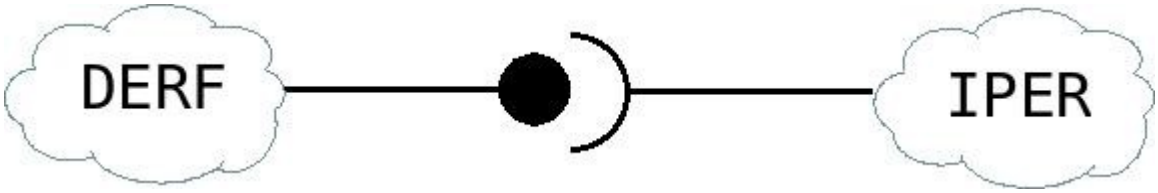
Title: **ESI Server – The Platform Solution**

Introduction

At a very high level, the ESI can be depicted as a simple interface between two parties, the DER Facility (DERF) and the Interfacing Party with External Responsibility (IPER).



To understand the ideas presented in this paper, its important to understand that everything we are presenting here is enabled via two APIs, these APIs are independent of any platform that might implement them and they are independent of the Open Source reference implementation which we discuss in this paper also. The first API is a substrate (or supporting) API that enables an ecosystem that the second API lives within. These are the two APIs respectively:

1. https://github.com/SolarNetwork/solarnetwork/wiki/SolarQuery-API (substrate API)
2. https://pastebin.com/qpHxmLRN (this API is what we call the ESI)

The remainder of this introduction presents a very high level scenario showing the plug and play nature of this approach for a new DER Facility (DERF). This scenario is made possible by the two APIs presented above. This scenario is also useful in providing context to the reader when understanding the details which are discussed throughout the rest of this paper .

High level Scenario:

This high level scenario presented below is played out with interactions between a DERF and an IPER as they establish a new relationship between each other in preparation for the exchange of commercial Grid services:

<<< START OF SCENARIO >>>

DERF: Login or create an account on a server that supports the ESI API for service providers in the local area e.g. on a server hosted at https://data.esiserver1.net/esiuser/login.do

DERF: Setup their “ResourceCharacteristics” on this server (this is done either directly via the API or possibly via a web user interface). Resource Characteristics provide a high level overview to the IPER of the services that this DERF has to offer.

DERF: Setup their “PriceMap” on this server (this is done either directly via the API or possibly via a web user interface). The Price Map provides a very precise (electrical engineering) definition of the services and the pricing that the DERF has to offer to the IPER.

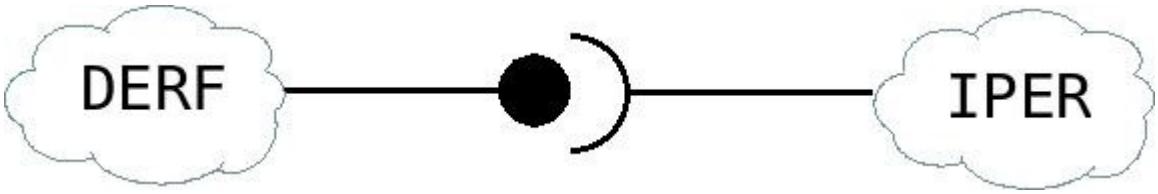
DERF: Grant permissions to their IPER(s) of choice (this is done either directly via the API or via a web user interface)

IPER: The IPER receives notification of the new services they have been granted permission to use (initial notification is via email, but this information is also available via the ESI). Using the ESI, the IPER enquires with the new DERF about PriceMap details, then provides offers to the DERF to engage with their services (done directly via the ESI API).

<<< END OF SCENARIO >>>

Description of ESI Functionality

As described in the introduction above, the ESI can be depicted as a simple interface between two parties, the DER Facility (DERF) and the Interfacing Party with External Responsibility (IPER).

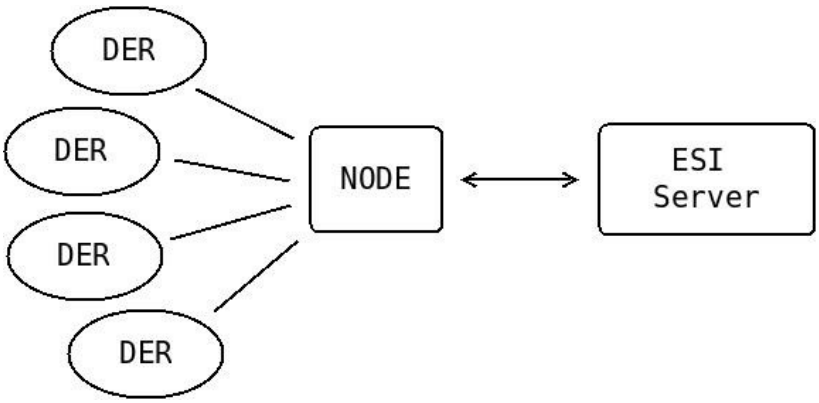


For a number of reasons, it is worth expanding the definition of the ESI to go beyond a simple interface as depicted in the diagram above. To achieve the stated goals of the DER Challenge, the ESI needs to be expanded into a platform that logically still provides an ESI, but physically looks vastly different.

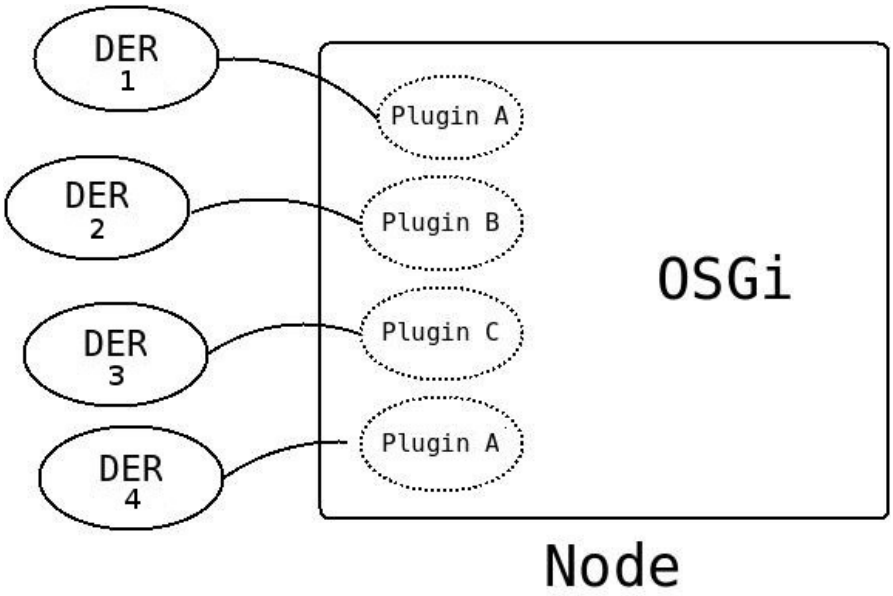
To satisfy all the requirements of the DER Challenge, we propose a cloud based platform to be known as the “ESI Server”. This ESI Server may potentially have many instances (possibly running in an AWS hosting centre or similar). The ESI Server is a conduit between the DERF and the IPER, communicating over secure SSL channels using a well defined API (using JSON for administrative messages and MQTT or gRPC for communication with more real-time demands).

Communicating with DER

Individual DER are communicated with via software plugins that run on industrial mini computers called Nodes (in some cases, they will be embedded within the DER). These mini computers or Nodes will run a cut down version of Linux (such as a minimal Debian image) that is easy to replace and upgrade.



The Node hosts DER-specific plugins within an OSGi framework. The software plugins for various pieces of DER hardware are matched to specific models of a product or more often to a manufacturer's product range. Mostly these plugins are provided as Open Source software modules (but occasionally they may be implemented as a proprietary plugin where the hardware provider requires this). These plugins are deployed on Nodes; small, possibly embedded computing platforms that are installed onsite within buildings or sometimes directly within a vendor’s DER equipment. OSGi is used to allow plugins to be easily deployed, upgraded or created if new protocols or DER hardware needs to be catered for.
In the image below DER devices 1 and 4 are from the same product line, hence they both use plugin A. DER devices 2 and 3 are from different product lines, hence they require different plugins B and C.

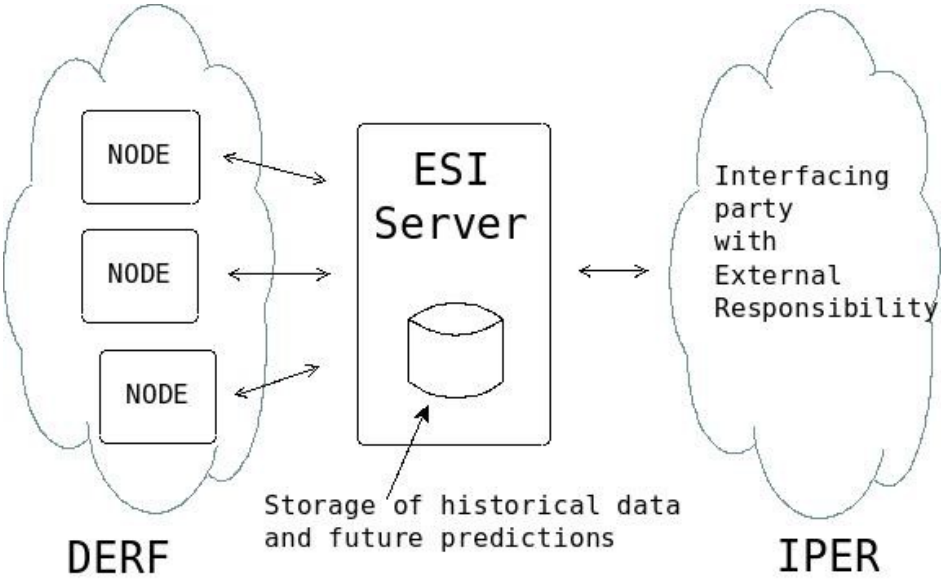


Nodes also have precise geographic locations and a set of source ids which can supply data and/or receive commands.

Example visualisation of Node geographic information

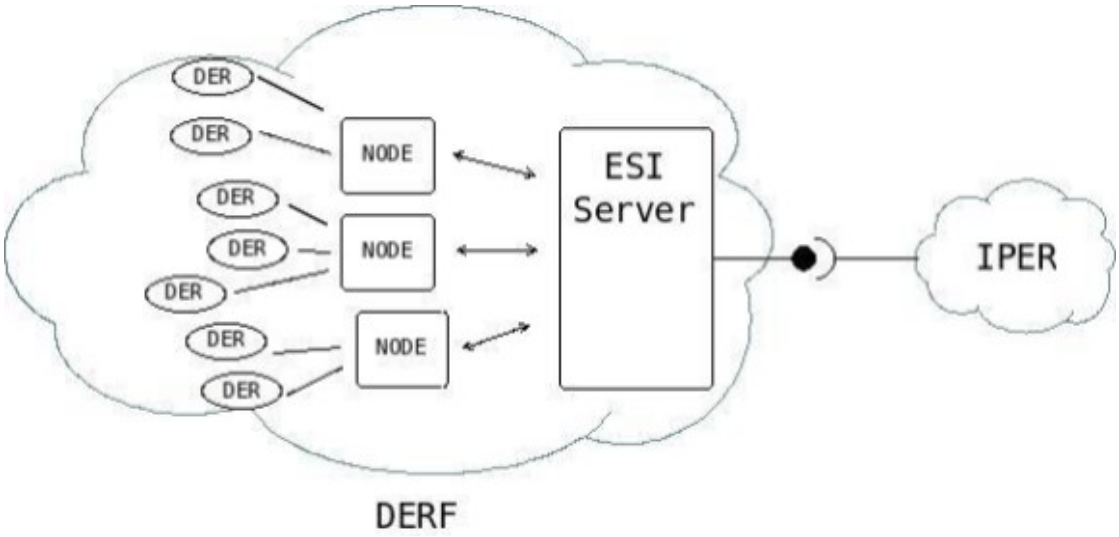
Looking at the actual functionality of the "ESI Server", and viewing it as a *platform* rather than a simple interface, we can explore the multiple services that it provides to both the Grid and DER providers. Some of the services that it provides are listed below:

1. Data Historian (records historical data from the nodes energy resources)
2. Data Forward Projections (stores future predictions for the nodes energy resources)
3. Control Gateway (i.e. conduit for the ESI messages)



Services of the ESI Server platform

Additionally to performing these services, the ESI Server can also itself host or be a DERF by aggregating many DERF within it and providing these as a single aggregated DERF to a higher level IPER.



A DERF can also be a virtual aggregator via the services of an ESI Server

ESI protocol

The actual ESI protocol implemented and supported within the ESI Server will likely use gRPC with Protocol Buffers. This provides a high performance protocol which easily and securely traverses standard HTTPS enabled networks. Combined with that it provides good features for backwards compatibility and iterative changes over time.

The logical structure of our ESI design (i.e. the logical interface as defined by the 'Interface Definition Language' or IDL) is intended to address the needs of the grid from a first principles perspective. This means focusing on services with parameters of time, location, energy and money.

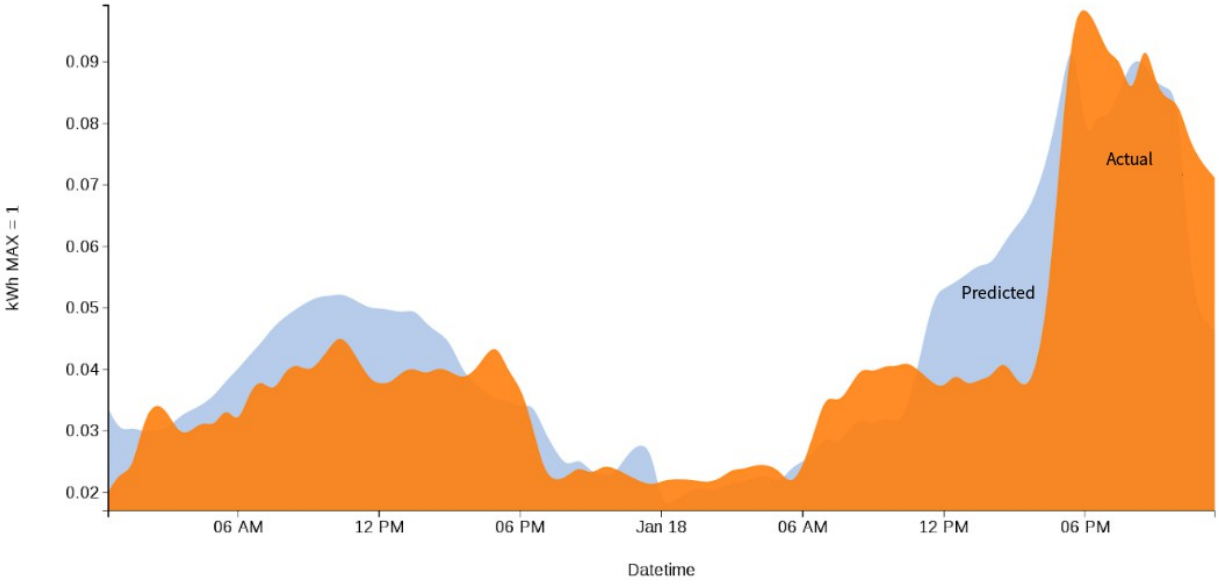
This approach results in two distinctly different types of service being supported within the ESI:

1. Real time Interactive Requests from the IPER to the DERF.
2. Dynamic Responses from DERF based on previously configured parameters.

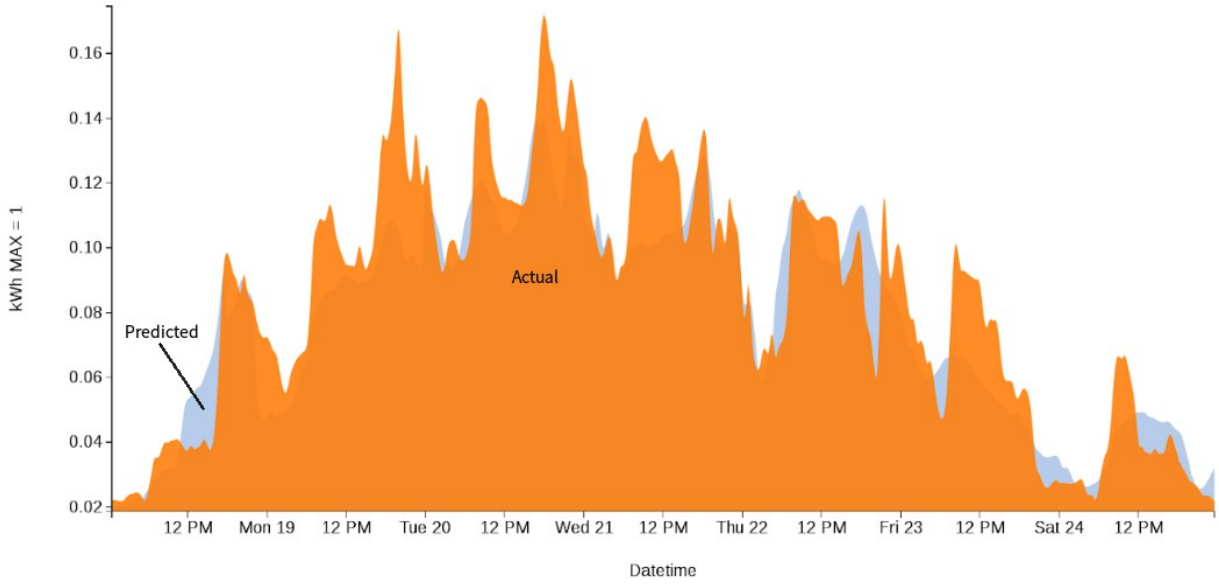
This approach leads to a very simple ESI which can address all the needs of the Grid. If engineering requirements, commercial entities or local regulations demand more rigid or predefined types of service structures, these can easily be built on top of this generic low level ESI.

Assumptions about Grid/DER Facility Relationship

Each DERF may include machine learning which can be utilised for the purposes of predicting upcoming behavior. This forward projection is driven by a NN (artificial Neural Network) which is continuously trained via historical data. It does its forward projections using time and weather inputs which allow the NN to provide accurate estimates of expected consumption and generation behaviour many days into the future. Plugins pulling data from “The Weather Company” or similar weather services provider can be used to provide accurate weather forecasts that in turn make for accurate *energy* predictions.



Machine learning predicts future energy needs (both consumption and generation)

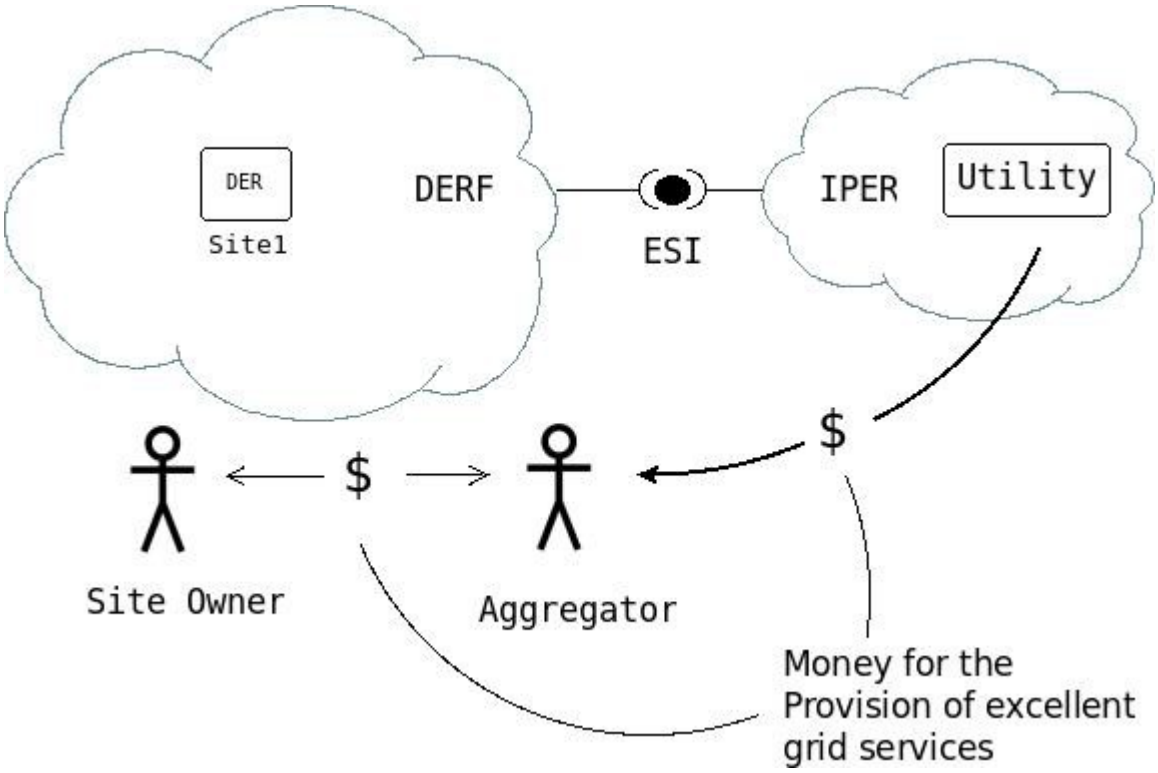


Predictions can span multiple days into the future

These energy predictions are stored within the ESI Server and can be used by an IPER with authority to access this data to make intelligent decisions about the need for demand response

services and the associated commercial value of various DERF service provider's offerings. When accurate forward projections are stored within the ESI Server, they can also be used to verify delivery of various demand response services (see the method DERF::getOneWeekHourlyEnergyProfile() on the ESI.idl for details on how this data is exposed to the IPER).

The high level architecture of the commercial relationships involved here looks like this:



Commercial Relationships (Site Owner may also interact directly with the IPER)

Needed Hardware/Software for Demo

Many of the components that we would like to demo already exist, aside from the configuration and physical preparation required to do a live demo, the main work to do is directly around the development and testing of the ESI features documented and described in ESI.idl shown in APPENDIX A of this document.

Here are the items we would like to demo:

- ESI Server (already exists as an instance of SolarNetwork in the cloud, the syntactic and semantic aspects of the underlying SolarNetwork protocol are documented here: https://github.com/SolarNetwork/solarnetwork/wiki/API-Developer-Guide)
- Node (already exist, we will bring a small DIN rail mini computer)
- OCPP enabled EV Charger Plugin (already exists)
- Building management via Luxone device Plugin (already exists), would show control of some simple building loads.
- Software simulation Plugin for use with a rampable Solar Inverter such as a Schneider Conext XW+ (already exists, likely needs adapting for this demo)
- Software simulation Plugin for Battery storage (already exists), could possibly demo a real physical device using a Schneider Conext XW+ or similar (Plugin development required).
- ESI realtime enhancements to existing SolarNetwork Open Source protocols (TODO - use ProtocolBuffers or MQTT)
- Interfacing Party UI (TODO – needed to allow the demo to be visualised from an IPERs perspective).

Integration Narrative

Narrative 1: **New DER deployed (within an existing DERF)**

In most Integration scenarios, the ESI Server will already be in existence and the DERF will already be familiar with these relationships, hence infrastructure and configuration related to the ESI Server will already be done (i.e. it will already be deployed, pre-configured and functional). Hence the common narrative will be focused on deploying a new Node within an existing DERF.

Deploying new DER within an existing DERF:

1. User installs product (electrician likely does this for them)
2. User provisions Internet connectivity.
3. User activates the product either directly on the products HMI or via a local LAN connection to the Node within the product. End result is that the Node says “hello” to the ESI Server and confirms connectivity (using the Internet).
4. User configures Smart Grid features based on their preferences e.g. prioritising cost savings over convenience (often the available options will be defined and/or constrained by the DERF management provider to suit their business model and to streamline the end user experience).
5. Product then behaves as per customer's expectations and the business model of the DERF.
6. At some time later, the User modifies their preferences.
7. Product behaves as per the customer’s new expectations.

Narrative 2: **DERF to the IPER Integration**

A DERF of any substance will likely be an aggregation of many nodes. This aggregation will be done via the services of the ESI Server. The ESI Server offers services for appropriately authorized external entities (such as an IPER) to list and aggregate nodes, or sets of nodes (i.e. DERFs) by a range of parameters including location.

The new DERF on the block:

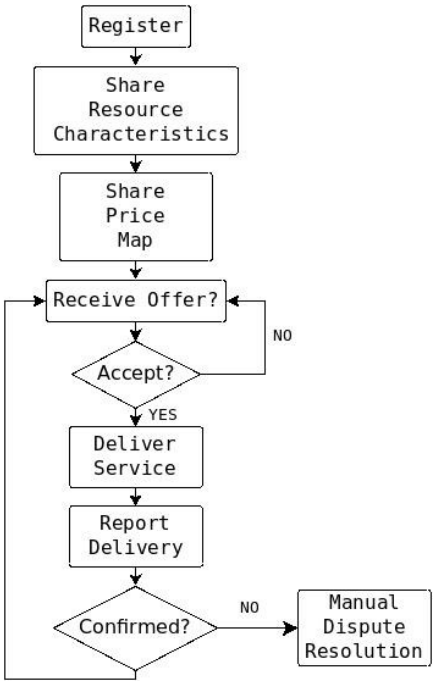
A new DERF called “DerfsRus” is coming online, this DERF has brought together a large group of industrial businesses who have recently had Solar PV and storage installed upon their buildings. The DERF is looking to offer the resulting services that these buildings can provide into the grid by leveraging an ESI. DerfsRus knows of one entity that will likely be willing to pay for grid services, that is the local Utility "ACME Power Inc" who has widely publicised that they are trying to defer spending on line and substations upgrades by engaging DR services during peak times and also during unexpected outages. The Utility has published their ESI Server handle which is simply their

email address within the local ESI Server which is “info@acmepowerinc.com”, they also publish the URL of the local ESI Server that they leverage which is "https://esiserver1.net". Now that DerfsRus have finished testing and configuring their facilities, they are ready to start selling their services. They look at other service providers on the market and read about average pricing of DR services over the previous 12 months. They know what the cost of offering their services are and realize that they have a very profitable service to offer. Using an Internet connected web browser, DerfsRus create their own account on esiserver1.net, they configure the ResourceCharacteristics and the PriceMap for all of their services and grant access/permission to “info@acmepowerinc.com” so that ACME Power Inc can view the DerfsRus nodes and engage with the associated ESI services.

Over the next few hours ACME Power Inc's system automatically enquires about the DerfsRus PriceMap (using the ESI) and discovers a number of price competitive services that will help to address a growing grid constraint problem in the north of the city. ACME Power Inc stores this information in its local database ready to request service during the next peak which is expected to occur at 1pm tomorrow. At 12:35pm the next day the grid constraint begins slightly earlier than expected ACME Power Inc issues an offer (using the ESI’s giveOffer method) to DerfsRus as well as a number of other DERFs in the area. Within 30 seconds ACME Power Inc sees a noticeable improvement on the grid and within 1 minute the grid constraint has been largely resolved.

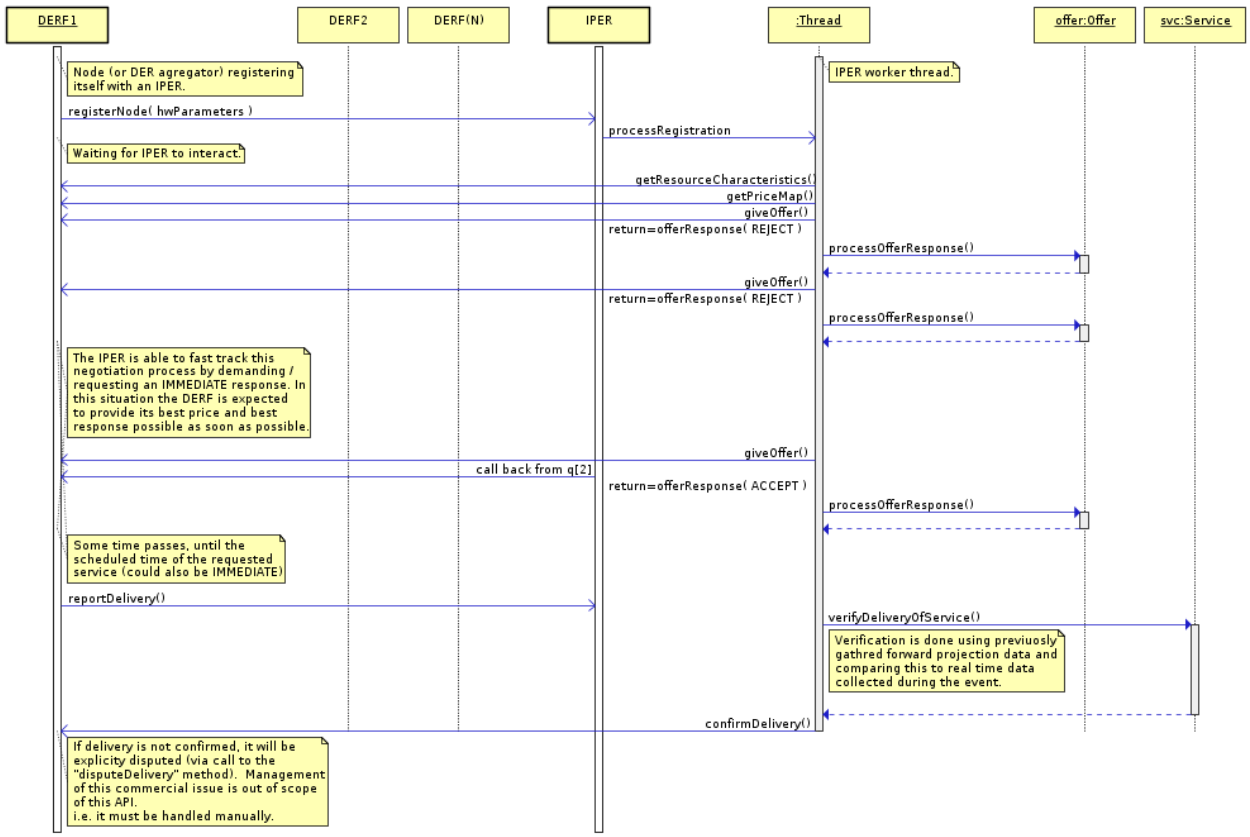
Note: *For additional information and ideas on this topic, please also see the “Business Models” section later in this document.*

Below is a State Machine within a DERF when it registers with the IPER and then subsequently negotiates pricing before delivering and confirming service:



Note if the IPER wants to avoid a potentially slow negotiation process due to immediate needs of the grid, it can simply supply an “Offer” to the DERF that matches their published PriceMap. Under this situation, the IPER would be expected to be able to deliver this service immediately if that is what the IPER requests (or if not immediately, at least within the “ResponseTime” figure published in the DERF’s PriceMap).

Below is a sequence diagram of the same. It is showing the API level interactions that happen during the provisioning of a new Node within a DERF and the associated commercial negotiation that can happen in real time with the IPER (likely to be a Utility) and the subsequent supply and verification of the requested commercial services.



ESI Sequence diagram - Interactive Requests

Please note that the “registerNode” method shown in the sequence diagram above is not part of the ESI.idl (see APPENDIX A). This node registration functionality is a core part of the platform that the ESI functionality sits upon (there are many other such supporting features within this platform that are not specifically mentioned here, but the registration process is documented in the section directly below for completeness of this example).

The Semantics of Node Registration

During the setup process, a node must first establish its unique identity and register itself with the ESI Server. In this sense the ESI Server is the “Registry” as described by GWAC (see details on page 17 here). Below are the semantics of this registration process (note: these steps and the APIs listed are already fully implemented by the Open Source SolarNetwork project).

During the setup process, a node must first establish its unique identify, it does this by requesting an invitation from the ESI Server (i.e. SolarNetwork):

https://github.com/SolarNetwork/solarnetwork/wiki/SolarIn-API#identity

The next step in the process is node association:

https://github.com/SolarNetwork/solarnetwork/wiki/SolarUser-API#associate-node

And finally, the node will eventually, at some point in the future, need to renew its certificate (it does this automatically as required):

https://github.com/SolarNetwork/solarnetwork/wiki/SolarIn-API#renew-node-certificate

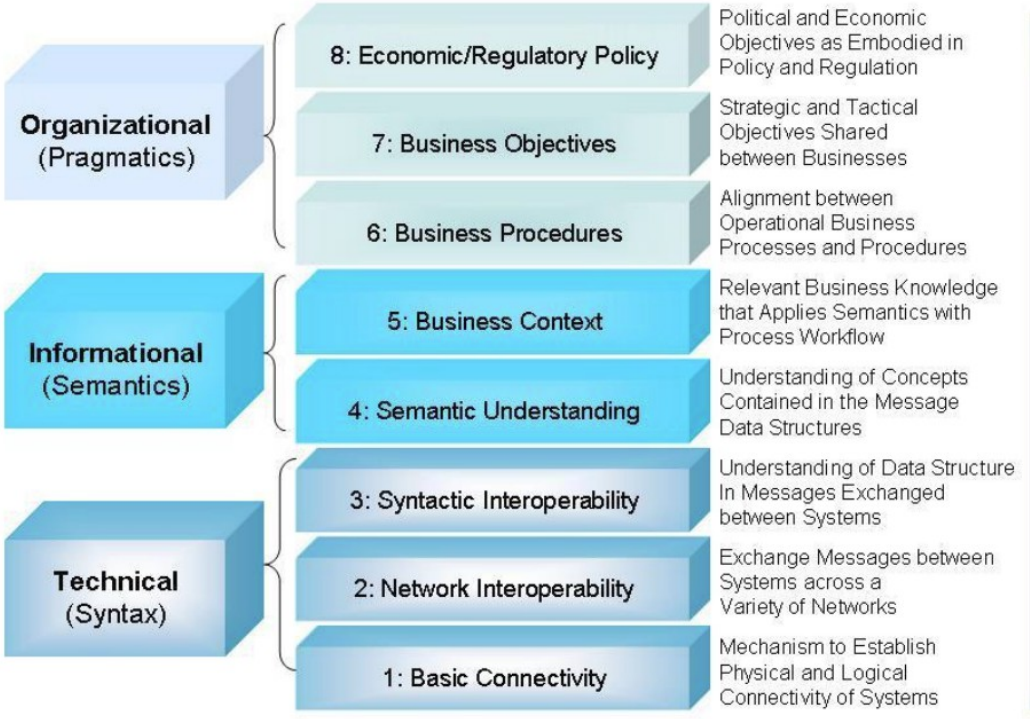
ESI Specification

The ESI has been specified using pseudo IDL (Interface Definition Language).

Here is a link to the IDL in pastebin online:
https://pastebin.com/qpHxmLRN

The full ESI.idl file has also been included in APPENDIX A of this document.

The remainder of this section shows how the ESI Server and its ESI interface maps onto the the information model of the GWAC stack shown below.



The GridWise Architecture Council (GWAC) Stack

The ESI has the following GWAC Stack mapping for communication between the DERF and the IPER (each number below is a reference to the equivalent section in the stack diagram above):

1) Any physical medium that supports TCP/IP. Likely to be Ethernet, Wifi or Cellular etc. Public networks can be easily leveraged and if additional security and/or management of bandwidth is required VPNs can and should be used.

2) HTTPS over TCP/IP

3) A mix of XML, JSON, MQTT and Protocol Buffers

4) For a Semantic understanding of the ESI Server proposal, please see ESI.idl, which is pseudo IDL (Interface Definition Language). It is documented in Appendix A of this document and it is also available online in pastebin here: https://pastebin.com/qpHxmLRN

The underlying XML and JSON messaging infrastructure that supports the base functionality within the ESI Server (i.e. the semantics of node registration, data history and forward projections) is documented here: https://github.com/SolarNetwork/solarnetwork/wiki/API-Developer-Guide

Currently this ESI proposal does not leverage any of the existing grid specific standards, this is because we have taken a very simple first principles approach to avoid the complexity of existing standards. It would be possible to adjust this first principles approach to leverage some aspects of CIM 61970 (or similar) if that was required or deemed to be desirable (e.g. adopting an existing standard's data types for Active and Reactive power etc).

5) The ESI Server and the associated ESI.idl is designed to generically support any Business Context. Hence from this perspective, the only purpose of a Business Context or a business context example is to exercise and confirm (or otherwise) the true generic nature of the ESI and the ESI Server and its ability to satisfy all the required scenarios.

6) The ESI Server and the associated ESI.idl is designed to be generic and hence support any Business Procedure that the Utility and/or the DER provider wish to implement.

7) Again, the ESI Server and the associated ESI.idl is designed to be generic and hence support any Business Objectives of the Utility.

8) And once again, the ESI Server and the associated ESI.idl is designed to be generic and hence support any Economic/Regulatory policy that may need to be implemented.

Business Models

Given that we have just stated that the ESI.idl is generic and does not specify or enforce any Business Model and associated Business Objectives, this section presents a couple of examples to allow us to explore how this approach might work in the real world.

Example 1: Real Time Pricing

Firstly, lets assume we are dealing with a Utility that wants to provide Real Time Pricing (RTP) with the business objective of delaying investment in new lines infrastructure. In this example, the Utility simply publishes the current price to DERFs via the IPER:getCurrentGridPrice method (and via a publish and subscribe equivalent which is not explicitly documented yet). The DERF will have implemented controls and a UI appropriate for the equipment onsite and the target user audience to allow them to manage price fluctuations appropriately. For example the DERF may be managing a HVAC system for a large commercial building. The owner of the building may have configured their system via the DERF HMI (human machine interface) so that if prices exceed 30 cents per kWh, that they will increase the HVAC upper setpoint to 78 degrees to reduce electricity consumption. The Utility knows that this will be the behaviour of the building, due to the published “CarefullyBuy” setpoint of the DERF being 30 cents, hence it knows what price points will produce a response from this building and others in the area.

Example 2: Unexpected Grid Events

In this example we have a Utility that wants to be able to manage emergency situations caused by unexpected outages or faults on the grid. They would like to be able to manage the outage on a particular feeder (for example) by requesting an immediate reduction in load from DER in the effected area. The Utility already knows what services are available to it, hence the Utility can sum up the aggregate total of all the PriceMaps from DERF in the area of interest. The Utility can select the DERF resources that best meet its needs (from both a kVAh, response time and price perspective) and then issue the command via a call (or calls) to DERF:giveOffer with a “when” parameter of NOW. Assuming all DERF respond positively to this offer, the job is done (if not the IPER will need to issue additional offers, until the energy response goal is meet). Once the DERF has completed delivery of the service it will inform the IPER via a call to

IPER:reportDeliveryComplete. The IPER will check that the response was as expected and acknowledge this fact via a call to DERF:confirmDelivery.

Note: *For further discussion on Security, or other cross cutting aspects of this model, please see the "Criteria Coverage" section below.*

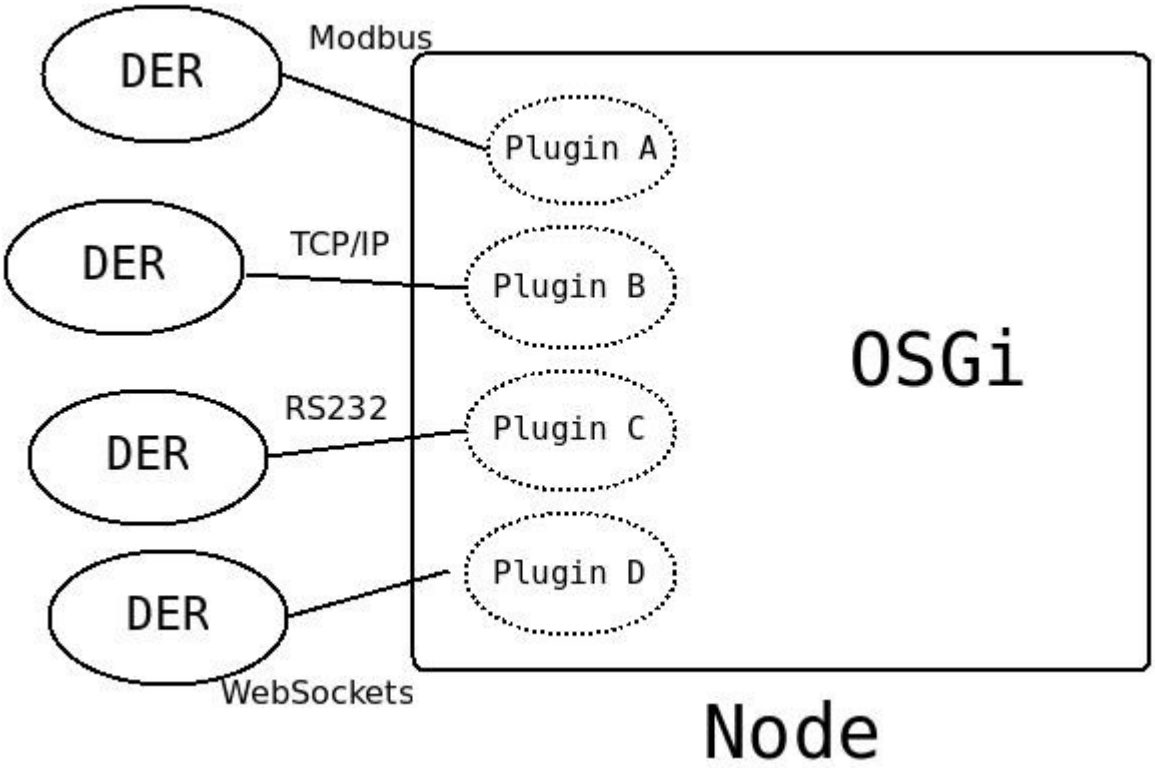
Discussion on Standards

As previously mentioned, we have specifically aimed to keep our ESI design as simple as possible, with a first principles approach. Consequently this has lead us to avoid leveraging existing grid related industry standards due to the complexity involved. Having said that, we are definitely leveraging standards from within the IT industry to facilitate the creation of a modern, efficient and secure computing infrastructure. Here is a list of some of the relevant computing infrastructure standards and protocols that we are using:

- TLS
- X.509
- HTTPS
- XML
- JSON
- MQTT
- ProtocolBuffers

DER Device to DER Facility Management Function Communication

Plugins are used to communicate between DER Devices and the Node's within a DER Facility (DERF) Management system. Plugins are implemented as Open Source or proprietary OSGi bundles. These plugins support whatever protocols are necessary to talk with the DER hardware components onsite (e.g. Digital IO, RS232, Modbus, proprietary XML over TCPIP etc).



Support for different protocols to cater for different DER hardware

Criteria Coverage

HIGH IMPORTANCE

► Configuration & Evolution

■What configuration methods exist to negotiate options or modes of operation (including support for user overrides if applicable)?

Within the DERF:

Plugins installed on the node are configurable by the owner of the Node. What configurations are available is determined by the hardware the plugin controls and the UI design associated with the plugin. Additional “control strategy” plugins can also be created specific to a scenario (e.g. solar PV backed with storage) so that the strategy and UI matches the actual DER configuration on site. Hence this allows a DERF to create a tailored user experience for their customer’s particular needs, without the customer having to purchase new equipment. In most instances, a DERF can deploy existing plugins from a plugin repository online to achieve this goal (i.e. only plugin configuration is required, not plugin development).

If new-to-market hardware is being deployed or new, never before seen combinations of hardware are being deployed that create new, never before seen opportunities to provide grid services, then the DERF may require new plugins to be developed. Before new plugins can be deployed on site in a customers production environment, this new hardware or hardware configuration will need to be tested in a lab. It will be tested in combination with the newly developed plugins to ensure it can be properly managed by a user and/or DERF and that they collectively offer services that conform to the expectations of the ESI Server.

The ESI Server supports the publishing of production-ready plugins and also the separate publishing of plugins under test. Commercial users would only have access to the production ready plugins. Software developers and test labs would have access to the repository that contains the plugins under test. Once testing and approval has been completed, test plugins can be elevated to production for commercial users to access.

Between DERF and IPER (i.e. the ESI):

When it comes to the ESI, the DERF is always in control of what services it offers. The DERF specifies the services that it makes available to the IPER and the price conditions under which it

will respond. There is a negotiation process available to the IPER which can be used to request more favourable pricing, but the actual set of services available are specified by the DERF and are not negotiable via the ESI. The ESI is designed such that it is easy for the IPER to select from a subset of the services that a DERF is offering and to cost effectively meet its needs.

In this way, the DERF can adjust its offerings over time and the IPER can choose from the ever evolving range of services provided by all the DERFs it connects with.

▪How they have the capability to revise and extend capabilities over time (versioning), while accommodating connections to previous versions of the interface?

The Plugins use OSGi which allows us to easily support versioning of the Plugins. Also, because Plugins are independent of the ESI, it is easy for hardware plugins to be upgraded without any impact on the functionality or backwards compatibility of the ESI.

Versioning of the ESI Server messages and the ESI API in particular is done by leveraging the ProtocolBuffers field numbering approach. If an existing message needs to be enhanced due to new requirements or deficiencies in previous implementations, then ProtocolBuffers easily supports the adding of new fields to existing messages WITHOUT breaking any existing implementations. Hence this allows the ESI Server and other components within the ecosystem to be upgraded as required, without the need to upgrade everything at once. Versioning information will also be explicitly built into the protocol to allow easy logical handling of changes that may occur within existing message structures.

▪How they accomplish unambiguous resource identification?

All nodes have a NodeID, this is unique to the instance of the ESI Server. Hence the combination of the NodeID and the URL of the ESI Server creates a Globally Unique ID. In addition to this, we allocate SourceIDs to each of the DERs on a Node. These SourceIDs are created with a hierarchy that allows easy human identification e.g.

Project / Site / System / DER

Where:

- **Project** = High level commercial identifier with specific meaning to the DERF (where DERF could be an aggregator).
- **Site** = A site is a physical location which could be large or small (e.g. a site may be a shopping mall with building management systems and solar array covered car parks)

- **System** = Specific system on a Site (e.g. a single instance of a battery backed solar array).
- **DER** = Specific hardware equipment within a System (e.g. an individual solar inverter).

Note: this organisation of entities is based on a flexible SunSpec OrangeButton design, which comes from 9 years of studying the distributed generation market, and reflects the actual setups of many stakeholders likely to participate in this program.

▪How they implement unambiguous resource discovery methods and its management?

The NodeID and ESI Server uniqueness is enforced by software infrastructure. The ESI Server allocates all NodeIDs following a Node registration and key exchange process that ensures uniqueness, security and non-repudiation (see “The Semantics of Node Registration” section above). SourceIDs are the responsibility of the DERF to manage as is appropriate for their needs (i.e. to allow them to best structure the DER resources under their control for the purposes of offering services via the ESI).

At the DERF and IPER level, assuming authority has been granted by the Node owner to do so, SourceIDs can be searched for by leveraging knowledge of the SourceID naming conventions described above.

From the perspective of the IPER, SourceIDs are irrelevant though. The IPER only cares about the services that a DERF is offering (remembering that a DERF’s offering includes specific location information, via the Node location data). **Hence the design of the ESI is successful due to its ability to achieve complete abstraction between the individual DER involved in providing a service and the necessary results that the IPER desires.**

► Safety & Security

▪What features address concerns for privacy and security, including how policies are defined, maintained, and aligned?

All communication between remote Nodes and the ESI Server(s) use HTTPS, with the minimum supported encryption version being TLS1.0

Posting data to the ESI Server requires a ESI Server issued X.509 client certificate for authentication (this ensures non-repudiation of the source of the data collected).

All actionable services performed by a Node requires a specifically allocated token with a paired secret key implemented with a physical security key. This prevents unauthorised agents from accessing the actionable services. These tokens can only be issued by the owner of the Node

(which will most likely be the owner of the DERF, but could also be the owner of the DER in some cases). Access to any resource is controlled via security tokens and their associated security policy.

The following external links describe these security features in further detail:

- https://github.com/SolarNetwork/solarnetwork/wiki/SolarNet-API-authentication-scheme-V2
- https://github.com/SolarNetwork/solarnetwork/wiki/SolarUser-API#associate-node
- https://github.com/SolarNetwork/solarnetwork/wiki/SolarNet-API-global-objects#security-policy

■How are failure modes dealt with, including policies and how they support the safety and health of individuals and the overall system?

Failure modes will be dealt with at the individual DER plugin level and also at the “control strategy” plugin level within each DERF. This way, strategies appropriate to the individual hardware components and the specific configuration can be developed or configured to match the likely and expected failure modes. This is the responsibility of the DERF, but due to the common needs of every DERF, the Open Source implementation of “control strategy” plugins and DER level plugins will converge on solutions that provide safe failure modes by default. (Note: this would not remove the possibility of badly configured systems failing in unsafe ways however.) These plugins and control strategies will be tested and confirmed before they are released publicly for use in commercial applications.

Test cases for common configurations need to be defined and industry testing of these configurations needs to be completed by labs or official bodies. Certification will be provided for particular official releases of this software stack and the associated configurations. DERFs may also develop their own plugins or control strategies. Unless the DERF is particularly skilled in the customisation of plugins and control strategy development, it would be advised to stick to using the officially released plugins. This advice should be followed for all commercial production deployments.

► Operations & Performance

■How the devices address time order dependency and sequencing of interactions?

NTP date time synchronisation is installed on all Nodes and ESI Server instances, this allows Nodes to easily maintain local time accurate to within tens of milliseconds (if accuracy to the nearest millisecond is required for the specific application at hand, a local (i.e. onsite) NTP server can be established to ensure this is the case). At a functional level, all time information is recorded and exchanged in UTC (although local time translation are often performed at the User Interface

level to provide ease of use for the operator). This allows any time dependant actions to be accurately executed and recorded.

▪What time synchronization requirements exist and how they are managed?

NTP date time synchronisation is installed on all nodes and ESI Server instances. All components need to have regular connectivity to NTP servers (standard Internet accessible NTP servers should be sufficient).

▪What capabilities the system has for managing transactions and device state?

Nodes are able to maintain state for any DER that it is responsible for (To achieve this, Nodes include runtime memory and industrial SLC SD media for permanent storage if required). This state information is communicated to higher level entities such as the DERF and IPER as required.

► Informational

▪How information models (i.e. semantic ontologies) are used in information exchange?

Our information model is fully defined by our ESI.idl. Other than this we have not created any additional information models.

►Technical

▪What the structure and format is of the communication transport used and its management?

The main administrative communication of this system is JSON over RESTful API endpoints. This is supplemented with a real time ESI component using ProtocolBuffers over gRPC for time critical interactions (such as spinning reserve or frequency regulation requests).

MEDIUM IMPORTANCE

► Configuration & Evolution

■The accommodation and migration path for integration between legacy and new components and systems shall be described?

DER Migration Paths:
New plugins can be written for legacy components if support is required. Plugins can also easily be written for new components as required. There are no limits on what components can be supported. As long as a hardware computing platform can be found with suitable physical IO to talk to the new or legacy component, then plugins can be developed. If it is possible to install and run a minimal Debian Linux distribution on the computing platform, then plugins can very easily be developed and deployed. What this does is provide a pathway for all hardware to fit into the framework as needed. This is achieved with both upgrade and migration strategies but also by keeping the opportunity open for any legacy situation where the manufacturer does not support the equipment any more.

ESI Migration Paths:
Due to the use of ProtocolBuffers to implement ESI.idl, it is easy for extensions or modifications to be added to any ESI version without causing backwards compatibility problems for old equipment or old service providers. If a new version of the ESI.idl offers new functionality, obviously this new functionality will not be available to old service provider infrastructure built specifically to support the previous ESI.idl version, but previous functionality will still be available and runtime incompatibilities can easily be avoided.

■How regional and organizational differences are supported shall be described?

The system is timezone aware. Because the ESI Servers can be deployed at any level of granularity, it is easy for regional or organizational differences to be catered for at whatever scale is necessary. For example, an aggregator could deploy an ESI Server for the purposes of managing a single town, city or state. The nature of aggregate offerings that are served up to the local utility would be designed to conform to the appropriate local laws. In many instances this can also be achieved while sharing a ESI Server that is used by other aggregators and utilities with different legal requirements. This is possible because the ESI Server platform and the associated Nodes are purely a conduit for information and executing of commands or requests. What commands or requests are made and what information is collected and how these commercial services are structured from a legal and financial perspective is a choice that can be made by the aggregator and the Utilities under the guidance of the relevant local laws.

Currency variations are not currently being addressed by the ESI Server (currently we are assuming all transactions are in a common currency such as USD).

■The ability of overall system operation and quality of service to continue without a disruption as parties enter or leave the system shall be supported.

By being able to aggregate DERF resources under its control, an IPER can easily determine what parties/resources it has under contract and hence it is also able to easily determine the impact of any one of these parties leaving.

► Safety & Security

■The requirements and mechanisms for auditing and for logging exchanges of information shall be described?

All logs on Nodes are accessible but must be viewed one at a time on each individual Node. We should consider using Fluentd to consolidate logs in a single location for DERF aggregators. Likely we would use Graylog as the front end.
e.g. https://www.fluentd.org/guides/recipes/graylog2

Beyond this mundane logging of the software's operational performance, the more relevant logging will be around the recording of services delivered over the ESI and the evidence supporting confirmation that service delivery was as contracted to be delivered. This is managed by the historical data recording and the recorded forward projections from the artificial Neural Network before the service was engaged.

■Performance and reliability requirements shall be defined?

Nodes should be very reliable, they should have a 99.99% uptime and last for at least 10 years. Nodes store and forward data, so intermittent comms is acceptable for logging and dynamic response purposes. Interactive real-time requests (such as spinning reserve) will likely require redundant communication layers with very high uptime if 99.99% delivery is required from a particular DER.

The ESI Servers shall be deployed in high uptime Data Centers with physical redundancy and load balancing strategies in place suitable to address the needs of its users. ESI Server instances should be designed and deployed to have a 5 nines uptime.

► Operations & Performance

■The way errors in exchanged data are handled shall be specified?

All data must pass validation checks before being passed on via an interface. Errors caused by a system's internal processing that are not able to handled appropriately by that systems business logic (i.e. its internal operating) shall be passed onto the calling system as an InternalError for the purposes of central logging and easy remote analysis of issues.

► Organizational

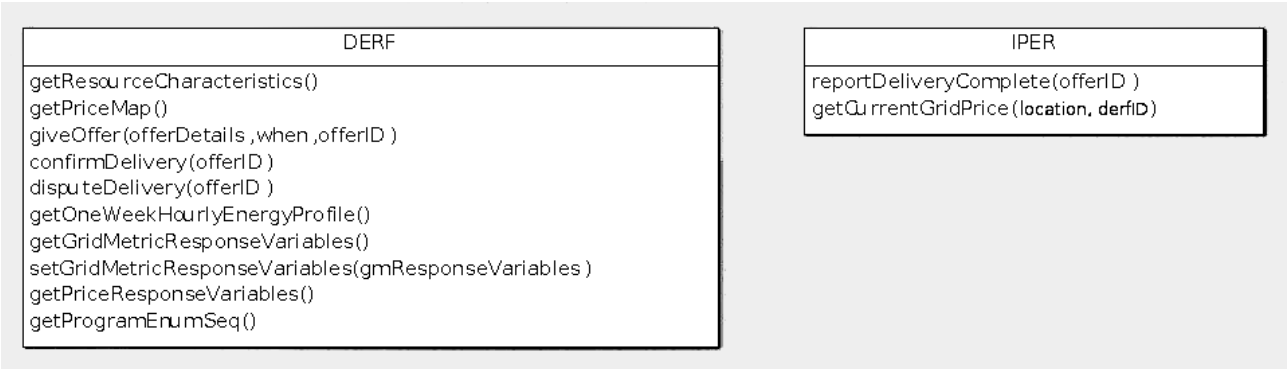
■Business conducted across the interface shall be aligned with jurisdictional economic and regulatory interoperability policies defined for the community?

The ESI Server platform and the associated Nodes are purely a conduit for information and executing of commands or requests. What commands or requests are made and what information is collected and how these commercial services are structured from a legal and financial perspective is a choice that can be made by the aggregator and the utilities under the guidance of the relevant local laws. Hence it is possible for individual communities to design and configure their Smart Grid using the ESI Server to meet the needs of their community and jurisdictional economic and regulatory requirements.

► Informational

■Information models relevant for the interface shall be formally defined using standard information modeling languages?

The interface is currently being modeled with CORBA IDL (this is effectively being used as pseudo code because that's what this author finds easy to do, this IDL will likely be converted to gRPC's Protocol Buffers during implementation). CORBA IDL is easy to convert to UML class diagram, a high level of this has been done below (please see ESI.idl for full details). Also the "Integration Narrative" above contains a UML sequence diagram for reference.



APPENDIX A

```
//--------------------------------------------------------------------------------
// DER Challenge
//--------------------------------------------------------------------------------
//        File: $RCSfile: ESI.idl,v $
//  Originator: pccourt
//        Date: Tue Aug 26 14:26:47 2018
// Application: ESI draft ideas
//      Module: ESI.idl
//       RCSID: $Id: $
//--------------------------------------------------------------------------------
// Last Edited: $Author: $
//        Date: $Date: $
//    Revision: $Revision: $
//--------------------------------------------------------------------------------
//
// Description : Psuedo IDL code for a draft ESI for the DER Challenge
//
// Assumptions : This is to be viewed as a logical description only, this will
//               be converted to the appropriate protocol/transport layer
//               technology when it is being implemented (likely to use MQTT or
//               gRPC).
//
//               This interface is intended to be a very low level first
//               principles API that contains full flexibilty for any business
//               model that might like to be implemented on top of it.  The
//               first principles approach is based on the concepts of time,
//               power, energy and money. It is a first draft that conveys
//               these ideas in a way suitable for a MVP demo, but expansion
//               of these ideas will likely be needed for a full production
//               ready v1.0 implementation to be completed.
//
//               This interface should also be viewed with the understanding
//               that an existing JSON protocol operates over https in parallel
//               with this one to provide timely data about power quality,
//               energy consumption and generation and future energy
//               predictions at particular locations within each DERF.
//               This provides a diffinitive historical account of events,
//               historical future expectations and evedence for responses
//               that occured onsite at each DERF for validation purposes.
//
// Dependencies: NONE (its intended to be IDL Psuedo code!)
//
//--------------------------------------------------------------------------------

#ifndef ESI_IDL
#define ESI_IDL

//---- Library includes -------------------------------------------------------
//---- Core library includes --------------------------------------------
#include <Pingable.idl>  //makes the interface pingable (useful for simple
                         //remote testing and measuring comms layer delays)

//---- Application includes ---------------------------------------------------
//---- Local includes ---------------------------------------------------------
//---- Interface Prototypes ---------------------------------------------------

module DER_CHALLENGE

/**
 * Encapsulates information about the functionality of an ESI.  Embodies the
 * communications gateways between the DER Facility (DERF) and the Interacting
 * Party with External Responsibility (IPER) e.g. Between a system operator
 * and a DER Facility.
 *
 * Communication between a DER Facility and its collection of DER Equipment is
 * NOT covered here. To see info on this, please check info on Plugins
 * which is covered on the SolarNetwork project's github wiki here:
 * https://github.com/SolarNetwork/solarnetwork/wiki/SolarNode-Development-Guide
 */
   module ESI
```

```
////////////////////////////////////////////////////////////////////////
//
// The ESI API must statisfy all of the following...
//
// A : Fast and immediate responses
// ----------------------------
//  e.g.
//  1) Spinning reserve (on standby)
//  2) Frequency regulation (4 sec interval signal from system balancing
//     authority)
//  3) Ramping (on standby to rapidly increase/decrease load)
//
// B : Fast dynamic responses
// ----------------------
//  e.g.
//  1) Artificial Inertia (i.e. complement the grids angular momentum)
//  2) Voltage management (adjust local reactive and/or real power
//     components to maintain specified voltage range)
//
// C : Slow planned responses
// ----------------------
//  e.g.
//  1) Peak Capacity Management (as needed, OpenADR like...?)
//  2) Contractual obligations (e.g. suppling capacity in the wholesale
//     energy market, OpenADR like...?)
//
// D : Slow dynamic responses
// ----------------------
//  e.g.
//  1) Market price response
//
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
//
// Data Structs
//
// This section contains some basic data types used throughout this API
//
////////////////////////////////////////////////////////////////////////

//Common data types within the ESI
typedef
const Version_T VERSION          //Version of the Interface being
                                 //implemented (allows backwards
                                 //compatibility logic to be introduced
                                 //at a later date if required)

typedef float kWh                //energy
typedef float kW                 //power
typedef float Dollars            //USD
typedef long MilliSeconds        //msec
typedef long Seconds             //sec

//Basic electrical power data types
typedef float kVAR      //Ractive Power = Q
typedef float kW        //Real Power = P

//kVA = S (Apparent Power)
struct kVA

   Version_T version
   kVAR Q    //Reactive Power
   kW P      //Real Power


// forward declarations

//Interacting Party with External Responsibility (eg System Operator)
interface IPER

/**
 * Interface: DERF
```

```
       *
       * The DER Facility (or a node that directly manages HW). Encapsulates
       * information about the functionality of a DER Facility (DERF).
       */
      interface DERF
         Ecogy::Corba::Pingable


         // Base characteristics of the hardware present on site (or for the
         // agregate devices below this DERF) In the final implementation,
         // these data structures will be defined with ProtocolBuffers IDL and
         // will inherently support backwards compatible extensions of these
         // data structures if that is needed. ESI specific strategies for
         // message or API versioning is easily acheived.
         struct ResourceCharacteristics

            Version_T version
            kWh      storage    //How much usable electrical energy can be stored on site (includes
heat energy and battery storage)
            kW       consumptionMaxPower  //Limited by hardware present on site (-ve = consumption)
            kW       consumptionMinPower  //Limited by hardware present on site (a value of zero
indicates this DERF can swtich to zero load if requested)
            kW       generationMaxPower  //Limited by hardware present on site (+ve = generation)
            long     responseTime     //Expected response time of resource (ms)
            float    powerFactor      //Expected power factor of these loads ???


         /**
          * Method: getResourceCharacteristics
          *
          * Gets the default ResourceCharacteristics for this DERF.
          * This provides a high level understanding to the IPER of what is
          * available from this resource. It does not however provide info on
          * how its services might vary over the immediate future (see
          * TODO ???? get30MinPrediction for this type of info).
          *
          * Parameters: NONE
          *
          * Return:
          *    ResourceCharacteristics - Characteristics for this DERF
          */
         ResourceCharacteristics getResourceCharacteristics
                   InternalErrorException )


         // PriceMap - An element in a sparse matrix of the form fn(x, y, z)
         // Note that S is actually a vector (rather than a single number),
         // hence the the 4D fn(x, y, z) analogy is incorrect (its actually a
         // 5D membrane). A PriceMap effectively defines a single physical
         // capability (or a agregated set of similar capabilities) that the
         // DERF has to offer.
         struct PriceMap

            Version_T version
            kVA S    //Apparent Power ("change in" apparent power that is being offered in this
PriceMap)
            Seconds duration
            MilliSeconds responseTime  //how long will this DERF take to respond when requested to
make this service immediately
            Dollars price  //indicative price per kVAh of service with these parameters

         typedef           PriceMap   PriceMapSeq

         /**
          * Method: getPriceMap
          *
          * Presents an indicative price map to the IPER (using a sparse
          * matrix, 4D surface). This sparse price map is easy to agregate
          * for any entity that may be managing multiple DERFs or for the
          * System Operator/Utility to do the same. For example the System
          * Operator could explore this 4D surface looking for the best and
          * most cost effective way to address the immediate or upcoming needs
          * of the grid.
          *
```

```
 * For some elements not specifically listed in the sparse matrix,
 * they are still available for request.  For example if the matrix
 * states that a service with a duration of 10 minutes is available,
 * then of course the IPER can request a similar service (with the
 * same kVA and response time), but with a shorter duration.
 *
 * Parameters: NONE
 *
 * Return:
 *    PriceMapSeq - The sparse matrix of pricing data
 */
PriceMapSeq getPriceMap
          InternalErrorException );

//*****************************************************************
// SECTION A : Fast and immediate responses
// SECTION C : Slow planned responses
//
// This section of API below supports both types A & C
//
//*****************************************************************

//
// OpenADR 2.0 should probably be supported in addition to APIs here
//

// OfferResponse is a structure that the DERF passes back to the IPER
// after receiving an offer. Note that the counterOffer will only be
// populated if this response indicates that the orginal offer is not
// accepted, in this case this counterOffer sequence will contain one
// element.
struct OfferResponse

   Version_T version
   boolean accept   //Did we accept this offer (yes/no)?
   GUID offerID   //unique identifier of the Offer that is being accepted or rejected
   PriceMapSeq counterOffer  //this sequence contains the counter offer (if one is made)


/**
 * Method: giveOffer (would possibly be better named requestService?)
 *
 * Called by the IPER to provide a price offer for service. The
 * DERF is receiving an offer from the IPER when this happens.
 * Note: DERF to IPER relationships are many to one.
 * Before calling this method, the IPER will have evaulated the
 * PriceMapSeq and decided that it would like to engage the services
 * of this DERF The IPER can match the requested pricing in the
 * PriceMap, or make an offer. Hence this methods name "giveOffer".
 *
 * Parameters:
 *    offerDetails - The details of the price being offered and the
 *                   nature of the service to be provided.
 *    when         - When the Service is requested (can also be NOW,
 *                   i.e. an immediate request).
 *    offerID      - A unique identifier of this offer (provided by
 *                   the IPER for later reference).
 *
 * Return:
 *    OfferResponse - Details of offer acceptance or counter offer
 */
OfferResponse giveOffer in PriceMap offerDetails  in DateTime_T when  in GUID offerID
          InternalErrorException );

// To report delivery of service i.e. the "is complete" message to the
// IPER, see the IPER interface at the end of this file...

//Method used by the IPER to acknowledege successful reciept of the
//requested service
void confirmDelivery  in GUID offerID
          InternalErrorException );

//Method used by the IPER to dispute reciept of the requested service
void disputeDelivery  in GUID offerID
```

```
                    InternalErrorException )

        //Energy profiles
        typedef     24HrHourlyEnergy       //24 hrs of predicted 'actual' future Energy needs (+ve
= generation, -ve = consumption)
        typedef                 OneWeekHourlyEnergyProfile     //Monday to Sunday

        /**
         * Method: getOneWeekHourlyEnergyProfile
         *
         * Gets predicted energy profile for the next 7 days.  Used by the
         * IPER to statistically measure the accuracy of the DERFs predictions
         * over time and to also verify any respose that the DERF may
         * contractually provide.
         *
         * TODO IPER may require more granular or longer duration predictions?
         *
         * Parameters: NONE
         *
         * Return:
         *    OneWeekHourlyEnergyProfile - One week (7 days) predicted energy
         *                                 profile for this DERF.
         */
        OneWeekHourlyEnergyProfile getOneWeekHourlyEnergyProfile
                    InternalErrorException )


        //*****************************************************************
        // SECTION B : Fast dynamic responses
        //
        // This section of API below supports type B responses
        //
        //*****************************************************************
        struct SafeParameterRange

           Version_T            version
           string               name    //Name of the paramter (e.g. Voltage or Frequency)
           string               units   //Engineering units of the paramter (e.g. V or Hz)
           EngineeringUnits     maxCriticalLimit    //Max value allowed (beyound this widespread
grid failures are likely)
           EngineeringUnits     minCriticalLimit    //Max value allowed (beyound this widespread
grid failures are likely)
           EngineeringUnits     maxAcceptable    //Acceptable safe upper limit
           EngineeringUnits     minAcceptable    //Acceptable safe lower limit
           EngineeringUnits     target    //Set point for this paramter


        struct GridMetricResponseVariables

           Version_T            version
           SafeParameterRange   voltage
           SafeParameterRange   powerFactor
           SafeParameterRange   frequency


        /**
         * Method: getGridMetricResponseVariables
         *
         * Request info about how this DERF dynamically responds to grid
         * parameters. This information informs the IPER as to the
         * likely/expected response of this DERF to real time grid changes.
         *
         * Parameters: NONE
         *
         * Return:
         *    GridMetricResponseVariables - info about grid parameter
         *                                  sensitivity.
         */
        GridMetricResponseVariables getGridMetricResponseVariables
                    InternalErrorException )

        /**
         * Method: setGridMetricResponseVariables
         *
```

```
         * Attempt to define how this DERF dynamically responds to grid
         * parameters (an authorized IPER can change these settings). This
         * information controls how the DERF will respond to grid changes.
         *
         * This method is not currently showing a price offer parameter, but
         * it should conceptually be designed to do this? Because if the DERF
         * is in a non regulated situation, it may prefer not to respond to
         * these constraints unless it is appropriately compensated?
         *
         * Parameters:
         *    gmResponseVariables - the grid parameters to maintain
         *
         * Return:
         *    true  - if update accepted (TODO maybe provide/offer a more
         *            nuanced response?)
         *    false - if it is NOT accepted.
         */
        boolean setGridMetricResponseVariables in GridMetricResponseVariables gmResponseVariables
                  InternalErrorException )


        //*********************************************************************
        // SECTION D : Slow dynamic responses (retail price behaviour drivers)
        //
        // This section of API below supports type D responses
        //
        //*********************************************************************
        struct PriceResponseVariables

           Version_T mVersion   //Its likely this data structure will become much more intricate
and complex to meet the needs of the customer
           Dollars       mAlwaysBuyPrice    //Price below which the customer will always consume if
they have a load running or scheduled to run
           Dollars       mCarefulBuyPrice   //Price below which the customer will be more selective
about consumption (what this means will be customer and DER HW specific)
           Dollars       mNeverBuyPrice     //Price above which the customer will never consume
(because it is too expensive) - suggests customer will use battery or shutdown


        /**
         * Method: getPriceResponseVariables
         *
         * Request info about how this DERF responds to prices (only the DERF
         * owner can change these settings).
         * This information informs the IPER as to the likely/expected response
         * of this DERF to real time price changes. These variables are
         * generally set by the owner of the property or the company in charge
         * of managing the DERs onsite on behalf of the owner (how these are
         * actually set is out of scope of this API).
         *
         * Parameters: NONE
         *
         * Return:
         *    PriceResponseVariables - info about price sensitivity
         */
        PriceResponseVariables getPriceResponseVariables
                  InternalErrorException )


        //The types of programs offered/supported by this DERF
        enum ProgramEnum_T
           Program_SPINING_RESERVE
           Program_FREQ_REGULATION
           Program_RAMPING
           Program_ARTIFICIAL_INERTIA
           Program_VOLTAGE_MANAGEMENT
           Program_PEAK_CAPACITY_MANAGEMENT
           Program_CONTRACTUAL_OBLIGATIONS
           Program_MARKET_PRICE_RESPONSE
           NUM_Program

        typedef           ProgramEnum_T   ProgramEnumSeq
        /**
         * Gets the full set of services this DERF provides.
```

```
         * For informational purposes only at this point... This concept
         * does not directly tie in with any of the other methods defined here
         * but its an easy way to communicate to the IPER what services
         * the DERF intends to try to offer with its capabilities.
         */
        ProgramEnumSeq getProgramEnumSeq
                  InternalErrorException );

        // end of DERF

      /**
       * Interface: IPER
       *
       * The Interfacing Party with External Responsibility (IPER) e.g. a System
       * Operator. Encapsulates information about the functionality of an IPER
       * that might be used by a DERF.
       */
      interface IPER
        Ecogy::Corba::Pingable


        //DERF calls this method on the IPER to confirm that the requested
        //service has been delivered
        void reportDeliveryComplete  in GUID offerID
                  InternalErrorException );


        // Allows a DERF to drive its dynamic price dependant behaviour.
        // TODO should also offer a broadcast service that DERFs can register
        // with to recieve real time price changes.
        Price getCurrentGridPrice in string Location  in string derfID
                  InternalErrorException );


        //============ Future Thinking ================

       // Not sure if we need the methods below (they imply another layer of
       // rigidity to the API that may not be needed here - since this API is
       // intended to be very generic, we can add this at another layer above
       // this ESI if it is requried e.g. if local regulations dictate a
       // certain shape to the smart grid programs, these specific structures
       // could be built above this ESI). i.e. the IPER could be implemented
       // in such a way that it only offered and accepted service engagements
       // that matched the local regulations (programs), these could be
       // implemented using the ESI interface documented here, but the
       // functionality actually delivered by the DERF<->IPER relationships
       // could be constrained by these programs...

       // registerForPrograms(in ProgramEnumSeq programs, in derfID)
       //    raises( InternalErrorException );
       // deregisterForPrograms(in ProgramEnumSeq programs, in derfID)
       //    raises( InternalErrorException );

        // end of IPER
      // end of ESI
    // end of DER_CHALLENGE

#endif // ESI_IDL
```