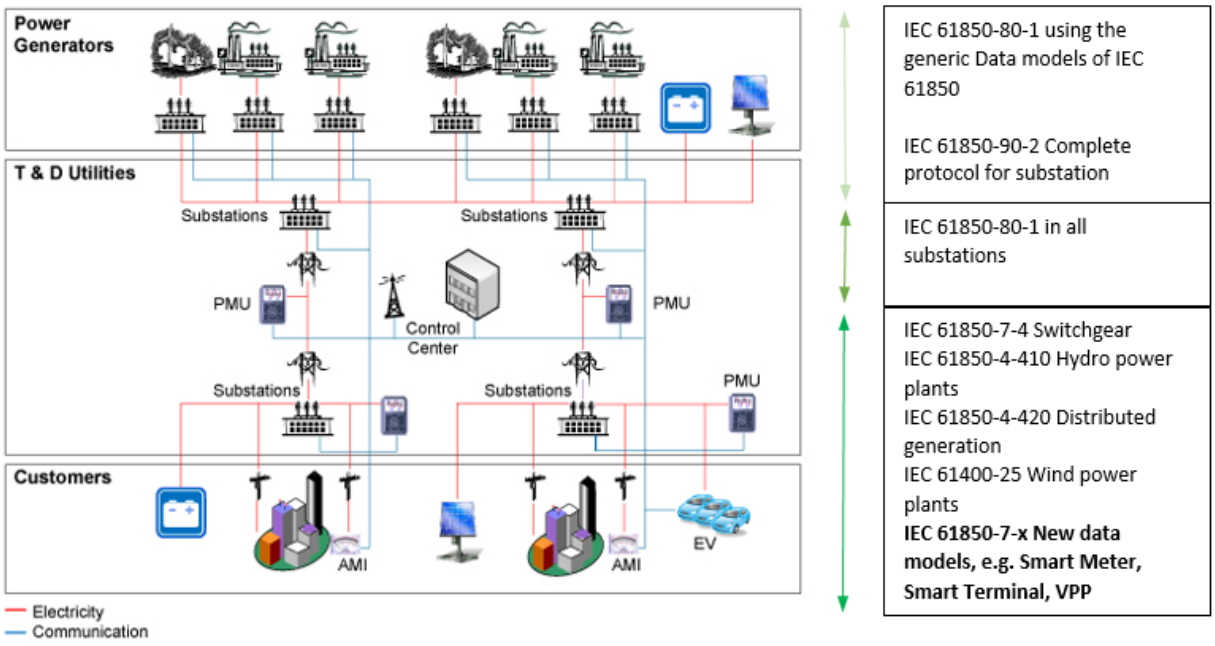

Figure 2: Relevant parts of the IEC 61850 standard [16].

Whereas the structure of the IEC 61850 standard is extensive and complex the focus is oriented towards communication parts of the standard defining requirements and mechanism for connectivity and data exchange between different components of the grid. From eBADGE point of view study of this standards is required in order to find a possible option to support pilot architecture of the project with IEC 61850 concept.

From telecommunication aspect the most relevant series of the IEC 61850 standard are 61850-6, 61850-7 and 61850-8. The IEC 61850-6 defines the SCL file format. In IEC 61850-7 series are given definitions of the communication models and data groups with sets of various data attributes. Mechanisms for embedding data in the MMS (Manufacturing Message Specification) are defined within the standard IEC 61850-8 [17]. Here is also defined the communication over IP and Ethernet for transmitting information via a telecommunication network.

In order to fully understand the concept of information sharing between different models defined within the IEC 61850 the basic information model should be well known. Architecture of the information model is presented in Figure 3.

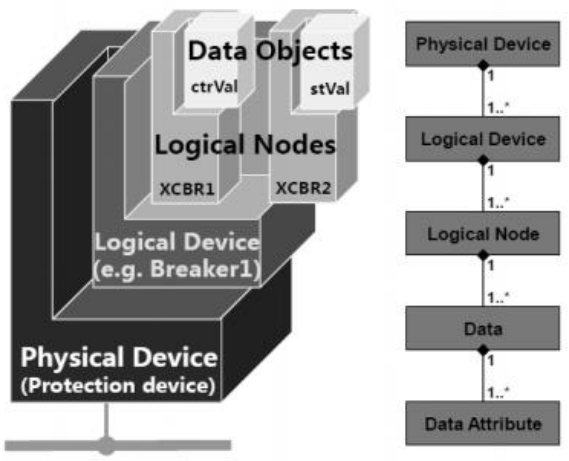

Figure 3: Information model defined according to the IEC 61750 [18].

A physical device (e.g. IED, PLC controller, etc.) is represented in nested form where the most important elements are logical device (LD) and logical node (LN). In this way the physical device is modelled as single or multiple devices that can perform various functions. Individual function (e.g. on or off the resource) is modelled with logical node. According to the IEC 61850-7-4 [19] standard series there are 92 different logical nodes defined that represents most basic real devices from power network. Majority of LN are defined and closely described in IEC 61850-7 standards series. The LN consist of different data groups (i.e. communication system and LN information, physical device information, measurement values, commands and status information and settings) that combines data with common content.

Individual data group consists of various data attributes, like voltage value, time stamp, etc. are defined within the IEC 61850-7-3 standard.

The final format of the object defined according to IEC 61850-8-1 is actually mapped to a specific protocol stack (based on MMS (ISO9506), TCP/IP, and Ethernet). For instance, objects can be send from (e.g. control centre, VPP, management system, etc.) to a server side and has the following structure:

LogicalDevice/LogicalNode$DataClass$DataAttribute$DataAttribute

The similar object format could be used for data exchange between the systems for providing demand response services like VPP [20]. The anatomy of the object references for a resource example is illustrated on Figure 4 bellow.

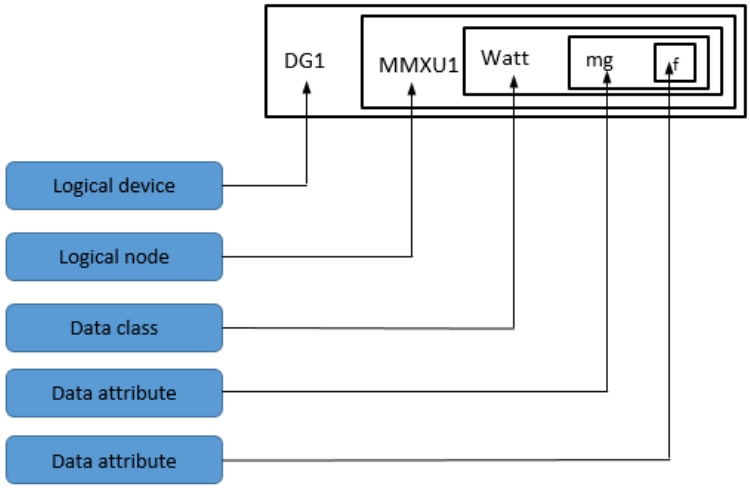

Figure 4: Object name defined by IEC 61850.

The message would consist from predefined LD, LN and data objects suitable for devices involved in demand response. In case, where VPP sends request for diesel generator (DG1) power measurement the following object name is sent to the DG1 resource.

DG1/MMXU1$Watt$mg$f

From the architectural point of view the server is at the top of a hierarchical object structure. In case of VPP, this is the controller which controls specific system (e.g. IED, RTU or PLC). The messages between the client and server side can be than exchange in the following manner. The VPP sends the request to server (e.g. DG1/MMXU1$Watt$mg$f) and server provides response with in XML file format. This message exchange procedure can be seen on Figure 5.

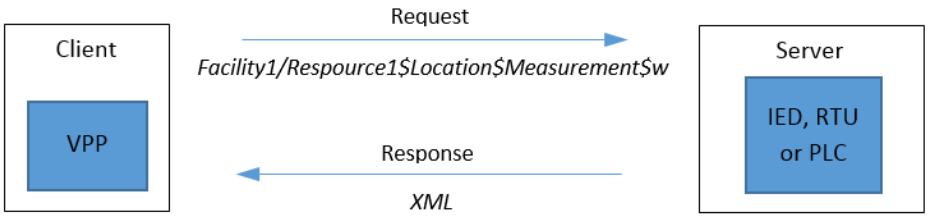

Figure 5: Message exchange between the server and client

3.2.1. Message exchange methods

Information exchange between the IED, RTU or PLC devices and control centres is carried out by using the abstract communication service interface (ACSI). On Figure 6 is presented placement of IEC 61850 information module into the different communication protocols.

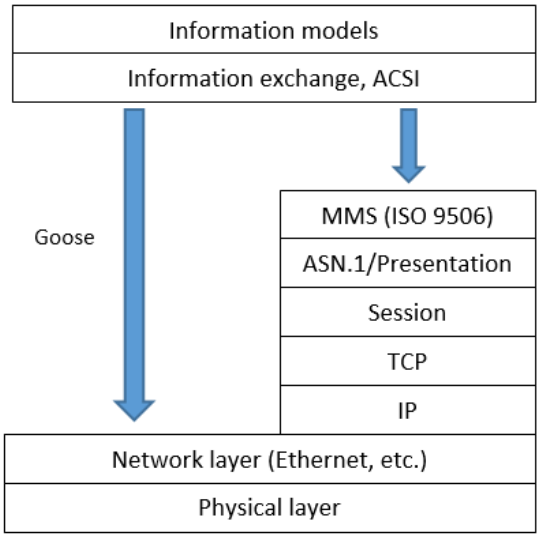

Figure 6: Placement of IEC 61850 information model in to the communication protocols.

ACSI enables connectivity and standardized access to the data that are stored within the logical nodes. Detailed description of data access is given in IEC 61850-7-2 standard [21]. In general, ACSI provides two abstract (virtual) interfaces.

First one is abstract interface that describes communication between client and server for query, data transmission, device management, notifications and event logging or file transfer (using MMS communication protocol). And the

second is abstract interface for fast and reliable event notifications from field devices based on generic object oriented substation events (GOOSE) protocol.

The primary task of ACSI is preparation of data structures and forwarding of this data in appropriate communication protocols and layers of ISO/OS protocol stack. That means that in client-server approach, model defined by IEC 61850 is firstly mapped into MMS protocol and then over ITC/IP protocol stack transmitted over network to control centre (right part of Figure 6).

In case of alarm notifications and sending individual measurement data the values are sent by GOOSE directly in Ethernet frame allowing short response times in case of quality and fast Ethernet connections.

Figure 7 lines up the IEC 61850 communication layers with those of the eBADGE message bus.

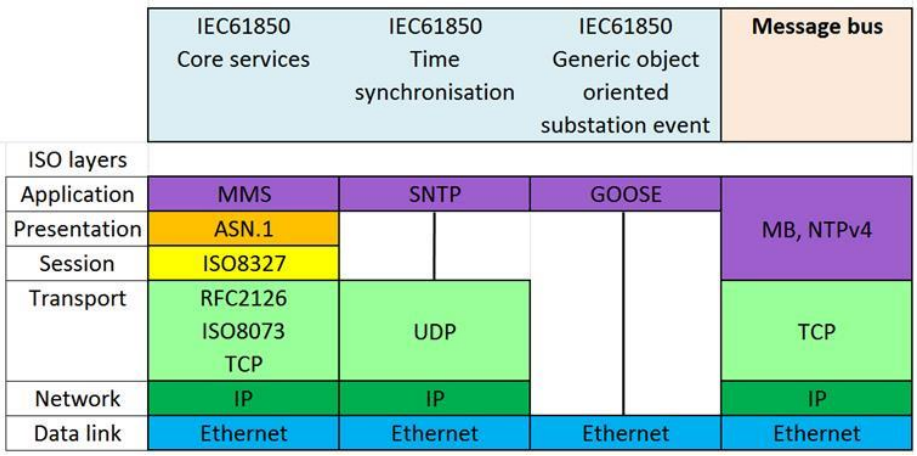

Figure 7: IEC 61850 vs eBADGE message bus layered on the ISO protocol stack.

3.2.2. Mapping of VPP requirements and available IEC 61850 entities

Within the IEC 61850-7 series are defined logical nodes and modules that are used for sending or receiving information regarding power, voltage, commands, etc. from various power system component. Here is presented review of general logical nodes that could be used for information sharing between the VPP systems, resources also high level architectures. The information types required for VPP operation are listed in Table 1 [20]. The information types provided by data sources are mapped to available logical nodes defined within the standard series.

Table 1: Mapping of required information to available IEC 61850 entities.

Information type	Data source	Description	IEC 61850	
			Logical Nodes	Standard part
Product prediction	Forecasting tool	Bid on the day-ahead spot market (e.g. 24 bids placed at 12:00 a.m. for next day).	Upcoming IEC61850-90-10 could be used	
Product prediction – intraday correction	Forecasting tool	Bid on intraday market (e.g. 1 bid per hour)	Upcoming IEC61850-90-10 could be used	
Real-time production of DER	SCADA system or IED	To determine accumulated production within present hour (used together with latest bid to estimate production).	MMXU.W	IEC 61850-7-4
Weather forecast data	Electricity retailers	Weather forecasts: wind speed, rainfall, for product prediction.	IEC 61850 not supporting (not expected)	

Historical levels of power flow	VPP	Determine if VPP shall attempt to lower peak flows to reduce tariffs or losses.	VPP historical statistical information model of MMXU.W	IEC 61850-7-2
Highest power flow in grid locations	VPP	Block power flow reduction if monthly maximum will not be exceeded.	VPP statistical information model of MMXU.W	IEC 61850-7-1 and 4
Real-time power measurements	SCADA system or IDE	Monitoring consumption/production of the resources	MMXU.W	IEC 61850-7-4
Reactive power in selected locations	SCADA system or IDE	Reactive power flow at points in the grid for which reactive power exchange is to be optimized	MMXU.Q	IEC 61850-7-4

3.2.3. Experiences with using IEC 61850

Today there are some demonstration projects and practical experiences from use of IEC 61850 standard. In the Danish Island of Bornholm reactive power support for frequency control and electrical vehicle charging is done by using IEC 61850 [20]. There is also an example of the VPP system that was constructed and implemented on the basis of the IEC 61850 in Germany [22].

One of interesting demonstration projects that involves VPP for electrical vehicles and use of IEC 61850 standard is Green eMotion project [23]. Within the project so called EDISON Virtual Power Plant (EVPP) was developed, where the server was implemented by using REST (Representational State Transfer)-based architecture. The IEC 61850 Data model was used in order to provide interoperability between the partners within the EDISON concept.

Here is necessary to point out another example of real implementation of IEC 61850 standard, more particularly communication part IEC 61850-7-420. One of the largest European utilities Vattenfall, launched in 2012 VHP Ready (Virtual Heat & Power Ready) demonstration project [24]. The aim of the project was implementation of VPP and use of renewable energy resources in a VPP system. For the communication purposes two standards were used IEC 60870-5-104 and IEC 61850-7-420. The VPP system used for this project mostly combines block-type combined heat and power plants (BCHP) heat pumps (HP), boilers and storage facilities.

Therefore, the logical devices like system, BCHP plant, HP, boiler are defined and are of equal rang arranged next to one another in the object model [24].

Based on this selection of resources the following logical nodes (Table 2) were selected from IEC 60850-7-420 standard for information exchange between the VPP system (command centre) and resources.

Table 2: Selected logical nodes from IEC 61850-7-420 standard.

Logical Nodes	Description
DCTS	Thermal storage
LN0	Common reports: contains relevant reports
LPHD	Physical device information
MENV	Emissions characteristics
MHET	Heat measurement
STMP	Temperature measurements
DRCC	Unit control actions
DRCS	Unit status
DRCH	Energy and/or ancillary services schedule
MFUL	Fuel characteristics
MMTR	Electricity metering
MMXU	Actual power system measurements
DCHB	Boiler system

These LN are representing beside LD, LN instance, function, data object and data attribute essential elements of the messages that are exchange between the server and clients.

IEC 61850 is a topical standard for power system area with support to different layers (e.g. information modelling, communications, etc.). Some parts of the standard are still under development. For instance, IEC 61850 will be able to support in future interoperable exchange of information from different renewable resources, scheduling and the aggregation required by a VPP system.

Certain objects and logical nodes (e.g. defined for diesel generators, micro turbines, CHP, BCHP, PV, etc.) defined within the IEC 62850-7-402 standard series are appropriate for use for demand response purposes. Nevertheless, there are still features that are needed to be standardised in order to cover all functionalities regarding demand response needs. Selected functionalities offered by IEC 61850 standard are suitable for use for VPP systems especially from technical point of view (get measurements, report actions, activate resources, etc.). But in the other hand there are still lack of object and models that could be used for information transfer regarding to market participation, scheduling and product prediction. These application layers are subject of upcoming standard IEC61850-90-10 [25] that is going to be released.

Based on today's trends of the prevalence of the IEC61850 standard some of utilities are moving towards the use of IEC 61850 for different fields of operation. More likely, some series of IEC 61850 standard are foreseen to be developed in order to cover needed parts of the demand response entities in the future. The LN, belonging to a specific device models, that are defined within the IEC 61850-7-420 standard are suitable for delivery of demand response signals to the various home energy hardware equipment within the customer's premises.

3.2.4. Compatibility with eBADGE Requirements

IEC 61850 targets physical infrastructure elements for power distribution, whereas eBADGE is concerned with communication between business entities. It is clear that on lower layers completely different set of constraints and goals exist for each, thus different implementation options emerge. As such these layers of IEC 61850 are not relevant for the discussion of the eBADGE data model.

Given the elaborate, top-down approach of IEC 61850, adopting it for the purpose of communicating the limited number of message types envisaged in eBADGE would be advisable if integration of existing artefacts based on it were planned in the pilot. In our case, on the other hand, a more straightforward implementation of our focused requirements suffices and will allow us to better concentrate on the project content. Furthermore, although open-source implementations exist, the standard itself is subject to licensing fees.

From the implementation point of view is necessary to point out that the implementation and support of the IEC 61850 is complex mainly because it contains different layers that needs to be supported from the eBAGDE systems side in order to be used for information exchange between the elements of the eBADGE cloud architecture. From this aspect the support of the standard for eBADGE purpose would be too extensive for the set time frame of the project, despite existing open-source implementation approaches that would need to be upgraded with additional functionalities.

3.3 Other Smart Grid Data Standards

3.3.1. iGorenje

iGorenje [26] is a simple HTTP/SOAP API for demand response intended primarily for smart home appliances. As a HTTP-based protocol, it requires either inbound connectivity into the clients (HEHs in our case) or periodic polling by the clients (asking the server "Do you have a command for me?"). Inbound connectivity is to be avoided for reasons of security, IP address scarcity etc. Periodic polling introduces performance problems for all but very small deployments, as experience from EUDT shows.

Content-wise, iGorenje fits the eBADGE requirements very well and could easily be extended by adding new methods to the API. Existing iGorenje semantics is thus a good starting point for definition of HEH-level messages. However, even if we re-implemented the semantics exactly but using another communication channel rather than HTTP, the software and devices based on our data model would not be compatible with iGorenje any more.

3.3.2. Open Smart Grid Protocol

OSGP is a large, complex binary standard for smart meters [27]. For example, to get the end device's status, the request (once decrypted, as OSGP uses encrypted commands) is
30 00 03 8E 56 E4 21 D6 2F 43 91 0C 48 A3 CC,
where:

- 30 is the OSGP command "Full Read",
- 00 03 is the ID of the table "BT03: End Device Mode Status",
- 8E 56 E4 21 is the OSGP sequence number of the request.

The response is an 8-byte sequence that encodes one integer and more than 40 boolean fields with a pre-defined meaning. As such, OSGP is difficult to implement and impossible to extend without breaking compatibility, which is in stark contrast to the eBADGE requirements.

3.3.3. OPC

The original OPC is a communication standard used in many fields of industrial automatics, including demand-response. The acronym originally stood for OLE (Object Linking and Embedding) for Process Control but was later re-defined as Open Platform Communications.

Being a communication standard, it only defines ways to access attributes (registers) and issue commands. The semantics of these attributes and commands are application/equipment-specific and each (type of) device requires additional configuration to define which registers can be accessed, what the data read from them means and what the commands that can be issued mean. It also requires Microsoft Windows. Although these properties make OPC unsuitable for eBADGE, certain existing OPC devices can be supported through separate protocol gateway services.

3.3.4. OPC UA

OPC UA (OPC Unified Architecture) is a completely new standard, although it also includes provisions for compatibility with the original OPC that it intends to replace. Unlike the original OPC, OPC UA is cross-platform [28]. It defines both a binary communication protocol as well as an HTTP/SOAP-based one. Multiple server implementations and client bindings for multiple programming languages are available from different software vendors.

OPC UA also contains the ability to model the semantics of the data objects. It allows data, alarms and events, and their history to be integrated into a single OPC UA Server. For example, OPC UA servers are able to represent a temperature transmitter as an object that is composed of a temperature value, a set of alarm parameters, and a corresponding set of alarm limits. Such models may be standardized for certain fields, which is then termed an *OPC UA companion standard* [29].

There are multiple OPC UA server implementations, including royalty-free, open-source ones from the OPC foundation and proprietary servers from various vendors. Similarly, client libraries for C/C++, Java, .NET, Python, and some other programming languages are available.

OPC UA has been considered as a transport layer for OpenADR [30] but was then rejected. The evaluation of OPC UA from the aspect of OpenADR would be very interesting for eBADGE as well; however, we did not find such an evaluation in the publicly available literature.

In our case, adopting OPC UA would imply both defining the eBADGE data model as an OPC UA semantic model and using OPC UA for all (or most of) the communication between the entities, that is, as a message bus equivalent. It has to be noted that we do not plan to use any existing OPC UA-based products within the pilot. Thus, rather than investing into the adoption of OPC UA, we have decided for a more straight-forward approach that will allow a rapid implementation of our requirements and then let the consortium concentrate on the development of innovative VPP functionalities and business practices, the demonstration of which is a major goal of the project. Nevertheless, certain strengths of OPC UA are mimicked in our data model, such as automatic discovery of the home energy hub's capabilities through the *get_capabilities* request.

3.4 Other Energy Market Data Standards

The energy markets mostly use data exchange formats defined by the software used in the individual markets, without following any recognized standards. On the other hand, for the control of the transmission grids, older, well-known, stable standards are mostly used.

3.4.1. Existing Market Applications

The trading applications mostly provide an interface to enter bids manually and to import them from a simple format. For example, the Slovenian intra-day market uses the ComTrader application [31]. The bids can be entered through a web GUI, imported from XML and CSV formats, or submitted through an API that also uses the XML format. Each bid contains required fields such as whether it is a buy or sell bid, quantity, price, and product ID. Additionally, a bid may include a date/time when it will automatically expire, a text/comment field for bidder's notes, sender and receiver IDs. Similarly, bids for the GME market can be entered manually or imported from XML.

3.4.2. NYSE UTPDirect

As an example of a high-volume trading standard, we also analysed the API for NYSE and NYSE MKT Cash Equities Markets [32]. It is a high-performance, low-level binary format operating, as the name suggest, directly above the physical layer. For example, the first byte marks the type, the next two the length, the rest is message body. Prices are represented as fractions with integer numerators and power-of-ten denominators. As an extremely high volume, low level, binary standard, it is not at all suitable for eBADGE.

4. General Approach

This section defines the encoding of message contents and other general technical choices and conventions that the whole data model follows. It also makes comparisons to OpenADR where relevant.

4.1 Specificity versus Generality

The required data can be encoded either in specific messages with fully defined semantics (meaning and expected action on the receiver's side), general message types complemented by some kind of "model of the system" that defines the semantics, or a compromise between the two. In the first version we chose the first option so that the data model stands for itself and implementation of the communicating software can straightforwardly follow this specification.

In the second versions we have generalized certain messages. In particular, instead of the *get_load_report* message we now use a general *get_report,* which can report any measured signal, such as temperature, gas consumption etc. The messages through which the HEH describes its capabilities were expanded accordingly so that for each device the list of signals and the units used are also reported. The downside of this generality is that the units to be used can no longer be specified in the data model, thus the receiver of the measurements must be able to read any commonly used unit.

The approach that OpenADR uses is more generic and flexible. The VEN can define new types of reports and describe them in the *oadrRegisterReport* messages. However, this can result in the VTN being unable to understand the report. For example, during our tests, the open source VTN by EPRI could not display the temperatures reported by the Universal Devices ISY 994i VEN connected to a Zigbee demand-response capable thermostat.

4.2 Message Payload Encoding

The message contents can be encoded in one of the many binary and text formats.

4.2.1. Binary Encoding

Binary formats are compact and, depending on whether a native encoding of the used programming language is used, serialization and de-serialization may be extremely efficient. On the other hand, lack of human readability lengthens the learning process for developers and, more importantly, makes debugging and diagnosing problems significantly more difficult. Human readability may seem undesired from the security aspect; however, since the model has to be public anyway, non-human-readable messages will not deter a determined attacker and may even make it harder to detect forged or malicious messages.

If our model defined the exact binary representation, a major effort would be required to ensure consistency and to develop the encoding and decoding software. On the other hand, if it relied on a standard serialization method such as Java object serialization or Google Protocol Buffers, it would be difficult to ensure interoperability even with the current versions of various programming languages, while maintaining it for the foreseen lifetime of the prototypes and artefacts based on this model would be even harder.

Finally, binary encodings are less extensible because adding a single data field may break the format of the whole message. Extensibility points have to be clearly defined in advance, which further complicates the model and lessens the compactness advantage of binary formats.

4.2.2. Text-Based Encoding: XML vs. JSON

The two most widely used text-based encodings are XML and JSON. XML is traditionally used in business software because it was the first widely accepted text-based structured encoding and because the validity of any XML document can be verified against a suitable XML schema, which describes the allowed format of the document. The schema can also be used to generate document-to-class mapping for a number of widely used programming languages. XML namespaces also solve the problem of distributed extensibility.

JSON, at first mostly used in small web-based applications due to easy conversion to/from JavaScript objects, is becoming increasingly popular in other fields. Although both XML and JSON can be made quite compact by

choosing short field/tag names, JSON is still more compact. For example, a small message containing two string and six integer fields might be 158 bytes as a (namespace-less) XML, 128 bytes as JSON, 40 bytes serialized with Google Protocol Buffers and 28 as raw binary.

Finally, JSON's good integration with programming languages like Python and JavaScript eases agile software development. Thus we have based the eBADGE data model on JSON.

OpenADR, in contrast, chooses XML. Furthermore, it makes extensive use of long field names and XML namespaces, which brings many advantages discussed below. However, the messages are significantly larger, as is described in Section 7.

4.2.3. Field names

The choice of field names in JSON (or, likewise, XML) is always a compromise between readability and compactness. In this draft we stressed the former over the latter. If later the testing reveals that shorter messages would be beneficial and that using compression on top of them is not recommended, we may consider either defining shorter synonyms for the most often used fields or shortening those field names altogether.

All message types and field names shall contain only the alphanumerical characters plus the underscore [A-Za-z0-9_] and shall begin with [A-Za-z] so that they can be mapped to object/structures in standard programming languages.

All field names are case-sensitive, except when explicitly defined otherwise. However, in order to allow easy mappings to object/structures in case-insensitive programming languages, no two field names shall differ only in the case.

OpenADR's naming scheme is a bit different. Its advantage is that it can use XML schemas, which allow for automatic validation of any message (i.e. checking whether a message is syntactically correct according to the schema). Furthermore, it automates the process of converting messages to objects in memory as well as the writing of documentation. As an example, we have put online a reference in the JavaDoc format to all OpenADR 2.0 messages[9].

4.2.4. Encoding of Standard Data Types

As per JSON standard, all strings in eBADGE messages shall be UTF-8 encoded.

All JSON numbers are floating-point; nevertheless, the eBADGE data model may define certain fields to be integer, or positive, or contain at most two decimals etc. The JSON standard does not support the IEEE 754[10] special values NaN, Inf, or -Inf. Although some implementations add support for these values, our data model does not use them in order to remain fully JSON-compliant.

The units of all numerical fields in eBADGE are either defined in the data model or chosen by the HEH and specified in the capabilities reported by the HEH. Use of other units is forbidden. In this version, all prices and money amounts are in € and other currencies are not supported.

JSON does not define date or time formats, thus all times are encoded as strings according to the ISO 8601[11]. Although the latter allows „local time" without a specified time-zone, this is forbidden in eBADGE messages to avoid confusion; all times must always specify the time-zone explicitly. Any time zone may be used since most programming languages offer easy-to-use conversion between them.

OpenADR also uses ISO 8601 dates and times. Like for field names, XML schemas are used for specifying and validating the syntax and allowed values of certain individual fields.

[9] The reference is available at http://xpower.xlab.si/unofficial-javadoc-openadr/ , however, it is unofficial and should only be used in conjunction with material provided by the OpenADR alliance and only for the purposes and within the limitations specified by the latter.
[10] A floating-point arithmetic standard.
[11] A standard defining encoding of times, dates, and periods in a text-based format.

4.2.5. Extensibility

To ensure distributed extensibility, any party may introduce additional message types or additional fields in existing messages. This version of the data model does not define how the receiver should react to unknown extensions, so the sender must not include them unless it knows for certain that the receiver supports them.

The reserved prefix for extensions is "ext_" – no message type nor field name in the core data model will start with "ext_". The extension name should always include a unique identifier of the party defining the extension, such as their internet domain without special characters. For example, a new message type for water usage profile could be named "ext_at_eudt_water_profile" and a message field containing extended VPP description could be named "ext_com_cybergrid_vpp_extended_desc".

We have originally identified three efforts related to JSON extensibility: JSON-LD[12], JSON Schema[13], and the JSON namespace proposal[14]. The latter is not actively developed anymore. The original draft specification was published in October 2011 and has not been updated since.

The JSON-LD and JSON Schema are being actively developed; the former was accepted by the W3C in January 2014. However, none of them are used even remotely as often as XML namespaces and are not supported natively in the most common programming languages, thereby requiring additional libraries. We have decided that removing the above described naming convention for extensibility and introducing either JSON-LD or JSON Schema as compulsory part of the data model would not bring enough advantages to be worthwhile.

OpenADR has the advantage of using XML namespaces here. The XML namespace capability is natively supported in all major programming languages. Any extension can thus simply be defined in a new namespace to ensure its XML tag will not clash with existing ones nor with extensions made by other parties. In fact OpenADR 2.0 already extends the OASIS standard “Energy Interoperation Version 1.0” [33].

4.3 Sender and Receiver Identification

The messages will always travel over some kind of communication channel; in the eBADGE project, this channel is the message bus developed in task T3.2. Most standard communication channels either connect a single sender to a single receiver or identify the sender and the receiver of each message in the message headers, thus it is not necessary to specify them explicitly in the message. This is also true for the specific case of eBADGE message bus [2] and for both transport types of OpenADR (HTTPS and XMPP)

4.4 Asynchronous and Synchronous Communication

This data model is agnostic to whether a synchronous communication protocol (i.e. immediate response built into the protocol, such as HTTP) or an asynchronous one (such as AMQP) is used. Whenever an asynchronous protocol is used and a message requires a response, the latter can be provided by a generalized message type *response* (see Section 6.1.2 for the exact specification).

OpenADR always uses a synchronous protocol, more specifically, the request-response communication pattern. All OpenADR requests have specifically prescribed response messages, except for *oadrDistributeEvent*, where the only a transport-level response is required (an empty HTTP response or an empty XMPP stanza).

4.5 Advice to Implementers

A reference Python implementation is available in the eBADGE project public deliverable D3.2.3, entitled "The eBADGE Message Bus, Final version" [2]. It is published under a permissive open source license and may thus be used as is, as a basis for other implementations, or as a reference.

[12] http://json-ld.org/
[13] http://json-schema.org/
[14] http://www.watersprings.org/pub/id/draft-saintandre-json-namespaces-00.txt

5. Specification of Messages at the Home Energy Hub Level

The flow of messages at the HEH level is quite simple. Only the following scenarios are possible:

- the VPP or other controlling entity sends requests for reports or activation orders, to which the HEH responds,
- the HEH autonomously sends certain data measurements and control reports, if configured to do so,
- the VPP sends pricing data.

The rest of this section first summarizes the changes compared to the previous version [1] and then specifies in detail all the message types at this level. For each message type its full contents are given and fields described based on an example. The sender and receiver of each message are also listed.

5.1 Changes from the Previous Versions

The biggest change is the generalization of reports. The load_report and generation_report were merged into a single report that can contain multiple measured signals. Consequently, the requests for reports are merged as well and must specify which signals the requesting party is interested in, and the device capabilities must include the list of supported signals. An important consequence is that the message count as well as their cumulative size are reduced whenever the HEH is periodically reporting more than one signal. The electricity_profile message was also dropped because it is now covered by the generalized reports.

An additional business case was proposed in WP4 where the VPP is a separate entity from the HEH operator/data collector. The latter provides measurements and activation capabilities as a service to the VPP. Consequently, an optional field heh_id was added to multiple messages, so that the two can express which HEH they are talking about. The heh_id represents a unique identifier of the HEH, which in our pilot is equal to the MAC address of the HEH's on-board network interface.

For each message type with this optional field one of the following must be true:

1. heh_id is null and the sender/receiver are as given in the message description (base setting, communication between HEH and controlling entity), or
2. heh_id is not null and in sender/receiver roles, controlling entity and HEH must be substituted for VPP and controlling entity, respectively (setting where VPP uses services provided by the HEH controlling entity).

It has to be noted that the dual role of the controlling entity looks similar but is not the same as in OpenADR, where a single entity can be a VTN and a VEN at the same time. The difference is that the controlling entity in the eBadge pilot merely offers access to certain HEH data (real-time or historical) and capabilities, rather than aggregating them or changing the message semantics.

The new message get_activation_capacity and respective response activation_capacity were also added due to WP4 requirements.

The prediction messages were removed from the data model because prediction was implemented separately in task T3.4 and integrated with the VPP as a cloud service, thus the HEH currently has nothing to do with it in our pilot.

The capabilities message was expanded with the optional fields device_sn, device_ip and device_mac for easier installation and maintenance of the HEHs. In the eBADGE pilot we also use certain extensions, e.g. a message type ext_si_xlab_led_signal is used to blink the LEDs on the HEH, which is helpful during installation. However, these extensions are not a part of the data model and are thus not listed here.

Finally, in this final version the reports (measured signals) are no longer divided into devices (appliances, measurement points) connected to the HEH. Instead they are referred to as, for example, *total.p* for the total real power measured by the HEH (typically the whole household) or *WaterHeater.i* for the current of the device called *WaterHeater*. Compared to the second version of the data model, this required the removal of the field *device* from

the messages *get_report, get_periodic_report,* and *report.* The *device_capabilities* message was also amended with the additional field *desc* with a human-readable description for each signal that can be reported.

5.2 Energy consumption/generation, and other measurements

5.2.1. Message type: get_report

Meaning: request for an individual measurement.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
from	start of period for which the report is to be returned	"2013-07-21T10:00:00.000Z"
to	end of period for which the report is to be returned	"2013-07-21T10:05:00.000Z"
resolution	in seconds	60
signals	the list of device.signals to report for (each device can measure multiple signals); in this example, voltage on all three phases in V (as measured centrally at the HEH, not per-device) and real power of the device WaterHeater01 in kW. The units used are reported by the HEH in the capabilities message.	["total.ua, total.ub, total.uc, WaterHeater01.p"]
heh_id	the ID of the HEH this request is for (optional; only used when the HEH owner provides measurements to third parties as a service, not used when sending directly to the HEH)	"1b8d6a00-0296-11e4-9191-0800200c9a66"

Raw JSON example: {"msg":"get_report", "from":"2013-07-21T10:00:00.000Z", "to":"2013-07-21T10:05:00.000Z", "resolution":60, "signals":["total.ua, total.ub, total.uc, WaterHeater01.p"], "heh_id":"1b8d6a00-0296-11e4-9191-0800200c9a66"}

5.2.2. Message type: get_periodic_report

Meaning: request for periodic reports; overrides previous setting.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
interval	in seconds, -1 to disable periodic reports	300
first_from	start of period that the first report should cover; if null, first report period may start at any time between 'now' and 'now+interval'	"2013-07-21T10:00:00.000Z"
resolution	in seconds	60
signals	list of signals; in the example, total real power summed over all devices in kW, real and reactive power of the device HeatPump01 in kW and kVAR, respectively, and outside temperature in °C	["total.p", "HeatPump01.p", "HeatPump01.q", "total.temperature_outside"]
heh_id	the ID of the HEH this request is for (optional; only used when the HEH owner provides measurements to third parties as a service, not used when sending directly to the HEH)	null
request_id	the ID of the periodic report request (optional)	"5f844f36-1747-4742-a619-3c87b90ea9aa"

Raw JSON example: {"msg":"get_periodic_report", "interval":300, "first_from":"2013-07-21T10:00:00.000Z", "resolution":60, "signals":["total.p", "HeatPump01.p", "HeatPump01.q", "total.temperature_outside"], "heh_id":null,"request_id":"5f844f36-1747-4742-a619-3c87b90ea9aa"}

5.2.3. Message type: report

Meaning: report/measured values of signal(s), sent either as response to individual request or periodically.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
from	start of period covered by this report	"2013-07-21T10:00:00.000Z"
to	end of period covered by this report	"2013-07-21T10:05:00.000Z"
resolution	in seconds	60
values	the signals being reported; for each signal, array of *n* values, where *n=(to-from)/resolution*	{"total.p":[1.73,1.68,0.43,0.33,0.45], "HeatPump01.p":[1.33,1.21,0.02,0.03,0.02], "HeatPump01.q":[0.13,0.11,0.01,0.01,0.01], "total.temperature_outside":[3.2,3.2,3.3,3.5,3.4]}
heh_id	the ID of the HEH this request is for (optional; only used when the HEH owner provides measurements to third parties as a service, not used when sending directly from the HEH)	null

Raw JSON example: {"msg":"report", "from":"2013-07-21T10:00:00.000Z", "to":"2013-07-21T10:05:00.000Z", "resolution":60, "values":{"total.p":[1.73,1.68,0.43,0.33,0.45], "HeatPump01.p":[1.33,1.21,0.02,0.03,0.02], "HeatPump01.q":[0.13,0.11,0.01,0.01,0.01], "total.temperature_outside":[3.2,3.2,3.3,3.5,3.4]}, "heh_id":null}

If the period requested for the report (get_report.(to-from) or get_periodic_report.interval) is not a multiple of resolution then the actual reported period's shall be longer, i.e. the 'to' time will be later than requested.

Details for field: values (depends on signals requested in get_[periodic_]report)

Field	Description/comment	Example value
total.p	total real power summed over all devices in kW	[1.73,1.68,0.43,0.33,0.45]
HeatPump01.p	real power of the device HeatPump01 in kW	[1.33,1.21,0.02,0.03,0.02]
HeatPump01.q	reactive power of the device HeatPump01 in kVAR	[0.13,0.11,0.01,0.01,0.01]
total.temperature_outside	outside temperature in °C	[3.2,3.2,3.3,3.5,3.4]

5.2.4. Message type: get_energy_events

Meaning: request for report on energy-related anomalous events.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
from	start of period for which the report is to be returned	"2013-07-23T16:10:00.000Z"
to	end of period for which the report is to be returned	"2013-07-23T17:20:00.000Z"
severity	minimum severity to include in report	1

Raw JSON example: {"msg":"get_energy_events", "from":"2013-07-23T16:10:00.000Z", "to":"2013-07-23T17:20:00.000Z", "severity":1}

5.2.5. Message type: get_energy_events_realtime

Meaning: request to report events immediately when they happen.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
severity	minimum severity to report immediately; set very high to disable real-time reporting	2

Raw JSON example: {"msg":"get_energy_events_realtime", "severity":2}

5.2.6. Message type: energy_events

Meaning: report about energy-related events, sent automatically on severe events or on request.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
events	list of events	[{"severity":2, "type":"voltage_low", "start_time":"2013-07-23T16:33:56.345Z", "end_time":"2013-07-23T16:33:58.645Z"}]

Raw JSON example: {"msg":"energy_events", "events":[{"severity":2, "type":"voltage_low", "start_time":"2013-07-23T16:33:56.345Z", "end_time":"2013-07-23T16:33:58.645Z"}]}

Details for field: events (array)

Field	Description/comment	Example value
severity		2
type	list of events	"voltage_low"
start_time	higher is more severe	"2013-07-23T16:33:56.345Z"
end_time	possible values: voltage_low, voltage_high, frequency_low, frequency_high; other values may be defined in next version of data model	"2013-07-23T16:33:58.645Z"

5.3 Activations and tariff updates

5.3.1. Message type: activate

Meaning: activation order.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
id	a unique ID of this activation order, assigned by the VPP	"938f2b97-314c-49e8-9860-f441df2284a1"
modification_count	increased by VPP each time a modified activation is sent; the pair uniquely identifies the activation order; if an activation with the same	0

	ID but lower modification count was previously accepted, this message implies modification of the accepted activation	
from		"2013-07-24T11:10:20.000Z"
to		"2013-07-24T11:14:55.000Z"
quantity	in kW, positive to increase generation/decrease load, negative for vice versa	3.4
device	device ID; use null for "any/total"	"ECAR01"
heh_id	the ID of the HEH this request is for (optional; only used when the HEH owner provides activation capability to third parties as a service, not used when sending directly to the HEH)	"304c3c50-0296-11e4-9191-0800200c9a66"

Raw JSON example: {"msg":"activate", "id":"938f2b97-314c-49e8-9860-f441df2284a1", "modification_count":0, "from":"2013-07-24T11:10:20.000Z", "to":"2013-07-24T11:14:55.000Z", "quantity":3.4, "device":"ECAR01", "heh_id":"304c3c50-0296-11e4-9191-0800200c9a66"}

5.3.2. Message type: accept_activation

Meaning: activation accepted.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
id	the ID of the activation being accepted	"938f2b97-314c-49e8-9860-f441df2284a1"
modification_count	the modification count of the activation being accepted	0

Raw JSON example: {"msg":"accept_activation", "id":"938f2b97-314c-49e8-9860-f441df2284a1", "modification_count":0}

5.3.3. Message type: reject_activation

Meaning: activation rejected.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
id	the ID of the activation being rejected	"938f2b97-314c-49e8-9860-f441df2284a1"
modification_count	the modification count of the activation being rejected; if an activation with the same ID but lower modification count was previously accepted, this implies rejection of the modification and preservation of the previously accepted version of the activation	0

Raw JSON example: {"msg":"reject_activation", "id":"938f2b97-314c-49e8-9860-f441df2284a1", "modification_count":0}

5.3.4. Message type: modify_activation

Meaning: suggestion for modification of the activation; implies rejection of original parameters; if VPP agrees to modification, must send a new activation order with suggested modifications.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
id	the ID of the activation being rejected	"938f2b97-314c-49e8-9860-f441df2284a1"
modification_count	the modification count of the activation being rejected	0
from	all fields must be present, including the ones with unchanged values	"2013-07-24T11:11:25.000Z"
to		"2013-07-24T11:14:55.000Z"
quantity		"3.6"
device		"ECAR01"

Raw JSON example: {"msg":"modify_activation", "id":"938f2b97-314c-49e8-9860-f441df2284a1", "modification_count":0, "from":"2013-07-24T11:11:25.000Z", "to":"2013-07-24T11:14:55.000Z", "quantity":"3.6", "device":"ECAR01"}

5.3.5. Message type: get_activation_capacity

Meaning: request for estimate on capacity available for activation in this moment.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
device	the device to report for (optional; if left out, total is returned)	null
heh_id	the ID of the HEH this request is for (optional; only used when the HEH owner provides activation capability to third parties as a service, not used when sending directly to the HEH)	"697e91d0-0296-11e4-9191-0800200c9a66"

Raw JSON example: {"msg":"get_activation_capacity", "device":null,"heh_id":"697e91d0-0296-11e4-9191-0800200c9a66"}

5.3.6. Message type: activation_capacity

Meaning: response to get_activation_capacity.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
device	the device to report for (optional; if left out, total over HEH)	null
pos_capacity	estimated capacity in kW available for increasing production/decreasing load; note that the quality of estimation is up to the estimator, e.g., in simplest case the nominal power of the device can be returned, even if the device is a consumer and is already switched off	4.3
neg_capacity	estimated capacity in kW available for decreasing production/increasing load	0
heh_id	the ID of the HEH this request is for (optional; only used when the HEH	"697e91d0-0296-

	owner provides activation capability to third parties as a service, not used when sending directly from the HEH)	11e4-9191-0800200c9a66"

Raw JSON example: {"msg":"activation_capacity", "device":null,"pos_capacity":4.3, "neg_capacity":0, "heh_id":"697e91d0-0296-11e4-9191-0800200c9a66"}

5.3.7. Message type: contingency_activate

Meaning: contingency activation order; activate by as much as possible up to the specified quantity.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
id	unique ID	"3640aa93-28a7-420e-aebf-f4a7fc3a08d2"
from	if in past, activate ASAP	"2013-07-24T10:55:13.000Z"
to		"2013-07-24T10:56:18.000Z"
max_quantity	maximum quantity to activate if possible, in kW; null means no upper limit	120

Raw JSON example: {"msg":"contingency_activate", "id":"3640aa93-28a7-420e-aebf-f4a7fc3a08d2", "from":"2013-07-24T10:55:13.000Z", "to":"2013-07-24T10:56:18.000Z", "max_quantity":120}

The HEH does not need to acknowledge a contingency activation. If it would have to then it would not be any diffferent from an ordinary activation.

5.3.8. Message type: contingency_end

Meaning: signals when the contingency activation should end.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
id	ID of the contingency activation being ended	"3640aa93-28a7-420e-aebf-f4a7fc3a08d2"
end	when to end; use a past date for immediate end	"2000-01-31T23:00:00.000Z"

Raw JSON example: {"msg":"contingency_end", "id":"3640aa93-28a7-420e-aebf-f4a7fc3a08d2", "end":"2000-01-31T23:00:00.000Z"}

5.3.9. Message type: load_price

Meaning: informs HEH about changed price at which the end-user BUYS energy.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
from		"2013-07-27T23:10:00.000Z"
to		"2013-07-28T00:30:00.000Z"
price	new price in €/kWh	0.138

Raw JSON example: {"msg":"load_price", "from":"2013-07-27T23:10:00.000Z", "to":"2013-07-28T00:30:00.000Z", "price":0.138}

5.3.10. Message type: generation_price

Meaning: informs HEH about changed price at which the end-user SELLS energy.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
from		"2013-07-27T23:10:00.000Z"
to		"2013-07-28T00:30:00.000Z"
price	new price in €/kWh	0.117
device	the ID of the generator, as the generation price depends on (at least) the type of the generator	"PV01"

Raw JSON example: {"msg":"generation_price", "from":"2013-07-27T23:10:00.000Z", "to":"2013-07-28T00:30:00.000Z", "price":0.117, "device":"PV01"}

5.3.11. Message type: get_all_prices

Meaning: a request for all available pricing info; sent by HEH on first boot and if for some reason it lost the information.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value

Raw JSON example: {"msg":"get_all_prices"}

5.4 Other

5.4.1. Message type: get_status_report

Meaning: a request for current status and all events that happened in the given interval.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
from		"2013-07-24T10:55:13.000Z"
to		"2013-07-24T10:56:18.000Z"
severity_threshold	events with this and higher severity will be reported	2

Raw JSON example: {"msg":"get_status_report", "from":"2013-07-24T10:55:13.000Z", "to":"2013-07-24T10:56:18.000Z", "severity_threshold":2}

5.4.2. Message type: status_report

Meaning: status report - response to status_request or automatically sent on the most severe events.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
status	not sure what we need here...	"OK"
clock	current time on device, to check if it is in sync	"2013-07-24T10:33:59.873Z"
events		[{"time":"2013-07-23T03:23:12.342Z", "severity":4, "type":"network_down"}, {"time":"2013-07-23T03:24:45.932Z", "severity":2, "type":"network_up"}]

Raw JSON example: {"msg":"status_report", "status":"OK", "clock":"2013-07-24T10:33:59.873Z", "events":[{"time":"2013-07-23T03:23:12.342Z", "severity":4, "type":"network_down"}, {"time":"2013-07-23T03:24:45.932Z", "severity":2, "type":"network_up"}]}

Details for field: events (array)

Field	Description/comment	Example value
time	current time on device, to check if it is in sync	"2013-07-23T03:23:12.342Z"
severity		4
type		"network_down"

5.4.3. Message type: set_clock

Meaning: set clock - issued if local clock in status_report is found to be wrong.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
offset	offset in seconds to add to the HEH's local clock; if null, the HEH should re-sync its clock over default means (ie NTP)	-3600

Raw JSON example: {"msg":"set_clock", "offset":-3600}

5.4.4. Message type: set_smart_mode

Meaning: sets smart/non-smart mode of HEH (e.g. in an emergency).

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
mode	normal, passive (all activations cancelled, all messages except set_smart_mode ignored, local logging still on), off (physical shutdown, requires manual intervention to return to normal)	"passive"
reset	whether HEH should also forget all activations, pricing etc and reboot itself; ignored if mode=off	false

Raw JSON example: {"msg":"set_smart_mode", "mode":"passive", "reset":false}

5.4.5. Message type: get_capabilities

Meaning: request to report the capabilities of HEH and/or individual devices.

Sent by: controlling entity **to:** HEH.

Field	Description/comment	Example value
device	device ID to report for; null for HEH itself	null

Raw JSON example: {"msg":"get_capabilities", "device":null}

5.4.6. Message type: capabilities

Meaning: report about HEH/total capabilities.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
device_name		"eBADGE Home Energy Hub"
device_version		"1.0"
device_sn	serial number of HEH (optional; in eBADGE, DID of measurement board)	"4524590"
device_ip	IP address of HEH (optional; in eBADGE, the WAN-side IP of LTE router	"172.21.0.42"
device_mac	MAC address of HEH (optional; in eBADGE, the MAC address of rPI's built-in Ethernet port)	"90:2b:34:5f:7c:dd"
devices	list of device IDs the HEH knows about, possibly including *total*	["ECAR01", "PV01", "PV02", "WASHDR01", "device01", "device02"]
ext_com_cybergrid_vpp_extended_description	a third-party extension field of the message, identified by ext_company_domain_ prefix	"some non-standard stuff"

Raw JSON example: {"msg":"capabilities", "device_name":"eBADGE Home Energy Hub", "device_version":"1.0", "device_sn":"4524590", "device_ip":"172.21.0.42", "device_mac":"90:2b:34:5f:7c:dd", "devices":["ECAR01", "PV01", "PV02", "WASHDR01", "device01", "device02"], "ext_com_cybergrid_vpp_extended_description":"some non-standard stuff"}

5.4.7. Message type: device_capabilities

Meaning: report about device capabilities.

Sent by: HEH **to:** controlling entity.

Field	Description/comment	Example value
device	signals that this is about the device with ID ECAR01	"ECAR01"

classes	subset (possibly empty) of: consumer, generator, storage	["consumer", "storage"]
type	type of device; hierarchical description with standardized possible values	"car/electric"
device_name	a descriptive name, e.g. commercial name of a car make and model	"Electric Cars Model E1"
version		"0.9 beta"
signals	list of signals that this device can measure/report; see below for explanation	[{"name":"p", "desc":"real power (load)", "unit":"kW", "range":[0,6.6]}, {"name":"p_gen", "desc":"real power (generation, vehicle-to-grid)", "unit":"kW", "range":[0,3.3]}, {"name":"battery_level", "desc":"level of charge from 'empty' to 'full'", "unit":"1", "range":[0,1]}]
ext_com_electriccars_ecar_battery_capacity	extension example, in this case battery capacity in kWh, extension defined by the company Electric Cars	14.2
can_predict_profile	this device can predict its profile (how it communicates the profile to the HEH is out of the scope of this document)	true
can_predict_curtailment_capacity	this device can predict its curtailment capacity	true

Raw JSON example: {"msg":"device_capabilities", "device":"ECAR01", "classes":["consumer", "storage"], "type":"car/electric", "device_name":"Electric Cars Model E1", "version":"0.9 beta", "signals":[{"name":"p", "desc":"real power (load)", "unit":"kW", "range":[0,6.6]}, {"name":"p_gen", "desc":"real power (generation, vehicle-to-grid)", "unit":"kW", "range":[0,3.3]}, {"name":"battery_level", "desc":"level of charge from 'empty' to 'full'", "unit":"1", "range":[0,1]}], "ext_com_electriccars_ecar_battery_capacity":14.2, "can_predict_profile":true, "can_predict_curtailment_capacity":true}

Details for field: signals (array)

Field	Description/comment	Example value
name	signal name	"p"
desc	human-readable description/meaning of signal	"real power (load)"
unit	the unit that all the values will be reported in (SHOULD be kW/kVA/kWh/A/V/Hz for electricity signals)	"kW"
range	optional; the possible range of values	[0,6.6]

6. Specification of Messages at the Market Level

The message flow at this level is more complicated, partly because of stricter consistency requirements and partly because the market model is also not trivial. We can divide it into the following scenarios:

1. On the national balancing reserve market, the TSO collects balancing reserve bids from the BSPs until gate closure. It then selects the appropriate subset of bids to accept, informs the respective TSOs about it, and informs all the BSPs that the market has been cleared.

2. On the national balancing energy market, the TSO collects balancing energy bids from the BSPs until gate closure and builds the (national) merit order list. When imbalance is detected, the BSP of the first bid from the list is instructed to activate the balancing energy that is the subject of that bid.

3. On the international balancing energy market, all the TSOs collect balancing energy bids from their respective BSPs and forward (some or all of) them to the super-TSO (the international market operator), who builds a common merit order list. When a TSO detects imbalance, it requests balancing from the super-TSO. The super-TSO may decide that it can be netted with another TSO's imbalance and/or that some bids need to be activated, taking into account both the merit order and the transmission capacities. It then informs the unbalanced TSO(s) that the balance is/will be (partly or fully) restored and the TSO(s) where the bid(s) come from that the bid(s) need to be activated.

The following subsections detail the messages for these three scenarios and also illustrate them with sequence diagrams. Note that the market level message specifications have not changed at all from the first version.

6.1 Balancing reserve market

Balancing reserve market will only be modelled later in the project. However, some messages are the same as in the balancing energy market so they are listed here.

6.1.1. Message type: balancing_reserve_bid

Meaning: balancing reserve bid.

Sent from: BSP **to:** the TSO responsible for the BSP's balancing zone.

Field	Description/comment	Example value
msg_id	UUID of message assigned (created) by BSP; all market-level messages that require a response have msg_id, others do not	"938f2b97-314c-49e8-9860-f441df2284a1"
bid_id	UUID of bid assigned by BSP, used to refer to it later (e.g. to accept or cancel it)	"3640aa93-28a7-420e-aebf-f4a7fc3a08d2"
product	product description - described below	{"positive":true, "start":"2013-07-21T10:00:00.000Z", "end":"2013-07-21T10:15:00.000Z"}
quantity	in MW	45.3
divisible	false for all-or-nothing; optional, default is true	true
reserve_price	reserve price in €/MWh	1.4
energy_price	energy price in €/MWh	8.32

Raw JSON example: {"msg":"balancing_reserve_bid", "msg_id":"938f2b97-314c-49e8-9860-f441df2284a1", "bid_id":"3640aa93-28a7-420e-aebf-f4a7fc3a08d2", "product":{"positive":true, "start":"2013-07-21T10:00:00.000Z", "end":"2013-07-21T10:15:00.000Z"}, "quantity":45.3, "divisible":true, "reserve_price":1.4, "energy_price":8.32}

The price is floating-point because JSON has no fixed-point numbers. No other currencies are supported in this version of the data model.

Details for field: product

Field	Description/comment	Example value
positive	direction - positive for generation/injection/upward reserve, negative for consumption/withdrawal/downward reserve; part of product description so that 'one product, one merit list'	true
start	start of period covered by this product	"2013-07-21T10:00:00.000Z"
end	end of period covered by this product	"2013-07-21T10:15:00.000Z"

6.1.2. Message type: response

Meaning: common response format for the messages that need acknowledgment or rejection.

Sent from: the receiver of the original message **to:** the sender of the original message.

Field	Description/comment	Example value
msg_id	ID of message we are responding to	"938f2b97-314c-49e8-9860-f441df2284a1"
response_code	error code; like OpenADR, we can use a subset of HTTP response codes, 200 being OK	200
response_subcode	optional message-specific additional code that further specifies error condition	200
response_desc	OK (code 200) or explanation of error (all other codes)	"OK"

Raw JSON example: {"msg":"response", "msg_id":"938f2b97-314c-49e8-9860-f441df2284a1", "response_code":200, "response_subcode":200, "response_desc":"OK"}

All messages that contain msg_id require a 'response' message as acknowledgement/rejection (except, of course, for the 'response' message itself).

Until the sender receives the response, it MUST NOT rely on the message being delivered or acted upon.

There is no cancel_bid message; bids can be changed by re-bidding the same bid ID, and cancelled by re-bidding with zero quantity.

There is no gate closure notification message - closure time is specified in the market rules and thus known in advance to all players.

6.1.3. Message type: bid_accepted

Meaning: bid accepted, BSP must confirm and then reserve the capacity.

Sent from: TSO **to:** the BSP that sent the bid.

Field	Description/comment	Example value
msg_id		"fd127905-8005-44cc-8870-00ac8b6d27e3"
bid_id	the bid that was accepted	"3640aa93-28a7-420e-aebf-f4a7fc3a08d2"
quantity	the accepted quantity	33.1

Raw JSON example: {"msg":"bid_accepted", "msg_id":"fd127905-8005-44cc-8870-00ac8b6d27e3", "bid_id":"3640aa93-28a7-420e-aebf-f4a7fc3a08d2", "quantity":33.1}

6.1.4. Message type: market_cleared

Meaning: a notification that the market is cleared, thus all bids not yet accepted will never be accepted.

Sent from: TSO **to:** all BSPs that sent any bids (optionally also to the rest of BSPs).

Field	Description/comment	Example value
product	the product whose market is cleared	{"positive":true, "start":"2013-07-21T10:00:00.000Z", "end":"2013-07-21T10:15:00.000Z"}

Raw JSON example: {"msg":"market_cleared", "product":{"positive":true, "start":"2013-07-21T10:00:00.000Z", "end":"2013-07-21T10:15:00.000Z"}}

6.2 National balancing energy market

In the national balancing energy market, bids are submitted with the balancing_energy_bid message, which is almost the same as reserve_energy_bid. The acknowledgement message 'response' is exactly the same. There are no bid_accepted and market_cleared because bids are only inserted into the merit order list.

In addition, an 'activate' message signals both that the energy bid was accepted AND that it has to be activated in the specified period/quantity.

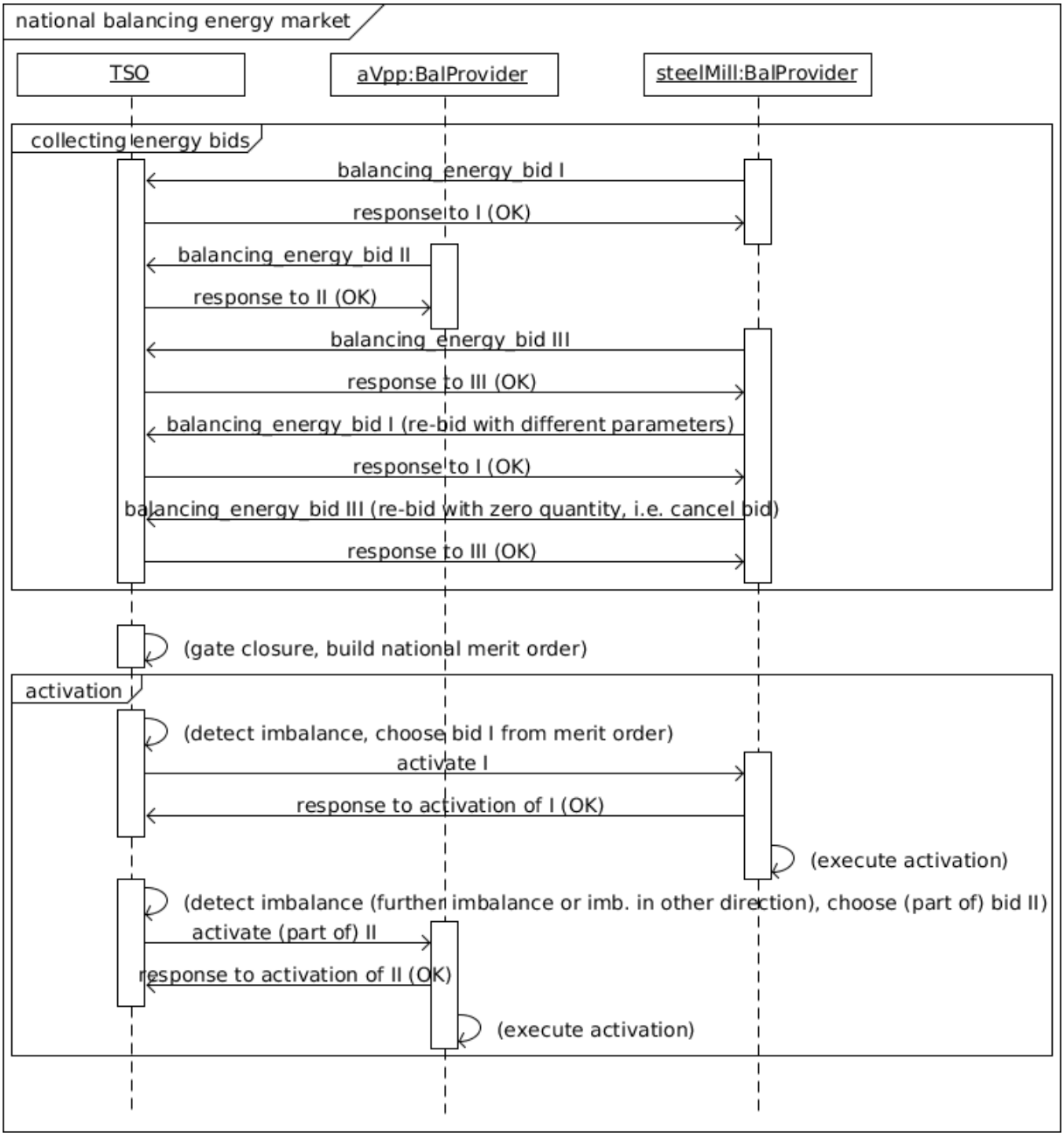

Figure 8: A typical message sequence in a national balancing energy market

6.2.1. Message type: balancing_energy_bid

Meaning: balancing energy bid.

Sent from: BSP **to:** the TSO responsible for the BSP's balaning zone.

Field	Description/comment	Example value
msg_id		"6c4de4fa-c950-4054-8a74-258088c29947"
bid_id	UUID of bid, used to refer to it later (e.g. to accept or cancel it)	"9b42269d-5b80-4029-8cce-79462eaf986e"

product	product object as shown above	{"positive":true, "start":"2013-07-21T10:00:00.000Z", "end":"2013-07-21T10:15:00.000Z"}
quantity	in MWh	15
divisible	false for all-or-nothing; default is true	false
energy_price	energy price in €/MWh	14.11

Raw JSON example: {"msg":"balancing_energy_bid", "msg_id":"6c4de4fa-c950-4054-8a74-258088c29947", "bid_id":"9b42269d-5b80-4029-8cce-79462eaf986e", "product":{"positive":true, "start":"2013-07-21T10:00:00.000Z", "end":"2013-07-21T10:15:00.000Z"}, "quantity":15, "divisible":false, "energy_price":14.11}

6.2.2. Message type: activate_bid

Meaning: activation, which BSP must confirm and then activate the capacity.

Sent from: TSO **to:** the BSP that sent the bid.

Field	Description/comment	Example value
msg_id		"b10c501d-a10d-47d6-a0bf-068951eb6ae7"
bid_id	bid to activate	"9b42269d-5b80-4029-8cce-79462eaf986e"
quantity	quantity to activate	15
from	activation start; use null for 'ASAP'	"2013-07-24T11:10:20.000Z"
to	activation end; use null for 'till end of product'	"2013-07-24T11:14:55.000Z"

Raw JSON example: {"msg":"activate_bid", "msg_id":"b10c501d-a10d-47d6-a0bf-068951eb6ae7", "bid_id":"9b42269d-5b80-4029-8cce-79462eaf986e", "quantity":15, "from":"2013-07-24T11:10:20.000Z", "to":"2013-07-24T11:14:55.000Z"}

Once activated, a bid cannot be deactivated; it remains active until scheduled 'to' time or until end of product.

Should the TSO be able to modify an activation (e.g. increase quantity, postpone activation end time)? Since there are no deactivations we might need this.

6.3 International balancing energy market

On the international market, the TSOs forward (some or all of) the bids to the super-TSO with the same balancing_energy_bid message. For each forwarded bid, the super-TSO must also store the originating (forwarding) TSO.

The possibility of unshared bids does not affect the data model. The unshared bids are simply not forwarded.

Since the TSOs are not the market organizers any more (the latter role being trusted to the super-TSO), additional message types are required for the TSOs to request balancing energy, and for the super-TSO to grant and/or deny these requests.

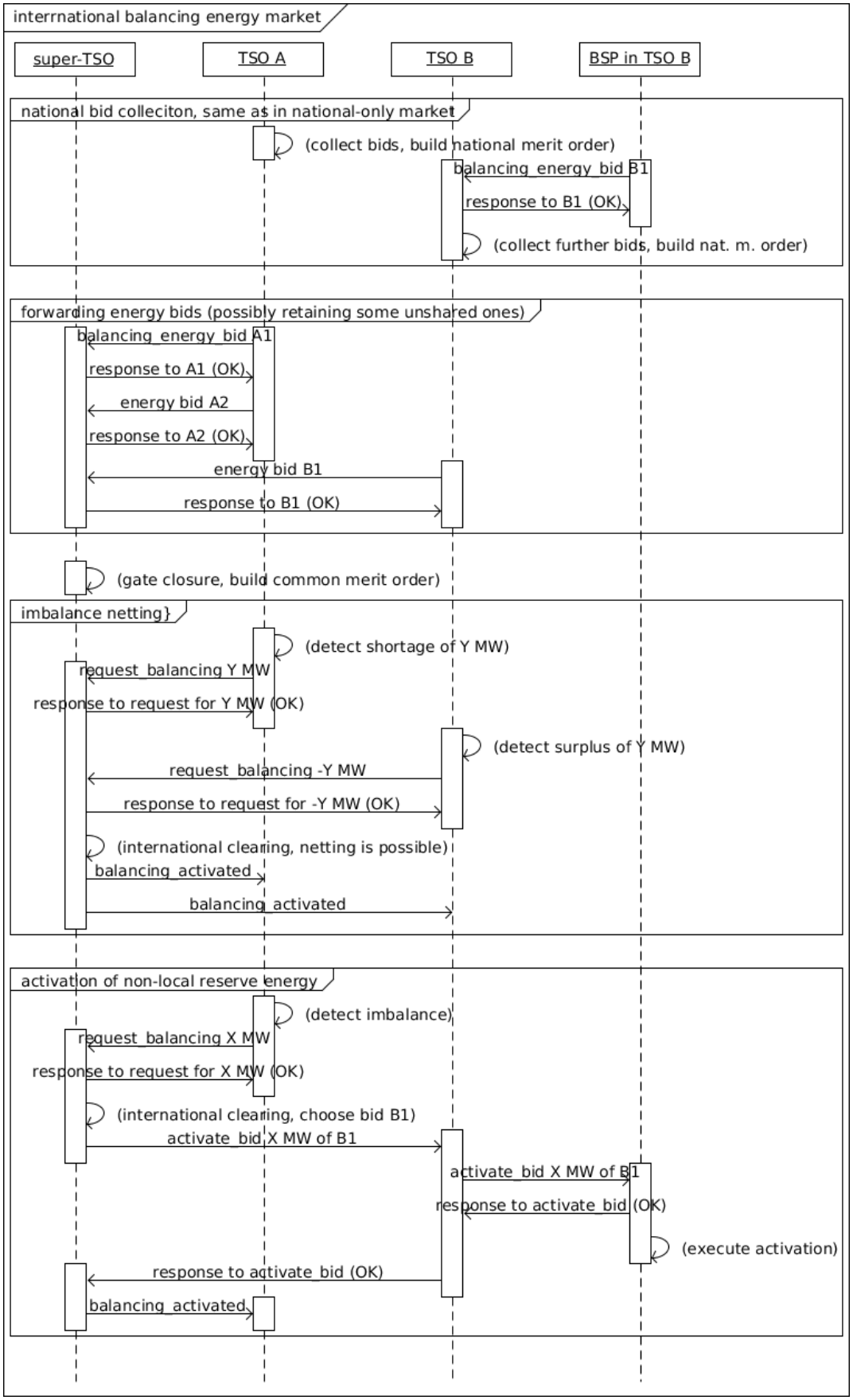

Figure 9: A typical message sequence in an international balancing energy market

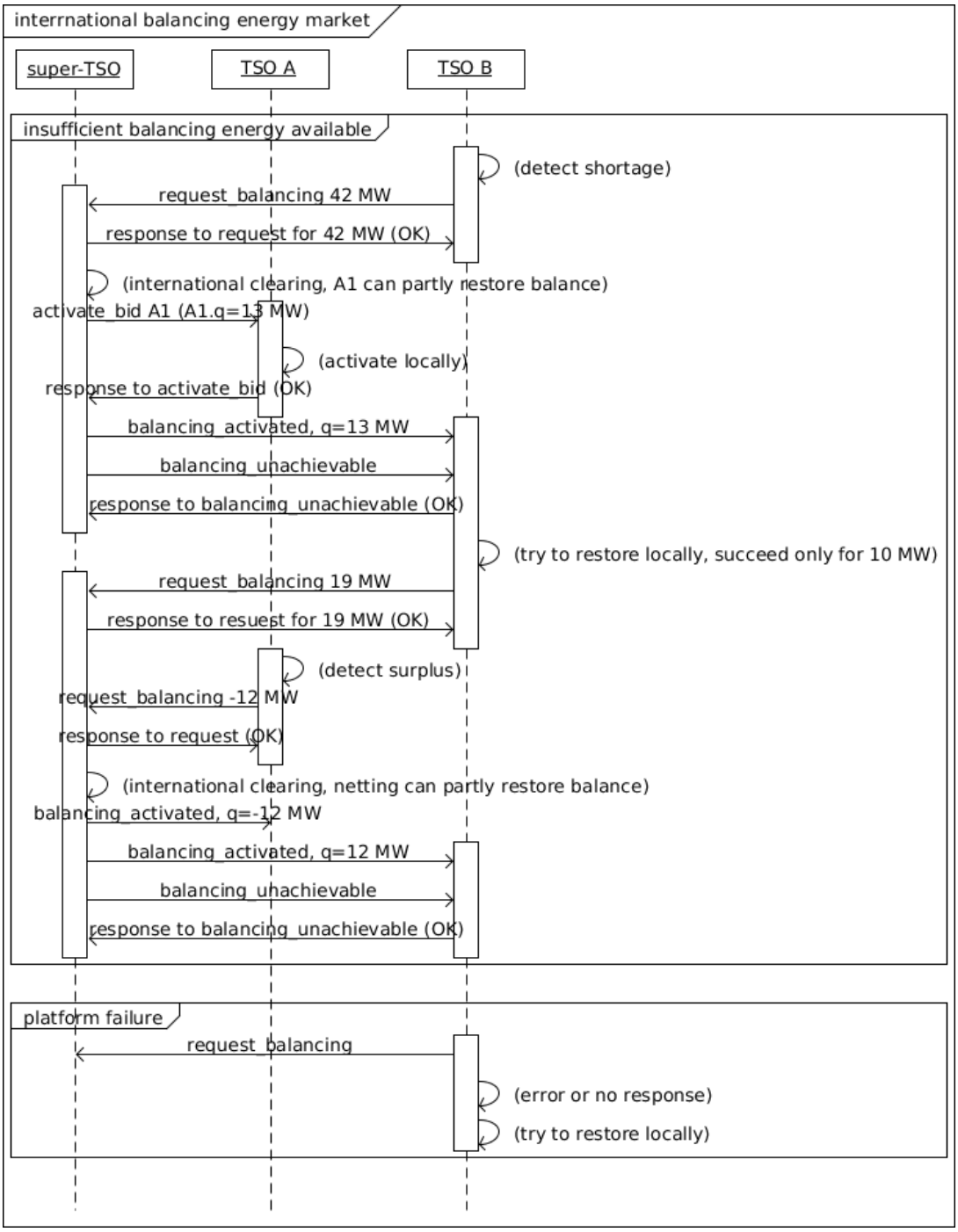

Figure 10: Two examples of unsuccessful balancing request in an international market

6.3.1. Message type: request_balancing

Meaning: notification about imbalance/request to activate balancing energy.

Sent from: TSO **to:** super-TSO.

Field	Description/comment	Example value

msg_id		"e8c99f81-97c2-43a5-90ad-98b7279a4a28"
quantity	quantity to activate; positive for 'shortage/consumption-higher-than-generation/injection-required', negative for 'surplus/consumption-lower-than-generation/withdrawal-required'	-4.5
from	time when balancing is expected to be needed; use null for NOW	"2013-07-24T11:10:20.000Z"
to	time when not needed anymore; use null for UNTIL FURTHER NOTICE	"2013-07-24T11:14:55.000Z"

Raw JSON example: {"msg":"request_balancing", "msg_id":"e8c99f81-97c2-43a5-90ad-98b7279a4a28", "quantity":-4.5, "from":"2013-07-24T11:10:20.000Z", "to":"2013-07-24T11:14:55.000Z"}

When this message is acknowledged by the super-TSO, it is ADDED TO the imbalance (position) of the sending TSO (shortage or surplus) as known by the super-TSO. A request may be cancelled by sending another request_balancing message with the same quantity but opposite sign.

An activation is the same as in the national market, except that they are first sent by the super-TSO to the originating TSO (the TSO that the balancing energy bid came from), which forwards it to the BSP. The acknowledgements are, of course, sent and forwarded in the opposite direction.

A typical message sequence would be:

1. A TSO sends request_balancing.
2. The super-TSO acknowledges.
3. The super-TSO selects (one or more) bid(s) from the common merit order list and activates it/them by sending the appropriate messages.
4. The super-TSO informs the requesting TSO that the request was (partly or fully) granted with (one or more) balancing_activated message(s).

Note that activation of a bid originating from a certain TSO does NOT imply that the balance is restored in its control area; it may have been activated to restore balance anywhere else. If the TSO requesting balancing is the same as the one the selected bid comes from, it will receive both 'activate' and 'balancing_activated' messages.

6.3.2. Message type: balancing_activated

Meaning: informs the destination TSO that its balance is/will be (partly) restored by energy from originating TSO.

Sent from: super-TSO **to:** the destination TSO, i.e. TSO that sent request_balancing.

Field	Description/comment	Example value
quantity	start of activation	-4.5
from	end of activation	"2013-07-24T11:10:20.000Z"
to	originating TSO	"2013-07-24T11:14:55.000Z"
originating_tso	bid ID; null if the imbalance of two TSOs was netted	"TSO-A"
bid_id		"9b42269d-5b80-4029-8cce-79462eaf986e"

Raw JSON example: {"msg":"balancing_activated", "quantity":-4.5, "from":"2013-07-24T11:10:20.000Z", "to":"2013-07-24T11:14:55.000Z", "originating_tso":"TSO-A", "bid_id":"9b42269d-5b80-4029-8cce-79462eaf986e"}

The balancing_activated message implies that the TSO's imbalance/position for this period changes by the specified quantity, i.e. the quantity is added to the TSO's position.

The balancing_activated message has no acknowledgement. The activation will happen anyway, even if e.g. destination TSO's computer is down, thus an acknowledgement would not serve any purpose.

The destination TSO does not know the contents of the bid, but it can save the ID in case of later disputes.

6.3.3. Message type: balancing_unachievable

Meaning: informs a TSO that the referenced balancing request could not be (fully, or at all) fulfilled.

Sent from: super-TSO **to:** the TSO that sent request_balancing that cannot be fully fulfilled.

Field	Description/comment	Example value
msg_id		"79b25a8f-67b1-4303-a980-fc43f7cc6f63"
request_id	the msg_id of request_balancing message that this refers to	"e8c99f81-97c2-43a5-90ad-98b7279a4a28"

Raw JSON example: {"msg":"balancing_unachievable", "msg_id":"79b25a8f-67b1-4303-a980-fc43f7cc6f63", "request_id":"e8c99f81-97c2-43a5-90ad-98b7279a4a28"}

This message may or may not be preceded by balancing_activated messages that partly fulfil the request, but it implies that no later balancing_activated messages will be sent that would balance that request.

After receiving an acknowledgement of this, the super-TSO will assume that the TSO will to restore the remaining imbalance by other means (e.g. with unshared bids). The super-TSO shall not try to restore the remaining imbalance, not even if it could be netted with the imbalance in the other direction requested later by another TSO.

If the TSO that remains imbalanced wants the super-TSO to re-try, it must send a new request_balancing message.

7. Semantic Comparison to OpenADR

In this chapter we compare our data model to OpenADR's. We give schematic overviews for the main four OpenADR services and their eBADGE counterparts. We describe the attributes used at each level in both data models. We also identify gaps of each relative to the other.

OpenADR is a relatively loose standard, thus the message workflows presented here are just one, albeit the most usual, possibility. Likewise, the content of the messages, such as which optional fields are present, is also only loosely described in the documentation. The development process for our VEN and VTN was thus based not only on the standard specification but also on early cross-testing with the EPRI VTN, which is one of the implementations approved by the OpenADR Alliance. After both entities (VTN and VEN) were developed and their integration was tested with various use cases, a comparison with existing eBadge message bus could finally be made.

Please note for the sake of simpler comparison of the two, throughout this chapter we will use the term VEN interchangeably for OpenADR VENs and eBADGE HEHs. Similarly, the term VTN will be used both for OpenADR VTNs and for VPPs or other entities controlling the HEHs over the eBADGE message bus.

7.1 Registration

OpenADR defines VEN registration as part of its data model. The respective message flow is shown in Figure 11. Normally, as soon as the VEN is online it starts the Registration part of the workflow shown, which also lets the VTN know that the VEN is now online.

Nevertheless, part of the registration must be done out-of-band before, otherwise any device or computer could register with any VTN. Both the EPRI VTN that was used as a reference and our VTN implementation require the administrator to enter the VEN ID and VENs with other IDs will not be allowed to register. For adequate security the access control must also be done on the transport level. For HTTPS, only VENs with a trusted client certificates should be allowed, and for XMPP a VEN without the correct credentials will not be able to connect to the XMPP server, and thus also unable to send or receive messages. All of these reduces the point of the *oadrCreatePartyRegistration* message to merely notifying the VTN that the VEN is now online.

Other workflows shown are auxiliary:

- The VTN can request the VEN to re-register. However, without any human intervention it will most likely register in exactly the same way as originally.
- The VEN can query certain VTN's parameters, such as any extensions or the supported transports and profiles. However, the latter two are of little value since the standard declares that the VTN must support both profiles (a and b) and both transports (HTTPS and XMPP).
- The VEN can cancel the registration, effectively logging off the VTN, as opposed to simply going off-line.

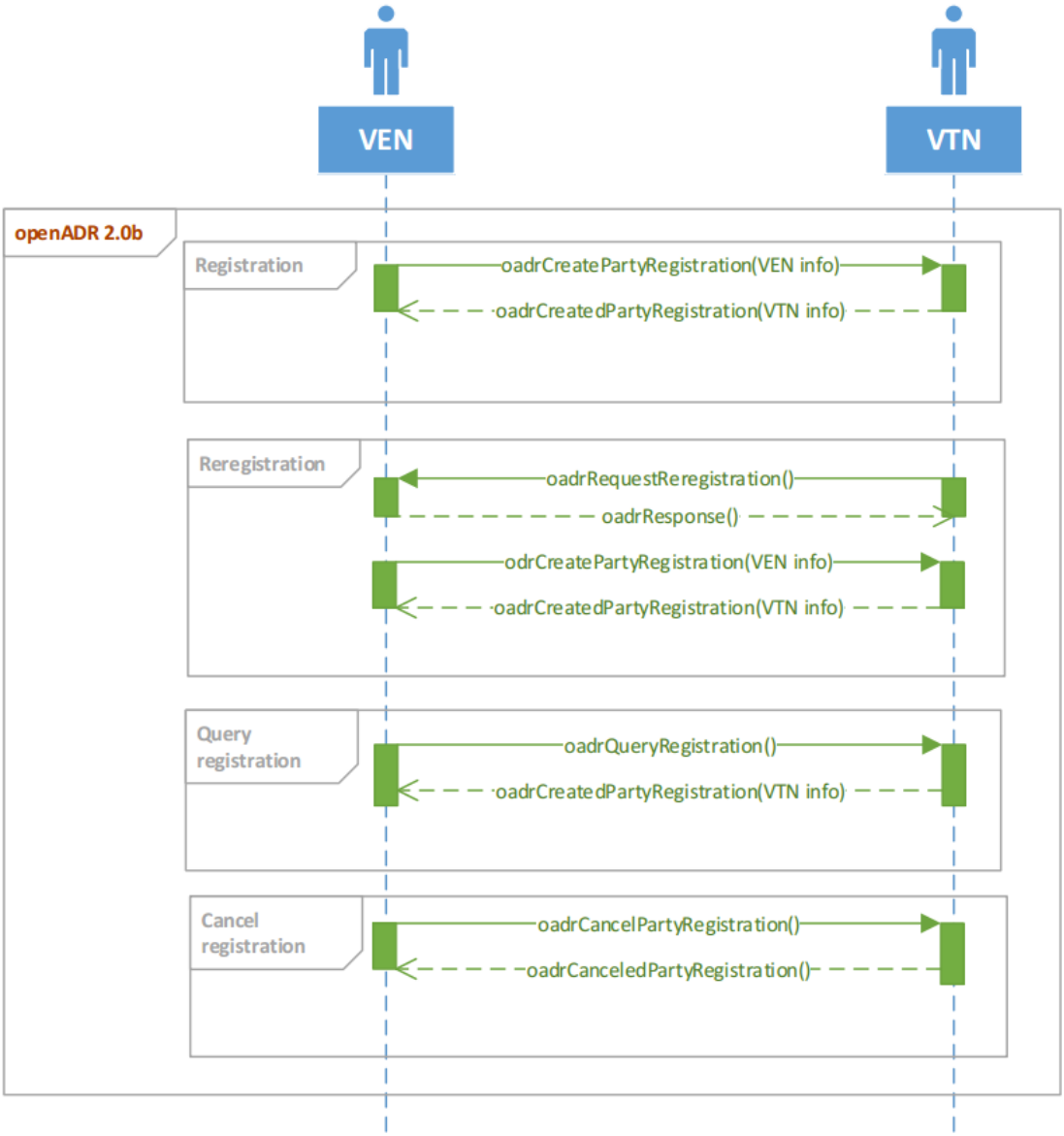

Figure 11: OpenADR VEN registration

The eBADGE data model, on the other hand, does not include registration at all. All access control is performed at the transport level, as described in [2]. The security implications of this are discussed and tested in [3]. However, to let the VTN know that the VEN is online, it sends the *capabilities* message automatically on each boot, as shown in Figure 12.

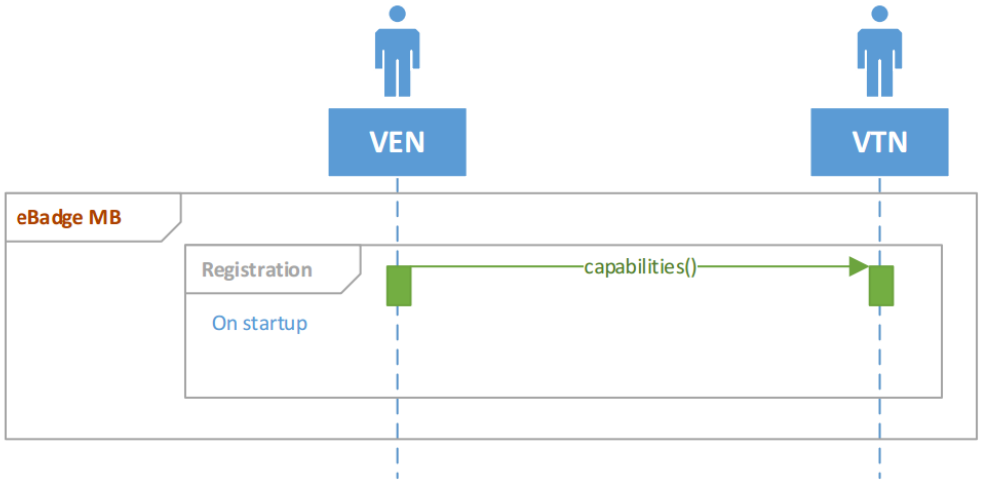

Figure 12: eBADGE HEH startup

This trivial workflow is the equivalent of the Registration workflow in Figure 11. The below table summarizes the differences in the main fields of both and the message sizes. Certain less important fields were left out. Please note that OpenADR message sizes are given as implemented by the VEN and VTN software developed in the project; depending on inclusion of optional fields the size could vary greatly and still be standard-compliant. To a lesser extent both OpenADR and eBADGE message lengths also depend on the chosen identifiers, names etc.

	oadrCreatePartyRegistration	**capabilities**	
Request message size (bytes)	1033	316	
Field in request			**Description**
profile name	oadrProfileName	not needed (no profiles in eBADGE)	the profile (A or B) that the VEN wishes to use
transport name, push vs pull	oadrTransportName, oadrHttpPullModel	not needed (always MB/RabbitMQ with push)	the transport (XMPP or HTTPS) and mode (push or pull) that the VEN wishes to use. Transport is redundant because this message already travels over the chosen transport.
ability to sign messages	oadrXmlSignature	not supported	whether the VEN supports signing messages
ability to control loads	oadrReportOnly	available separately on VTN request, with more detail (get_activation_capacity)	whether the VEN supports just reporting or can also accept events
VEN name	oadrVenName	device_name	
VEN version, serial number, IP and MAC address	not supported	device_version, device_sn, device_ip, device_mac	
list of devices	not supported	devices	the devices, appliances, resources connected to, and meterable/controllable by, the VEN
vendor-defined extensions	yes	yes	
	oadrCreatedPartyRegistration	**(no response in eBADGE)**	
Response message size (bytes)	2825		
Field in response			**Description**
registration ID	registrationID		registration ID assigned by VTN
VTN ID	vtnID		unique VTN ID
supported profiles	oadrProfiles		list of profiles supported by VTN; currently redundant because VTN must support both profiles
vendor-defined extensions	yes		

The other workflows in Figure 11 do not have eBADGE equivalents. The lack of log-out capability similar to oadrCancelPartyRegistration prevents an eBADGE VEN from notifying the VTN that it is about to stop being controllable. The VTN thus has to detect VEN's intended unavailability in the same way as if the VEN would suddenly stop responding, e.g. because of a technical failure.

7.2 Reporting

Reporting capabilities are typically used for providing metering, flexibility data and status reports. The relevant workflows of OpenADR are shown in Figure 13. The VEN first lets the VTN know what kinds of reports it can send and the VTN subscribes to the subset it is interested to. The VEN then continues sending reports until the subscription runs out or is explicitly cancelled by the VTN.

The figure shows the message flow for the cases where VEN can receive messages, e.g., where the XMPP transport is used. In case of HTTPS polling additional *oadrPoll* messages are required.

OpenADR also supports VTN-to-VEN reporting, but we do not see a use case for this in our pilot and have thus not analyzed it in detail.

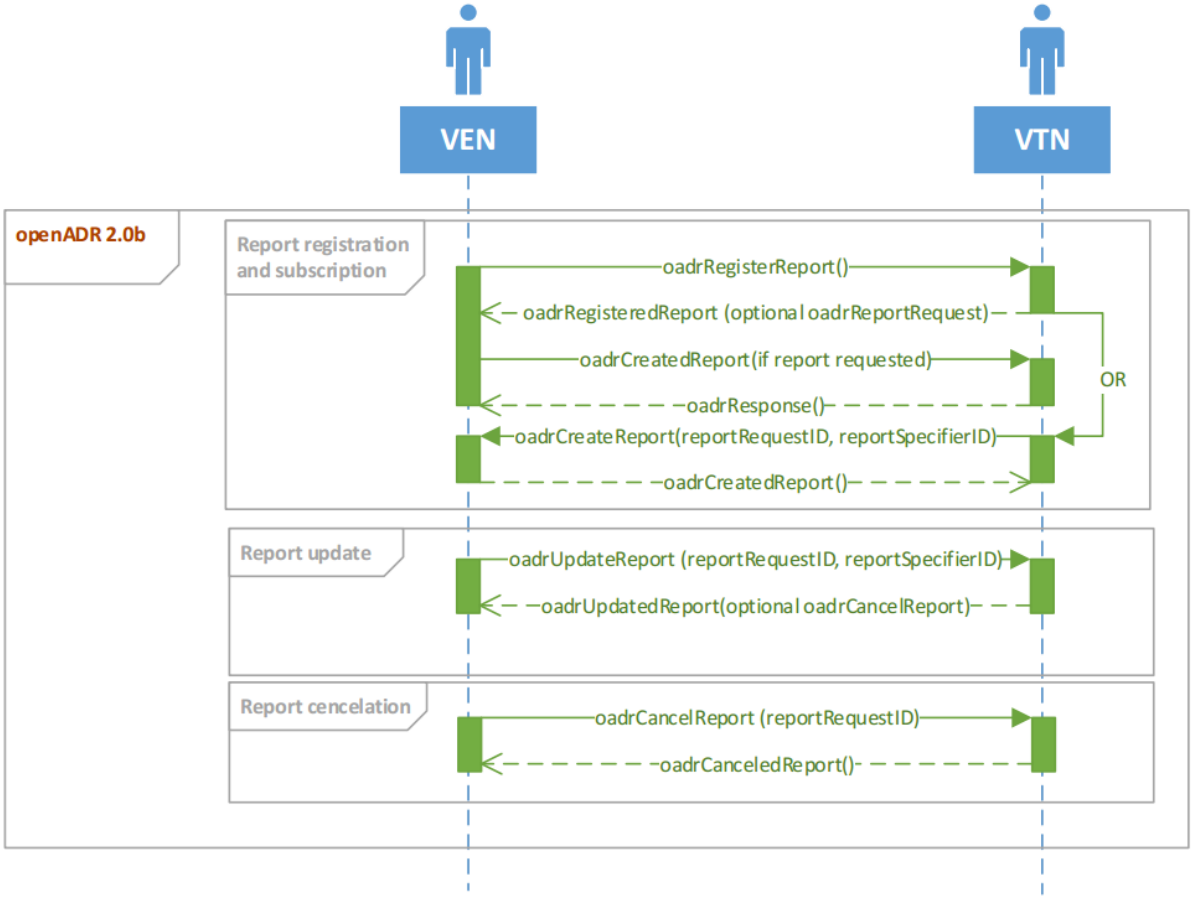

Figure 13: Reporting workflows in OpenADR

The eBADGE workflows are very similar. The two key differences are that in eBADGE data model the VTN can also request report registration with the *get_capabilities* message, rather than just waiting for the VEN to send it, and that the eBADGE report subscriptions never run out by themselves.

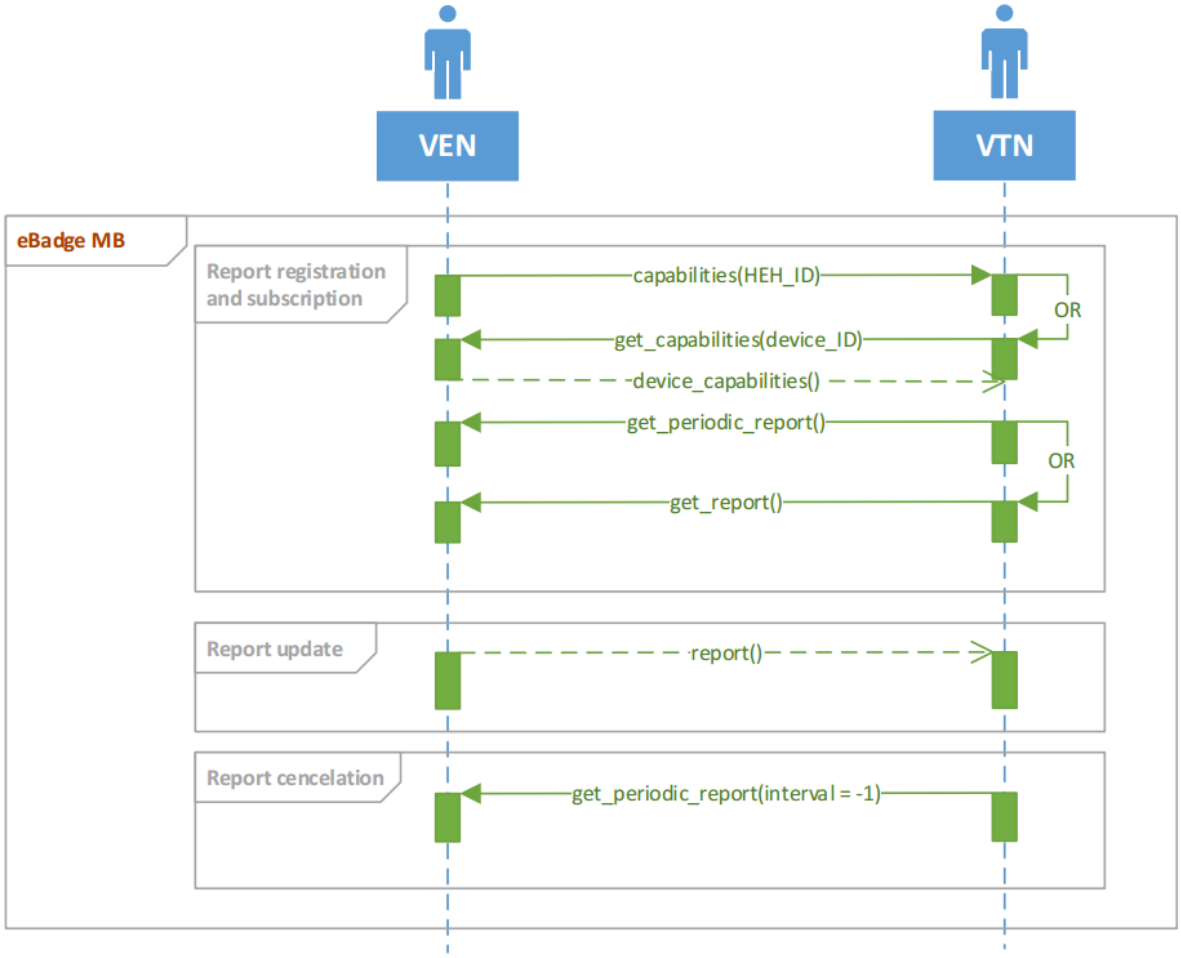

Figure 14: Reporting workflows in eBADGE

The fields of the requests for report registration are given below. The response messages *oadrRegisteredReport* and *oadrCreatedReport* are trivial and do not have eBADGE counterparts, thus their fields are not given in the below tables.

	oadrRegisterReport	**device_capabilities**	
Request message size (bytes)	11.020	573	
Field in request			**Description**
maximum duration of report	duration	not supported (always infinite)	the longest period that can be covered by a single report
minimum and maximum sampling rate	oadrSamplingRate	not supported	limits of sampling rate supported by VEN
measurement type, meaning, and unit	rID, reportType, readingType	signals	machine-readable description of signal
market context	marketContext	not supported	Market contexts are a way to divide a demand-response programme into multiple separate domains, similar to namespaces.
device/resource/appliance the report relates to	resourceID	device	
class of device (consumer, generator, storage)	not supported	classes	see Section 5.4.7 for details
hierarchical description of device type	not supported	type	
descriptive device name	not supported	device_name	
device version	not supported	version	

	oadrRegisteredReport	**(no response in eBADGE)**
Response message size (bytes)	1305	

	oadrCreateReport	**get_periodic_report**	
Request message size (bytes)	4650	254	
Field in request			**Description**
request ID	requestID + one or more reportRequestIDs	request_id	required for later reference (e.g. to cancel the reporting)
type of report	reportSpecifierID	not supported	must be equal to the one specified in oadrRegisterReport
report period	reportBackDuration	interval	period of time to be covered by each individual report
signal resolution	granularity	resolution	
reporting start	reportInterval.dtstart	first_from	beginning of interval that should be covered by the first sent report
reporting duration	reportInterval.duration	not supported; however, just a single report can be requested with the *get_report* message	after this duration the reporting is stopped
signals	specifierPayload (can appear multiple times)	signals (array)	list of signals to be reported
	oadrCreatedReport	**(no response in eBADGE)**	
Response message size (bytes)	1327		

The most frequent messages are the regular metering reports. The listed message size was observed with reports that covered 6 signals over a 3-minute period with 1-minute temporal resolution, thus containing the net payload of 18 floating-point values. The response *oadrUpdatedReport* contains no significant fields and is thus not detailed.

	oadrUpdateReport	**report**	
Request message size (bytes)	10.375	343	
Field in request			**Description**
start	dtstart	from	start of period covered by report
end	reportSpecifierID	to	end of period covered by report
resolution	N/A (given inside each interval, see field intervals)	resolution	
signals	intervals (array; each interval contains start time, duration, and all signal values)	map of values (for each signal contains an array of temporally equidistant values)	the actual signal values reported
report request ID	reportRequestID	not supported	as assigned by VTN in oadrCreateReport
	oadrUpdatedReport	**(no response in eBADGE)**	
Response message size (bytes)	1355		

Finally, report cancellation differs the least between the two data models.

	oadrCancelReport	**get_periodic_report**	
Request message size (bytes)	1358	254	
Field in request			**Description**
ID of report to cancel	reportRequestID	request_id	as assigned at subscription
report following	reportToFollow	not supported	true if VEN should send one final report after cancellation
other		interval; must be -1 to signal cancellation	in eBADGE, cancellation is marked by negative interval rather than by special message type

	oadrCanceledReport	**(no response in eBADGE)**
Response message size (bytes)	1127	

The eBADGE data model contains separate messages for reporting exceptional energy-related events, such as overvoltages (messages *get_energy_events, energy_events*) and the status of the HEH (*get_status_report, status_report*). In OpenADR these are implemented by various types of reports.

7.3 Events

Event is a general OpenADR term that covers the activation of demand-response resources, the distribution of price changes, and more. We can only directly compare here the case of demand-response activation request, which is the only type of event currently used in the eBADGE pilot. The OpenADR flow, shown in Figure 15, is very simple: the VTN distributes information about the event, the VEN acknowledges it and informs the VTN whether it will participate in the event.

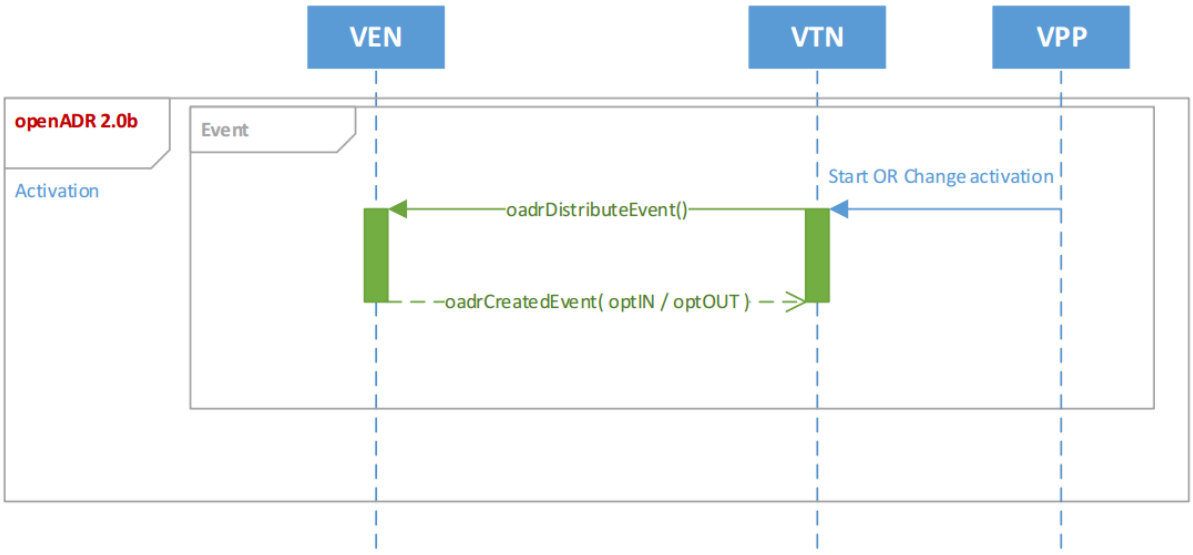

Figure 15: Activations in OpenADR

The eBADGE HEH has more options: it can accept the activation request as is, reject it, or reject it but suggest that it could accept a modified activation. There is also a contingency activation meant for urgent cases, whereby the VEN should reduce the consumption as much as possible.

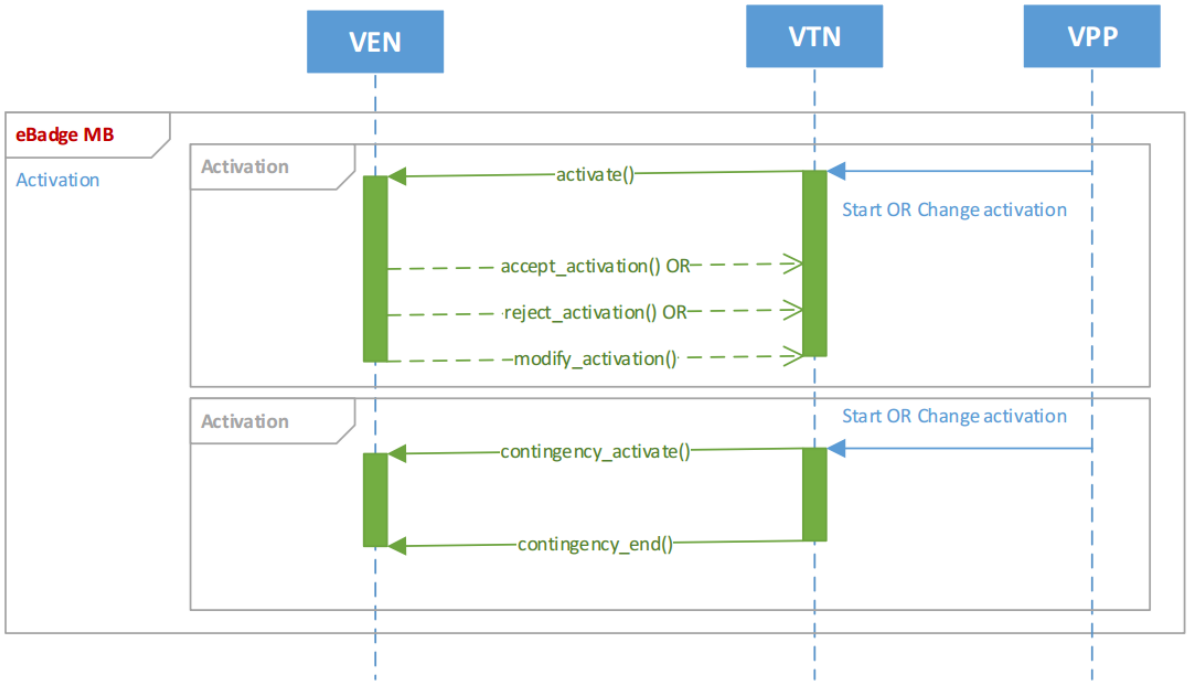

Figure 16: Activations in eBADGE

The comparison of both is given below. Note that both eBADGE responses given correspond to the *activate* messages and the *contingency_activate* has no response. Note also that a single *oadrDistributeEvent* message can carry multiple events, although in the residential demand-response this will typically not be the case.

	oadrDistributeEvent	**activate**	**contingency_activate**	
Request message size (bytes)	4670	239	164	
Field in request				**Description**
event ID	eventID + modificationNumber	id + modification_count	id	e.g. for premature cancellation
priority	priority	not supported	implicitly highest priority	in case of conflicting events, VEN must follow higher priority
market context	eiMarketContext	not supported	not supported	Market contexts are a way to divide a demand-response programme into multiple separate domains, similar to namespaces.
creation time	createdDateTime	not supported	not supported	time when event was created at VTN
status of event	eventStatus	not supported	not supported	e.g. event already completed
test event	testEvent	not supported directly; use quantity = 0	not supported	true if this is test-only event
active period	eiActivePeriod (contains fields dtstart and duration)	from, to	from, to	
signal	intervals (array; each interval contains e.g. a float value, signal name, target, and duration)	device + quantity	max_quantity	the “command”, or description what VEN should do during event
	oadrCreatedEvent	**option 1: accept_activation, reject_activation**	**option 2: modify_activation**	
Response message size	2130	96	201	**Description**

(bytes)				
event ID	eventID	id + modification number	id + modification number	must match eventID from oadrCreateEvent
acceptance status	optType (optIn or optOut)	implied from message type (accept, reject)	implied reject	whether VEN will proceed with event, e.g. shed load
event modification	not supported		from, to, quantity, device	suggestion for alternative event that VEN would accept

The device and quantity in eBADGE activations simply mean “please reduce/enlarge (depending on sign of quantity) energy usage of this device by this much”. If device is omitted, the HEH can decide which device(s) to apply the activation to.

In OpenADR, on the other hand, the semantics of events are not defined and depend on the specifics of the demand response programme. In other words, both parties have to reach a separate agreement about what kinds of events the VTN will distribute and what the VEN should do with them. For the purpose of our comparison the demand response programme uses the signal name LOAD_*CONTROL*, the target corresponds to the device in an eBADGE *activate* message and the float value corresponds to the quantity.

OpenADR is also more flexible in that a single event can have a non-constant profile. For example, a VEN could be asked to decrease energy usage by 2 kW for the first hour and by 3 kW in the second hour. In eBADGE data model this requires two activations with no way for the VTN to demand that the VEN opts either in or out of both of them. However, in our pilot it never happened that such flexibility would be needed.

The eBADGE data model has a separate set of messages for conveying energy pricing to the HEH (*load_price, generation_price, get_all_prices*). In OpenADR these are implemented as special kinds of events.

7.4 Opt Service

OpenADR also has the opt service, through which the VEN can let the VTN know in advance when it will or will not be available for events. This is typically used to convey human knowledge about future (un)availabilities to the VTN and is shown in Figure 17.

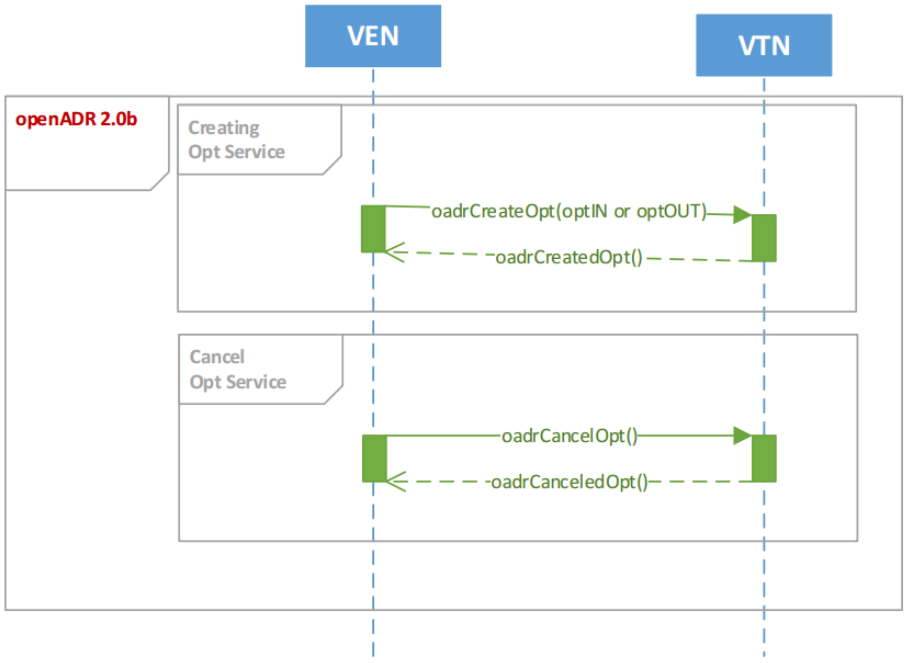

Figure 17: Opt service in OpenADR

The eBADGE data model has no directly comparable capability. The VTN can only inquire the VEN about its current availability by sending it a *get_activation_capacity* request message.

8. Conclusions

The requirements analysis has shown that the communication and data model can be divided into two levels:

- HEH-level, where the metering data is collected, individual demand-response commands are sent etc.,
- market level, where the BSPs send balancing energy bids and the TSOs activate selected ones as needed.

At the beginning stages of the project we decided to develop our own data model rather than use the OpenADR 2.0, which was just being standardized at the time, because the latter did not seem to cover all our use cases. OpenADR 2.0 was later widely adopted, particularly in the USA, and the documentation was improved, thus we adopted it in parallel to our data model for the HEH level.

The analysis of IEC 61850 has shown that it is not aligned to our requirements closely enough to warrant implementation, particularly as the most relevant parts of the standard are only now being developed. All other standards from the Smart Grid domain are even less appropriate for reasons of inextensibility, lack of fitness for our objectives, or dependency on inbound connectivity into the home energy hubs. On the market level, no suitable standards exist either. The existing markets, such as the Slovenian intra-day market, use the data models and exchange formats defined by the respective vendors of the market software.

After the definition of a new data model, JSON encoding was chosen as a suitable compromise between compactness, human-readability, interoperability and extensibility. The detailed descriptions and examples of all message types for the final version of the eBADGE data model are given in this document, accompanied by sequence diagrams for the more complex communication scenarios. The data model covers the specific requirements of the eBADGE pilot but the implementation strives to be applicable beyond this project and allows third-party extensions.

The parallel adoption of OpenADR allowed direct comparisons between the two. In this document we are concerned strictly with the message content and meaning in both. The biggest semantical difference is that the eBADGE data model prescribes strict meanings to the messages, i.e. it is apparent from the specification what actions the message receiver should do. OpenADR, on the other hand, is more vague and depends on the specifics being agreed to separately between the parties. OpenADR also uses the advantages of XML encoding, such as schema-based validation and native namespace support, which our JSON-based data model cannot. However, the eBADGE data model messages are at least an order of magnitude shorter.

References

[1] eBADGE Project, “The eBADGE Data Model Report - Intermediate version (deliverable 3.1.2),” 2014.

[2] eBADGE Project, “The eBADGE Message Bus – Final version (deliverable 3.2.3),” 2015.

[3] eBADGE Project, “The eBADGE ICT Interfaces Testing report - Final version,” 2015.

[4] eBadge Project, “Prototype of a Home Energy hub and home energy Cloud - Final version (deliverable 4.2.2),” 2015.

[5] S. Coe, “Comparison of Demand Response Communication Protocols,” 2011. [Online]. Available: http://www.nrcan.gc.ca/sites/www.nrcan.gc.ca/files/canmetenergy/files/pubs/2011-061_e.pdf.

[6] J.-F. Martínez, J. Rodríguez-Molina, P. Castillejo and R. d. Diego, “Middleware Architectures for the Smart Grid: Survey and Challenges in the Foreseeable Future,” *Energies,* vol. 6, pp. 3593-3621, 2013.

[7] Lawrence Berkeley National Laboratory, “Open Automated Demand Response Communication Specification (Version 1.0),” [Online]. Available: http://openadr.lbl.gov/pdf/cec-500-2009-063.pdf.

[8] J. S. John, “In California, Building Codes and Demand Response Align,” 2013. [Online]. Available: http://www.greentechmedia.com/articles/read/in-california-building-codes-and-demand-response-align.

[9] OpenADR Alliance, “OpenADR 2.0 Profile Specification – B Profile,” 2013. [Online]. Available: http://www.openadr.org/specification.

[10] OpenADR Alliance, “List of OpenADR 2.0 Certified Products,” [Online]. Available: http://www.openadr.org/products.

[11] Universal Devices, “ISY-994i series home automation solution,” 2014. [Online]. Available: https://www.universal-devices.com/residential/isy994i-series/.

[12] K. Schwarz, “News on IEC 61850 and related Standards. Compilation of the IEC 61850 blog until August 4th, 2013,” [Online]. Available: http://www.nettedautomation.com/download/Newsletter/IEC61850-Blog_until_2013-08-05_o.pdf.

[13] IEC, “Core IEC Smart Grid Standards,” [Online]. Available: http://www.iec.ch/smartgrid/standards/.

[14] S. Sučić, A. Martinić and A. Kekelj, “Utilizing standards-based semantic services for modeling novel Smart Grid supervision and remote control frameworks,” in *IEEE International Conference on Industrial Technology*, Athens, 2012.

[15] IEC, “IEC standards,” [Online]. Available: http://www.iec.ch/smartgrid/standards/.

[16] R. E. Mackiewicz, “Overview of IEC 61850 and Benefits,” in *Power Engineering Society General Meeting*, 2006.

[17] IEC, “IEC 61850-8-1 Communication networks and systems in substations,” 2014.

[18] J. Zhang and C. A. Gunter, “IEC 61850 – Communication Networks and Systems in Substations: An Overview of Computer Science,” [Online]. Available: http://seclab.illinois.edu/wp-content/uploads/2011/03/iec61850-intro.pdf.

[19] IEC, “IEC 61850-7-420, Communications systems for Distributed Energy Resources (DER) - Logical nodes”.

[20] N. Etherden, “Communication Requirements of a Virtual Power Plant using IEC 61850 to provide Grid Services,” in *IEEE SmartGridComm*, 2013.

[21] IEC, “IEC 61850-7-2 Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI),” [Online]. Available: http://read.pudn.com/downloads93/doc/comm/365472/IEC61850-7-2.pdf.

[22] B. M. Buchholz, C. Brunner, A. Naumann and A. Styczynski, “Applying IEC standards for communication and data management as the backbone of smart distribution,” in *IEEE PES General Meeting*, San Diego, 2012.

[23] Green eMotion Project, “Business Analysis (D3.1),” 2012.

[24] Vattenfall Europe Wärme AG, “VHP Ready (Virtual Heat & Power Ready), Technical requirements specification,” 2012. [Online]. Available: http://www.vattenfall.de/de/file/VHP-READY-3.0-englisch.pdf_30408907.pdf.

[25] IEC, “IEC 61850-90-10 Object Models for Scheduling”.

[26] Gorenje, d.d., “iGorenje specification v1.5 (classified),” 2012.

[27] ETSI, “Open Smart Grid Protocol (OSGP),” 2012. [Online]. Available:

http://www.etsi.org/deliver/etsi_gs/OSG/001_099/001/01.01.01_60/gs_OSG001v010101p.pdf.

[28] OPC Foundation, "OPC Unified Architecture," [Online]. Available: http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC.

[29] OPC Foundation, OPC Unified Architecture Specification, Part I: Overview and Concepts, Release 1.01, 2009.

[30] "OpenADR Version 2 – Potential Transport Mechanisms," 2011. [Online]. Available: http://cio.nist.gov/esd/emaildir/lists/h2g_interop/pdf00007.pdf.

[31] BSP d.o.o., "Trgovanje znotraj dneva [intra-day market]," [Online]. Available: http://www.bsp-southpool.com/trgovanje-znotraj-dneva.html.

[32] Ney York Stock Exchange, "NYSE UTPDirect (CCG Binary) API Specification – NYSE and NYSE MKT Cash Equities Markets," 2012. [Online]. Available: http://usequities.nyx.com/connecting/ccg-binary-api.

[33] OASIS Open, "Energy Interoperation Version 1.0 - an OASIS Standard," 11 June 2014. [Online]. Available: http://docs.oasis-open.org/energyinterop/ei/v1.0/os/energyinterop-v1.0-os.pdf.