Next Generation Automation – Effective Platform Design and Practical Implementation

Daniel Evans, Sam Hendley, Adam Crain, John S. Camilleri

NCSU Centennial Park Campus
Venture IV Building
1730 Varsity Drive, Suite 500
Raleigh, NC 27606

devans@greenenergycorp.com, shendley@greenenergycorp.com, acrain@greenenergycorp.com, jcamilleri@greenenergycorp.com

Keywords: Automation, SOA, ESB, Messaging

Abstract

There are a number of challenges in meeting the defined NIST architecture while driving interoperability. Application modularity, integration with existing enterprise data models, leveraging existing enterprise architecture and supporting increasing volumes of diverse field data (including meter data, phasor measurements, traditional telemetry for operations, power quality and new remote sensing devices) are among some of the key challenges today.

This paper describes a service-oriented architecture for data acquisition and control leveraging some of the latest concepts in software engineering. By addressing the business requirements to support role-based applications and services, the platform removes the "islands of automation" issue and allows a vertical decomposition of the application stack to lower the barrier to entry for new applications. The platform supports horizontal scalability for performance and is distributed, by design, for reliability and high availability. In addition, this paper addresses the detailed design issues around support, performance, reliability, security and scalability.

Finally, we will showcase a practical implementation of the basic concepts of data acquisition and control while leveraging an industry standard enterprise data model to support interoperability between heterogeneous systems. This platform design shows how today's well defined standards coupled with emerging standards can be leveraged in a modular approach to minimize the customer's risk and reduce overall development support for vendors.

1. KEY CHALLENGES

The Electric Utility Industry is under an ever increasing pressure to support the demands of the "Smart Grid". The challenge is complex due to the variety of systems and business functions that exist today and those that are being considered the smarter grid.

Some of these challenges are identified as the following:

- Bridging the operational technologies(OT) and information technologies (IT) to support increasing interoperability needs
- Avoid architecture brittleness
- Remove the island of automation
- Exponential growth in remote sensing data
- manage the effects of a diminishing workforce
- Harmonization of enterprise data models
- Leveraging/Enhancing the legacy IT investment

The Guidelines for Smart Grid Cyber Security - NIST IR 7628 [1] help identify the overall number of systems, interoperability and guidelines for cyber security requirements. The main thrust of this paper is to describe an open platform approach that addresses the challenges identified above and leverages the NIST IR as a standard reference model.

2. DESIGN PRICINPLES

In order to address the key challenges the platform supports the following design principles. With these principles in place barriers to innovation are removed.

2.1. Open Platform

The concept of openness is typically focused on a specific protocol, standard or programming interface rather than a holistic view of a system. The word "open" has become a marketing buzzword with no real meaning. Therefore it requires a clear understanding of the platform lifecycle to evaluate with "open" in the context of the specific system or platform really is a feature or red herring.

The main objective of an open platform is to allow adoption by those entities that can support and employ the platform. Adoption is enforced through common standards that a community agrees to use. They are numerous standards that are currently being reviewed by the Smart Grid Interoperability Panel. Many of these standards like IEC 61850, DNP3, C22.12 are specific to the electric utility. Other standard bodies like IETF, OASIS, and W3C support a vast array of technologies in the software community. *The principle of this platform is to leverage information technology standards across the software and industry domains.*

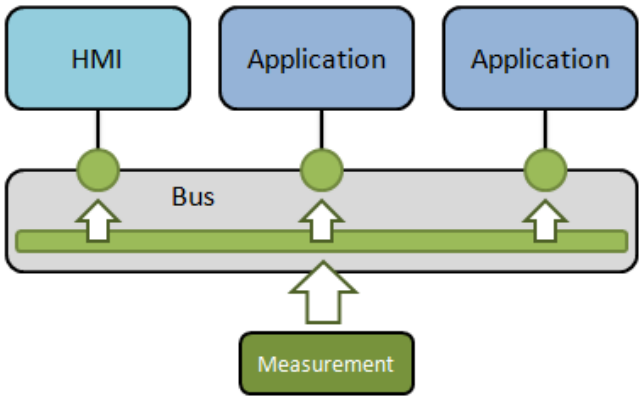


Figure 1: Subscribe Model

2.2. Open Source

The use the open source in commercial software is a growing. In many cases the end customer does not even realize that some of the code they are running is based on open source. Many companies leverage stable open source projects to support their product. Running stable projects [2] is complex set of community interactions and solid software engineering lifecycle principles. Many of these projects rival some of most complex and propriety code bases in the world due to the hundreds of developers and level of complexity. One key use of open source in this platform is to leverage hundreds of thousands of lines of code from stable community projects addressing common issues.

A complexity of open source projects is the licensing options. The open source initiative [3] maintains a list of approved licenses. Some licenses are incompatible with each other and require users to package their solution differently or abandon their use. *The principle use of open source with this platform is to allow community involvement across the core platform components and support liberal licensing policies for independent software vendors to use the platform.*

2.3. Cross Platform

In order to support a diverse set of applications, platforms and human resource skill sets the platform should leverage architectures and technologies that allow multiple languages and client architectures. *This principle much like the use of open source is to help drive innovation to increasing the tool sets and approaches of domain experts.*

2.4. Service Oriented Architecture

A service oriented architecture (SOA) provides for modular growth capabilities and is especially designed for continuous integration support [4]. Figure 1 shows the subscribe model supported by SOA. Adherence to this type of architecture forces service design to be generalized and reusable. Architecture brittleness is reduced and nicely supports a dynamic business and technical environment such as the evolution to a smarter grid. *This principle supports innovation, serviceability, continuous integration.*

3. DOMAIN TRIAD DESCRIPTION

In order to support continuous integration of business objectives related to managing the electric power system, providing value added solutions to the end customers, and managing the rate of change on business requirements we first have to identify the domain triad**.**

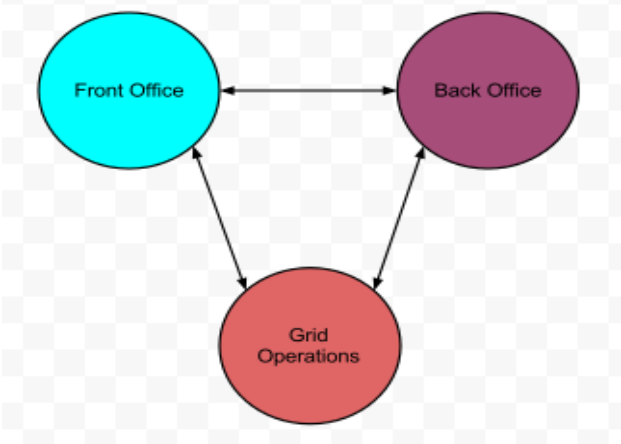


Figure 2: Domain Triad

For simplicity, we can identify three main domains of systems, business processes and technologies for a Utility. Back Office systems are dedicated to running the business of the utility such as accounting, billing, and account management. Front Office systems are customer facing

such as support and new services. Grid Operations refers to the management of the power system. The conventional domain model from SGIP contains seven domains, but for simplicity the three mentioned above are sufficient for discussions in this paper. Figure 2 shows a simple depiction of the domain triad.

To build out an platform architecture the is open, leverages best practices and solutions from other industries, and supports a robust ecosystem of solution providers requires basic architecture building blocks to interface all three domains. For this paper we will focus primarily on the building blocks for the Grid Operations Domain.

4. ARCHITECTURE BUILDING BLOCKS

In many cases the IT Enterprise solves integration needs for business systems using Enterprise Integration Patterns [4]. The common term for this type of integration architecture is called an Enterprise Service Bus (ESB) [5]. An ESB leverages service oriented architecture (SOA) [6] for implementing the interaction and communication of services and applications. The SOA design also supports distributed computing and promotes an asynchronous message oriented design.

The information technology fabric used to bridge the domain triad is called a message oriented middleware. The heart of the fabric is the Advanced Messaging Queuing Protocol (AMQP) which is an open standard application layer protocol.

4.1. AMQP

AMQP is the heart of the message-oriented middleware platform that bridges the domain triad. At minimum, AMQP provides queuing, routing, message orientation, reliability and security. The specification [7] calls out Ubiquity, Safety, Fidelity, Applicability, Interoperability, and Manageability as key requirement areas. The origination of this open standard is from the financial industry. Their objective was to address a critical business need of providing high speed reliable messaging with an open standard to avoid vendor-lock in. This work started in the mid 2000's and AMQP 1.0 is just now being ratified under OASIS.

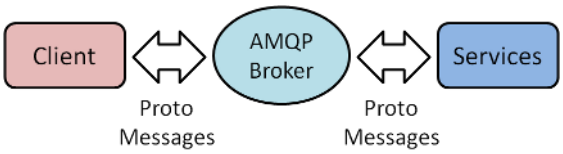
The requirements of AMQP from the financial industry are applicable to the utility industry as well. The two major requirement areas for supporting ubiquitous messaging within the Grid Operations domain are performance and security. AMQP addresses these requirements through the specification and real world implementations. In a cited performance test a commercial implementation of AMQP achieved over 6 million messages/sec on an 8-core box using Gigabit Ethernet [8]. There are a number of other industries adopting AMQP as the wire protocol for the same reasons required by the Utility industry.

Brokers and clients communicate with each other by the AMQP specification. Because the specification is open multiple vendor implementations can be deployed to work together which gives the end customer true interoperability and the ability to choose best-in-class client and broker solutions avoiding vendor lock-in.

The function of the broker is to receive messages from the clients and create routes, queues and generally manage all incoming/outgoing messages of the system. Clients are services and/or applications.

4.2. Configuration

Brokers can be configured in clusters for High Availability Messaging. Clients, also referred to consumers and producers, are really an extension of a simple client server model. When an client publishes messages to the broker, the broker, for instance, might configure an exchange with topics for subscription by other clients. For this platform a simple example is the collection of measures from an RTU in a substation where the client subscribes to all measurement in that particular substation.



The deployment models of the brokers and clients are driven by the requirements of the architecture. Clustering, federation, message persistence (durability), and other implementation can be tuned to support the deployment architecture.

The configuration supports how the messages move throughout the system, their availability, and reliability. The payload of the messaging system for this type of system requires a terse payload envelope to maintain high performance. The payload schema in this platform leverages protocol buffers developed by Google.

4.3. Protocol Buffers

Google protocol buffers are licensed under an Apache 2.0 license and have C++, Java and Python implementations among others. The protocol buffers use an interface description language within the serialization format to provide simple yet effective interface. The lightweight schema format that loosely resembles C structs:

The following example shows how measurement quality is encoded as a protocol buffer. In this case, the IEC 61850

definition is used. The reef project uses protocol buffers to define service resources and the REST framework.

```
// mirror the iec61850 quality (CIM uses these too)
message Quality {
   enum Validity {
      // No abnormal condition of the acquisition function or
      // the information source is detected
      GOOD = 0;

      // Abnormal condition ""
      INVALID = 1;

      // supervision function detects abnormal behavior,
      // however value could still be valid.
      // Up to client how to interpret.
      QUESTIONABLE = 2;
   }

   enum Source {
      // value is provided by an input function from
      // the process I/O or calculated by application
      PROCESS = 0;

      // value is provided by input on an operator
      // or by an automatic source
      SUBSTITUTED = 1;
   }

   optional Validity validity = 1 [default = GOOD];
   optional Source source = 2 [default = PROCESS];
   optional DetailQual detail_qual = 3;

   // classifies a value as a test value, not to be used for
operational purposes
   optional bool test = 4 [default = false];

   // further update of the value has been blocked by an
   // operator. if set, DetailQual::oldData should be set
   optional bool operator_blocked = 5 [default = false];
}
```

4.4. REST

Representational State Transfer (REST) is an architectural style for distributed systems. REST is commonly thought of as a "nouns and verbs" alternative to Remote Procedure Call (RPC), but it is actually a proper subset of RPC that has some interesting properties. Factoring a distributed system in a RESTful style has the following advantages (which can also be thought of as constraints) as applicable to the discussed platform:

- *Uniform interface* - Service consumers/providers communicate through a uniform interface (i.e. GET, PUT, POST, DELETE) which decouples client server development.
- *Opaque transport* - Clients are unaware of whether they are directly connected to the end service or a proxy. Proxies can be used for things like load balancing or to enforce security.
- *Stateless* - Every request contains all of the information necessary to satisfy the request. Any server-side state must be addressable in some way.

5. CORE SERVICES

Within this platform there are a number of services. We will focus a set of core services that support the automation functions of the platform. Figure 3 shows the basic services of Reef.

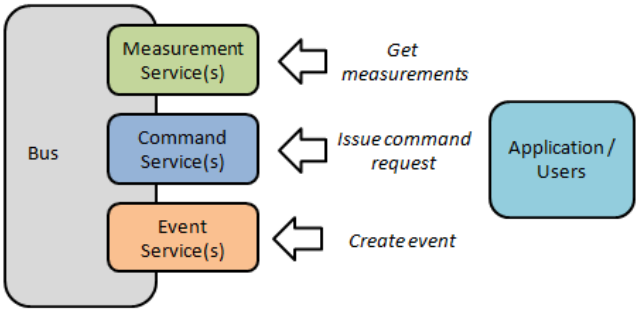


Figure 3: Basic Services of Reef

5.1. Measurements

Measurements are the basic units of data published by *data sources* which are processed and monitored by the system. Often, measurements are acquired using communications protocols, and are used to represent the state of remote *field devices*. Measurements may also be generated or manually entered by agents. The basic fields provided by the measurement service are Value, Quality, Time, and Unit.

Measurements enter the system either from communication *front-ends* or directly using the Reef Project APIs. The measurement *stream* is then processed -- configured transformations are applied (scaling, value mapping) and side-effects are triggered (event/alarm generation) . Finally, measurements are stored in the database and published to the bus. The system also maintains *measurement history*, the chronological stream of previous measurement values.

5.2. Commands

Commands are the indications agents use to interact with and modify the state of the system. Commonly, commands are tunneled by the communications processors to remote field devices in order to exercise control over their functions. Whereas measurements constitute the flow of data from the field, commands form the information moving outwards.

Commands may or may not contain an associated value. In the field communications world, commands with values are usually referred to as *setpoints*, and may refer to a *target value* the system or end device is intended to reach. Values are therefore used when a simple imperative cannot convey the proper message, such as "set temperature to 65 degrees celsius".

When multiple agents/clients will be using the system concurrently, simultaneously making modifications to the same subsystems can lead to undesired results and indeterminate behavior. Furthermore, it is frequently the case that some field devices need to be declared "off limits" for safety or maintenance reasons. The following objects are used to regulate access to commands:

- *Selects* are acquired by clients to grant exclusive access to a command or set of commands.
- *Blocks* are used to prevent *any* client from accessing a command or set of commands.

The architecture allows for a unique opportunity to support multiple clients in the grid operations domain. A real world example would be clients like load shedding, remedial action schemes, and switch order management applications. In this case the architecture allows for each client to be from different vendors since they are operating on the same equipment and communication models of the platform. They can subscribe to the measurement stream needed and perform automatic /manual operations without concern of indeterminate behaviors.

Another capability of this architecture is a term called scatter-gather. This is an Enterprise Integration Pattern that allows a client to broadcast many requests and receive an aggregate message when all the recipients have responded. A practical application is broadcasting a select command to multiple switching devices and then waiting for the response from all devices. The client does not need to manage this but instead allows the platform to perform the work.

5.3. Events

Events are objects that record a meaningful occurrence or change of state in the system. They are used both to monitor the system in real-time and to provide an audit log of system history.

Events are configured with the parameters *type* and *severity*, and contain context information such as the originating subsystem and associated agent. The "message" of the event may contain a further description, and contain any relevant data attributes.

Ultimately, the definition of events, as well as the conditions under which they are triggered is highly configurable. Events, as a whole, are a tool system designers use to characterize system behavior and to provide clients and administrators necessary information.

Many or most commands issued by clients will qualify as events. Other events may not be tied to operational data, but instead will record system activity such as application errors and agent authentication actions.

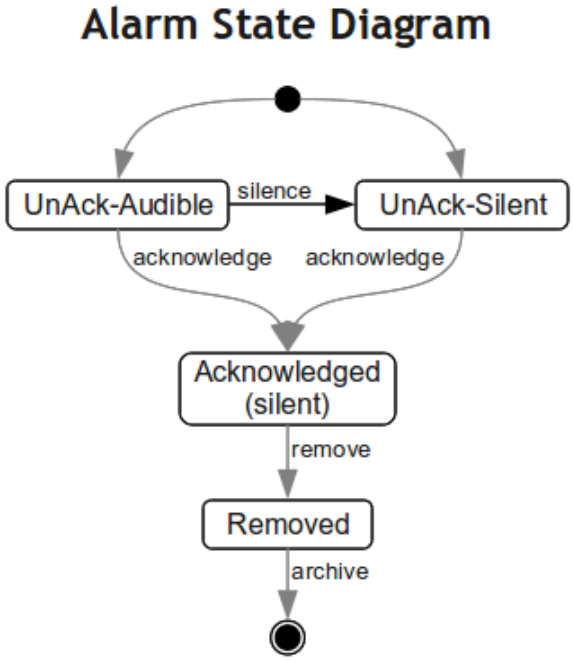


Figure 4: Alarm State Diagram

5.4. Alarms

Alarms are a refinement of events which identify system occurrences that require operator intervention. Alarm objects are tied closely to event objects. All alarms are associated with events, but not all events cause alarms. In contrast to events, alarms have persistent state. The three principal alarm states are *unacknowledged*, *acknowledged*, and *removed*. The transitions between these states constitute the *alarm lifecycle*, and manipulation of the states involves *workflow*. Transitions in alarm state may themselves be events, as they are part of the record of client operations.

6. PROJECT SERVICE LIST

Table I: Reef Project Service List

Name
AgentService.java
AlarmService.java
AllScadaService.java
ApplicationService.java
AuthTokenService.java
CommandService.java
CommunicationChannelService.java
ConfigFileService.java
EndpointManagementService.java
EntityService.java
EventConfigService.java
EventCreationService.java
EventService.java
MeasurementOverrideService.java
MeasurementService.java
PointService.java

Table I list the services in the Reef Project at http://totalgrid.org. These services provide the automation plumbing and core services.

7. SERVICE EVENTS & SUBSCRIPTIONS

In order to create a highly scalable system, the platform has to be event driven. Clients can subscribe to various services based on what information they require. Subscribing reduces the wasted conversation time of a reply/request method. The method used is typically referred to a pub/sub.

Another aspect of this approach is that multiple clients can subscribe for the same service topics (routing keys). For example a client could be any number of thick or thin applications all subscribing the same information.

This is an extremely powerful tool in the grid operation domain to share automation data between many clients within and across domains. With appropriate authentication and authorization access to the automation data-non operations personal can access real time data for analysis and planning.

8. MODELING

From an automation context modeling refers to the act of representing the power system data acquisition model. Modeling of grid operations systems has always been a complex design issue due to user experience (UX), data federation, and integration of other sources of record systems such as a Geographical Information System / Asset Management System. The modeling system uses a color directed graph to support relationships like “own”, “uses”, “source” and many others. This approach is not specific to the electric utility. With this approach the platform can support modeling for water, gas, building, manufacturing, electric automation system individually or combined. For instance, a Municipal Utility may have operational support for both the electric and water systems for their customer base. Modeling for both systems can reside in the same system while still supporting different operational business units and processes. Another more interesting case is the integrated utility services of a microgrid. The platform allows simpler integration of applications that manage multiple and interdependent services.

9. DEPLOYMENT CONTAINER

Another feature of the platform is the deployment container based on a specification of the Open Services Gateway Initiative (OSGi). The components of the Reef project are deployed as bundles in within this container. Figure 5 shows the Reef Karaf shell.

Figure 5: Reef Karaf shell

The Karaf provides easy interface to load services, access resources and overall management of the platform. This technology supports the following key features:

- Cross platform support (Windows and Linux)
- Hot pluggable support
- Command Line Interface (CLI) to service interfaces

The ability to load and unload bundles while the system is running is a very powerful feature. A practical application

of this feature is the ability to patch a protocol bundle like DNP3 without shutting down the system. Since the system can be distributed to multiple nodes the operational impact of such an act can be limited.

Another feature that supports developers is the ability to expose service interface through the karaf shell. Client application can leverage the same interfaces therefore the developer can explore the interfaces through the CLI.

10. PERFORMANCE

Performance of the platform can be explained in terms IO, CPU, and network bound. Persistence of time series data is the typical performance bottleneck. This type of IO constraint has been addressed by leveraging new technologies like Cassandra, which is a highly scalableable NOSQL solution.

CPU and Network performance bottlenecks are addressed through horizontal scaling of commodity hardware. The service oriented architecture allows the design of this type linear expansion.

11. SECURITY

For message oriented middleware a fundamental security requirement is to ensure safe messaging. The AMQP specification calls out Simple Authentication and Security Layer (SASL).

This method provides authentication support for the clients. There are optional authentication methods and capabilities to negotiate protection on other protocol interactions.

The underpinnings Role Based access is support for multiple users with course permission sets to service resources and/or the entity model. Granular permission will be supported in future releases.

12. OPEN SOURCE MATRIX

All of the software components discussed in this paper are licensed under various open source licenses. The following table is a partial list of the main components.

Table II: Platform Open License List

Component	License	Description
Reef Core	AGPL	http://reef.totalgrid.org
Reef API	Apache 2.0	http://reef.totalgrid.org
DNP3	Apache 2.0	http://dnp3.totalgrid.org
Cassandra	Apache 2.0	http://cassandra.apache.org
Postgreql	GPL	www.postgresql.org
qpid	Apache 2.0	qpid.apache.org
Spring		www.springsource.org
Protocol Buffers	Apache 2.0	Google Protocol Buffers

13. IMPLEMENTATION

The following are various implementation of the reef project.

13.1. FREEDM Lab - MicroGrid Implementation

The NSCU FREEDM Lab (http://www.freedm.ncsu.edu/) uses the Reef platform to preform data acquisition to various components in the microgrid. The figure below represents a conceptual model of the microgrid. DNP3 is used to communicate to the field devices. This is an NSF funded site that supports research in solid state devices to control the flow of power. The platform allows new applications development and leverages a real time HMI to manage the network. The configuration of the platform is a single node running on Ubuntu 10.4 LTS. The model is less than 100 measurements but support a number of control options and can be configured to support the new solid state devices.

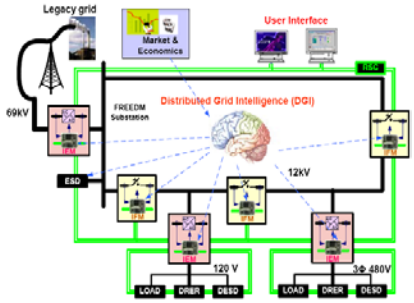


Figure 6: FREEDM Microgrid

13.2. EV Charging Station

The figure below shows the automation platform reef with a special type of protocol connected to the frontend processor (FEP) of reef. The protocol bundle provides the mapping and the specific charging station protocol. A state machine is used as a poller. The Spring Web Service provides the

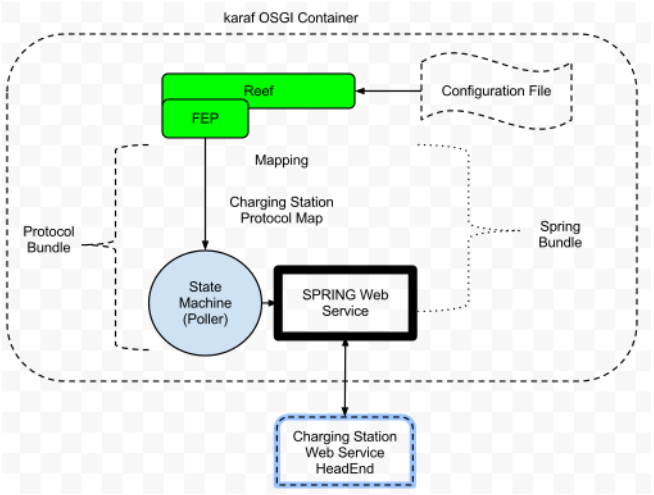


Figure 7: Charging Station FEP

interface to the charging station web service headend. Within the karaf shell the protocol and spring are bundled to provide a deployable package. The Configuration file is a general XML modeling file for the system. It provides the modeling parameters for the FEP and protocol bundle. This implementation provides demand information and is processed by the platform so applications on the platform can monitor and control the charging station. Technologies like SOAP, WS-Security and HTTPS are used to leverage a standard reusable approach and a authenticated connection.

13.3. Grid Operation / Back Office Integration

Another integration problem is providing back office integration with an outage management system (OMS) and the Reef platform. In this configuration the web service stack can use approaches like a JMS connector to communicate with the platform and subscribe to breaker status changes to support the OMS fault location algorithms. The data model used is the point-to-point standard MultiSpeak®. The platform provides a number of advantages over a straight application to application connection.

First, the integration of the MultiSpeak® client does not impact the operation of the reef platform. Therefore, the client can be modified and / or patched without any direct impact on the platform's ability to communicate with field devices. Second, the integration cost of standing up another version of the same interface is minimized to just days of integration and testing. Finally, configuration issues are focused on the client and not the platform.

14. CONCLUSION

The discussed message oriented middleware automation platform leverages a next generation messaging protocol and diverse IT technologies to support ubiquitous messaging across multiple domains. The platform is based on open source components to drive innovation and provide a path for industry adoption. The purpose built services and messaging infrastructure also provide a reliable and preformat platform to enable utilities to build out a non-brittle IT infrastructure for the realization of a smarter grid.

15. REFERENCES

[1] NISTIR-7628 Vol 1. - http://csrc.nist.gov/publications/nistir/ir7628/nistir-7628_vol1.pdf

[2] Chris DiBona, Danese Cooper, Mark Stone, *Open Sources 2.0: The Continuing Evolution*, O'Reilly, ISBN: 0-596-00802-3, 2006.

[3] OSI - http://www.opensource.org/licenses

[4] Gregor Hohpe, Bobby Woolf, *Enterprise Integration Patterns, Designing, Building and Deploying Messaging Solutions*, Addison-Wesley, 2004.

[5] David A. Chappell, *Theory in Practice Enterprise Service Bus*, O'Reilly Media, 2004.

[6] Thomas Erl, *Service-Oriented Architecture Concepts, Technology, and Design*, Printice Hall, ISBN-10 0-13-185858-0, 2005.

[7] AMQP Specification V1.0, Revision:1350, 07 Oct 2011. http://svn.amqp.org/svn/amqp/trunk

[8] http://www.redhat.com/mrg/messaging/features/#aio

16. BIOGRAPHY

Mr. Evans is a Software Engineer working on Green Energy Corp's Green Bus Platform and open source initiatives. Mr. Evans holds a Bachelor of Science in Computer Science from the University of North Carolina-Chapel Hill. Since 2006, Mr. Evans has worked on a variety of efforts as part of Green Energy's work in power systems automation. His domain experience includes software support for substation integration, the development of "smart grid" load reduction automation for a major utility, and the development and open-source release of the industry-standard DNP3 protocol. He has been responsible for quickly prototyping proof of concepts for thin-client SCADA interfaces, rich internet application-based operator HMIs, and distributed field device communications using low-power digital radios. Mr. Evans is currently developing core applications and services as part of the Green Bus Platform team.

Mr. Hendley is a development lead within the Production Development group of Green Energy Corp. He holds a Bachelor of Science in Physics from the University of North Carolina-Chapel Hill. Mr. Hendley worked on a number of games and products and built an extensive framework for porting games to a range of phone hardware increasing the team productivity. Mr. Hendley also made contributions to a number of open source projects necessary to use testing and coverage tools on the RIM Blackberry device. Since Mid 2006 Mr. Hendley has been a software developer at GEC. Mr.Hendley has worked on many projects including "Advanced Applications" built on top of a large utility EMS, a custom control system for a sun tracking solar collector, a web-based HMI system for a renewable energy plant and a number of communication protocol stacks.

Mr. Crain leads the GreenBus platform development and open source efforts. His experience in the power industry includes SCADA systems, substation automation, control centers, field devices, protocol stacks, and concentrating photovoltaic systems. He is the principal author of Green Energy Corp's first open source release, DNP3, a high performance implementation of Distributed Network Protocol. Adam was the principal software engineer behind Skynet, a control system for arrays of robotic telescopes. Skynet (http://skynet.unc.edu) is now an integrated part of the high school and college curriculum in North Carolina. Adam holds a BS in Physics and Computer Science from the University of North Carolina at Chapel Hill.

Mr. Camilleri is the EVP of Product Development and Chief Product Owner of GreenBus at Green Energy Corp. He has worked as a distribution and transmission engineer for PECO Energy. During his time with ALSTOM-GRID John integrated SCADA/EMS/DMS systems to numerous utilities. Later he served in the Operations and R&D divisions providing leadership to the engineering and product development organizations as well as working closely with utility customers. For Microsoft, John managed developments of security features and operating system security architecture for several windows mobile product lines. John holds both a BS and MS in Electrical Engineering from Tennessee Technological University, is a Senior Member of IEEE.