OpenADR 2.0
Profile Specification
A Profile

Revision Number: 1.0
Document Status: Completed
Document Number: 20110712-1

Contact:

OpenADR Alliance
275 Tennant Avenue, Suite 202
Morgan Hill, CA 95037
help@openadr.org

Legal Notice

CONTENTS

OPEN AUTOMATED DEMAND RESPONSE

OpenADR 2.0 Profile Specification

FOREWORD

The development of the **Open Automated Demand Response Communications Specification**, also called OpenADR, began in 2002 following the California electricity crisis. The California Energy Commission Public Interest Energy Research Program funded an OpenADR research program through the Demand Response Research Center (DRRC) at Lawrence Berkeley National Laboratory (LBNL). OpenADR development began in 2002 to support California's energy policy objectives to move toward dynamic pricing to improve the economics and reliability of the electric grid. Initial field tests focused on automating a number of event-based DR utility programs for commercial and industrial (C&I) customers. The DRCC research set out to determine if today's communications and information technologies could be used to automate Demand Response (DR) operations using standardized electricity price and reliability signals. This research, development, and deployment have led to commercial adoption of OpenADR. Today, utilities and governments worldwide are using OpenADR to manage the growing demand for electricity and peak capacity of the electric systems. This low cost communications infrastructure is used to improve the reliability, repeatability, robustness, and cost-effectiveness of DR.

OpenADR is a fundamental element of U.S. Smart Grid interoperability standards being developed to improve optimization between electric supply and demand. OpenADR is designed to facilitate automated DR actions at the customer location, whether it involves electric load shedding or shifting. OpenADR is also designed to provide continuous dynamic price signals such as hourly day-ahead or day-of real time pricing. OpenADR has been field tested and deployed in a number of DR programs in U.S and worldwide. While the scope of OpenADR focuses on signals for DR events and prices, significant work focuses on DR strategies and techniques to automate DR within facilities. OpenADR interacts with facility control systems that are pre-programmed to take action based on a DR signal, enabling a response to a DR event or a price to be fully automated, with no manual intervention.

The DRCC OpenADR 1.0 specification was donated to the Organization of Structured Information Standards (OASIS) to create a national standard for OpenADR. The OASIS' Energy Interoperation (EI) Technical Committee (TC) developed a standard to describe "an information model and a communication model to enable collaborative and transactive use of energy, service definitions consistent with the OASIS SOA Reference Model [SOA-RM], and XML vocabularies for the interoperable and standard exchange of dynamic price signals, reliability signals, emergency signals, communication of market participation information such as bids, load predictability and generation information. Considering that the goal of OASIS EI TC was more than DR and Distributed Energy Resources (DER), the EI TC created profiles within the EI Version 1.0 standard for specific applications within the Smart Grid. The OpenADR Alliance used the OpenADR profile as the basis for this OpenADR 2.0 Profile Specification, which is referred to OpenADR 2.0 in this document. OpenADR 2.0 utilizes the EI Version 1.0 standard to create profiles that are specific to OpenADR applications for DR and Distributed Energy Resources (DER), while keeping in mind the requirements of the diverse market and stakeholder needs.

INTRODUCTION

Development of the Demand Response (DR) market has resulted in a transition from manual DR to OpenADR in Automated DR (Auto-DR) programs. As of 2012, over 250 MW was enrolled in California commercial and industrial customers Auto-DR programs using OpenADR 1.0.[1] DR is defined as "…action taken to reduce electricity demand in response to price, monetary incentives, or utility directives so as to maintain reliable electric service or avoid high electricity prices." [2] OpenADR was developed to support Auto-DR programs and California's energy policy objectives to move toward dynamic pricing to improve the economics and reliability of the electric grid. The recent developments has expanded the use of OpenADR to meet diverse market needs such as ancillary services (Fast DR), dynamic prices, intermittent renewable resources, supplement grid-scale storage, electric vehicles, and load as a generation. For example, with real-time price information, an automated client within the customer facility can be designed to continuously monitor these prices and translate this information into continuous automated control and response strategies. This rationale is the fundamental element of the United States (U.S.) Smart Grid interoperability standards, which are developed to improve dynamic optimization of electric supply and demand.

OpenADR Communications have the following defining features:

- **Continuous, Secure, and Reliable** - Provides continuous, secure, and reliable two-way communications infrastructures where the end points at the end-use site receive and acknowledge the receipt of DR signals from the energy service providers.
- **Translation** - Translates DR event information to continuous Internet signals to facilitate DR automation. These signals are designed to interoperate with energy management and control systems, lighting, or other end-use controls.
- **Automation** - Receipt of the external signal is designed to initiate automation through the use of pre-programmed demand response strategies determined and controlled by the end-use participant.
- **Opt-Out** - Provides opt-out or override function to any participants for a DR event if the event comes at a time when changes in end-use services are not desirable.
- **Complete Data Model** - Describes a rich data model and architecture to communicate price, reliability, and other DR activation signals.
- **Scalable Architecture** - Provides scalable communications architecture to different forms of DR programs, end-use buildings, and dynamic pricing.
- **Open Standards** - Open standards-based technology such as Internet Protocol (IP) and Web services form the basis of the communications model.

OpenADR is a "communications data model," which facilitates information exchange between two end-points, the electricity service provider and the customer. It is not a protocol that specifies "bit-structures" as some communications protocols do, but instead relies upon existing open standards such as eXtensible Mark-up Language (XML) and Internet Protocol (IP) to as the lingua franca and framework for exchanging DR signals. In some references the term "system," "technology," or "service" is used to refer to the features of OpenADR.

OpenADR is designed to facilitate automation of DR actions at the customer location, whether it involves electric load shedding or load shifting. We are often asked if the communications data model can be used for continuous operations. The answer is **yes**. Many emergency or reliability DR events occur at specific times when the electric grid is strained. The OpenADR

---

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

[2] U.S. Federal Energy Regulatory Commission (FERC), 2007 Assessment of Demand Response and Advanced Metering, Staff Report, available: http://www.ferc.gov/legal/staff-reports/09-07-demand-response.pdf

communications are designed to coordinate such signals to facility control systems (commercial, industrial, and residential). OpenADR is also designed to provide continuous dynamic price signals such as hourly day-ahead or day-of real time pricing. With such price information an automated client can be configured to continuously monitor these prices and translate this information into continuous automated control and response strategies within a facility. Several reports present the history of OpenADR 1.0 research.[3] This OpenADR 2.0 profile specification covers the signalling data models for price and reliability signals to both wholesale and retail markets in the U.S..

OpenADR provides the following benefits:

- **Open Specification**–Provides a standardized DR communications and signalling infrastructure using open, non-proprietary, industry-approved data models that can be implemented for both dynamic prices and DR emergency or reliability events.
- **Flexibility**–Provides open communications interfaces and protocols that are flexible, platform-independent, interoperable, and transparent to end-to-end technologies and software systems.
- **Innovation and Interoperability**–Encourages open innovation and interoperability, and allows controls and communications within a facility or enterprise to build on existing strategies to reduce technology operation and maintenance costs, stranded assets, and obsolesce in technology.
- **Ease of Integration**–Facilitates integration of common Energy Management and Control Systems (EMCS), centralized lighting, and other end-use devices that can receive Internet signals (such as XML).
- **Supports Wide Range of Information Complexity –** Can express the information in the DR signals in a variety of ways to allows for systems ranging from simple end devices (e.g. thermostats) to sophisticated intermediaries (e.g. aggregators) to receive the DR information that is best suited for its operations.
- **Remote Access**– Facilitates opt-out or override functions for participants to manage standardized DR-related operation modes to DR strategies and control systems.

The OpenADR Alliance is the primary authority for the development and adoption of OpenADR, leveraging the OpenADR 1.0 activities and OASIS Energy Interoperation (EI) Technical Committee's Version 1.0 standard.[4] The OpenADR profiles within OASIS EI Version 1.0 standard are utilized for OpenADR 2.0 profile specifications and are referenced as appropriate to provide specific OpenADR applications for DR and Distributed Energy Resources (DER), while keeping in mind the requirements of the diverse market, product, and stakeholder needs.

---

[3] These reports are available at http://drrc.lbl.gov/drrc-pubsall.html:

Piette, M.A., S. Kiliccote, G. Ghatikar, Design and Implementation of an Open, Interoperable Automated Demand Response Infrastructure, Proceedings of the Grid-Interop Forum, October 2007, LBNL-63665.

Koch, E., M.A. Piette, Architecture Concepts and Technical Issues for an Open, Interoperable Automated Demand Response Infrastructure. Proceedings of the Grid-Interop Forum,. October 2007. LBNL-63664.

Piette, M.A, D. Watson, N. Motegi, S. Kiliccote Automated Critical Peak Pricing Field Tests: 2006 Pilot Program Description and Results. August, 2007. LBNL-62218.

Motegi, N., M.A. Piette, D.S. Watson, S. Kiliccote, P. Xu. Introduction to Commercial Building Control Strategies and Techniques for Demand Response. May 2007. LBNL-59975.

[4] Energy Interoperation OASIS Committee Specification, Energy Interoperation Version 1.0, December 2011. http://www.oasis-open.org/committees/download.php/44364/energyinterop-v1.0-csprd03.zip

1   Scope

The OpenADR 2.0 profile specification is a flexible data model to facilitate common information exchange between electricity service providers, aggregators, and end users. The concept of an open specification is intended to allow anyone to implement the two-way signalling systems, providing the servers, which publish information (Virtual Top Nodes or VTNs) to the automated clients, which subscribe the information (Virtual End Nodes, or VENs).

This OpenADR 2.0 profile specification covers the signalling data models between VTN and VEN (or VTN/VEN pairs) and does include information related to specific DR electric reduction or shifting strategies, which are taken at the facility. In particular, OpenADR 2.0 supports the following services from OASIS EI Version 1.0 standard or subset thereof. Extensions to these services are included to meet the DR stakeholder and market requirements:

1. **Registration (EiRegisterParty)**: Register is used to identify entities such as VEN's and parties. This is necessary in advance of an actor interacting with other parties in various roles such as VEN, VTN, tenderer, and so forth.
2. **Enrollment (EiEnroll)**: Used to enroll a Resource for participation in DR programs. This establishes a relationship between two actors as a basis for further interactions. (Planned for future releases)
3. **Market Contexts (EiMarketContext)**: Used to discover program rules, standard reports, etc. Market contexts are used to express market information that rarely changes, and thereafter need not be communicated with each message. (Planned for future releases)
4. **Event (EiEvent)**: The core DR event functions and information models for price-responsive DR. This service is used to call for performance under a transaction. The service paramaters and event information distinguish different types of events. Event types include reliability events, emergency events, and more – and events MAY be defined for other actions under a transaction.
5. **Quote or Dynamic Prices (EiQuote)**: EiDistributeQuote for distributing complex dynamic prices such as block and tier tariff communication. These are sometimes referred to as *price signals*; such signals are indications of a possible tender price – they are not themselves actionable.
6. **Reporting or Feedback (EiReport)**: The ability to set periodic or one-time information on the state of a Resource (response).
7. **Availability (EiAvail)**: Constraints on the availability of Resources. This information is set by the end node and indicates when an event may or may not be accepted and executed by the VEN with respect to a Market Context. Knowing the Availability and Opt information for its VENs improves the ability of the VTN to estimate response to an event or request. (Planned for future releases)
8. **Opt or Override (EiOpt)**: Overrides the EiAvail; addresses short-term changes in availability to create and communicate Opt-in and Opt-out schedules from the VEN to the VTN.

These OpenADR 2.0 services in this specification provide information that is pertinent to DR, pricing, and DER communication requirements. These services make no assumption on specific DR electric load control strategies within the resource or market-specific contractual or business agreements between electricity service providers and their customers.

OpenADR uses an application-level data model, which is independent of transport mechanisms. For the purposes of interoperability, OpenADR 2.0 provides basic transport mechanisms and their relevant interaction patterns (e.g. PUSH information vs. PULL information) to address different stakeholder needs.

OpenADR 2.0 specifies the necessary level of security that is essential to meet the U.S. Cyber Security requirements for such purposes as data confidentiality, integrity, authentication and message-level security. Such security requirements are essential for non-repudiation and to mitigate any resulting Cyber Security risks.

OpenADR 2.0 provides a clear set of mandatory and optional attributes within each of the services to meet the broader interoperability, testing and certification requirements, while creating feature-sets with different product profiles to address today's market needs as well as future requirements that are closely aligned to meet OpenADR goals and national interoperability requirements for Smart Grid standards.

The different product certification levels for OpenADR include *OpenADR 2.0a*, *OpenADR 2.0b*, and *OpenADR 2.0c* profiles, and their services (all from OASIS EI 1.0 standard) and transport mechanisms for interaction between VTN and VEN are summarized in Table 1 below and are described later in more detail:[5]. NOTE that the OpenADR 2.0b and 2.0c profiles are not completed yet and are only included in this table for reference. For the final 2.0b and 2.0c features, please refer to the respective specifications.

	Device Types & Profile	VTN			VEN		
		A	**B**	**C**	**A**	**B**	**C**
Services and Functions Support							
Operational Services							
EiEvent							
Limited Profile (this specification)		M	M	M	M	M	M
Full Profile		NA	M	M	NA	M	M
EiQuote							
Full Profile		NA	NA	M	NA	NA	M
Transactional		NA	NA	M	NA	NA	M
EiOpt							
Full Profile		NA	M	M	NA	M	M
EiReport							
Full Profile		NA	M	M	NA	M	M
EiRegisterParty							
Full Profile		NA	M	M	NA	M	M
Transport Protocols							
Simple HTTP		M	M	M	M	M	M
XMPP		O	O	O	O	O	O
Security Levels							
Standard		M	M	M	M	M	M
High		O	O	O	O	O	O
		M - Mandatory		O - Optional	NA - Not available for profile		

Figure 1: OpenADR 2.0 Certification Levels

2   Normative References

- [**OASIS EI 1.0**]: Energy Interoperation OASIS Committee Specification, Energy Interoperation Version 1.0, December 2011. http://www.oasis-open.org/committees/download.php/44364/energyinterop-v1.0-csprd03.zip

- [**OASIS EMIX 1.0**]: **EMIX** OASIS Committee Specification Draft 04, *Energy Market Information Exchange 1.0*, September 2010. http://docs.oasis-open.org/emix/emix/v1.0/csd04/emix-v1.0-csd04.html

- [**OASIS WS-Calendar**]: WS-Calendar OASIS Committee Specification 1.0, WS-Calendar, July 2011, http://docs.oasis-open.org/ws-calendar/ws-calendar-spec/v1.0/cs01/ws-calendar-spec-v1.0-cs01.pdf

---

[5] M indicates mandatory requirement while NA indicates not applicable

- **[RFC2246]**   T. Dierks, C. Allen *Transport Layer Security (TLS) Protocol Version 1.0*, http://www.ietf.org/rfc/rfc2246.txt, IETF RFC 2246, January 1999.

- [**SOA-RM**]   SOA-RM OASIS Standard, *OASIS Reference Model for Service Oriented Architecture 1.0*, October 2006 http://docs.oasis-open.org/soa-rm/v1.0/

- [**RFC2616**]   R. Fielding *Hypertext Transfer Protocol -- HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt, IETF RFC 2616, June 1999.

- [**RFC6120**]   P. Saint-Andre *Extensible Messaging and Presence Protocol (XMPP): Core Version 1.0*, http://www.ietf.org/rfc/rfc6120.txt, IETF RFC 6120, March 2011.

- **[RFC6122]**   P. Saint-Andre *Extensible Messaging and Presence Protocol (XMPP): Address Format*, http://www.ietf.org/rfc/rfc6122.txt, IETF RFC 6122, March 2011.

3   Non-Normative References

- UCA OpenSG OpenADR security profile

- IRC and NAESB requirements/use-cases

- NIST Special Publication 800-131A

- Annex A – Registration Scenarios

4   Terms and Definitions

The OpenADR Alliance: The OpenADR Alliance is comprised of industry stakeholders that are interested in fostering the deployment of low-cost price- and reliability-based demand response communication protocol by facilitating and accelerating the development and adoption of OpenADR standards and compliance with those standards. These include de facto standards based on specifications published by LBNL in April 2009, as well as Smart Grid-related standards emerging from OASIS, UCAIug, NAESB, and IRC.

OpenADR 2.0 Profile Specification: The OpenADR 2.0a, b, or c Profile Specifications provide specific implementation related information in order to build an OpenADR enabled device or system. Developers shall use the Profile Specification in conjunction with the schemas, sample payloads, PICS and test plans.

OASIS Energy Interoperation (EI): Energy Interoperation standard describes information and communication model to coordinate energy supply, transmission, distribution, and use, including power and ancillary services, between any two parties, such as energy suppliers and customers, markets and service providers, in any of the domains defined in the Smart Grid. The EI 1.0 standard was used as a basis for OpenADR 2.0 Profile Specification.

Demand Response: A mechanism to manage customer load demand in response to supply conditions, such as prices or availability signals.

Slow DR: Demand Response where the signals are sent significantly before the events are called, such as day-ahead.

Fast DR: Fast Demand Response or Fast DR refers to programs that require a (much) faster than usual response time. While typical peak shaving DR programs have minutes, if not hours or days, of lead time, these programs have lead times of seconds (e.g. 4 second response time) used for load balancing and frequency stabilization (Ancillary Services)

PUSH/PULL operations: OpenADR 2.0 can be used in either pull mode (VEN pulling information from VTN or vice versa) or in a push mode.

5   Abbreviations

AutoDR: Automated Demand Response

DER: Distributed Energy Resources

DR: Demand Response

DRRC: Demand Response Research Center

DUT: Device Under Test

EI: Energy Interoperation

HTTP: Hyper Text Transfer Protocol

IRC: ISO/RTO Council

JID: Jabber Identifiers

LBNL: Lawrence Berkeley National Laboratory

MEP: Message Exchange Pattern

NAESB: North American Energy Standards Board

OASIS: Organization of Structured Information Standards

OpenADR: Open Automated Demand Response

PICS: Protocol Implementation Conformance Statement

SASL: Simple Authentication and Security Layer

SOAP: Simple Object Access Protocol

TC: Technical Committee

UCAIug: UCA International Users Group - http://www.ucaiug.org/default.aspx

VEN: Virtual End Node

VTN: Virtual Top Node

XMPP: XML Messaging and Presence Protocol

6   Overview

This section gives an overview of the message exchanges, the roles, and actors supported within OpenADR 2.0 Profile Specifications. It contains the following elements, used to develop test and certification framework for Smart Grid and customer systems interoperability:

1. A set of data models derived from the OASIS Energy Interoperation 1.0 standard.
2. A set of services for performing various functions and operations for the exchange of the data models, also derived from the OASIS Energy Interoperation 1.0 standard.
3. A set transport mechanisms for implementing the services. The transport mechanisms rely upon standard-based IP communications such as HTTP and XML Messaging and Presence Protocol (XMPP).
4. A set of security mechanisms for securing each of the transport mechanisms.
5. OpenADR 2.0 Schemas (separate document)

The OASIS Energy Interoperation (EI) 1.0 standard describes the coordination of energy supplies, transmission, distribution, and usage. However, as shown in Figure 2, the features required within the OpenADR 2.0 Profile Specifications are a subset of this Energy Interoperation 1.0 standard. The reason for this development is twofold – 1) By having a strict and clear subset of features to support OpenADR, devices can be clear about what features they must support in order to participate in the OpenADR ecosystem; 2) by being a referenced subset, devices can validate against the Energy Interoperation 1.0 standard, and participate in that ecosystem as well. These requirements are critical to maintain interoperability.
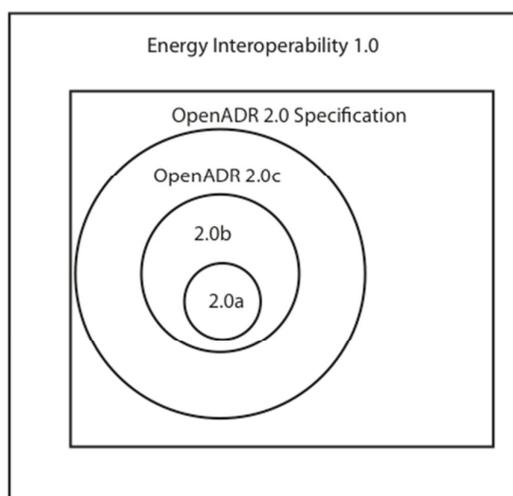
Figure 2 – Relationship between OASIS Energy Interoperation 1.0 Standard and OpenADR 2.0 Profiles

The above figure 2 shows the relative relationship between the complete data model and services features of the Energy Interoperability 1.0 standard and OpenADR 2.0 profiles. The diagram also shows how the different profile subsets of OpenADR2.0 relate to the complete OpenADR 2.0 feature set.

Message exchanges in OpenADR 2.0 support services related to communicating information about Demand Response events. Networks of OpenADR nodes must be able to query for active or pending events, register themselves, schedule events, report on the status of events, request prices for energy rates. OpenADR nodes must also be able to refine and update previously sent information. For instance, an OpenADR node reporting DR events to nodes downstream must be able to cancel a previously scheduled event if this becomes necessary.
Nodes in these networks are divided into two groups: nodes, which publish and transmit information about events to other nodes (e.g. Utilities), and nodes, which receive the communications respond to that information (e.g. end users). The upstream nodes that publish information about upcoming events are called Virtual Top Nodes (VTNs); the more downstream nodes that receive this information are called Virtual End Nodes (VENs).
These nodes may communicate using a variety of protocols. They may communicate using http in either PUSH mode (where the VTN initiates communication) on in a PULL mode (the VEN requests information from the VTN to begin a series of message exchanges). The VTNs/VENs may also communicate over other transport mechanisms such as Simple Object Access Protocol (SOAP), XML Messaging and Presence Protocol (XMPP).

Under the OpenADR specification, VTNs must support all operations described in this specification. However, the specification currently defines 3 levels of support for VTNs and VENs, in order to support more or less complicated end devices and systems. The levels are 2.0a, minimal support, 2.0c for full support, and an intermediate 2.0b level of support.

6.1 Node and Device Types

Following the notion from the OASIS Energy Interoperability 1.0 standard, OpenADR 2.0 uses the definitions of Virtual Top Nodes (VTNs) and Virtual End Nodes (VENs). For any interaction between actors using OpenADR 2.0 to communicate, one actor is designated the Virtual Top Node and the remainder are the Virtual End Nodes. All communications are between a VTN and one or more VEN's. There is no peer-to-peer communication in OpenADR 2.0, i.e. VTNs do not communicate directly with other VTNs and likewise VENs do not communicate directly with other VENs.

Generally in an interaction, the VTN acts as the server, providing information to the VEN, which themselves respond to the information. For instance, a VTN would be the entity to announce a DR event; VENs hear about DR events and respond. The response may be to reduce power to some devices. The response could also be to propagate the signal further downstream to other VEN's. In this case, the VEN would become the VTN for the new interaction. (FIG 2)

For the purpose of device development, the OpenADR Alliance always tests the interface between a VTN and a VEN, whereas either node can be the Device Under Test (DUT). Intelligence build into the systems not related to the OpenADR 2.0 message exchange is not part of the OpenADR Alliance testing program.

Virtual Top Node (VTN): An entity that is responsible for communicating grid conditions (e.g. prices, reliability events, etc.) to other entities (i.e. VEN's) that control demand side resources. The VTN is able to communicate with both the Grid and the VEN devices or systems in its domain. A VTN may take the role of a VEN interacting with another VTN.

Virtual End Node (VEN): The VEN has operational control of a set of resources and/or processes and is able to control the electrical energy demand of these in response to an understood set of smart grid messages (i.e. DR signals). The VEN may be either a producer or consumer of energy. The VEN is able to communicate (2-way) with a VTN receiving and transmitting smart grid messages that relay grid situations, conditions, or events. A VEN may take the role of a VTN in other interactions.

Although within any interaction, one actor is designated the VTN and the remainder are VENs (moreover, most interactions have exactly one VTN and one VEN), sets of actors can be arranged in any hierarchy, by allowing actors to act as VENs for some interactions and VTNs for others.
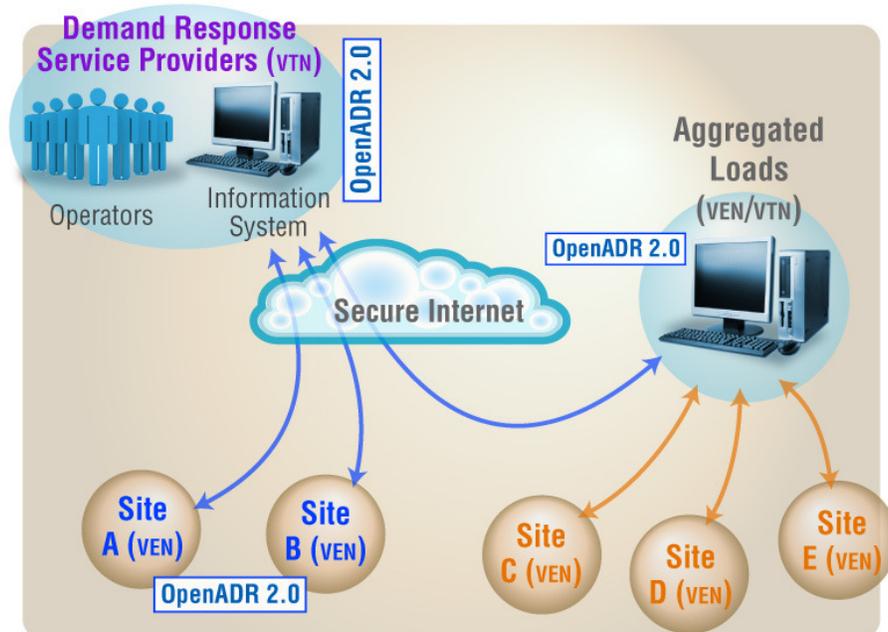
Figure 3 – Possible relationships of VTN and VEN

As illustrated in Figure 3 above, any combination of VTN and VEN is possible through a Utility/ISO (service provider or server) to Sites (customers). Also, as shown above, systems can function as a VEN to a VTN in a higher layer of the hierarchy and as a VTN to subordinate VENs. In either of these architectural scenarios, an operation can be initiated by the VTN to a VEN (PUSH pattern) or a VEN can request it from a VTN (PULL pattern). The exchanged events in either direction can be independent from each other and the OpenADR Alliance does not define how the nodes react to the information. In nodes, which support both the VTN and VEN interfaces (e.g., aggregators) there are no specifications or constraints on how messages arriving at the VEN interfaced is coupled or translated into any subsequent messages that may be sent from the VTN interface and vice versa. They are treated as completely independent interfaces and both will be evaluated and tested independently to assure adherence to the profile specification and interoperability. A specific deployment scenario depends on an agreement between the Utility/ISO and the participating Sites.

6.2   Energy Interoperation Services

The OASIS Energy Interoperation specification encompasses a great number of services for collaborative and transactive use of energy of which only 8 services are applicable for OpenADR. OpenADR 2.0 uses a profiled subset of the Energy Interoperation services tailored to meet the OpenADR needs, but still conforming to the Energy Interoperation Specification.

These services are EiRegisterParty, EiEnroll, EiMarketContext, EiEvent, EiQuote, EiReport, EiAvail, and EiOpt. For further information on these services, consult section 1.

6.3   Feature Sets

The OpenADR 2.0 specification has been designed to support a variety of end-use devices, from simple to more sophisticated. Accordingly, not all devices need to support all OpenADR capabilities. For example, although OpenADR describes how VTNs may report lists of active and pending future events, a simple residential programmable thermostat may only need information about the next pending event. In order to prevent onerous requirements on all devices, OpenADR has defined three feature sets, each of which represent a subset of OpenADR functionality. The three feature sets are 2a, 2b, and 2c. The purpose of the profiles

is to create a range of functionality that can be supported by devices as simple as a thermostat (profile A) to full blown IT based systems such as might be used by aggregators (profile C), and something in-between (profile B). Additional feature sets can be established to accommodate additional market requirements.

7   OpenADR 2.0 Feature Set Profiles

The OpenADR Alliance has defined several feature sets to accommodate a variety of different devices for varying applications. Each Feature Set Profile describes the required services to be implemented as well as the accompanying operations and attributes. Please refer to the OpenADR 2.0 Protocol Implementation Conformance Statement (PICS – refer to separate document for alliance members only) for more information regarding the required services. Additional profiles will be added if required by the market participants.

The OpenADR 2.0 services discussed in the Feature Set Profiles can be found in section 8. All services are subsets of the OASIS Energy Interoperation Specification and validate individually with the relevant schemas.

This document outlines the OpenADR 2.0a profile. OpenADR 2.0b and all the following profiles will have their own sets of documents and will follow. Summary information for these profiles is described in this document.

7.1   OpenADR 2.0a Feature Set Profile

The OpenADR 2.0a Feature Set was developed for simple or resource-constrained devices with limited computing and memory capacity with basic functionalities to participate in a DR program. As the technology evolves, such devices will be able to support other OpenADR 2.0 profiles.

7.1.1   Supported Services

   a) EiEvent Service – section 8.1

OpenADR 2.0a devices only need to support limited EiEvent services.

7.1.2   Transport Mechanism

Supported transport mechanisms are as follows. The mechanisms are described in section 9.

   a) Simple HTTP is mandatory

   b) XMPP is optional

7.1.3   Security

Supported security details are outlined in sections 9 and 10. The following security levels apply to OpenADR 2.0a.

   a) Standard Security – mandatory

   b) High Security - optional

8   OpenADR 2.0a Services and Data Models Extensions

8.1   OpenADR 2.0a Profile

8.1.1   EiEvent Service

Events are generated by the VTN and sent to the VEN using the oadrDistributeEvent payload containing one or more events described by the oadrEvent element. Some events require a response and others do not as indicated by the oadrResponseRequired element in the event description. If a response is required, the VEN acknowledges its opt-in or out-out disposition by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent.

Either a push or pull interaction patterns may be used. For push, the VTN will deliver events to the VEN using an oadrDistributeEvent payload. If an application level response is required, the VEN asynchronously sends an oadrCreatedEvent back to the VTN in a second message. These sequences are illustrated in Figure 4. VEN must support pull interactions as a minimum requirement.
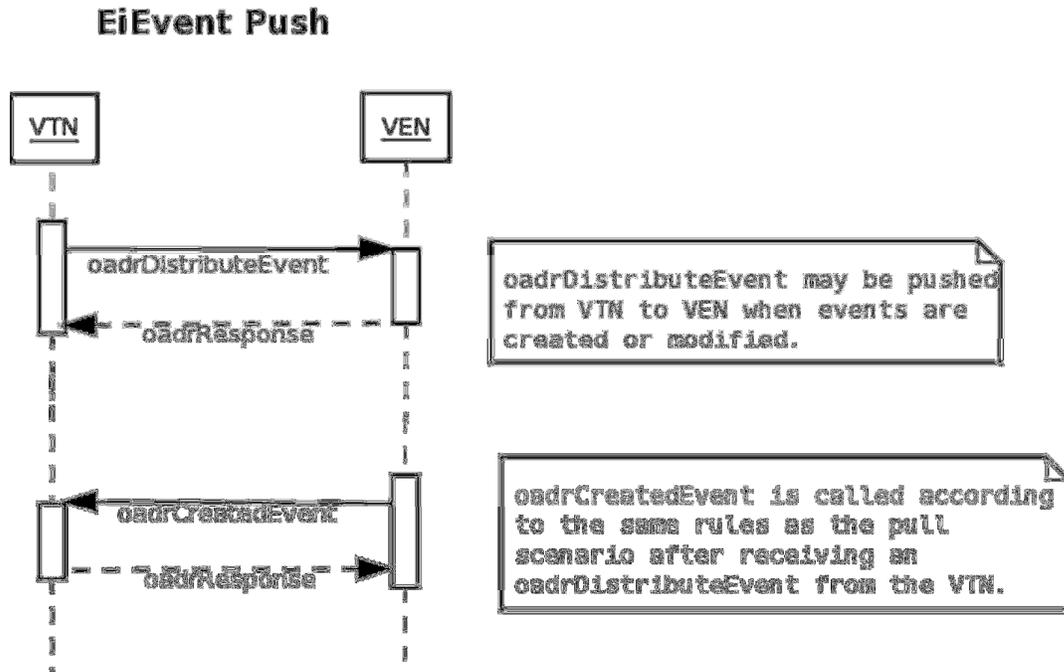
EiEvent Push

Figure 4: EiEvent Push Pattern

For the pull case the VEN requests events by sending an oadrRequestEvent to the VTN. The VTN responds with an oadrDistributeEvent. From this point the VEN response is exactly the same as in the push interaction. These sequences are illustrated in Figure 5.

EiEvent Pull

oadrRequestEvent may be periodically called one or more times until the VEN receives new or modified events.

oadrRequestEvent

oadrDistributeEvent

Periodic poll

oadrCreatedEvent may be called 0..n times after an oadrRequestEvent operation. It must be called at least once whenever an event has changed (based on its <modificationNumber>) or is encountered the first time by the VEN, or whenever <responseRequired> is 'always'.

oadrCreatedEvent

oadrResponse

Figure 5: EiEvent Pull Pattern

The details of how these interactions are performed within the context of a specific transport mechanism are covered in Section 9.

When an event requires a response, an initial oadrCreatedEvent is always sent from the VEN to the VTN. If a given program allows a VEN to later change its opt-state during an event it may do so by issuing subsequent oadrCreatedEvent message containing the new state for a given event. Detailed descriptions of VTN and VEN event processing are given in the following paragraphs

Note that for both push and pull operations, an oadrDistributeEvent payload will always contain all events applicable to the VEN it is communicating with.

Events are conveyed in the oadrDistributeEvent payload using one or more oadrEvent elements. The oadrDistributeEvent payload has the following components:

- A requestID to uniquely identify this request and any contained events.
- A vtnID identifying the VTN sending the request.
- Zero or more oadrEvent elements.

The requestID uniquely identifies the request and any contained events. Its value is set by the VTN using whatever scheme they desire, including using the same value in every request if its use is not needed by the VTN. The receiving VEN must use this requestID in the oadrCreatedEvent event responses.

oadrEvent Description

oadrEvent elements describe individual events, signal values, and time periods that apply to signals. Each oadrEvent has an eiEvent element containing detailed event information and an oadrResponseRequired element that controls whether a VEN must respond with an oadrCreatedEvent.

The responseRequired field indicates how the VEN must respond to contained events. A value of "always" indicates that the VEN must reply to each event whether or not the event state (eventID, modificationNumber) has changed. A value of "never" implies a "broadcast" event and the VEN must not send any responses in this case.

The eiEvent eventDescriptor has the following fields:

- eventID – a unique ID for this event. The ID is unique within the context of a VTN.
- modificationNumber – A sequence that starts at zero and is incremented by 1 each time the VTN modifies the event.
- Priority – An indication of the event priority with 0 being no priority
- marketContext – Identifies a particular program or application defined grouping that pertains to an event.
- createdDateTime – The time the payload containing the event was created.
- eventStatus – The status of the event, indicating if the event is "near", "far", "active" or "canceled".
- testEvent – If not false, indicates this is a test event.
- vtnComment – Arbitrary comment provided by the VTN.

A single eiActivePeriod defines the start time and duration of the event. The start time is defined by an ISO 8601 time descriptor and an iCalender duration string specifies the duration.

The event signals that get applied over the entire active period are defined in an eiEventSignals element. This element contains one (for OpenADR 2.0a) or more eiEventSignal elements, each with a sequence of durations, the sum of which must equal the full duration of the active period. Each signal element contains a signalType such as level or price. The signalPayload contains the relative values of "normal", "low", "moderate" or "special" for each duration.

The EiTarget may be used to provide information to explicitly specify the entities that apply to events. It may contain one or more venIDs, groupIDs, resourceIDs, or partyIDs. These may be used, for example, when a VEN is acting as an aggregator and has multiple resources behind it. Exactly how these IDs are handled is beyond the scope of this specification and must be dealt with by provisioning at the VEN and VTN. If the EiTarget element is not present the VEN must assume that it is the primary and only resource targeted by the events.

oadrCreatedEvent Description

When one or more received events require a response, the VEN creates and populates an oadrCreatedEvent element and posts it to the VTN. The oadrResponse element contains an application level responseCode and responseDescription and a requestID. An eiResponses element contains one or more eventResponse elements corresponding to each event. These are matched to specific events using thequalifiedEventID, which contains an eventID and modificationNumber. The optType may have a value of "optIn" or "optOut" to indicate the VENs disposition for a given event. '

An initial oadrCreatedEvent response must be sent for each event requiring a response. Subsequent EiCreatedEvents may also be sent to change the opt-state of a VEN when this is allowed for a given marketContext.

The grouping of events in an oadrCreatedEvent is completely up to the VEN and does not necessarily correspond to the grouping of events in an oadrDistributeEvent. The VEN is free to send one event per payload or group multiple pending events into a single oadrCreatedEvent payload.

8.1.1.1   Data Model

The OpenADR Alliance has its own schema that defines alliance specific extensions and references sub-setted elements from Energy Interoperation and related schemas.

For each Schema referenced by Energy Interop and used by OpenADR profiles, the Alliance created a separate subset schema xsd file. These schema subsets will maintain the same schema hierarchical element relationships, ordering of elements, mandatory cardinality, type definitions, and restrictions. However, the subset schemas may not use the same structural elements, such as abstract types and substitution groups, as the Energy Interop and related schemas. We will associate namespace prefixes with the same URIs used by the Energy Interop and related schemas.

URIs referenced in the OpenADR schema can be mapped to either the alliance subset schema files or the Energy Interoperation 1.0 standard and related schema files. Individual components of the OpenADR payloads will successfully validate against the OpenADR schema and the OASIS Energy Interoperation 1.0

8.1.1.2   UML Models

Figure 6 below shows the OpenADR 2.0a EiEvent Service using the OASIS EI 1.0 standard. The Green highlights are the elements, which apply to OpenADR 2.0a with an indication if these elements are mandatory (M) or optional (O)..
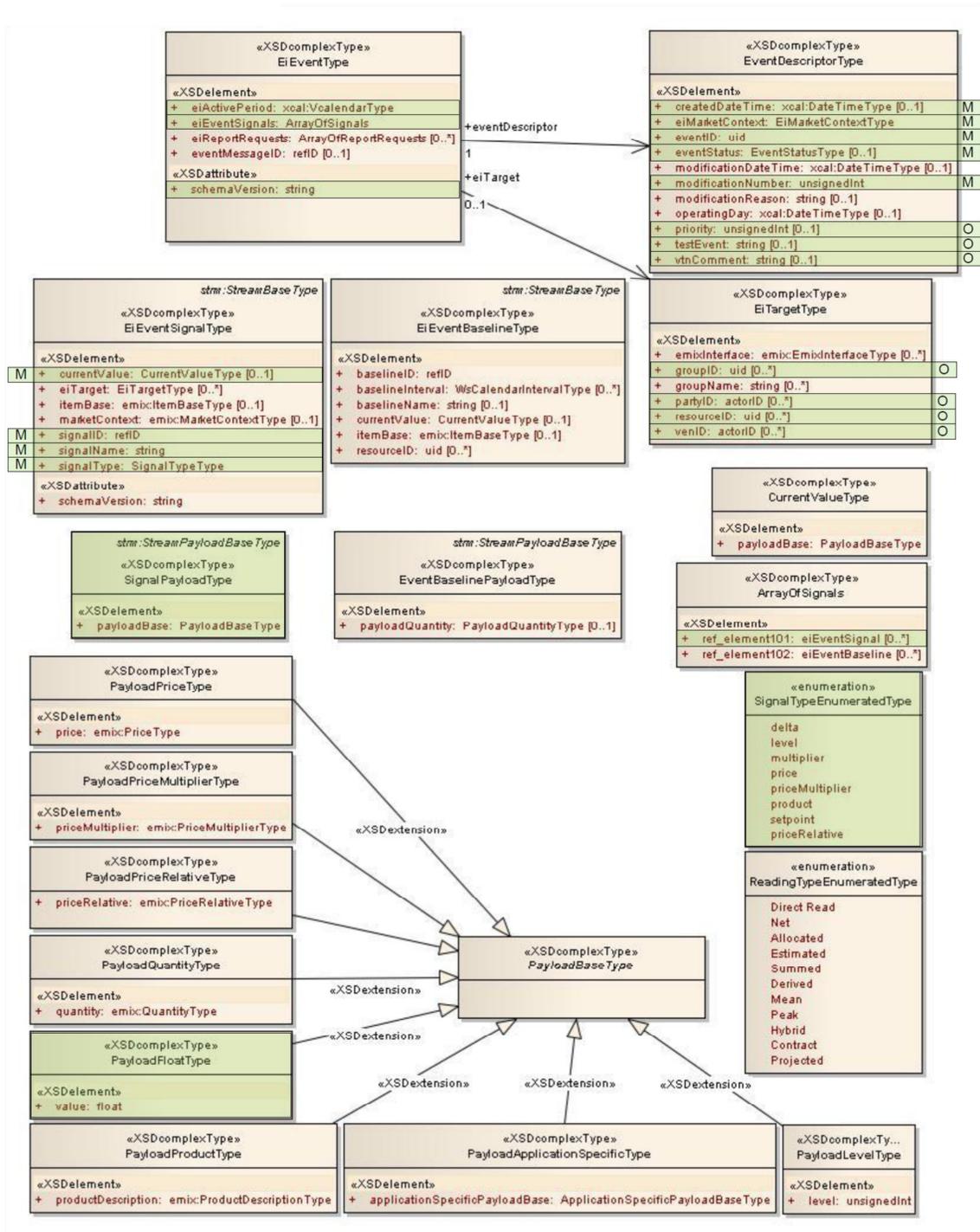
Figure 6: EiEvent UML

9   Transport Protocol

OpenADR 2.0 will support a small number of transport protocols to accommodate different deployment scenarios. For resource constraint end devices, a Simple HTTP transport was used. Further transports include XMPP and potentially other mechanisms. Note that top-level VTNs must support all transport protocols while VENs can chose the most applicable mechanism(s).

9.1   Simple HTTP

9.1.1   PUSH and PULL implementation

9.1.1.1   PUSH Definition

In a push MEP, messages may be sent from the VTN to VEN (pushed.) In order to use push, the VEN must expose HTTP URI endpoints (an HTTP server) to which the VTN may send requests such as oadrDistributeEvent. While this is the most efficient way to execute OpenADR2 over HTTP, it presents technical difficulties as the VEN may reside behind a network firewall.

9.1.1.2   PULL Definition

Using the pull MEP, all operations are initiated by the VEN to the VTN. This can be thought of as a 'polling' mode, where the VEN periodically asks for updates from the VTN. The pull mode removes the requirement for an HTTP server on the VEN, avoiding the technical limitation presented by the possibility of a network firewall in front of a VEN. However, the pull MEP has its own limitations, namely latency (due to limited polling frequency) and increased bandwidth requirements.

The pull MEP may involve a 'two-phase' execution to complete some operations. This is due to the nature of the VEN initiating the HTTP request. In a push model, the VTN may notify a VEN of a new event via the oadrDistributeEvent operation. The VTN would send a request with an oadrCreatedEvent payload, to which the VEN would respond with an oadrCreatedEvent payload.

However in the pull model, the VEN requests events from the VTN using oadrRequestEvent, to which an oadrDistributeEvent payload is sent in the response. After parsing the response, the VEN still needs to acknowledge the creation of any new events by making a *second* request using the oadrCreatedEvent operation on the VEN.

9.1.1.3   Service Endpoint URIs

The endpoint names will be of the form:
https://<hostname>(:port)/(prefix/)OpenADR2/Simple/<service>
- "prefix" is an optional URI path prefix that may be used to separate OpenADR services from other services that may reside on the same HTTP server.
- "Simple" indicates the simple XML over HTTP protocol (others might be include "SOAP", "BACnet", etc.).
- <service> is the name of the EI service (e.g., "EiEvent", "EiReport", etc.).

The "operation" portion of a service is defined by the XML payload sent in a request. E.g. a oadrRequestEvent payload root element specifies the oadrRequestEvent operation.

Note that for implementations that expose both a VEN and VTN (such as an aggregator,) these implementations MUST use different URI endpoints for their VTN and VEN interfaces. For example, https://mycompany.com/**myVTN**/OpenADR2/Simple/EiEvent and https://mycompany.com/**myVEN**/OpenADR2/Simple/EiEvent.

9.1.1.4   HTTP Methods

All messages will be sent using the HTTP POST method. This helps to avoid caching and allows all operations to contain a payload in the HTTP request body.

9.1.1.5   Failure Conditions

The following failures can occur for a given operation:
- TCP (or below) fails
- HTTP fails (http error code)
- application acknowledgement fails (application error code)
- response failure (timeout or application error code)

The proper action for each failure condition depends upon the application and the operation being attempted. Since all operations are idempotent, it is safe to retry any operation.

In the case of TCP failure, it is ALWAYS recommended to retry the operation.
In the case of an HTTP failure, the behavior must be according to HTTP status codes defined in section 8.1.1.6.
Application-level errors are defined elsewhere in OpenADR 2.0.
Timeout failure handling is defined below.

9.1.1.6   HTTP Response Codes

See (2) for more details on HTTP status code definitions.

200 OK - any response that the endpoint was able to handle completely and send a valid OpenADR response payload. This includes responses that may indicate an error at the application level (e.g. 'you gave me an invalid event ID.') Errors that indicate a failure at the transport level are handled by transport- level HTTP error codes:

401 Unauthorized - the requestor is not authorized to perform the given operation, likely due to missing or misconfigured authorization credentials. The requestor MUST NOT re-send the request until credentials have been modified.

404 Not Found - the VEN or VTN does not support the requested operation. The requestor MUST NOT re-send the request.

406 Not Acceptable – If a payload is sent that does not validate against the EI schema, or if a request content-type is unsupported. The requestor MUST NOT re-send the request without first modifying it.
501 Not Implemented – if any request is made with an unsupported HTTP method. The requestor MUST NOT re-send the request without fixing the HTTP method.

503 Service Unavailable – indicates that the server is temporarily unavailable, possibly due to inability to handle the current request load. This error in particular must indicate to the requestor that it must execute quiesce logic in order to not put further strain on the server. The requestor MUST retry the request after the proper quiesce period.

500 Internal Server Error – undefined or unexpected server error. The requestor MAY retry the request after a quiesce period.
For all error (non-200) codes, the content body of the response is undefined. The server MAY choose to send some informational message in the response, but the requestor is NOT obligated to parse or understand it.
All application-level error conditions are conveyed through the status code element of oadrResponse payload.

9.1.1.7   Message Timeouts

There are no prescribed connect or response timeout thresholds for OpenADR 2.0. In general, implementations must allow configurable timeouts in order to handle IP networks with different latency characteristics. HTTP clients MUST use a request timeout of at least 5 seconds.

9.1.1.8   Message Retry/ Quiesce Behavior

When a request fails for any reason (either due to physical or network-level failure or a timeout) the requestor must institute 'back-off' or quiesce logic to avoid flooding the network or receiver with requests.
Clients must begin quiesce at some small interval (say, 1 second) plus or minus some random 'jitter,' which is a small percentage of that interval (say, 10%). So for example, the first quiesce interval for a device might be between 0.9 and 1.1 seconds. Then the device may retry the request. If subsequent retries fail, quiesce interval must double from the prior interval, again adding a random jitter of plus or minus 10% of that interval. This doubling for subsequent failures must continue up to some maximum, probably dependent on the poll interval in the case of a VEN polling a VTN. This is known as a "truncated binary exponential back off algorithm."

9.1.1.9   Pull Timing

The ability to configure the poll interval for a pull MEP is not defined in OpenADR 2.0, however VEN implementations MUST allow the poll interval to be configurable at a millisecond resolution.

VEN clients MUST also be configurable to induce some 'jitter' - a random offset from the absolute poll interval - in order to avoid request spikes on the VTN caused by many VENs initiating a pull request at the exact same instant within the poll interval.

9.1.1.10   HTTP Headers

The following HTTP headers (3) must appear in requests or responses (where indicated):

Accept

The accept request header specifies the expected content-type of a response. Since responses are always "application/xml", the Accept header MAY be omitted. However, if it is included, the value of the Accept request header must always be "application/xml".

Accept-Encoding

This request header indicates if a client supports content compression of the response payload. A VEN MAY include this header in a request if it supports content compression such as gzip or deflate. If the VEN includes this header, the VTN MUST honor it and compress the response content using one of the methods given in the request header.

Authorization

The authorization header MUST be used where required to transmit necessary credentials to identify the requestor. See the "Security and Authentication" section below for more details.

9.1.1.11   Content-Encoding

If a VTN is responding to a request for which it has compressed payload, it MUST include a Content-Encoding response header indicating the correct encoding method, such as gzip or deflate.
Push operations from a VTN MUST NOT utilize the content-encoding header in the request, since it would require the VTN to have a priori knowledge of which content-encodings are supported by each and every VEN.

9.1.1.12   Content-Length

The content-length header MUST be used according to (3) to indicate content body size of all request and response payloads.
Special Note: 'chunked' transfer encoding (where content-length is unknown) is not a requirement for OpenADR 2.0, although it may be supported inherently by many HTTP/1.1 implementations. Implementers must assume that the total content body length is known when the response headers are sent, and must not attempt to send chunked responses.

9.1.1.13   Content-Type

Must be used for both request and response messages, indicating payload MIME type. The appropriate value is "application/xml". The content-type MAY also specify a character encoding. For OpenADR2, the only supported character encoding is UTF-8. If a charset is included, the entire header value must appear as "application/xml; charset=utf-8".

9.1.1.14   Host

The 'Host' header must be included in all requests per HTTP/1.1 Section 14.23 (3).

9.1.1.15   User-Agent

The User-Agent header MAY be included by the requestor but its presence must not be relied upon, nor must it materially affect the behavior of the server handling the request.

9.1.1.16   WWW-Authenticate

When a server responds with a 401 status, it MUST include this header to indicate proper authentication mechanisms. The requestor may then re-send a modified request, with proper Authorization header(s) included.

9.1.2   Transport Specific Security

TLS 1.0 (5) must be used to encrypt all traffic, regardless of the authorization method used. The client must always validate the server's SSL certificate given during the handshake.

9.1.2.1   SSL/ TLS Client Certificate

Client certificates must be used for HTTP client authentication. The entity initiating the request (the client) must have a X.509 certificate that is validated by the server during the TLS handshake. If no client certificate is supplied, or if the certificate is not valid (e.g. it is not signed by a trusted CA or if it is expired) the server must terminate the connection during the TLS handshake.

If the certificate appears valid during the TLS handshake, the connection is established and the HTTP request proceeds. Once the HTTP request is received by the server, the server must perform authorization, given the credentials in the client certificate. The VTN should use the certificate public key as the primary identification mechanism. The client credentials must be compared to ensure they match the venID that appears in OpenADR payload of the client request. If credentials do not match, or if the server otherwise determines that the request is not authorized, it must respond with a HTTP 401 error.

9.2   XMPP - DRAFT

XMPP is a stateful, bi-directional protocol ideal for transmitting messages in XML format. The core protocol is specified in RFC-6120, and addressing is defined in RFC-6122. Additional information including accepted extensions can be found at http://xmpp.org/.

9.2.1   PUSH and PULL implementation

By nature, XMPP is a bi-directional, stateful protocol. As such, any client utilizing XMPP can implement both PUSH and PULL operations seamlessly. PULL operations are simply those initiated by the VEN, and PUSH operations are initiated by the VTN.

9.2.2   Service Endpoints

Both VTN and VEN service endpoints are defined by a Jabber Identifier JID, which is similar to an email address. The fully qualified JID performs the same function as an endpoint URI in an HTTP implementation. The definition of JIDs is further described in section 9.2.4.1 of this document.

9.2.3   Service Execution

Because XMPP is a message-based protocol, execution of OpenADR services occurs by passing XMPP messages that contain an OpenADR XML payload.

9.2.4   Implementation of XMPP Features for OpenADR

9.2.4.1   JIDs

While the prospect of using of multiple resources per JID opens some interesting possibilities (e.g. exposing each building at a facility as a different resource,) this strategy should be used with some caution, as same mechanism is not supported by other OpenADR transports such as HTTP.

VEN clients **should** always authenticate with a fully-qualified JID (rather than allow the server to assign a random resource ID.)

VTN clients **may** choose to expose OpenADR operations either as an XMPP service or as a client JID. Although most OpenADR service operations will be 'pushed' from VTN to VEN, a VEN may still need to send some requests to the VTN. Rather than use some configuration to define the JID for VTN services, OpenADR services shall be discovered at runtime via service discovery (described below.)

9.2.4.2   Use of the Packet 'type' Attribute

All operations MUST use the 'set' IQ type attribute for all OpenADR service operations.

9.2.4.3   Use of Message versus IQ Packets

Any OpenADR2 service operations which require an application-level response (e.g. an OpenADR2 response payload) or a transport-level response code **must** use an IQ packet. The XMPP protocol then mandates a response IQ from the recipient.

Any OpenADR2 service operations which do not require a response, such as a broadcast operation (or possibly feedback,) **may** use a message packet to send the OpenADR2 payload. Handling of the packet should be identical to if it were an IQ, except that no response is sent by the recipient. This is a bandwidth optimization for use cases where a response is not expected, and a response IQ would be unnecessary.

9.2.4.4   Error Handling

9.2.4.4.1   Transport-Level Status Codes

Under normal conditions, an IQ response will have a `type='result'` attribute. This is analogous to an HTTP 200 response. OpenADR2 service operations which define an empty HTTP 200 response will simply look like an IQ element with no child payload element:

```
<iq type='result'
    to='ven1234@xmpp.myvo.net/client'
    from='vtn.xmpp.somevtn.net'
    id='1' />
```

9.2.4.4.2   Transport-Level Errors

Operations which result in a transport-level error will have a `type='error'` and a child <error> element that indicates the transport-level status code as described in RFC-6120 section 8.3[i]. For example:

```
<iq type='error'
    to='ven1234@xmpp.myco.net/client'
    from='vtn.xmpp.somevtn.net'
    id='2'>
    <error type='modify'>
        <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    </error>
</iq>
```

Since XMPP does not use numeric status codes, direct mapping of HTTP status codes is not possible, and logic for handling transport-level errors is outside the scope of this document. Guidelines for handling different types of XMPP errors can be found in RFC-6120 section 8.3.3.

In particular, if an OpenADR Profile B request is made to a Profile A client, the client should respond with a `feature-not-implemented` error as defined in 8.3.3.3.

If a client receives an OpenADR payload that does not adhere to the OpenADR schema, a client should respond with a `bad-request` error as defined in section 8.3.3.1.

Note that application-level errors must be handled the same as in HTTP. That is, a normal 'result' XMPP response should be returned with the application-level error details in the OpenADR payload.

9.2.4.5   Presence

VEN clients **must** use presence packets to notify subscribed entities of online and offline status as described in RFC-6121 section 4. All other status broadcasts from a client should be considered informational.

After a VEN completes session negotiation, it **must** send an 'available' presence stanza as defined in Section 4.2.1:

```
<presence />
```

If VEN deliberately terminates its XMPP connection (e.g. due to a graceful shutdown, not an unexpected, sudden connectivity loss,) it **must** first send an 'unavailable' presence stanza as defined in Section 4.5.1:

```
<presence type='unavailable'/>
```

9.2.4.6   Authentication

All clients **must** support SSL/TLS and authorization as defined in section 13.8 and 13.9.4 of RFC-6120. In particular, servers **must never** allow PLAN or DIGEST-MD5 authentication over a channel that is not secured with TLS. All clients **SHOULD** support DIGEST-MD5 authentication. Clients may also implement other authentication mechanisms such as Simple Authentication and Security Layer SASL EXTERNAL in order to use certificate authentication without passwords. The exact authentication mechanism used is not mandated and should follow auth feature negotiation as defined in RFC-6120.

9.2.4.7   Rosters

While rosters are a core feature of XMPP and commonly occur during XMPP session initiation, they have no defined role within OpenADR2 and therefore their use is undefined within the scope of OpenADR2.0. A roster request **may** be performed by a client, however under most circumstances, the server may simply respond with an empty roster. Optimally, a client (particularly VEN clients) **SHOULD NOT** request a roster from the server for the purposes of OpenADR.

More specifically, VEN clients should not typically add other entities to its roster, as this will result in additional network overhead due to presence broadcasts.

VTN clients might choose to use the roster as a mechanism to track presence of online VEN clients, however this is not strictly a feature or requirement of OpenADR.

9.2.4.8   Service Discovery

XMPP Service Discovery (XEP-0030)[ii] shall be used to define the address for OpenADR services on the VTN. This allows the VTN flexibility in how it implements services, while removing the need for additional configuration on the VEN, which would be outside the scope of OpenADR.

9.2.4.8.1   Discovery of the OpenADR Feature

After a VEN has completed session initiation, it may perform an XEP-0030 'info' query[iii] to the bare domain to which the VEN has authenticated. For example, if a VEN has connected as 'ven1@xmpp.myco.net/client' then its initial service discovery query would look like the following:

```
<iq type='get'
    from='ven1@xmpp.myco.net/client'
    to='xmpp.somevtn.net'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

To which the server would respond:

```
<iq type='result'
    from='xmpp.somevtn.net'
    to='ven1@xmpp.myco.net/client'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='http://openadr.org/openadr2'/>
    <feature var='http://jabber.org/protocol/disco'/>
  </query>
</iq>
```

This indicates, in an XMPP-compliant fashion that the XMPP server supports the OpenADR2 protocol.

9.2.4.8.2   Discovery of OpenADR Service Endpoints

Then VEN may then perform an 'items' query, with the 'node' set to the OpenADR2 namespace 'http://openadr.org/openadr2#services' as such:

```
<iq type='get'
    from='ven1@xmpp.myco.net/client'
    to='xmpp.somevtn.net'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'
        node='http://openadr.org/openadr2#services'/>
</iq>
```

The VTN shall then respond with the JIDs used for each OpenADR2 service, for example:

```
<iq type='result'
    from='xmpp.somevtn.net'
    to='ven1@xmpp.myco.net/client'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'
        node='http://openadr.org/OpenADR2#services'>
    <item jid='event.openadr2.xmpp.somevtn.net'
        node='http://openadr.org/OpenADR2/EiEvent' />
    <item jid='feedback.openadr2.xmpp.somevtn.net'
        node='http://openadr.org/OpenADR2/EiFeedback' />
  </query>
</iq>
```

Note that all services could use the same JID or different JIDs and JIDs may be subdomains (which are common for XMPP services) or fully-qualified with a 'localpart' and resource.

9.2.4.8.3   Service Discovery on VEN Clients

VEN clients may choose to implement service discovery to provide supplemental information about their client (such as client IP address, available disk space, etc.)   However such supplemental information is outside the scope of OpenADR. At the bare minimum, if a VEN supports XEP-0030 queries, it must indicate support for the 'http://openadr.org/openadr2' feature, as described in the "Discovery of the OpenADR Feature" section above.

9.2.5   Security Considerations

Beyond authentication (as defined above) additional steps are required to secure an XMPP network. In particular, VEN clients must not be allowed to communicate with each other (at least, not within the scope of OpenADR.) Clients may opt to implement some sort of whitelist to control access from different JIDs, which would be similar to implementing a firewall in an HTTP push scenario.

However, ultimate responsibility for controlling access between XMPP clients must be held at the XMPP server level. In most cases, the XMPP server will be logically deployed alongside or as an integrated part of a VTN. The XMPP server can then implement access controls to keep VENs logically isolated so they can only communicate with the VTN.

10 OpenADR 2.0 Security

OpenADR 2.0 aspires to conform to NIST Cyber Security requirements and to follow the guidelines provided by the "Security Profile for OpenADR" prepared by The UCAIug OpenADR Task Force and SG Security Joint Task Force. Furthermore additional NIST guidelines were taken into account to determine the appropriate security for OpenADR profiles. This background exercise was intended to make sure that the OpenADR meets the NIST Cyber Security and as well as any DR service provider deployment requirements. The OpenADR Alliance provides this basic security framework for testing to meet these stringent requirements and understands that the final security scheme would be determined by the DR program deployments.

It is expected that many OpenADR2.0 implementations will make use of existing cloud computing services and platforms. The OpenADR Alliance has therefore decided to allow currently available security protocols and cypher suites until they are deprecated for the intended use. All certified OpenADR 2.0 Products must be upgradable. The mechanism of the upgradability is left to the manufacturers' implementation.

OpenADR 2.0 uses common security mechanisms, e.g. TLS, without any modification.

Manufacturers shall refer to the latest version of the NIST Special Publication 800-131A when choosing their security algorithm.

10.1   Security Overview

There are two types of actors we are concerned about for OpenADR 2.0, the VTNs (Demand Response (DR) Server) and the VENs (Client systems). In the DR scenario, the VENs connect to one of more VTNs for receiving DR signals. The VTNs provide services to one or more VENs. The VENs are installed at end-points such as consumer homes, commercial, or industrial premises, which choose to participate in the DR programs.

10.2   Architecture and Certificate Types

To provide security services like authentication, confidentiality and integrity, the VENs and VTNs use Public Key Infrastructure (PKI) certificates. Two levels of security are defined, called 'Standard' and 'High,' are defined for OpenADR 2.0. The 'Standard' security uses TLS for establishing secure channels between the VTNs and VENs for communication. 'High' security uses TLS for establishing secure channels between the VTNs and VENs and it additionally uses XML signatures providing stronger non-repudiation.

OpenADR 2.0 adopts an open architecture for security and will not restrict itself to some specific or proprietary technologies. There are primarily two public key cryptography algorithm options for using PKI certificates, namely RSA and ECC. While RSA is more widely acceptable, ECC provides the benefits of more efficient cryptographic operations like encryption and digital signatures. For the same cryptographic strength, ECC key sizes are much shorter as compared to RSA keys. This is especially important for embedded devices, which favor efficient cryptography. As mentioned, RSA is more widely accepted on the Internet and has multiple certificate providers making it easier to select a provider which best matches the requirements.

To retain the benefits of both options and for the purposes of interoperability, the VTNs will support both, ECC and RSA certificates, each from a well-known certificate authority (CA). The final design may have more than two providers but it will be ensured that it has at least more than one ECC and one RSA certificate authorities are included. The VENs can choose to use one or more PKI certificates on the devices. The only restriction is that the VENs must implement at least one certificate from the approved list of certificate and CAs for the VTNs. This list will be decided upon in advance and will be publicly available. To establish a secure communication channel between the VTNs and VENs, the VTNs will have to support all the certificates in the approved list of certificates types and CAs. Upon initiating the communication, the VEN will have the choice to use any one of them.

The following parameters will be used for the certificates:

- **ECC** – 256 bits or longer keys.
- **RSA** - 2048 bits or longer keys.
- **Certificate types** – X.509v3

See references.[6]

---

6 References

1. NIST Special Publication 800-131A.

2. X.509 public key certificates version 3, http://tools.ietf.org/html/draft-igoe-secsh-x509v3-07

3. http://www.certicom.com/images/pdfs/ecc/ds-ecc-cu-102210.pdf

4. http://www.entrust.net/ecc-certs/index.htm

5. Apurva Mohan, ECC vs. RSA performance benchmarks on embedded platforms, draft document submitted to the OpenADR alliance security group

10.3   Certificate Authorities

Manufacturers shall obtain CA certificates from commercial vendors and shall use those certificates to issue OpenADR certificates. The OpenADR Alliance will establish contractual relationships with 3-5 approved root CA providers and shall make a list available to OpenADR device manufacturers and implementers.

10.4   Certificate Revocation

Certificate misuse in OpenADR 2.0 is very unlikely due to the strong relationship between the server and the client. Instead of a globally accessible revocation list, OpenADR 2.0 VTNs shall support a whitelisting feature to which device certificates are added upon registration and removed at de-registration. The OpenADR Alliance will publish a list of known compromised certificates.

10.5   TLS and Cypher Suites

As discussed earlier, OpenADR 2.0 manufacturers shall review the NIST SP 800-131 for deprecation dates and current security requirements. Based on current availability, the following mechanisms shall be the baseline for the OpenADR 2.0 Standard security.

VTN and VEN shall support TLS 1.0 and may support higher versions of TLS provided that they can still interoperate with TLS 1.0 implementations. The default cipher suite selection shall be as follows:

- The VEN client shall offer at least at least one of the default cipher suites listed below
- The VEN server shall must support at least one of the default cipher suites listed below and must select one of the default cipher suites regardless of other cipher suites that may be offered by the VTN client
- The VTN client must offer both the default cipher suites listed below.
- The VTN server must support both of the default cipher suites listed below and must select one of listed the default cipher suites regardless of other ciphers that may be offered by the VEN client

Default cipher suites:
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA

Note that a VTN or VEN may be configured to support any TLS version and cipher suite combination based on the needs of a specific deployment. However in the absence of changes to the default configuration of the VTN or VEN, the behavior of the devices shall be as noted above.

Individual DR programs can chose any security level required by the involved entities.

10.6   System Registration Process

Registration is defined as the VTN becoming "aware" of the VEN, e.g. exchange of credential and possibly other information so that the VTN can properly authenticate the VEN. Note that this specifically does not include enrolling in DR programs. OpenADR 2.0 provides manufacturers and program operators with a flexible solution. Therefore it is left to the individual program how client devices might register to the server. However, basic mechanisms are required. See Annex A for more details. All current registration processes involve a certain amount of out-of-band information exchange to establish the initial connection. Program operators and VTN vendors must implement a secure way of performing this registration process. However, the procedure is outside the realm of this specification.

10.6.1   Certificate Fingerprints

VENs shall facilitate registration by providing a "certificate fingerprint" which can be easily transmitted out-of-band to the VTN. The fingerprint may then be used by the VTN to identify a VEN when it first connects to the VTN.

The certificate fingerprint will be generated as follows:

1. Perform an SHA-1 hash on the bytes of the DER-encoded client certificate

2. Take the last 10 bytes (from the 20-byte SHA-1 hash), represented as pairs of hexadecimal digits, separated by the colon (ASCII 58) character.

This fingerprint may be generated by the common 'openssl' command line tool from a PEM certificate as follows:

```
$ openssl x509 –in client_cert.pem –fingerprint
SHA1
Fingerprint=B8:50:FE:63:E4:C4:73:BD:16:D2:20:95:07:CA:49:8F:E6:A6:12:
C4
```

The last 29 characters of the SHA1 hash (20:95:07:CA:49:8F:E6:A6:12:C4) shall be used as the "certificate fingerprint." This fingerprint shall be printed or otherwise distributed with the VEN so it can be transmitted out-of-band to the VTN during installation time. See Addendum A for usage examples.

The fingerprint could also be computed in python as follows:

```
import ssl, hashlib

bin_cert = ssl.PEM_cert_to_DER_cert( open('client_cert.pem').read() )
sha_hash = hashlib.sha1(bin_cert).digest()
print ':'.join( '%02X' % ord(c) for c in sha_hash[-10:] )
```

11 Conformance

11.1 OpenADR 2.0 conformance statement

In order to claim conformance to this profile specification, a VTN, VEN or VTN/VEN combination must conform to all statements made in this document as well as the OpenADR 2.0 PICS document. Product variations can be found in the product matrix. Further, a product must be tested at an authorized test service provider and undergo certification managed by the OpenADR Alliance.

11.2 OpenADR 2.0 Profile Conformance Rules

11.3 OpenADR 2.0a Conformance Rules

Conformance Rule	Requirement
1	**VTN** The time, date, or date and time MUST be specified using [ISO8601] utc-time (also called *zulu time*). Example: 2011-12-15T15:26:44Z
2	**VTN, EiEvent Service, oadrDistributeEvent Payload** Within a single oadrDisributeEvent eiEventSignal, UID must be expressed as an interval number with a base of 0 and an increment of 1 for each subsequent interval
3	**VTN, EiEvent Service, oadrDistributeEvent Payload** oadrDisributeEvent priority element - This is the priority of this event relative to other events. The lower the number higher the priority. A value of zero (0) indicates NO priority is the lowest priority by default.
4	**VTN, EiEvent Service, oadrDistributeEvent Payload** New events start with a modificationNumber of 0.
5	**VTN, EiEvent Service, oadrDistributeEvent Payload** Each modification of the oadrDisributeEvent eiEvent object, excluding

	createdDateTime, eventStatus, and currentValue, cause the modificationNumber to increment by 1 and an updated event sent to the VEN.
	Exception: An eventStatus change to "cancelled" shall cause the modification number to increment by 1
6	**VEN, EiEvent Service, oadrDistributeEvent Payload** The presence of any string except "false" in the oadrDisributeEvent testEvent element is treated as a trigger for a test event.
7 **A Profile Only**	**VTN, EiEvent Service, oadrDistributeEvent Payload** The oadrDistibuteEvent eiEvent object must contain only one event signal with a signalName of "simple".
8	**VTN, EiEvent Service, oadrDistributeEvent Payload** oadrDisributeEvent eventSignal interval durations for a given event must add up to eiEvent eiActivePeriod duration.
9	**VTN, EiEvent Service, oadrDistributeEvent Payload** oadrDisributeEvent eiEventSignal's with a signalName of "simple", must use signalPayload values of 0=normal; 1=moderate; 2=high; 3=special.
10	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN must change the oadrDisributeEvent eventStatus to "cancelled" when communicating an event cancellation to the VEN. Note that the modificationNumber is incremented by 1 when issuing a cancellation.
12	**VEN, EiEvent Service, oadrCreatedEvent Payload** The VEN must respond to an event in oadrDistributeEvent based upon the value in each event's oadrResponseRequired element as follows: Always – The VEN shall respond to the event with an oadrCreatedEvent eventResponse . This includes unchanged, new, changed, and cancelled events Never – The VEN shall not respond to the event with a oadrCreatedEvent eventResponse Note that oadrCreatedEvent event responses SHOULD be returned in one message, but CAN be returned in separate messages.
13	**VTN, EiEvent Service, oadrDistributeEvent Payload** EventStatus must always transition from NONE to FAR to NEAR to ACTIVE. The transition to NEAR occurs at the start of the x-eiRampUp period if defined. If x-eiRampUp is not defined the transition will move from FAR to ACTIVE at the dtStart time.
14	**VTN, EiEvent Service, oadrDistributeEvent Payload** The currentValue must be set to 0 (normal) when the event is not active.
15	**VTN, EiEvent Service, oadrDistributeEvent Payload** A list of events returned by the VTN in response to a pull EiRequestEvent must be ordered as follows: 1.Active events have priority over pending events 2.Within Active Events, priority is determined by Priority in the Event Descriptor.

	3.Between active events with the same priority, the one with the earlier start time has the higher priority.
	4.Between pending events the one with the earlier start time has the higher priority
	5.After processing rules 1-4, if Priority is still indeterminate within a set of Intervals, then the order is indeterminate within that set. A Reply containing Events with indeterminate Order MUST maintain that order in response to successive Requests while they remain indeterminate.
	Note: A cancellation should be ordered according to whatever rules were applied before the event was cancelled
16	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN must recognize the state of the eiCreatedEvent optType element, both optIn and optOut
17	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN must recognize an async eiCreatedEvent optOut for a previously acknowledged event.
18	**VEN/VTN, EiEvent Service** The VEN/VTN must honor the following rules with regards to overlapping active periods... *DR events with overlapping active periods may be issued, but only if they are from different marketContexts and only if the programs have a priority associated with them. DR events for programs with higher priorities supersede the events of programs with lower priorities. If two programs with overlapping events have the same priority then the program whose event was activated first takes priority.* The behavior of a VEN is undefined with respect to the receipt on an overlapping event in the same market context. The VTN shall not send overlapping events in the same market context, including events that could potentially overlap a randomized event cancellation. Nothing in this rule should preclude a VEN from opting into overlapping events in different market contexts.
19	**VEN, EiEvent Service, oadrDistributeEvent Payload** If an oadrDistributeEvent payload has as mix of valid and invalid events, the implementation shall only respond to the relevant valid events and not reject the entire message.
20	**VTN, EiEvent Service, oadrDistributeEvent Payload** At any time, a VTN can change any element or attribute of a pending or active event as long as it does not pertain to the past.
21	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** If venID, vtnID, or EventID is included in payloads, the receiving entity must validate the ID values are as expected and generate an error if no ID is present or an unexpected value is received. Exception: A VEN shall not generate an error upon receipt of a cancelled event whose eventID is not previously known.
22	**VEN, EiEvent Service, oadrDistributeEvent Payload** If no sub elements are present in oadrDistributeEvent eiTarget, the presumption is that the recipient is the intended target of the event. If multiple criteria are present in eiTarget subelements, the values are OR'd

	togther to determine whether the VEN is a target for the event. However, the VENs behavior with respect to responding to an event when it matches one of the eiTarget criteria is implementation dependent.
23	**VEN/VTN, EiEvent Service, oadrRequestEvent Payload** oadrRequestEvent many only be sent in the VEN to VTN direction
25	**VEN/VTN, EiEvent Service** VTN and VEN: The following rules must be followed with respect to application level responses with respect to multiple events: 1)If the Response indicates success, there is no need to examine each element in the Responses. 2)If some elements fail and other succeed, the Response will indicate the error, and the recipient should evaluate each element in Responses to discover which components of the operation failed. Exception: For oadrCreatedEvent, the presence of a failure indication in eventResponse:responseCode shall not force a failure indication in eiResponse:responseCode. Typical behavior would be for the VEN to report a success indication in eiResponse:responseCode and indicate any event specific errors in eventResponse:responseCode. The VTN should specifically check each eventResponse for the optType state.
27	**VTN, EiEvent Service, oadrRequestEvent Payload** If a value is provided in the oadrRequestEvent replyLimit element, the VTN must only return the number of events specified by the value in its oadrDistributeEvent response.
29	**VTN, EiEvent Service, oadrDistributeEvent Payload** The oadrDisributeEVent currentValue for each eiEVent signalType must accurately reflect the signalPayload value for the active interval in an executing event.
30	**VEN, EiEvent Service, oadrDistributeEvent Payload** The VEN must randomize the dtstart time of the event if a value is present in the startafter element. Event completion times are determined by adding the event duration to the randomized dtstart time. Modifications to an event should maintain the same random offset, unless the startafter element itself is modified.

31	**VEN, EiEvent Service, oadrDistributeEvent Payload**
	The VEN must recognize and act upon values specified in the subelements of activePeriod including:
	
	    • dtStart
	    • duration
	    • tolerence
	    • x-eiRampUp (positive and negative)
	    • x-eiRecovery (positive and negative)
	
	Note: x-eiRampup and x-eiRecovery are not testable requirements

32	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload**
	 The VEN must recognize and act upon values specified in the subelements of intervals including:
	
	    • duration
	    • signalPayload

33	**VEN/VTN** The implementation must provide an application layer error indication as a result of the following conditions: • Schema does not validate • Missing expected information • Payload not of expected type • ID not as expected • Illogical request – Old date on new event, durations don't add up correctly, etc. • Etc.
35	**VEN, EiEvent Service, oadrCreatedEvent Payload** The eiResponses element in oadrCreatedEvent is mandatory, except when an error condition is reported in eiResponse.
36	**VEN, EiEvent Service, oadrCreatedEvent Payload** An event cancellation received by the VEN must be acknowledged with an oadrCreatedEvent with the optType element set as follows, unless the oadrResponseRequired is set to 'never": optIn = Confirm to cancellation optOut = Cannot cancel Note: Once an event cancellation is acknowledged by the VEN, the event shall not be included in subsequent oadrCreatedEvent payloads unless the VTN includes this event in a subsequent oadrDistributeEvent payload.
37	**VEN** A VEN Implementation must support pull model and can optionally also support push
38	**VTN** A VTN must support both a push and pull model
40	**VTN, EiEvent Service, oadrDistributeEvent Payload** The optional eiResponse object in oadrDistributeEvent must be included when responding to oadrRequestEvent.
41	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN must send a requestID value as part of the oadrDistributeEvent payload. Note: The requestID value is not required to be unique, and in fact may be the same for all oadrDistributeEvent payloads. That there are two requestID fields in oadrDistributeEvent. The feild that must be populated with a requestID is located at oadrDistributeEvent:requestID
42	**VEN, EiEvent Service, oadrCreatedEvent Payload** A VEN receiving an oadrDistributeEvent eiEvent must use the received requestID value in the EiCreatedEvent eventResponse when responding to the event. This includes any and all subsequent EiCreatedEvent messages that may be sent to change the opt status of the VEN. The eiResponse:requestID in oadrCreatedEvent shall be left empty if the payload contains eventResponses. The VTN shall look inside each eventResponse for the relevant requestID
43	**VEN, EiEvent Service, oadrDistributeEvent Payload**

		The VEN must make no assumptions regarding the uniqueness of requestID values received from the VTN in the oadrDistributePayload
44		**VEN/VTN** With the exception of oadrDistributeEvent and oadrCreatedEvent payloads, requestID may be an empty element in other payloads and if a requestID value is present, it may be ignored
45		**VEN/VTN** Messages sent between VENs and VTNs shall *not* include a schemaLocation attribute
46		**VEN/VTN** Optional elements do not need to be included in outbound payloads, but if they are, the VEN or VTN receiving the payload must understand and act upon those optional elements
47		**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** An event with an overall duration of 0 indicates an event with no defined end time and will remain active until explicitly cancelled.
48		**VEN/VTN** When a VTN or VEN receives schema compliant oadr payload that has logical errors, the receiving device must provide an application layer error indication of 4xx. The detailed error message number is informational and not a requirement for response to a specific scenario. If the error is in an event contained in an oadrDistributeEvent payload, it should be reported in the eventResponse element of oadrCreatedEvent. The following logical errors must be detected by implementations: • VEN receives non-matching market context • VEN receives non-matching eiTarget • VEN receives unsupported signalName • VTN receives non-matching eventID in oadrCreatedEvent Response • VTN receives mismatched modificationNumber in oadrCreatedEvent
49		**VEN/VTN** The following responseCodes may be used by implementations in reporting application layer error conditions: 401 Unauthorized 404 not found (maybe if attempting to change a non-existing event ID) 405 not allowed - if you try to change an event in the past? 408 request timeout - you took too long to opt in/out of an event 409 conflict - you're trying to create an event ID that already exists
50		**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** In both the push and pull model, oadrDistributeEvent MUST contain all existing events which have the eventStatus element set to either FAR, NEAR, or ACTIVE. Events with an eventStatus of cancelled MUST be included in the payload upon change to the modificationNumber and MAY be included in subsequent payloads.
51		**VEN/VTN** Implementations that support both push and pull models may operate in a dual mode, exchanging messages using either model during any communication session.
52		**VTN, EiEvent Service, oadrDistributeEvent Payload**

	If a VTN requests acknowledgment of a cancelled event with oadrResponserequired of always, the VTN shall continue to send the cancelled event to the VEN until the event is acknowledged, eventStatus transitions to the complete state, or some well defined number of retries is attempted
53	**VEN/VTN** Shall implement the simple http transport. Including support for the following mandatory http headers: • Host • Content-Length • Content-Type of application/xml
54	**VEN** HTTP PULL VEN's MUST be able to guarantee worst case latencies for the delivery of information from the VTN by using deterministic and well defined polling frequencies. The VEN SHOULD support the ability for its polling frequency to be configured to support varying latency requirements. If the VEN intends to poll for information at varying frequencies based upon attributes of the information being exchanged (e.g. market context) then the VEN MUST support the configuration of polling frequencies on a per attribute basis.
55	**VEN** HTTP PULL VEN's MUST NOT poll the VTN on average greater than some well defined and deterministic frequency. THE VEN SHOULD support the ability for the maximum polling frequency to be configured.
56	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with an eventID that the VEN is not aware then it should process the event and add it to its list of known events
57	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with an eventID that the VEN is already aware of, but with a higher modification number then the VEN should replace the previous event with the new one In its list of known events.
58	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with an eventID that the VEN is already aware of, but which has a lower modification number than one in which the VEN is already aware then this is an ERROR and the VEN should respond with the appropriate error code. Note that this is true regardless of the event state including cancelled.
59	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with the eventStatus set to cancelled and has an eventID that the VEN is aware of then the VEN should cancel the existing event and delete it from its list of known events.
60	**VEN, EiEvent Service, oadrDistributeEvent, oadrCreatedEvent Payload** If the VTN sends an oadrEvent with the eventStatus set to cancelled and has an eventID that the VEN is not aware of then the VEN should ignore the event since it is not currently in its list of known events, but still must respond with the createdEvent if required to do so.by oadrResponseRequired
61	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends the oadrDistributeEvent payload and it does not contain an event for which the VEN is aware (i.e. in its list of known events) then the VEN must delete it from its list of known event (i.e. implied cancel). Exception: A VEN that has an active event that cannot be immediately stopped for operational reasons, may leave the event in its data store until

	the event expires or the event can be stopped.
62	**VEN, EiEvent Service, oadrDistributeEvent, oadrCreatedEvent Payload** The VEN must process EVERY oadrEvent event message (new, modified, cancelled, etc.) that it receives from the VTN in an oadrDistributeEvent payload and it MUST reply with a createdEvent message for every EIEvent message in which the responseRequired is set to always. Furthermore if the responseRequired is set to never, the VEN MUST NOT respond with a createdEvent message. It is at the complete discretion of the VTN as to whether responses are required from the VEN. Note that this rule is universal and applies to all scenarios including the following: • The event is one in which the VEN is already aware. • The event is being cancelled and the VEN did not even know it existed • It does not matter how the EIEvent payloads were delivered, i.e. PUSH, PULL or as the result of being delivered in an ALL payload
63	VTN: The VTN must not include more than one venID in the oadrDistributeEvent eitarget.
64	**VEN, EiEvent Service** A pull VEN shall respond to all received events before initiating another polling cycle.
65	**VEN, EiEvent Service, oadrDistributeEvent, oadrCreatedEvent Payload** When an event containing a randomization value in the startafter element is cancelled, either explicitly or implicitly, the VEN MUST randomize its termination of the event. The randomization window should be between 0 and a duration equal to the value specified in startafter.
66	**VEN/VTN, EiEvent Service, oadrDistributeEvent, Payload** If a VTN sends an oadrDistributeEvent payload containing an event with a startafter element with a value greater than zero, the VTN must continue to include the event in oadrDistributeEvent payloads, even if the event is complete, until current time is equal to dtStart plus duration plus startafter. The receipt of an eventStatus equal to completed shall not cause the VEN to change its operational status with respect to executing the event.
67	**VEN/VTN** VTN and VEN shall support TLS 1.0 and may support higher versions of TLS provided that they can still interoperate with TLS 1.0 implementations. The default cipher suite selection shall be as follows: • The VEN client shall offer at least at least one of the default cipher suites listed below • The VEN server shall must support at least one of the default cipher suites listed below and must select one of the default cipher suites regardless of other cipher suites that may be offered by the VTN client • The VTN client must offer both the default cipher suites listed below. • The VTN server must support both of the default cipher suites listed below and must select one of listed the default cipher suites regardless of other ciphers that may be offered by the VEN client Default cipher suites:

	- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA - TLS_RSA_WITH_AES_128_CBC_SHA Note that a VTN or VEN may be configured to support any TLS version and cipher suite combination based on the needs of a specific deployment. However in the absence of changes to the default configuration of the VTN or VEN, the behavior of the devices shall be as noted above.
68	VEN/VTN Both VTNs and VENs shall support client and server X.509v3 certificates. A VTN must support both an ECC and RSA certificate. A VEN must support either an RSA or ECC certificate and may support both. RSA certificates must be signed with a minimum key length of 2048 bits. ECC certificates must be signed with a minimum key length of 224 bits. ECC Hybrid certificates must be signed with a 256 bit key signed with a RSA 2048 bit key.

11.4 Cardinality

The OpenADR 2.0 profile specification modifies the cardinality of the OASIS Energy Interoperation Specification in order to ensure consistent implementations. The following table maps the relevant changes.

Object	Element	EI	OADR 2.0a
EiEventType	eventDescriptor:eventStatus	0 – 1	1
EiEventType	eventDescriptor:createdDateTime	0 – 1	1
EiEventType	eiActivePeriod:properties	0 – 1	1
EiEventType	eiActivePeriod:properties/dstart	0 – many	1
EiEventType	eiActivePeriod: properties:duration	0 – many	1
EiEventType	eiActivePeriod: properties:tolerance	0 – many	0 – 1
EiEventType	eiActivePeriod:properties:x-eiNotification	0 - many	1
EiEventType	eiActivePeriod: properties:x-eiRampUp	0 – many	0 – 1
EiEventType	eiActivePeriod: properties:x-Recovery	0 – many	0 – 1
EiEventType	eiEventSignals:eiEventSignal	0 – many	1 - many
EiEventType	eiEventSignals:eiEventSignal:intervals	0 – 1	1
EiEventType	eiEventSignals:eiEventSignal:currentValue	0 – 1	1

EiEventType	eiEventSignals:eiEventSignal:intervals:interval	0 – many	1 - many
EiEventType	eiEventSignals:eiEventSignal:intervals:signalInterval:duration	0 – many	1
EiEventType	eiEventSignals:eiEventSignal:intervals:signalInterval/uid	0 – many	1
EiEventType	eiTarget	0 – 1	1

11.5 Services used from OASIS Energy Interoperation V1.0 Standard

The OpenADR 2.0 Profile Specification uses a number of services defined and required by the OASIS Energy Interoperation standard. The following services are currently supported:

- EiEvent

- EiOpt (not in OpenADR 2.0a)

- EiQuote (not in OpenADR 2.0a)

- EiReport (not in OpenADR 2.0a)

- EiRegister (not in OpenADR 2.0a)

11.6 Services not used from OASIS EI

The OpenADR 2.0 Profile Specification currently does not use the following services included in the OASIS Energy Interoperation OpenADR profile

- EiAvail

- EiEnroll

- EiMarketContext

Annex A – Registration - Non-Normative

A.1   Summary

The purpose of this Annex is to outline common registration scenarios between VEN and VTN. Registration is defined as the VTN becoming "aware" of the VEN, e.g. exchange of credential and possibly other information so that the VTN can properly authenticate the VEN. Note that this specifically does not include enrolling in programs.

Note also that in all scenarios described below, the XMPP connections would work the same as "HTTP pull" since in both cases, the VEN acts as a client (party that initiates a connection to a server.)

A.2   Out-of-Band Registration

"Out-of-band" means simply, outside of any OpenADR communication. It is covered here because it is a necessary prerequisite for OADR communication to occur. Program operators and VTN vendors shall implement appropriate security measures for this communication.

A.3   Scenario 1: Registration through VTN Web Portal

Assume VENs have a factory-installed certificate, and a public key fingerprint that is for example printed on a label on the VEN hardware.

At installation time, the user logs into the VTN portal from a browser. They then register the VEN by entering the public key fingerprint along with the account information that the VEN should be associated with.

(An aside: If the "user" is a residential customer, the account info might be implied by the user's portal credentials. If the "user" is a field technician or customer with multiple properties, then some additional information would likely be input at the portal to associate the VEN with a particular site/ meter, etc.   This is all specific to the deployment and is ultimately inconsequential to the registration mechanics here.)

Given the public key fingerprint, the VTN now has the ability to authenticate the VEN the first time it connects via OpenADR. The OpenADR connection will be made over TLS. When initiating the TLS connection, the VEN will include its client certificate. The certificate chain is then validated against the VTN's root CA store, and the certificate's public key fingerprint is compared to what was given during registration.

On the VEN side, a pull configuration would require inputting the VTN's server address and possibly the venID.

A pull VEN does not need any specific information about the VTN's server certificate prior to connection, so long as the VEN has a list of root CAs and can validate the VTN's certificate chain against a trusted root CA. The VEN would also validate the CN field of the certificate against the domain of the server.

In an HTTP push scenario, registration would be almost the same, but data input through the VTN portal would also include the IP/ host address of the VEN. The VTN, upon connecting to the VEN, would now validate the VEN's server certificate (which under most scenarios should be the same certificate as the "client certificate" used in the pull scenario.) The VEN would also need to know the fingerprint of the client certificate used by the VTN, which would presumably be input in a similar fashion through some user interface provided on the VEN.

A.4   Scenario 2: Certificate Installation on the VEN

In this scenario, the VTN owner (the utility, aggregator, etc) issues certificates, which are installed on the VEN through some out-of-band interface. Then when a VEN client connects, the VTN can validate the certificate with the information it already has.

Note that for the HTTP push case, for the VEN to have a server certificate issued by the VTN would require the VEN to operate its push server on multiple ports or virtual hosts if it wanted to interact with more than one VTN. It is more likely in this case that the VEN would issue a client certificate which the VTN would use to connect, and the VEN would use either a factory-installed certificate, or a certificate that is issued by a trusted CA to match the VEN's server hostname (e.g. the same way a VTN will get a server certificate.)

A.5   In-Band Registration

While this scenario is not 100% "in-band," the bulk of the interaction may occur via an OpenADR message, through the use of a pre-shared key.

One example of an in-band registration process could work like this:

The VEN owner obtains a "registration key" from the VTN owner, e.g. through logging into a utility portal and generating an alphanumeric key. The user then inputs this key into the VEN, along with the connection information (e.g. hostname/IP) for the VTN. The VEN then connects to the VTN, and sends an "oadrRegister" payload, which includes the pre-shared key. Note that at this point, TLS negotiation has already occurred, meaning the channel is encrypted but VTN must not attempt to validate the client certificate yet.

Now, the VTN can validate the pre-shared key. If the key is valid, the VTN will associate the given client certificate with the account information which was used when the pre-shared key was generated. Both parties may then exchange additional registration information, such as the VTN telling the VEN what its venID should be, etc.

Note that these pre-shared keys should be one-time use. Once registration is complete, both parties should discard the pre-shared key: TLS certificates are used for authentication after that point (the same as prior described scenarios) and the pre-shared key is only used to authenticate the VEN during the initial "handshake."

A.6   A Note on Server Certificates

Most VTNs will likely be sold without any server certificate. When the VTN server is deployed, VTN owner will buy a server certificate which will match the hostname on which the VTN operates (for instance openadr2.pge.com, oadr2.enernoc.com, etc.) In order for any VEN to authenticate the VTN when it connects, the VEN needs a list of trusted root CAs from which all VTNs will purchase their server certificates. This list of trusted CAs will likely be installed on the VEN at manufacture time. How or whether the trusted CA list is updated should be a deployment concern/ vendor feature.

---

i http://xmpp.org/rfcs/rfc6120.html#stanzas-error
ii http://xmpp.org/extensions/xep-0030.html
iii http://xmpp.org/extensions/xep-0030.html#info