OpenADR Open Source Toolkit: Developing Open Source Software for the Smart Grid

Charles McParland
Lawrence Berkeley National Laboratory

July 2011

DISCLAIMER

OpenADR Open Source Toolkit: Developing Open Source Software for the Smart Grid

Charles McParland,

Computational Research Division,

Lawrence Berkeley National Laboratory

***Abstract*—Demand response (DR) is becoming an increasingly important part of power grid planning and operation. The advent of the Smart Grid, which mandates its use, further motivates selection and development of suitable software protocols to enable DR functionality. The OpenADR protocol has been developed and is being standardized to serve this goal. We believe that the development of a distributable, open source implementation of OpenADR will benefit this effort and motivate critical evaluation of its capabilities, by the wider community, for providing wide-scale DR services.**

***Index Terms*—OpenADR, open source, demand response.**

I. Introduction

While demand response systems have been an active area of interest and research over the past decade, the Smart Grid initiative, which mandates some form of demand response load shedding [1], has spurred increased interest in this field. One of the goals of the Smart Grid, namely, the achievement of a common, unified and standards-driven grid communications architecture, has motivated a closer examination of software frameworks and protocols capable of supporting wide-scale demand response functionality. One of the protocols undergoing evaluation is OpenADR.

Much of the current research in demand response is focused on relatively isolated pilot implementations. However, the prospect of a common, grid-wide demand response protocol requires the shared focus of a much larger and coordinated community of experts. The open source software movement [2] has succeeded in creating and coordinating such "communities of interest" around efforts of similar scale (e.g. the Linux OS). In looking at the level of analysis and effort required to implement a truly Smart Grid–capable demand response protocol, the open source community model can provide both the level of skills and breadth of experience needed to guide a new protocol into the wider grid community. We have attempted to seed this effort by producing an open source version of the OpenADR demand response protocol and are in the process of packaging and releasing it to the research community. Our goal is to provide a common code base that can drive both innovative experiments and formal protocol verification activities and, ultimately facilitate ubiquitous demand response capabilities within the emerging Smart Grid.

Manuscript received November 30, 2010. This work was sponsored in part by the Demand Response Research Center which is funded by the California Energy Commission (Energy Commission), Public Interest Energy Research (PIER) Program, under Work for Others Contract No. 500-03-026 and by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

C. McParland is with the Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley CA. (Phone 510-486 6956; e-mail: mcparland@lbl.gov).

II. Demand Response architectures

Early demand response efforts were primarily manual in nature. Requests to reduce demand were typically made a day or more in advance and communicated to the end user through fax or telephone messaging. Once received, local energy management system set points were altered to reduce consumption in accordance with the communicated request and contractual requirements. There was little automation in either the generation/distribution of the request or in effecting an appropriate response.

In the 1990s, the growing availability of inexpensive computing equipment and wide scale data communications infrastructure allowed the development of systems capable of automating both the communications and response aspects of DR activities. From these experiments, there emerged a conceptual view of demand response as a power grid system behavior that could be operationally used to alter energy demand levels in both an automated and time-bounded manner – thus the term "automated demand response or ADR. As demand for energy continued to approach existing generation capacity, the importance of ADR as a key methodology for matching energy generation capacity and demand also increased. Figure 1 illustrates the system load "shaping" behavior measured in a single building [3].

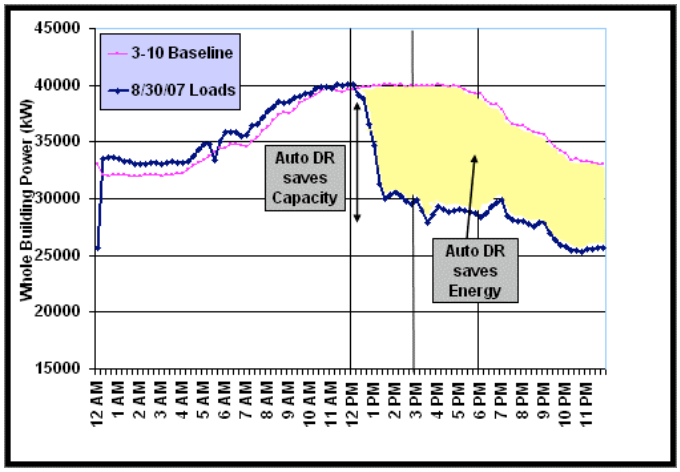

Figure 1 – Automated DR Load Shaping.

III. OpenADR Architecture

OpenADR [4] (Open Automated Demand Response) was developed at the Demand Response Research Center [5] (DRRC) as part of an ongoing effort to help building and facilities managers implement automated demand response within their facilities. It was designed to allow buildings to invoke pre-planned demand shedding strategies quickly and automatically when requested by utility operations. By integrating automated building responses into everyday grid operations, it also enabled utilities to promote commercial and industrial participation in new power pricing programs that leverage automated demand response behavior from end users.

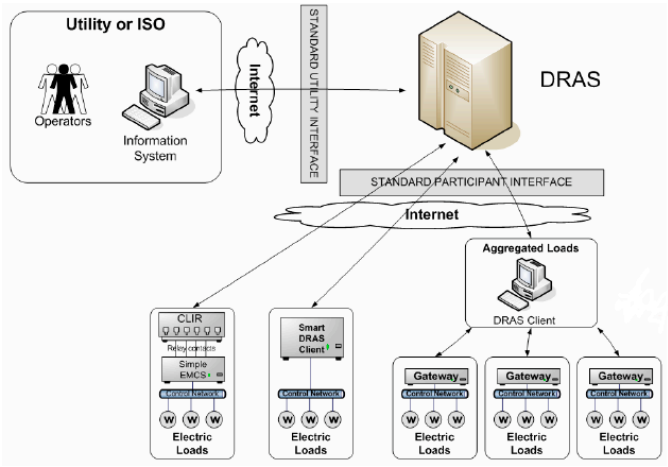

Figure 2 – OpenADR Architecure.

At its core, the OpenADR architecture defines a data model for energy cost and reliability that is common among and relevant to an energy provider and its customers. By providing a common data model among these parties and defining the semantics for accessing and changing elements within this model, energy providers and consumers can efficiently exchange demand response requests based on both price and grid reliability criteria.

One particular feature that differentiates OpenADR from other automated DR architectures is that utility DR requests contain no information about specific devices or operations that should be curtailed or stopped. OpenADR only conveys the utility's request for demand reduction to the customer - either by direct request or through distribution of increased energy cost rate schedules that will, in turn, motivate load shedding and reduced demand. Specific consumer-side responses to these requests are formulated by and completely under control of the end user. The overall result is that energy consumers can respond to utility DR requests in ways that are most effective and convenient for each local. In addition, the OpenADR messaging protocol has provisions for customers to individually respond to DR requests with an "opt out" signal – further increasing flexibility permitted for customer response. The end result is a system that promotes automated responses to utility DR requests while maximizing the local flexibility exercised in responding to those requests.

Take, for example, the case where a utility must shed load to maintain grid stability. It knows that one or more pre-registered clients are participating in a particular DR "program" and that they have agreed to respond, within certain pre-arranged limits, to utility requests to shed load over a given time interval. The magnitude of the expected load shed and the speed at which it can be affected are fully specified by the specific DR "program" to which a client has subscribed. The utility issues a request ("DR event") to participating clients to shed load and receives verification that that the request has been received. Clients then begin shedding load in keeping with pre-agreed criteria. Failure to perform as agreed, as indicated by interval-recording revenue meter data, will result in penalties at a later date when utility rate charges are reconciled. An OpenADR client may optionally indicate that it will not participate in a DR program ("opt out") for some period of time. While this typically does not override a client's obligations to comply with DR program requests, such a response may ease the complexity of utility dispatch operations.

A second DR use case is one in which utilities modify energy costs in an attempt to motivate reduction in demand. In this case, an OpenADR server can issue "price events" that describe elevated energy costs over a particular time interval. Participating clients can respond to these requests by shedding load, according to the communicated schedule, and reduce their energy costs. Conversely, some clients may decide to accept the "opportunity costs" of higher energy rates because local conditions require energy consumption to continue at normal rates. Since utility back office operations are ultimately responsible for local "time of use" readout and revenue settlement, local DR behavior can remain relatively independent of utility DR requests while accurate contractual or revenue obligations can be insured.

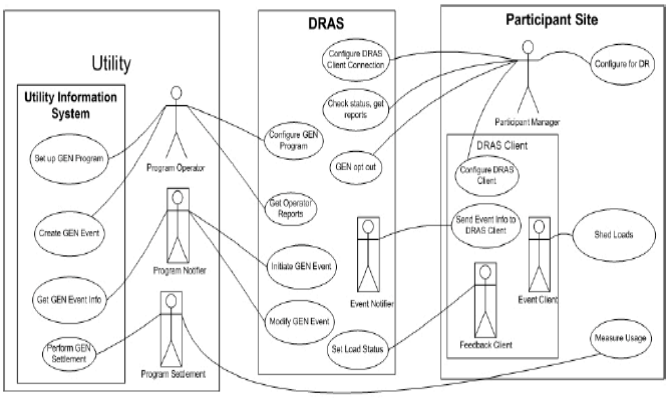

Figure 3 – Generic Roles, Event-based Use Case.

In practice, OpenADR is implemented as a client/server system that highly leverages design elements found in similar successful commercial Internet designs. It is readily adapted to widely-available Internet communications infrastructures and, due to its hierarchical design can mirror the layered organizations typically found in energy distribution systems – namely utilities, aggregators, and end users. The following diagram depicts typical OpenADR elements. (note: DRAS acronym commonly used for Demand Response Automation Server).

Clearly, the exchange of model data element between client and server is critical to this design. In order to effectively share model data between participating entities, OpenADR defines an extensive set of XML (eXtensible Markup Language [6]) formatted messages that describe model element identifiers and their values. These XML-formatted messages are used to communicate current and future energy pricing, time of use pricing schedules and as well as explicit demand reduction requests between the OpenADR Demand Response Automation Server (DRAS) and its clients.

```
<?xml version="1.0" encoding="UTF-8" ?>
<p:eventStateprogramName="LBNL" eventModNumber="0"
  eventIdentifier="LBNL-102141" drasClientID="akua1"
  eventStateID="0" schemaVersion="1.0" drasName="DRAS 1.0"
  testEvent="false" offLine="false" xmlns:p="urn:EventState"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
     "urn:EventStatehttp://openadr.lbl.gov/src/1/EventState.xsd">
  <p:simpleDRModeData>
     <p:EventStatus>ACTIVE</p:EventStatus>
     <p:OperationModeValue>MODERATE</p:OperationModeValue>
     <p:currentTime>125</p:currentTime>
     <p:operationModeSchedule>
     <p:modeSlot>
        <p:OperationModeValue>MODERATE</p:OperationModeValue>
        <p:modeTimeSlot>120</p:modeTimeSlot>
     </p:modeSlot>
  </p:operationModeSchedule>
  </p:simpleDRModeData>
  <p:drEventData>
     <p:notificationTime>2009-03-26T10:25:00.0</p:notificationTime>
     <p:startTime>2009-03-26T10:27:00.0</p:startTime>
     <p:endTime>2009-03-26T10:30:00.0</p:endTime>
  </p:drEventData>
```

Figure 4 – Example OpenADR XML-formatted Event.

When this common data model is widely adopted by energy consumers, OpenADR provides the following benefits:

- Simplify and reduce DR-related costs by use of standardized messaging formats.
- Promote interoperability between utility servers and multi vendor clients.
- Increase customer participation and reduce operating cost associated with manual responses to DR requests through use of automation.
- Allow energy consumers to customize local response to utility DR requests.
- By promoting standardized architecture and message formatting, encourage wide-scale integration and embedding of DR capabilities into consumer devices.

IV. Placing OpenADR in to the Smart Grid Environment

The cornerstone of the Smart Grid is the requirement that data and control communications essentially "follow" the flow of power from the generation facility, through the distribution automation infrastructure, into the end user facility and, ultimately to the load itself. While much of the current power grid head end is already extensively interconnected, substantial effort, within the Smart Grid initiative, is being focused on "securitizing" these grid communications against cyber attacks.

However, at the other end of the grid – that of the power consumer, there is little (i.e. pre-Smart Grid) communications infrastructure. At present, new meter communications systems (Advanced Metering Infrastructure, or AMI [7]) are being installed across the US in order to extend grid communications to the residential meter – the traditional power grid/consumer demarcation boundary. Furthermore, these new AMI systems interconnect, within the facility, with yet another new data network – the Home Area Network (HAN [8]) – which is intended to be the utility-connected control network for major appliances and sub systems within a facility or residence. While a detailed evaluation of these new Smart Grid communications networks is beyond the scope of the paper, it is worth discussing their potential impact on DR architectures and implementation.

All DR architectures require some communications mechanism for signaling load shedding requests – either directly or, by implication, through notification of increased rate schedules. While the communications path described above does constitute a way to communicate between utilities and their customers, it is just now being deployed widely within the US and, operationally, is still in a formative stage. AMI systems and interfaces are proprietary by design and vary across the entire Smart Grid topology – occasionally, different AMI systems serve customers within the same city. Furthermore, the potential interfaces through which non AMI messages can be injected into these systems vary widely or are non-existent- across AMI vendors. The result is that, at this point in time, a DR standard that targets wide-scale

deployment cannot effectively reference details of these communications channels.

As a result, successful DR protocols and standards need to avoid explicit references to network transport layers and should be capable of readily adapting to the communications channels available in a given local. The most prevalent of these channels, with market penetration of between 70% and 80% [9] is the Internet, which is typically accessed through either broadband or DSL technologies. The OpenADR V1.0 specification, while making no explicit demands on underlying communications layers, has been developed, tested and successfully deployed using these standard IP (e.g. Internet) protocols. In terms of advancing DR research in anticipation of its wide-scale deployment, we believe that use of existing Internet communications standards provides the best foundation for large pilot deployments and ongoing research. Furthermore, by focusing attention at the application – and not the communications – level, we ensure the required level of adaptability needed for future ubiquitous deployment within the Smart Grid.

V. Relationship between OpenADR and Smart Energy Profile (SEP)

As mentioned earlier, the desire to control or modify energy loads has led to a number of messaging systems, each with their own formats and semantics. Currently, two specifications, OpenADR and SEP (Smart Energy Profile) 2.0 [10], appear most frequently in discussions about wide scale, automated DR architectures. Both of these are intended to complete the standards process and be suitable candidates for communicating DR requests between utilities and consumers.

Both OpenADR and SEP 2.0 are considered to be application-level protocols and, as such, are focused on providing services and functions that are relevant within their specific domain or application area. They typically specify how to encode high level functions, e.g. "change thermostat set point" or "here is tomorrow's energy price schedule", into specific messages and codify all legal responses to such messages during normal operations. With few exceptions, they are not concerned about the actual mechanics of message transmission and reception (i.e. message transport) – those responsibilities are left to lower level communications protocols (e.g. IP). At their heart, application protocols are focused on describing the behaviors of each communicating component. Each as its origins and strengths in a particular part of power grid operations – as described below:

OpenADR grew, in part, out of early utility-level DR programs that communicated price and reliability information to buildings and end users thus allowing them to reduce demand on a voluntary basis. One of the key innovations of OpenADR was the creation and communication of a standardized data representation (using XML) of these pricing and supply conditions that would allow end user control systems to respond in an automated and timely way. As an applications level protocol, OpenADR leveraged existing Internet communications infrastructure. And, as the reach and reliability of the Internet grew, so did its ability to interact with power consuming clients over a wide geographic ares. Like earlier manual DR programs, end users still defined the manner in which they responded to these signals, but the OpenADR protocol allowed truly automated response to DR signals.

SEP 2.0 has taken a different evolutionary path. It has grown out of an effort to create a standardized application layer for use with low-power, 802.15.4 [11] radio equipped hardware platforms that supported the Zigbee software network protocol. The Zigbee Alliance has energetically specified both a software and hardware "ecosphere" that supports applications in a number of key areas (e.g. device-level energy management) using computing platforms that were known to be sufficiently inexpensive to make their ubiquitous inclusion in home appliances both attractive and certain. Since the primary function of automation within the home has always been the actual control of devices, the software model found in the energy application specification (SEP 1.0) naturally focused explicitly on the control of devices. With SEP's evolution to SEP 2.0 through the adoption of the IP protocol stack, it became possible for this programming model to function in wide area networks as well.

At this point in time, we feel these two DR-capable models complement each other. OpenADR, implemented primarily on web-base transport infrastructures, is a good fit at the enterprise end of distribution grid operations. And, SEP 2.0, growing out of the cost effective device control domain, is a good fit within the home-based, device control world. While technical advancements in both the micro-controller and networking domains can allow each of these protocols to extend across the entire utility to customer to device spectrum, we feel each has a natural domain within which they are best suited to meet application needs.

VI. Motivation – does open source make sense?

The open source development model has been widely adopted within the Internet community and the larger software development world. Although the rules governing behavior within this community take on varying forms, the general precept of "shared responsibility for development of quality software" is well accepted and ubiquitous across both the academic and commercial domain. While some companies initially viewed open source efforts as eroding the value invested in proprietary software, over time it has been shown that, where individual applications share common functionality, open source programs can provide code that is more correct and reliable than proprietary implementations. Individual and corporate justification for actively supporting open source projects vary from creating an increased sense of professional "community" to allowing start-up companies to focus on value-added portions of their product while leveraging common open source-supported web frameworks. Regardless of individual motivations, there can be little argument about the success and usefulness of open source

projects such as Linux and the Apache Foundation's Tomcat web application server in the commercial world.

However, while many facets of the current power grid resemble the Internet, they are, in fact, very different software environments and the question "does open source have a place in Smart Grid" is worth addressing. The software infrastructure of the power grid is purpose-built and, in general, a very conservative programming environment. Overall power grid stability and safety are paramount and inadvertent or intentional destruction of key system elements can have catastrophic monetary consequences. Thus, within the power generation and distribution automation portion of the grid, the advantages of the open source model may not be justified.

Some areas, such as utility back office operations, are essentially IT environments, and freely use open source software solutions where appropriate. These business support services, which are technically still part of the greater Smart Grid model, may find open source tools attractive for cost and reliability reasons. However, since they exist outside of the utilities operations domain, they have little impact on the grid per se.

So, what is the nature of the open source opportunity in the DR world? It can be argued that, with advancement of the Smart Grid initiative, some degree of automated demand response will be required by all operating utilities within the US and these DR systems will reach most, if not all, of their energy customers. Successful DR systems of this scale will use or, at a minimum, leverage proven Internet-based tools capable of scaling to service these large markets. This is precisely the environment in which open source tools have already proven their value. Furthermore, since the OpenADR architecture has evolved within and proven its capabilities in the Internet-mediated environment, it will very likely benefit from the attention of a developer community intent on providing high quality, reliable web-based services. Therefore, we believe there is both research and, potentially, commercial value in producing an open source OpenADR implementation and exploring and analyzing this protocol within the wider open source and DR communities.

VII. Open Source OpenADR goals

The open source OpenADR project has four primary goals. The first of these is to provide a vehicle for educating researchers, utilities and vendors about the OpenADR protocol. Much of the previous work in the DR domain has focused on direct control of customer loads. As noted above, this DR methodology is rooted in a centralized control model that is fundamentally different than OpenADR. Past discussions about the suitability of OpenADR for wide-scale deployment have made it clear that some confusion about the basic OpenADR behavioral model still remains. We believe that, by distributing a simple, research-quality OpenADR server and client demonstration program, we will address these misunderstandings through direct and meaningful demonstration. Furthermore, by giving the wider research community access to a simple, OpenADR software environment, we will promote widespread experimentation by potential OpenADR implementers.

The second goal is to provide a basic software "core" that is capable of supporting simple OpenADR demonstrations and pilots with a minimum amount of software development. As noted above, our intention is to produce a "research-grade" OpenADR server and client. In other words, there is no intention to produce, at least at this stage of development, an OpenADR server that is suitable for continuous, commercial applications. However, given the state of development in the DR domain, there are research areas that will clearly benefit from data gained through focused pilot programs. One of our goals is to facilitate these research activities by providing a simple framework capable of being altered to meet immediate research goals.

Thirdly, we want to produce an independent OpenADR implementation code base that is suitable for future protocol conformance testing activities. As noted above, OpenADR is currently a published, publicly available specification (OpenADR V1.0) and is undergoing formal standardization process within both the OASIS [12] and UCA [13] standards organizations. Looking forward to the successful completion of these efforts, the OpenADR community will require a compliance and interoperability testing framework for verification of new, third party OpenADR implementations. While we cannot foresee the exact outcome of these standardization efforts, we feel this project provided an opportunity to initiate the design of such a package.

And, lastly, we want to produce a modular OpenADR implementation that lends itself to integration into a variety of smart grid and energy market simulation frameworks. While utilities and independent system operators (ISOs) have used simulation programs to evaluate physical power distribution designs, few such frameworks are available to evaluate large, metropolitan-scale aggregations of energy consumers that are responding to energy price and reliability signals. Discussions with several utilities indicated that the usefulness of such simulation frameworks would be greatly enhanced if they could include modeling of auto DR signaling and response behaviors. Our last goal is to insure that the OpenADR code base produced within this project is sufficiently modular and granular to allow its integration into software frameworks that are substantially different, architecturally, from the "application server" environment required to satisfy our other goals.

VIII. Implementation specifics

As noted above, OpenADR is essentially a client/server architecture that should easily integrate into the existing Internet Web Service environment. We chose to implement both the server and client portion of this project in the Java language. While other choices were available to us (e.g. C++), programming in the Java language enhanced the

number of potential open source tools available for use in addressing other aspects of the implementation. The prevalence of Java in web-based software packages and the availability of language specific tools for creating, manipulating and binding XML documents to program elements argues strongly for its use on the server side. On the client side, Java environments are readily available in PC-based systems and in many embedded Linux environments. While we have seen a number of minimal OpenADR clients programmed in either C or C++, we decided that a reference Java client implementation would be of great benefit to users.

The decision to use Java as the primary implementation language prompts the question – what is the appropriate execution environment for a research-grade OpenADR software implementation? Given recommendations of the present OpenADR V1.0 specification, server functions must, at a minimum, respond to standard Web Service requests (i.e. REST [14]) posted by OpenADR clients. While it is possible to compose and structure a stand-alone Java program that will accept http connection requests, interpret Web Service-formatted messages, and respond appropriately, open source frameworks capable of supporting such transactions are widely available and well supported. Java application servers, such as Apache Tomcat [15], Jetty 16] and JBoss 17], are widely used in both academic and commercial contexts and, most importantly, have large followings in the software development community. While we considered basing our implantation on the JBoss application environment, we decided that, in keeping with our original set of goals, the Apache Tomcat application server was the most appropriate open source web framework for the current software package. It is readily available, simple to install on both Windows and Linux environments and is generally well known in the web developer community.

Information about participating clients, utility program rate programs and applicable schedules is maintained in OpenADR's database. Regardless of the object oriented nature of our implementation, traditional relational databases were considered most appropriate for this role. However, differences between popular relational databases prompted lengthy discussions about which particular database would prove most useful. We considered MySQL [18], Oracle [19] and Postgres [20] as database candidates. Since, in keeping with the open source tradition, we wanted to only incorporate freely obtainable software components, we limited the Oracle implementation to the freely distributed Oracle Express package. All candidates implemented standard SQL[21] query languages and shared features typically found in modern databases – namely customized database table trigger functions. However, each used different commands for initial database configuration, table creation and user privilege management. Discussions with back office staff from several utilities and further discussions with vendors serving the utility marketplace led us to ultimately chose Oracle Express. This decision was based on the communities overall familiarity with the Oracle command interface and by the percieved ease of integration – even if only for pilot demonstration purposes – with existing utility operations and dispatch systems.

IX. Discussion

A. Security

The role of security in OpenADR has several aspects. In general, any Internet application that exchanges sensitive user information now receives increased scrutiny from designers viz. implementation of security features. Since OpenADR transactions involve no direct funds exchange, it could be considered a low security risk application. However, OpenADR messages do convey cost saving opportunities and malicious activities that manipulate the contents of these messages could create "opportunity costs" that, in effect, turn into very real utility bill differentials that must be reconciled. Therefore, OpenADR V1.0 has specified that, at a minimum, client/server messaging should follow best commercial web practices for security. In practice, commercial OpenADR DRAS systems implement Transport Layer Security (TLS [22]) for all critical message passing operations. It should be noted that the security measures described here exist as layers, recommended by, but not implemented as part of, OpenADR. Therefore, in designing an open source version of OpenADR, we have not added an explicit security layer as the details of this layer are outside the scope of the specification. In future releases, we will add layered security to promote use of this package in demonstrations with modest security requirements. These security layers will not explicitly interact with the core OpenADR code base.

It should also be noted that, for at least two of our motivating guidelines, explicit security layers will prove counter-productive. In particular, when used as part of a large simulation framework, the presence of a security fuctions as an explicit part of the OpenADR implementation would add unnecessary complexity and computational overhead. Furthermore, when using elements of this codebase as part of certification suite, the presence of security features would be a distraction for use case testing. Since OpenADR is a DR messaging content specification, it contains no explicit security behaviors. Therefore, while security issues play a critical role in overall design of a DR system, these issues have not role in proving the correctness of OpenADR message content.

B. Utility Operator Interface

The OpenADR specification focuses primarily on the interface between the utility and the consumer. Detailed use case descriptions and messaging examples delineate exchanges between these two entities. However, OpenADR does not proscribe or define the process by which a utility decides to declare DR events or the mechanism through which a utility operator instructs the OpenADR messaging system to initiate distribution of such events to participating clients. In practice, these details are dictated by utility policies and, with respect to the actual mechanism used to interact with the utilities DRAS, ddictated by the utility's IT environment.

Therefore, our open source implementation provides a simple interface through which an external operator can cause the server to evaluate its present state and, if required, formulate and emit events to clients as appropriate. Rather than include all the required utility state information in this interface, we assume that such information will already be present in the OpenADR server database prior to invoking event processing. In practice, this design closely maps onto typical utility and ISO (Independent System Operator) architectures. Rather than providing a very rich – and very specific – web service interface as part of event scheduling, utility-specific sub-systems continually update OpenADR tables with current DR targets and potential price schedule changes. On recognition of a need for system-wide demand reduction, an operator requests that an event be created and distributed through a simple server interaction.

X. Conclusion

The increased interest in demand response systems and the growing influence of the Smart Grid initiative has lead to the requirement for national standards for demand response protocols. Given the potential scale in which such standard protocols will operate, parallels to open source software efforts of similar scale are apparent. We have specifically developed an open source implementation of the OpenADR protocol with the purpose of facilitating the analysis and research of this protocol within the larger research and development community. The current effort has already promoted interactions with a number of independent OpenADR developers and we anticipate wider collaborations based on wider use of this open source codebase.

Acknowledgment

This work was sponsored in part by the Demand Response Research Center which is funded by the California Energy Commission (Energy Commission), Public Interest Energy Research (PIER) Program, under Work for Others Contract No. 500-03-026 and by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] David A. Wollman, "Status of NIST's EISA Smart Grid Efforts," Mar. 26, 2009, Congressional Noontime Briefing, R&D Caucus. Available: files.asme.org/asmeorg/NewsPublicPolicy/GovRelations/21405.pdf
[2] Open Source Intiative, http://www.opensource.org/
[3] Pacific Gas and Electric, http://www.pge.com/mybusiness/energysavingsrebates/demandresponse/adrp/
[4] Mary Ann Piette, Girish Ghatikar, Sila Kiliccote, Ed Koch, Dan Hennage, Peter Palinsky, and Charles McParland. "Open Automated Demand Response Communications Specification (Version 1.0)", 2009. LBNL-1779E. Available: http://openadr.lbl.gov
[5] PIER Demand Response Research Center, http://drrc.lbl.gov
[6] Extensible Markup Language (XML). Available: www.w3.org/XML
[7] Advanced Metering Infrastructure (AMI), Electric Power Research Institute.Available: http://www.ferc.gov/eventcalendar/Files/20070423091846-EPRI%20-%20Advanced%20Metering.pdf
[8] Home Area Network (HAN) Overview, Edison Foundataion. Available: www.edisonfoundation.net/iee/issuebriefs/PG&E_HAN_January_2009.pdf
[9] Consumer Electronics Association, "Broadband in America: Access, Use and Outlook". http://www.bizreport.com/2007/07/threequarters_of_us_households_have_broadband.html#
[10] "ZigBee Smart Energy 101", Available: www.zigbee.org/imwp/download.asp?ContentID=16600
[11] IEEE 802.15.4, Available: www.ieee802.org/15/pub/TG4.html
[12] OASIS, www.oasis-open.org
[13] UCA International Users Group, www.ucaiug.org
[14] Representational State Transfer, en.wikipedia.org/wiki/Representational_State_Transfer
[15] Apache Tomcat, http://tomcat.apache.org
[16] Jetty Web Server, www.mortbay.org
[17] JBoss Application Server, www.jboss.org
[18] MySQL, www.mysql.com
[19] Oracle, www.oracle.com
[20] PostgresSQL, www.postgresql.com
[21] SQL Tutorial, www.sql.org
[22] The TLS Protocol Version 1.0 (RFC 2246), Internet Engineering Task Force, www.ietf.org/ifc/rfc2246.txt