University of Groningen

Software Engineering

Requirements Document

GreenerSimulation

Team:
Chris Worthington
Sjoerd Hilhorst
Victor-Cristian Florea
Duncan Saunders
Mariya Shumska

Client:
Jim van Oosten
Jasper Clarijs
(Greener Power Solutions)

May 4, 2020

1 Introduction

Greener Power Solutions provides sustainable energy for the temporary energy market, by delivering large-scale batteries, energy forecasting and monitoring software. Greener software controls actual hardware, based on the data provided by the batteries themselves. The Greener software (called GreenerEye) acts as a client that reads the batteries values, and decides on that data which task to perform.

Testing a new software release is done on a physical battery that gets energy from various power sources and delivers it to the real loads. However it is time consuming and errors might lead to costly and even dangerous situations.

The goal of the project is to set up simulations that model the batteries, the power sources they are connected with, and the loads that they deliver power to in order to speed up testing and make it more environmentally friendly and repeatable.

2 Stakeholder(s)

For this project, we identified only one stakeholder as he is the only one who interacts with the software

- **Client:** wants a software product that mimics a battery either by simulation or historic data in order to test his existing software more easily.
  The delivered software has to flexible and modular, it should allow electrical engineers to configure formulas according to their needs.

The client is familiar with programming, therefore some of the requirements might use low level of language and computer science terms (section 4.2).

3 Critical functional requirements

3.1 Implement the Battery model

- As a User I want to have a model of the Battery in order to test my existing software
- As a GreenerEye I want to monitor the state of the Battery in order to decide what next to do

In order to run the tests without physical equipment, the virtual model of the battery is required. It should contain all necessary fields of its current parameters like State of Charge (SoC), mode, status, access to power sources and electrical loads, etc. Moreover, the 'digital' battery should update its state according to the simulated input using appropriate mathematical relations.

3.2 Model energy input with historic data

- As a User I want to fill the battery with historic data in order to test my existing software

In order to test the model battery you need to provide test data into the battery. One way to do this is with historic data. The client has data files of historical real world scenarios. The user should be able to load the data file into the program and perform tests with it.

3.3 Model energy input with simulated data

- As a User I want to fill the battery with simulated data in order to test my existing software

Another option to test the model battery is to perform simulations. By doing simulations rather than historic data new environments can be modeled, this ensures more extensive testing and makes it possible to model upcoming events. The user should be able to specify parameters on which the simulation should be run. The program then executes the simulation and feeds the data to the battery.

4 Important functional requirements

4.1 Mapping battery registers

- As a User I want to be able to map the registers in order to unify the software so it can be used regardless of the battery provider

Greener power solutions relies on batteries from several different providers. While every provider lets their battery communicate via modbus, the storage location or even which fields are available are not unified. The user should therefore be able to easily define register mappings in order to test different batteries.

4.2 Float storing functionality

- As a User I want to be able to choose the option to store floats, such that I can test batteries from different providers

Modbus registers define themselves how to store floats, different battery providers implement different methods. In general, there are 2 ways:

1. Storing floats with a scaling factor. The battery provider multiplies every float with a specified scaling factor and rounds it to an integer. When retrieving the value it is divided by the scaling factor again to obtain the original value

2. Storing floats in 2 registers. This means splitting up the float in an integer part and a decimal part. The user should be able to define which storing option he wants to use to test different batteries.

4.3 Interaction via GUI

- As a User I want to interact with the software via GUI in order to simplify manual testing

Greener Power Solution would appreciate the possibility to interact with the software via GUI, since it is the most user-friendly option and is the easiest to use while manual testing.

5 Useful Functional Requirements

5.1 Write battery state to a file

- As a User I want to save the Battery state for documentation

The battery updates it's state regularly, having the ability to go back and see what changes happened where is important. The program should therefore a generate an output file where all states are chronologically documented.

5.2 Define update speed of simulation

- As a User I want to be able to speed up the simulation such that CPU time usage is decreased

The client has expressed that he wants to use the software for manual testing and for automated testing. When testing manually, real time updating will suffice, but for automated testing speeding the client wants to speed up the tests in order to decrease CPU time. the User should be able to define the update speed of the simulations in order to test in realtime or speed up tests.

6 Non-Functional Requirements

6.1 Implement the battery storage with modbus registers

The batteries our client uses make use of modbus server to store the batteries data. In order to make our models directly compatible with the software of Greener Power Solutions, the data needs to be stored in a modbus server, where the client's software can directly retrieve it, as if it were an actual physical battery.

6.2 Modularity

The software should be flexible and modular. Configuration files should be easily understandable by the electrical engineers: after a total of two hours training they will be able to fully use the program.

7 Won't Do

At this point in time there is no requirements we have decided we are definitely not going to implement. There may be some we decide are not necessary or too difficult in the future.

8 Meeting Log

8.1 21st February - Video meeting

In the first meeting the client provided a general explanation of the project. Greener power solutions have software that control actual hardware, errors in the software could be dangerous. To test the software the client wants us to set up simulations that model batteries, connected power

sources and loads.

Various inputs like state of charge, input power and output power should be modeled. These variables should be stored on a modbus server as that's where the the data of physical batteries are stored and the clients software communicates with.

Overview:

1. Greener software controls hardware, errors can be costly, maybe dangerous
2. Software needs testing before production
3. Setup simulations that model
    - Batteries
    - Connected power sources
    - Connected Loads
4. Mimic batteries, diesel gen sets, grid connections by implementing modbus servers.
5. Battery state of charge, gen set, diesel consumption and varying power demand could all be modelled on these servers
6. Recommended software
    - Gitlab.com for VCS
    - Python 3.5
    - Sqlite 3 for databases

Information from Alex

1. Sprint Start - Monday
2. Translate clients information into wishes
3. Use task tracking software

Client Meeting

1. Batteries power source and load connected simultaneously.
2. Inputs:
    - battery has capacity
    - calculate state of charge percent based on hours
    - Using kW and kW/h
    - in $\rightarrow$ capacity
3. I/O Load Defined Profiles
4. GUI is up to us
5. Data can be retrieved from database, generated with maths (sin function plus noise with gaussian).
6. Output: Read registers, collect data, during simulation via the modbus protocol, modbus protocol has a master slave setup, server insides battery needs to be virtual.

8.2 24th February - physical meeting

In this meeting the client expanded on the general explanation and gave us clear guidelines on the details of the project. Modeling of the battery can be done in 3 ways:

- Random data: Fill registers with a random value
- Historic data: Fill registers with values from historic data, however you cannot model new situations with this.
- Simulations: simulate real world appliances that draw power and give power

1. Batteries:
   (a) Limited Capacity
   (b) Used with other power sources
   (c) Controlled by supplier software
   (d) Needs to follow same electrical field (a/c) as generated by power
   (e) 3 phase cable
   (f) GreenerEye, a small pc internetted for automation
2. Power Source - optimally a grid connection, or a genset. Batteries optimise how much diesel is used.
3. GreenerEye - Remote monitoring and control, automation, independent of battery supplier.
4. Modbus Protocol
   (a) Access's registries
   (b) Each register has an address
   (c) Size of register depends on addresses, 1 bit, 8 bit, 16 bits etc...
   (d) Registry definition determines data type, UINT, SINT, FLOAT, etc...
   (e) Can be read only or read write.
   (f) Server hosts registry and updates all registers
   (g) Client sends requests to the slave.
   (h) Can be big or little endian
5. Filling the Registry
   (a) Fill each register with random data, according to the registers data type
      - Easiest
      - Least Similar to reality
   (b) Fill each register with historical data
      - Load data from JSON/CSV.
      - Can never simulate new situations.

(c) Make simulations

- Hardest
- Simulate power source, electrical load and battery.

6. Questions/Answers

(a) Protocols: Modbus over TCP.

(b) Python prefered, Gitlab prefered.

(c) Docker would be nice

(d) Turn on genset by writing boolean true to specific modbus register

8.3 2nd April - Call

1. (important) addresses in JSON need to be changed to a tuple to split header and address number to distinguish e.g. register 3, 10 from register 310.
2. (future) Implement assignment of constants, inital values and state of charge in JSON configuration file for simulation.
3. (future) Implement ability to add new fields using JSON configuration file. e.g. more active_power_in fields for more different power inputs.
4. (future) Eventually make possible to make changes in command line and then GUI while running.
5. (future and optional) Implement client side UNIT tests to check the simulation works properly

8.4 22nd April - Call

During the call the client showed the presentation with explanation about actual physical processes. Some important points:

1. If the input contactor (`input_connected`) is open, power cannot flow through the input:
   - (re)active power in must be 0
   - input currents must be 0
   - voltages can be non zero
2. If the ESI (`converter_started`) is not turned on:
   - the SoC of the battery cannot change
   - (re)active power converter must be 0

Some more wishes:

1. When starting the battery simulation, a user should select the powers, voltages, and currents that should be fed to the simulation (e.g. from historical data).

2. The simulation should be able to react to events:

    - When a user turns off the ESI (via de modbus protocol), the (re)active power converter should be 0.00, and the SoC should no longer change.

Main message:
There is no need to develop our own mathematical simulation model. Instead we should make a software that is easy for electric engineers to use, they will deal with the tricky formulas

9 Change Log

Date	**Comment**
25.03.2020	Mariya: Useful functional requirement (5.2) was added
26.03.2020	Sjoerd: General updating and refactoring of all sections
27.03.2020	Mariya: Important functional requirement (4.3) was added
28.03.2020	Mariya: Critical functional requirement (3.1) was modified; Non-functional requirement (6.1) was modified
02.04.2020	Mariya: Critical functional requirement (3.2) update
02.04.2020	Chris: Requirements update call (8.3) was added
28.04.2020	Mariya: Update of sections (2), (3); (8.4), (6.2) added
04.05.2020	Mariya: Update of 'Stakeholders' section (2)