T.C.

YEDITEPE UNIVERSITY

FACULTY OF ENGINEERING

GREEN ENERGY INTEGRATED SMART GRID SYSTEM

Instructor: Mert Özkaya

ABSTRACT

Power systems, communication, software and hardware technologies, instrumentation, big data, and other engineering practices are all interconnected by green energy integrated smart grid system. The design must realize complex legacy power grid systems while also coping with modern Information and Communication Technologies (ICT). The software architecture in a smart grid system is expected to be distributed and to allow for local control. Furthermore, smart grid architecture should be flexible, adaptable, and sensitive to technological advancements.

Kamran Musayev

Nursena Karayılanoğlu

Burak Koçak

CSE 447

Table of Contents

1. INTRODUCTION ..... - 2 -

1.1 PURPOSE OF THE DOCUMENT ..... - 2 -
1.2 GLOSSARY ..... - 2 -
1.3 GENERAL KNOWLEDGE ABOUT THE DOMAIN ..... - 3 -
1.4 CUSTOMERS AND USERS ..... - 3 -
1.5 ENVIRONMENT ..... - 3 -
1.6 TASKS AND PROCEDURES CURRENTLY PERFORMED ..... - 3 -
1.7 COMPETING SOFTWARE ..... - 4 -

2. PROBLEM INDENTIFICATION ..... - 4 -

3. REQUIREMENTS ..... - 5 -

3.1 FUNCTIONAL REQUIREMENTS ..... - 5 -
3.2 USE-CASE DIAGRAM ..... - 7 -
3.3 NON-FUNCTIONAL REQUIREMENTS ..... - 7 -
3.4 VOLERE TEMPLATES ..... - 9 -

4. ARCHITECURE – AN EMBEDDED SOFTWARE VIEW ..... - 13 -

4.1 LAYERS ..... - 13 -
4.2 INTERACTIONS ..... - 14 -
4.3 COMPONENTS ..... - 14 -
4.4 XCD MODELLING APPROACH ..... - 17 -

5. TRACEABILITY ..... - 26 -

5.1 FUNCTIONAL REQUIREMENTS ..... - 26 -
5.2 NON – FUNCTIONAL REQUIREMENTS ..... - 26 -

6. CONCLUSION ..... - 27 -

1. INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

The domain is 'Electric Power Distribution'. The goal of this domain analysis is to create a new system that improves on an existing one by combining energy and information technology infrastructure to integrate and connect all users (producers, operators, marketers, and consumers) in order to efficiently balance demand and supply across an increasingly complex network. Furthermore, the green energy integrated smart grid platform will eliminate flaws and deficiencies in electric power distribution, which has been condemned as wasteful in many ways. The smart grid software architecture is defined as a large-scale, multidisciplinary, highly networked, data-driven system. There has been a rush of suggestions and trials in smart grid research to investigate software architecture techniques.

Motivation

The application of software engineering approaches in the domain of power grids is a multidisciplinary, research-intensive field. Designing and developing software architecture for smart grids involves extensive study across multiple fields. To achieve the goals of smart grid, the software components are organized in layers. The software system runs on the embedded electronics in the electrical grid, which are located at the numerous smart grid substations. The ability of the electronics to connect with one another and use feedback transforms the electrical grid into a smart grid. The adoption of an operational smart grid has the potential to alleviate some of the issues confronting the energy sector. This is due to the increased penetration of variable energy supplies enabled by a flexible management system. When compared to traditional grids, the smart grid concept has proven to be more reliable.

The effective deployment of smart grid technologies enables the integration of renewable energy, the reduction of CO2 emissions, and the management of electricity consumption. It is now possible to meet demand needs with the help of smart grid. Furthermore, smart grid offers efficiency-driven reaction and dependability which will be deeply explored in this report.

1.2 GLOSSARY

The electronics and firmware architecture configures, monitors, maintains, and regulates energy equipment on the electric power grid. The essential elements of smart grid design are dependability and definitive control over power grid equipment, as well as security. As a result, much of the domain's special terminology will be computer-related, but there will also be terminology connected to electrical equipment, distribution and hardware equipment.

1.3 GENERAL KNOWLEDGE ABOUT THE DOMAIN

Smart electricity grids include sensors and smart meters that allow for two-way communication. As a result, software systems that do demand response optimization require a platform that can enable data production and calculation. However, in order to implement this platform, statistics on power outages, used equipment, power distribution structure, and other relevant data should be collected and analyzed in order to determine the level of performance your system requires. Furthermore, you should be aware of the various types of power outages and how the existing system handles each of them, as well as the current configuration of the system's layout, how long it takes to deal with an emergency situation, where the power comes from, and what the average distance from source to destination is.

1.4 CUSTOMERS AND USERS

This architecture aids in the efficient and intelligent distribution of energy to end consumers and the integration of renewable energy sources, as well as their creation and delivery. Also, it helps to have communication between consumers and the utility control center over the appropriate network. This will result in more dependable, efficient, cost-effective, self-repairing, self-optimizing, and environmentally sensitive power services. However, you should be aware of what clients and users know and what they need to learn while dealing with an emergency. Some engineers may be opposed to the introduction of new software because they have accumulated extensive expertise with the current system and are concerned that a new system would render their skills outdated, or perhaps put them out of work. Knowing this allows you to take efforts to address their problems while avoiding any political complications.

1.5 ENVIRONMENT

Since energy providers and customers are distributed in a smart grid setting, the communication network will take on a hybrid structure with core networks and multiple edge networks that link all suppliers and customers. As a result, the computers, software, and distribution equipment currently in use should be investigated. Your clients may be eager to update generic hardware, but your system will undoubtedly need customized hardware that is expensive to replace.

1.6 TASKS AND PROCEDURES CURRENTLY PERFORMED

The processes presently used by power distribution providers will help you determine which functions you must adopt. These protocols are sometimes unclear, resulting in delays even in the midst of a crisis. Procedures include, for example, how priorities are formed in times of crisis, what happens when there aren't enough electricians to go around during a power outage, how are the records created and used? Examining these techniques will help you determine which components of the program may be improved and how the software can become a valuable asset

to your customer. You should also get familiar with the rules and regulations that may prohibit or restrict the software's use, either totally or partially.

1.7 COMPETING SOFTWARE

Smart gird architecture connects traditional power system automation (such as SCADA systems) to distributed energy resources. National Institute of Standards and Technology (NIST) standards define the architectural concepts for software layer organization and grid sub-system interfaces. These standards are intended to spur innovation, highlight best practices, and open up global markets for Smart Grid devices and systems. However, the industry is relatively immature, with numerous potentials for software to prosper. You may be possible to develop a generic product rather than a personalized product with more effort.

2. PROBLEM INDENTIFICATION

The conventional grid systems and process of deployment of renewable resources are inefficient and suffering with the integration of new technologies like AI, Big Data, smart house, smart solar panels etc. To overcome these problems, we propose a software system that will be the baseline for the upcoming technologies and bring efficient scalability for renewable resources and control over the conventional grid system. Another benefit we are expecting from the system is that it will enable us to harvest high amount of data that can be used in big data technologies to make the system evolutionary and adaptable with the aid of current developments in the field of AI. To point out the details of the current problems with conventional grid system we should investigate the struggles of development on this field. First of all, there is no power loss recovery system, to obtain such a robust technology we will be using intelligent node system in our software to redirect the electricity flow to prevent power loss on the end systems. Secondly, in the conventional grid system the energy distribution is highly imbalance in a day. Our proposed system will be dealing with this problem by the concept of smart city and solar panel integrated homes. We are planning to control the flow of the energy with a control center and intelligent node system. In this way we will be able to schedule the energy conception of the city and lower the loss of energy this will also affect the price, so we will benefit in many ways with the little changes in the distributed system. And lastly, the issue with the scalability of the renewable energy resources is becoming larger every day and damages the attempts on transferring to renewable energy from fossil energy. To fix this issue and encourage the companies and governments, we introduce a solution, which we will integrate, a deploy and feedback for green energy system into existing smart grid system. In this way we will be able to give deploy analysis to those who are interested in investing in the renewable energy area. And by the feedback system, Renewable resources will be managed and maintained easily and will be evolving in the environment of the smart grid system.

3. REQUIREMENTS

3.1 FUNCTIONAL REQUIREMENTS

3.11 **User Interface** for registering and logging-in in order to access the necessary components for providing and receiving energy or controlling and adjusting energy consumption and flow.

3.12 **Activate / Deactivate Service** based on the conditions or simply on their own free will.

3.13 ***Market*** to browse through offers and select the offer that best suits his or her needs to receive or provide energy depending on parameters. On the other hand, users should be allowed to publish an advertisement that has been approved by moderators in order to sell renewable energy through the smart grid system.

3.14 ***Feedback*** system for storing and reviewing user complaints and opinions in order to better serve consumers and resolve any technological issues that may arise. Customers should also receive feedback from the system based on their complaints.

3.15 ***Energy Resource*** support for both renewable such as Solar energy, Wind Energy, Hydropower and classical non-renewable energy resources.

3.16 ***Voltage and Frequency*** of the energy flow to or from itself should be controllable by the user. Furthermore, in order to maintain supply-demand balance, there should be a limited capacity for giving or receiving energy.

3.17 ***Network Status*** will enable browsing, connecting with, and messaging other users, as well as viewing basic information. The network will collect information such as delivered and received energy values, basic information about each user, connect numerous users, and automate the receiving and delivering energy system without the need for manual intervention.

3.18 ***Disturbance*** of any kind should be investigated and logged by the system before being analyzed to optimize energy flow.

3.19 ***Energy Supply Restoration*** to ensure uninterrupted energy flow to the user. The system should be able to detect any type of energy cut and automatically supply an alternate power source.

3.110 ***Backup Power*** will consist of any power source that is not currently being used by any user and is offered and listed in the system will be regarded as a backup power source to be used in emergency situations.

3.111 ***Tariff Reduction and Power Consumption*** will be controlled by system which will be installed in the user's home with the required sensors and actuators to continuously provide and send information to the system that will be cross-referenced with the electricity prices throughout the day, and the optimal time will be chosen to suggest or automatically configure the time interval based on the user preference in which selected electrical machines will work.

3.112 ***Fault Tolerance and Recovery System*** is one of the primary concerns when it comes to continuous energy flow. A smart grid network is required to connect a large number of electric equipment in various places and communicate status information and control instructions. System-wide intelligence is only possible if information transmission among the many functional units is efficient, dependable, and trustworthy. As a result, the system should save backed up system logs that will serve as a recovery system if or when the system malfunctions or shuts down completely. Furthermore, when devices are utilized for storage backup, any failure in the devices will send a warning signal to the system. As a result, the operation sequence of protection devices during a fault is critical.

3.113 ***Data Authenticity*** is critical feature which provides safeguard against unauthorized information alteration or deletion, while ensuring information non-repudiation and authenticity. When an integrity or authenticity issue is detected, the system must ensure that the data is not exploited.

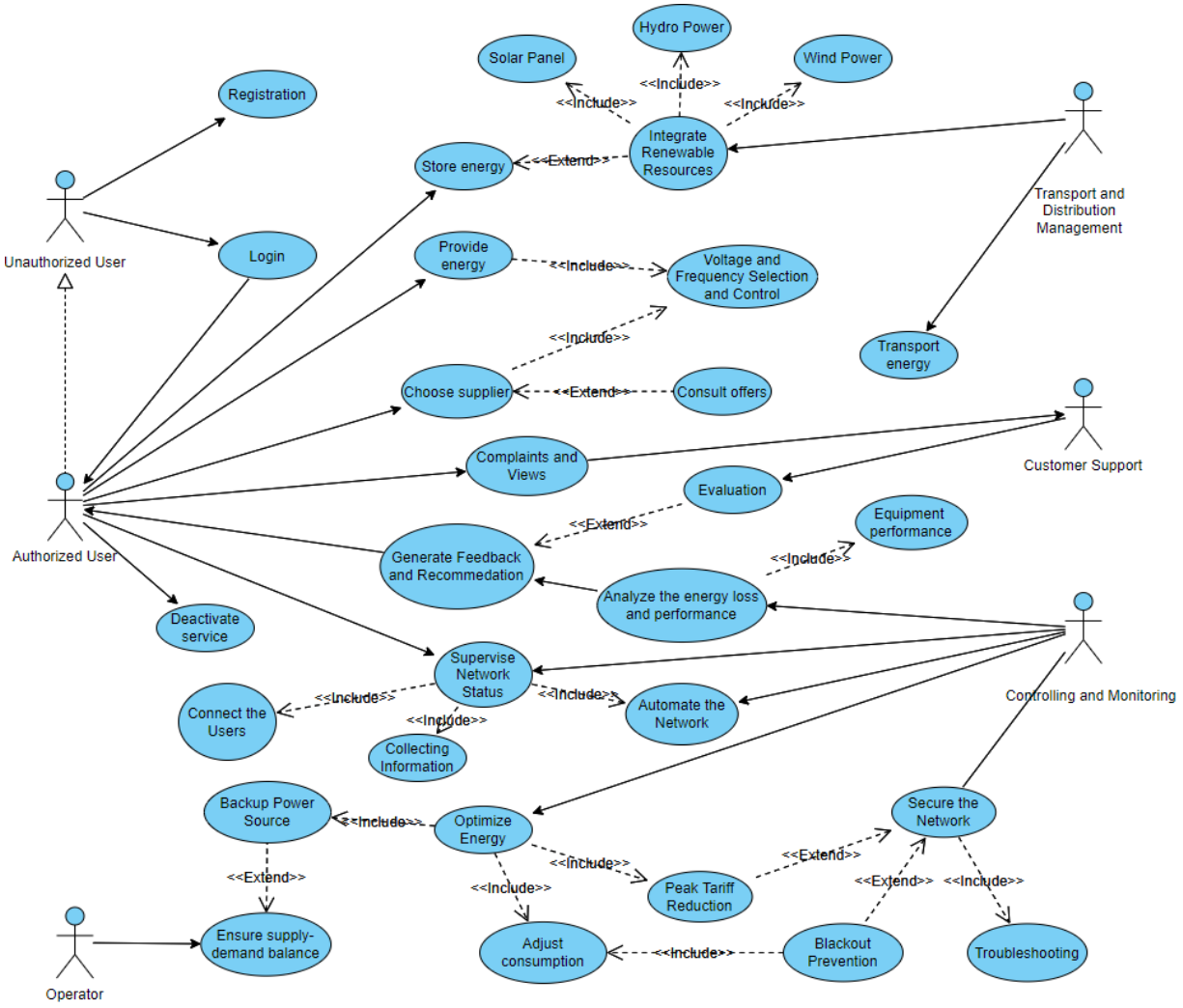
3.114 ***Data Storage System*** is concerned about grid services hosted on high-end resources such as expensive instruments and a dependable and high-speed database linking.

3.115 ***Information Access Limitation*** ensures that all types of information are protected by adequate safeguards against illegal access, alteration, disclosure, or destruction, as well as accidental loss or destruction, which eliminates the need to access client property.

3.116 ***Customer Evaluation*** of the network is critical for providing an optimal service; therefore, energy loss and performance should be examined, and appropriate recommendations and feedback should be made and sent to the customer.

3.117 ***Automated System*** to detect, analyze, and respond to changes automatically if human intervention is not required or the action is too crucial to wait for human input. Some systems, however, may still require manual programming of the components in their homes.

3.2 USE-CASE DIAGRAM



3.3 NON-FUNCTIONAL REQUIREMENTS

3.31 ***Security*** is one of the most critical features of software. Because the system will be linked to every component of a house, including solar panels and automobiles, the software must be capable of providing the end user with absolute privacy. Another factor to consider is that, because these systems have several distributed subsystems, we must protect both internal data and data transit between these systems. As a result, system security should be designed with this in mind.

3.32 ***Reliability*** is another critical feature of distributed systems. In our situation, the system should be fault-tolerant, so that the entire system can continue to function even if certain distributed sub-systems fail. This strengthens one of the system's primary principles.

3.33 ***Concurrency*** enables access and use the shared resource without interfering with other subsystems.

3.34 ***Openness*** ensures that the system's interfaces are standardized, and the specifications are published for future integrations.

3.35 ***Scalability*** is one of the major objectives of this system. As this criterion is met, we will be able to easily integrate the additional components into the system.

3.36 Transparency

- ***Access Transparency*** requires that objects are accessed with the same operations regardless of whether they are local or remote. That is, the interface to access a particular object should be consistent for that object, no matter where it is actually stored in the system.
- ***Location transparency*** is the ability to access objects without knowledge of their location. This is usually achieved by making access requests based on an object's name or ID which is known by the application making the request. A service within the system then resolves this name or ID reference into a current location.
- ***Replication transparency*** is the ability to create multiple copies of objects without any effect of the replication seen by applications that use the objects. It should not be possible for an application to determine the number of replicas, or to be able to see the identities of specific replica instances. All copies of a replicated data resource, such as files, should be maintained such that they have the same contents and thus any operation applied to one replica must yield the same results as if applied to any other replica.
- ***Performance transparency*** requires that the performance of systems should degrade gracefully as the load on the system increases. Consistency of performance is important for predictability and is a significant factor in determining the quality of the user experience.
- ***Migration transparency*** requires that data objects can be moved without affecting the operation of applications that use those objects, and that processes can be moved without affecting their operations or results.

3.37 ***Maintainability*** is a major problem because the system will be very dispersed and huge. The system should be kept in good working order with as little effort as possible. We can attain ease of maintenance with the aid of the loosely connected architecture.

3.4 VOLERE TEMPLATES

Requirement #: 1 (Security)	**Requirement Type:** NFR(Quality)
Description: The system shall provide security for the whole structure and interactions between distributed sub-systems.	
Rationale: The system stores and manages user and whole systems data thus system security is essential.	
Fit Criterion: The system must ensure that the whole data transfer is either encrypted or end to end with no intervention.	
Customer Satisfaction: 5	**Customer Dissatisfaction:** 5
Dependencies: Reliability, Maintainability	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 2 (Reliability)	**Requirement Type:** NFR(Quality)
Description: The system shall provide fault-tolerance, reliability and robust structure against the changes of the system and the environment.	
Rationale: To prevent one point failure and increase of the efficiency of the system.	
Fit Criterion: The system should not crash-stop even though some parts of the system crashes or stops.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: scalability, maintainability, concurrency.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 3 (Concurrency)	**Requirement Type:** NFR(Quality)
Description: The system shall provide concurrent interactions in the overall system.	
Rationale: To be able to achieve distrusted structure, the system should be able to work in concurrent behaviour.	
Fit Criterion: Even though the same resource is accesses by many sub-systems the resource management should be robust and access order is well-defined.	
Customer Satisfaction: 5	**Customer Dissatisfaction:** 5
Dependencies: scalability, access transparency, replication transparency, concurrency, performance transparency	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 4 (Openness)	**Requirement Type:** NFR(Quality)
Description: The system shall standardize the interfaces of the whole system.	
Rationale: To support scalability, integration and maintainability the openness requirement should be met.	
Fit Criterion: The interfaces of the system should be standard for all sub-systems and well-defined against ambiguity.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: scalability, access transparency, replication transparency, migration transparency.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 5 (Scalability)	**Requirement Type:** NFR(Quality)
Description: The system shall provide scalability with minimal effort.	
Rationale: With the introduction of new sub-systems like power-plant solar panel farms etc. the system should be able to handle these new sub-systems and integrated them to the system.	
Fit Criterion: After the introduction of the new sub-system the system should handle the newly introduced sub-system and integrate it into system within specified time limits.	
Customer Satisfaction: 5	**Customer Dissatisfaction:** 5
Dependencies: Location transparency, replication transparency, migration transparency.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 6 (Access Transparency)	**Requirement Type:** NFR(Quality)
Description: The system shall abstract the access to the objects.	
Rationale: With confirmation of this requirement the system will be able to abstract the data from the system and standardize the interactions no-matter where the object is stored.	
Fit Criterion: The objects should be accessible with different operations.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: scalability, replication transparency, openness, security, location transparency.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 7 (Location Transparency)	**Requirement Type:** NFR(Quality)
Description: The objects of the system shall be accessed without the knowledge of the location.	
Rationale: To support scalability, concurrency, and most importantly security.	
Fit Criterion: If the objects can be accessible without knowledge of the location, we can say that the system is met with this requirement.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: scalability, access transparency, replication transparency, migration transparency, security, maintainability.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 8 (Replication Transparency)	**Requirement Type:** NFR(Quality)
Description: The objects of the systems shall be replicable without any effects on the applications using replicated objects.	
Rationale: To support reliability, concurrency, maintainability, performance.	
Fit Criterion: There must be no effects on the applications using replicated objects.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: scalability, access transparency, migration transparency, concurrency, maintainability, performance, reliability, security.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

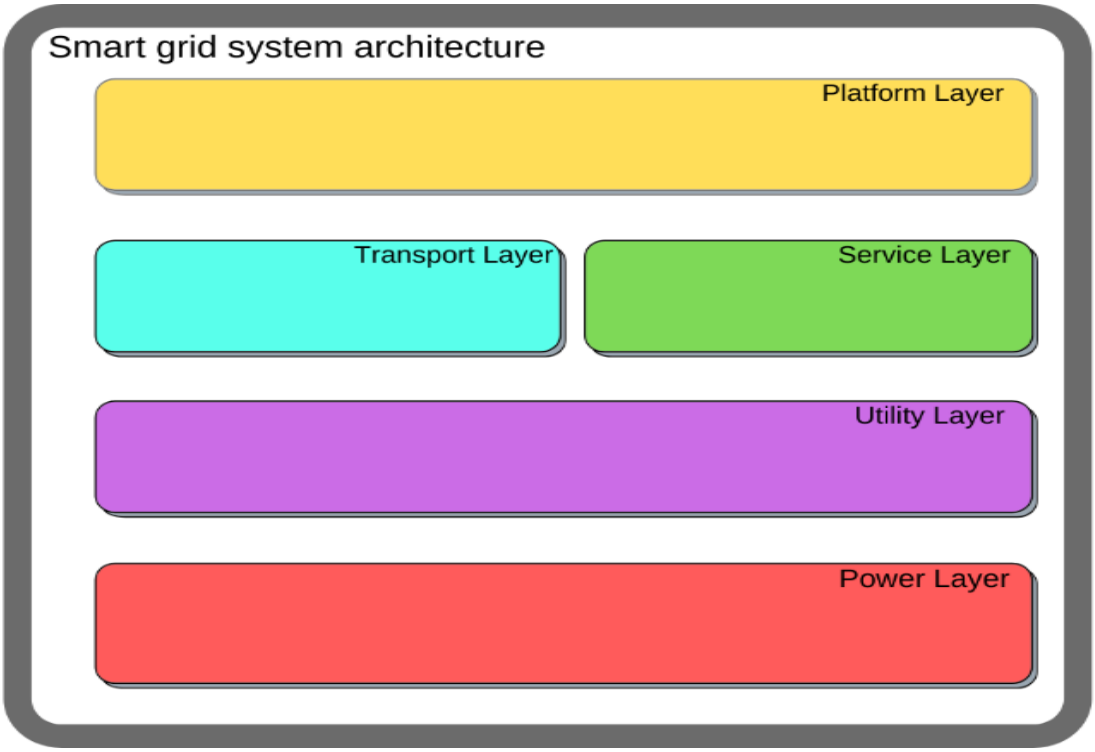
Requirement #: 9 (Performance Transparency)	**Requirement Type:** NFR(Quality)
Description: The system shall provide high performance in heavy load.	
Rationale: To make the system predictable and ensure high quality user experience.	
Fit Criterion: The performance should gracefully degrade as the load of the system increases.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: concurrency, reliability, scalability.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 10 (Migration Transparency)	**Requirement Type:** NFR(Quality)
Description: The objects and process of the system shall be moved without affecting operations or the results.	
Rationale: This is highly important for maintenance, reliability, and concurrency characteristics of the system.	
Fit Criterion: The objects and processes that interact with a moving object or a process they should not be able to detect that they are moved.	
Customer Satisfaction: 4	**Customer Dissatisfaction:** 4
Dependencies: scalability, maintainability, reliability, concurrency.	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

Requirement #: 11 (Maintainability)	**Requirement Type:** NFR(Quality)
Description: The system shall be maintainable in the long term.	
Rationale: When the system is designed maintainability in mind this reduced the costs of the maintenance and predictability for the system maintenance.	
Fit Criterion: The maintenance plans should fit in the criterion that will be specified later on.	
Customer Satisfaction: 5	**Customer Dissatisfaction:** 5
Dependencies: reliability, security,	**Conflicts:** None
Supporting Material History: Quality_Requirements.docx	

4. ARCHITECURE – AN EMBEDDED SOFTWARE VIEW

Green energy integrated smart grid architecture is visualized with electronics and software elements and features are realized by software algorithms that interface with grid sensors and actuators. The device drivers enabled with an operating system ensures the real time control and operation of the smart gird system. These components are placed in five layers of distributed software architecture:



4.1 LAYERS

4.11 Power Layer: The power layer will oversee the management of energy resources that will power the entire grid system as well as providing critical information to the utility layer regarding power plant specifications, energy types, and voltage rates.

4.12 Utility Layer: The utility layer will operate as a database storing the information, and a server transferring routing information, Backend service information, power plant specifications and the system condition to the transport layer.

4.13 Transport Layer: The transport layer will be handling routing, distribution, anomaly detection, supervise network status, blackout prevention, energy loss analyses, performance and power optimization.

4.14 Service Layer: The service layer will oversee the management of microservices such as consumption adjustment, peak tariff reduction, and authentication services via API gateway as well as perform service automation.

4.15 Platform Layer: The platform layer will specify and provide an environment in which to develop and run applications that will serve as gateways to the service layer. In addition, it will provide an application ecosystem to the developer community, allowing for flexibility, reusability, and integration of other systems into the smart grid system.

4.2 INTERACTIONS

4.21 Power-Utility Interaction: The interaction view from the standpoint of the power layer is one-way continuous data flow and functionality support from the utility layer. On the other hand, utility layer's role is storing continuous data and regulates the power layer component specifications.

4.22 Utility – Transport Interaction: The interaction between utility layer and transport layer consists of three types of data flow: continuous data flow, request/response style data flow, and event-based data flow.

4.23 Utility – Service Interaction: The service layer's microservices and the utility layer's database will be in continual interaction while delivering continuous data flow, event-based data flow, and request/response interaction.

4.24 Transport – Platform Interaction: One-way interaction from platform to transport layer is expected including listening and event-based notification types.

4.25 Service – Platform Interaction: The platform layer will send a request / API call to the service layer, which will respond with data / feedback based on the request.

4.3 COMPONENTS

Component Name	Functionality	Behavior	Layer
Electric Generator	Continuous data-flow and configuration logging	Notifies components that require the configuration files	Power Layer
Energy Storage	Log storage and supply rate information to database	Adjusts and logs energy distribution to specified values coming for events	Power Layer
Database	Stores all the information about the system	Handles parallel continuous data transfer	Utility Layer
PL Component Analyzer	Troubleshooting and feedback broadcasting	Periodically performing power generator equipment analysis and	Utility Layer

		broadcast unexpected results	
TL Component Analyzer	Troubleshooting and feedback broadcasting	Periodically performing transport equipment analysis and broadcast unexpected results	Utility Layer
Backend Service	API call response, microservice congestion control and collapse prevention	Responds API requests, and in parallel performs authentication checking and congestion control	Utility Layer
Backup Database	Critical Information Storage	Prevents data loss and periodically performs database performance and health checkup	Utility Layer
Intelligent Node	Route calculating, energy flow redirection, energy loss and blackout prevention, fluctuation detection	During an unexpected result performs rerouting and feedback generation	Transport Layer
Network Status	Intelligent Node management and current status logging	Periodically logs the network status and in parallel performs intelligent node functionality management.	Transport Layer
API Gateway	API handling	Accepts connection and performs gateway functionality	Service Layer
Microservice Authentication	Gives access to authentication specific microservices	Sequentially accepts request and sends conformation request and then accepts the user authenticity.	Service Layer
Scheduler	Cost analysis, device scheduling, daily cost metrics logging	After a scheduler event request received performs scheduling functionality. Periodically logs the information.	Service Layer

Microservice Support	Database query, user event logging, API requests	Listens API-gateway for events and performs requests base on authenticity	Service Layer
User Connection	Log user transactions, personal information and internal network details	Base on event calls performs logging information to the database	Platform Layer
User registration and authentication	Enables user access to the system	Waits for call events and sends API requests to the service layer	Platform Layer
Market	Handles marketplace API request and tracks transaction history	Enables advertisement and transaction based on event-calls	Platform Layer
User Status	Service activation and deactivation	Activates/deactivates service specific subscriptions/features.	Platform Layer

4.4 XCD MODELLING APPROACH

```
component Electric Generator {
    String configuration;
    int data;
    emmiter port data_flow {
        @functional {
            promises : /nothing;
            ensures: data := sensor_readings;
        }
        int Data_flow();
        @functional{
            promises: req_config := configuration.parameters();
            ensures : /nothing;
        }
        string send_configuration(req_config);
    }
 }
component Energy Storage {
    int stored_energy_amount;
    int avaliable_storage_amount;
    int ongoing_energy_supply_rate;
    emitter port log_data {
        @functional{
            promises: /nothing;
            ensures: stored_energy_amount := sensor_readings;
        }
        int log_ stored_energy_amount();
        @functional{
            promises: /nothing;
            ensures: avaliable_energy_amount := sensor_readings;
        }
        int log_avaliable_energy_amount();
        @functional{
            promises: /nothing;
            ensures: ongoing_energy_supply_rate:= sensor_readings;
        }
        int log_ ongoing_energy_supply_rate ();
    }
}
```

```
component Data Base {
    database power_plant_specifications;
    boolean plant_spec_query = false;

    database backend_service_information;
    boolean backend_query = false;

    database system_condition;
    boolean system_condition_query = false;

    provided port power_plant_port{
        @interaction {accepts: !plant_spec_query}
        @functional {
            requires : /nothing;
            ensures: result := power_plant_specifications;
        }
        database  get_plant_spec();

    }
    provided port backend_port{
        @interaction {accepts: ! backend_query }
        @functional {
            requires : /nothing;
            ensures: result := backend_service_information;
        }
        database  get_backened_service();
    }
    provided port condition_port{
        @interaction {accepts: ! system_condition_query}
        @functional {
            requires : /nothing;
            ensures: result := system_condition;
        }
        database  get_system_condition();
    }
    consumer port logport  {
        @functional{
            requires: requested_data := plant_spec_query  ;
            ensures: power_plant_specification := data;
            otherwise:
            requires: requested_data := backend_query  ;
            ensures: backend_service_information := data;
            otherwise:
            requires: requested_data := system_condition_query  ;
            ensures: system_condition := data;
        }
        bool log_data(bool requested_data, database data)
    }
}
```

```
component Power layer component Analyzer {
    database data;
    bool fault = false;
    analayse analysis_results;
    required port data_port {
        @functional {
            requires: query := true;
            ensures: data:= response;
        }
        void get_data(bool query, database response);
    }
    emitter port send_analysis{
        @interaction{accepts: fault == true;}
        @functional{
            promises: /nothing;
            ensures: result := analysis_results
        }
        analysis send_analysis();
    }
}
component Transport layer component Analyzed {
    database data;
    bool fault = false;
    analayse analysis_results;
    required port data_port {
        @functional {
            requires: query := true;
            ensures: data:= response;
        }
        void get_data(bool query, database response);
    }
    emitter port send_analysis{
        @interaction{accepts: fault == true;}
        @functional{
            promises: /nothing;
            ensures: result := analysis_results
        }
        analysis send_analysis();
    }
}
```

```
component Backend_Service {
    bool api_call = false;
    provided port API_response {
        @interaction {waits: !api_call}
        @functional {
            requires: /nothing;
            ensure: result = response();
        }
        database API_CALL();
    }
    required port authentication_port {
        requires: check_validity(authentication_key) == true;
        ensures: result:=  authenticate_database(authentication_key);
    }
    bool authenticate(int authentication_key);
}
component Intelligent_Node {
    database condition;
    vector route_info;
    bool route_change = false;
    emitter port route_port{
        @interaction{waits: !route_change;}
        @functional{
            requires: /nothing;
            ensures: result = route_info;
        }
        vector send_route_info();
    }
    required port read_condition{
        @interaction{waits: !periodic_scan();}
        @functional() {
            requires: /nothing
            ensure: condition := system_condition;
        }
        bool read_system_condition(database system_condition)
    }
}
```

```
component API_Gateway {
    string API_call;
    required port get_api  {
        @functional{
            requires: /nothing;
            ensures:  API_call = data;
        }
        boolean get_api_call(string data);
    }

    provided port send_api {
        @functional {
            requires: true;
            ensures: result = API_call;
        }
        string send_api_call();
    }
}
component MicroService {
    string API_call;
    required port get_api  {
        @functional{
            requires: /nothing;
            ensures:  API_call = data;
        }
        boolean get_api_call(string data);
    }
}

component user_connection {
    string data;
    provide port send_data {
        @functional{
            requires: /nothing;
            ensures:  result = data;
        }
        string send_user_data();
    }
}
```

```
connector databasetoPower_Analyzer (Power layer component Analyzer {data_port}, database{log_port}){
    role database {
        provided port_variable power_plant_port {
        get_plant_spec();
    }
    role Power layer component Analyzer {
        required port_variable data_port{
            get_data(bool query, database response);
    }
    connector async database2analyzed (Power layer component Analyzer {data_port}, database{log_port} );
}

connector databasetoTransport_Analyzer (Transport layer component Analyzed {data_port}, database{log_port}) {
    role database {
        provided port_variable power_plant_port {
            get_plant_spec();
        }
    }
    role Transport layer component Analyzed {
        required port_variable data_port {
            get_data(bool query, database response);
        }
    }
    connector async database2transport_analyzed (Transport layer component Analyzed {data_port}, database{log_port} );
}
connector databaseToBackend (Backend_Service { authentication_port}, database{ backend_port}) {
    role database {
        provided port_variable backend_port {
            get_backened_service();
        }
    }
    role Backend_Service {
        required port_variable authentication_port {
            authenticate(int authentication_key);}
        }
    connector async backend2database (Backend_Service{ authentication_port}, database{ backend_port});
    }
}
```

```
connector intelligent_node_todatabase (Intelligent_Node { route_port },  database{ log_port}) {
    role Intelligent_Node {
        emitter port_variable route_port {
            send_route_info();
        }
    }
    role database {
        consumer port_variable log_port{
            log_data(bool requested_data, database data) ;
        }
    }
    connector async intelligent2database (Intelligent_Node{route_port },  database{ log_port});
}
connector databasetointelligent_node (Intelligent_Node { read_condition },  database{ condition_port}) {
    role Intelligent_Node {
        required port_variable read_condition {
            read_system_condition(database system_condition);
        }
    }
    role database {
        provided port_variable condition_port {
            get_system_condition();
        }
    }
    connector async database2intelligent (Intelligent_Node{read_condition },  database{ condition_port});
}
connector api_getway_to_microservices (MicroService { get_api },  API_Gateway { send_api }) {
    role API_Gateway {
        provide port_variable send_api {
            send_api_call();
        }
    }
    role MicroService {
        required port_variable get_api {
            get_api_call(string data);
        }
    }
    connector async geteway2service (MicroService { get_api },  API_Gateway { send_api });
}
```

```
connector user_registerToapi_getaway (User_Registiration { send_api },  API_Gateway { send_api }) {
    role User_Registiration {
        provide port_variable send_api {
            send_api_call();
        }
    }
    role API_Gateway {
        require port_variable get_api {
            get_api_call(string data);
        }
    }
    connector async user_regis2gateway ( user_connection { send_data },  API_Gateway { send_api} );
}
```

```
connector generator_to_database (Elelctric_Generator{data_flow}, DataBase{log_port}){
    role Elelctric_Generator{
        emitter port_variable data_flow{
            send_configuration(string req_config);
        }

    }
    role DataBase{
        consumer port_variable log_port{
            log_data(bool requested_data, database data);
        }

    }

}
```

```
connector storage_to_database (Energy_Storage{log_data}, DataBase{log_port} ){
    role Energy_Storage{
        emitter port_variable log_data{
            log_stored_energy_amount() | log_availabe_energy_amount() | log_ongoing_energy_su

        }
    role DataBase{
        consumer port_variable log_port{
            log_data(bool requested_data, database data);
        }

    }
    }

}
```

```
component Smart_Grid_System(){
    component Elelctric_Generator electric_generator()[1];
    component Energy_Storage energy_storage()[1];
    component DataBase database();
    component Power_layer_component_Analyzer power_layer_analyzer();
    component Transport_Layer_Component_Analyzer transport_layer_analyzer();
    component Backend_Service backend();
    component Intelligent_Node intelligent_node()[1];
    component API_Gateway gateway();
    component MicroService microservice()[1];
    component user_connection connection();

    connector databasetoPower_Analyzer conn1(power_layer_analyzer{data_port}, database{log_port});
    connector databasetoTransport_Analyzer conn2(transport_layer_analyzer{data_port}, database{log_port});
    connector databaseToBackend conn3(backend{authentication_port}, database{log_port});
    connector intelligent_node_todatabase conn4[1](intelligent_node[@]{route_port}, database{log_port});
    connector databasetointelligent_node conn5[1](database{condition_port}, intelligent_node[@]{read_condition});
    connector api_gateway_to_microservices conn6[1] (gateway{send_api}, microservice[@]{get_api});
    connector user_registerToapi conn7(connection{send_api}, gateway{get_api});
    connector generator_to_database conn8(electric_generator[@]{data_flow},database{log_port});
    connector storage_to_database conn9(energy_storage[@]{log_data}, database{log_port});


}
component Smart_Grid_System config();
```

5. TRACEABILITY

5.1 FUNCTIONAL REQUIREMENTS

Our layered architecture enabled us to develop the smart grid system in a hierarchical way. By designing the system with four layers and five functional sections, we were able to separate the concerns for each functional requirement as shown in section ***4.3***.

- The platform player components satisfy Functional Requirements **3.11**, **3.12**, and **3.13**.
- Service layer components handle the functional requirements **3.110**, **3.112**, **3.113**, **3.115**, and **3.116**.
- Transport layer components handle requirements **3.15**, **3.16**, **3.17**, **3.18**, and **3.111**.
- The utility layer handles requirements **3.112**, **3.113**, and **3.114**.

And lastly, requirements **3.14** and **3.19** are handled by power layer.

As we can see, we were successful at incorporating our functional requirements into our design selections.

5.2 NON – FUNCTIONAL REQUIREMENTS

- ***Security***: Because our technology is intended to be distributed, security is critical. In certain ways, we were able to create a secure system. However, we cannot claim that the system is fully safe. To do this, we should add more components to improve overall system security.
- ***Reliability***: We developed the system as a layered stack to minimize coupling, and we included backup systems in our smart grid to achieve even lower coupling. We also have analyzer components to strengthen the system.
- ***Concurrency***: We can safely state that we achieved concurrency in our system and built it to manage large loads thanks to our design decisions.
- ***Openness***: We were able to achieve this need by employing a layered structure as well as interface abstracted components.
- ***Scalability, Replication Transparency, and Migration Transparency***: We primarily met this need by isolating each worry from one other, which resulted in abstract components. We also introduced reliability analysis components to make scaling the system even easier. And, with the aid of transparency, we were able to establish replication and migration transparency.
- ***Access and Location Transparency***: We achieved access and location transparency through standardized interfaces and robust database design. Layered architecture was also very helpful in achieving this goal because it abstracted lower layers from users, so they don't need to know where the object is or how to access it.

- ***Performance Transparency***: With the aid of our load and equipment analyzer components, we obtained a minimal degree of performance degradation even under large loads. In this manner, we are adjusting our system to match the intended performance.
- ***Maintainability***: The layered design was extremely beneficial in achieving the system's strong maintainability. And, by utilizing periodic analytic tools and backup systems, we were able to achieve predicted on-point system maintainability.

6. CONCLUSION

Smart grid architecture is a complicated and data-intensive system. The basic framework of smart grid architecture is determined by data gathering and interchange among consumers and distribution. In a smart grid setting, the bit-level protocol definition enables rapid, dependable, and secure communication. Cloud integration approaches might also allow for the incorporation of several technologies such as automotive and home automation into the smart grid sector.

This report focuses on how to build and improve the performance and efficiency of the smart grid system. There is still considerable work to be done in the future to increase the system's efficiency. Because of rising energy consumption and the greater use of renewable resources, energy-efficient smart grid devices must be designed on smart software architecture. In other words, while creating a security framework for the entire Grid is difficult and challenging, the end result will be worthwhile.