A GENERIC FRAMEWORK FOR COMMUNICATION OF DISTRIBUTED ENERGY RESOURCES THROUGH A CLOUD-BASED SERVICE

Lasse Dreisig Orda, s040884
Jesper Bach, s072458

July 2013

Informatics and Mathematical Modelling
Technical University of Denmark
IMM-M.Sc.-2013-72

A generic framework for communication of distributed energy resources through a cloud-based service

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-M.Sc.-2013-72

Summary

Power production has traditionally been very centralized, with large fossil fired combined heat and power plants carrying the vast majority of the load. In recent years, however, the interest in renewable energy has spiked and more and more dispersed energy resources (DER) are starting to appear. In Denmark the government has set an ambitious goal of 50% wind power penetration by 2020 [16], and a complete out-phasing of fossil fuels by 2050.

Because the power grid has to be kept in balance, a multitude of new control schemes are being researched in order to properly manage the new resources [62]. From a network-technical perspective, handling connections to a multitude of units constitutes a problem in itself. Not only does it require a significant amount of server capacity, but obtaining access to all DERs through firewalls and the like, is a rather daunting task.

By shifting these more tedious tasks to the cloud, the aggregator not only takes a great step towards true scalability, but also towards increased security. In connecting directly to the cloud, the aggregator no longer needs to keep sockets open for connecting DERs, effectively distancing the critical control algorithms from the outside world, leaving this to the cloud.

This thesis covers the development of a cloud-based framework, for receiving incoming WebSocket connections from DERs allowing for near real-time communication. To handle multiple DERs raw data, a generic module for each DER unit, allows for various data models to be mapped into a generic one, based on the well tested IEC 61850 standard. Using a cloud-based solution greatly aids the aggregators to scale and meet future demands.

The thesis also includes a concept of flexibility provided by iPower-net. The Flexibility Interface is currently being researched by iPower, and is mapped into the IEC 61850 standard as an additional logical sub-node. By mapping to the existing standards, no major changes would be needed to adapt existing systems.

Resume

Energi produktionen har traditionelt set været meget centraliseret, med store fossilbaseret varme- og kraft-værker, som bar størstedelen af belastningen på el-nettet. I de seneste år er interessen for vedvarende energi, specielt “Distribuerede Energi Ressourcer” (DER), begyndt at dukke mere og mere op. I Danmark har regeringen sat et ambitiøst mål om at 50% af alt energi skal være baseret på vindkraft inden 2020 [16], og en komplet udfasning af de fossile brændstoffer skal ske inden 2050.

Da el-nettet gerne skulle holdes i balance, bliver der konstant forsket i mange forskellige kontrolanordninger, for at hjælpe med styringen af de nye distribuerede ressourcer [62]. Ud fra et netværks-teknisk perspektiv, så er selv håndteringen af flere enheder et problem. Det kræver nemlig en betydelig mængde server ressourcer at håndtere den store mængde data og det at åbne en forbindelse til en DER enhed, igennem firewalls og lignende, kan være en besværlig opgave.

Ved at flytte en række af disse opgaver til skyen, så tager man ikke blot et stort skidt i at gøre aggregatoren mere skalerbar, men man øger også sikkerheden. Ved at tilslutte direkte til skyen, så er det ikke længere aggregatoren, som er nødt til at holde en åben forbindelse til de enkelte DERs. Derved distancerer man kritiske kontrol-algoritmer fra resten af version.

Denne afhandling omhandler udviklingen af et sky-baseret framework, som skal håndtere indgående WebSocket forbindelser fra DERs. For at håndtere de forskellige typer rå data som hver DER kan transmittere, så er der blevet udviklet et generisk modul for hver DER enhede, som konverterer DER data til en generisk data struktur, baseret på IEC 61850 standarden.

Afhandlingen indeholder også et koncept for fleksibilitet, leveret af iPower-net. Det såkaldte Flexibility Interface er stadig under udvikling af iPower og er brugt her, ved at bliver lagt ind i IEC 61850 standarden som en ekstra logisk node. Ved at indføre fleksibilitets konceptet til en eksisterende standard, så kræver det ikke væsentlige ændringer for at få eksisterende systemer til at indordne sig i det nye koncept.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modeling at the Technical University of Denmark in fulfillment of the requirements of acquiring an M.Sc. degree in Computer Science and Engineering.

This M.Sc. Thesis is written by

Jesper Bach
s072458

Bach

Date: July 15, 2013

M.Sc-student
Technical University of Denmark

Lasse Dreisig Orda
s040884

[illegible]

Date: July 15, 2013

M.Sc-student
Technical University of Denmark

Thesis advisors:

Bjarne Poulsen
Associate Professor, Ph.d.
DTU Compute
Technical University of Denmark

Lars Henrik Hansen
Senior Engineer, Ph.d.
DONG Energy

Anders Bro Pedersen
Ph.d.
DTU Electrical Engineering
Technical University of Denmark

Acknowledgement

We would like to thank our advisor **Bjarne Poulsen**, Associate Professor, Ph.d, DTU Compute, Technical University of Denmark. Bjarne has consistently kept our motivation high and encouraged us to put the best work possible into this thesis. He has provided us with an endless stream of relevant materials, that have brought the quality of the project to a higher level.

Anders Bro Pedersen, Ph.d, DTU Electrical Engineering, Technical University of Denmark. As an advisor for our project, Anders has continuously provided us with excellent points about the design and implementation of the prototype. He has been exceptionally helpful whenever we have had any questions, needed a helping hand with practicalities or his opinion on any written text.

Lars Henrik Hansen, Senior Engineer, Ph.d, DONG Energy. Lars has in the role as an external advisor, from DONG Energy and the iPower consortium, provided us with a thesis subject, that is actively being researched and requires an innovative approach using modern technologies. Lars' impressive ability to keep the big picture and connect all the people from different areas, have introduced us to multiple people and wide ranging perspectives on the subject of our thesis.

Arne Hedevang, Senior Energy System Architect, DONG Energy. Arne has provided us with great inspiration to the design of the networking and communication for the project prototype. Arne and DONG Energy has additionally and generously provided us with an embedded micro-controller, which has formed the base for the new DER Controller concept, that is introduced with this thesis.

Chresten Træholt, Associate Professor, Ph.d, CET/DTU Electrical Engineering. Chresten and DTU PowerLab.dk have provided us with access to a real world distributed energy resource, in the form of a combined and heat power generator from Senertec.

Carsten Voldstedlund Thuesen, Software Engineer, Dantherm Power; **Mads Møller Melchiors**, Project Manager, Dantherm Power and **Dantherm Power A/S** have kindly provided us with 3 real world Distributed Energy Resources, Dantherm DKμCHPs which have been used as a basis to develop the prototype in this thesis.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

The following chapter will describe the motivation, project description and the scope behind the project. The description contains an elaboration about the project, the stakeholders and the goals for the project. Lastly the scope of the project will describe the boundaries of the contents of what the thesis will cover.

1.1 Motivation

The wish to obtain sustainable energy has been present for many generations. Even though the electricity consumption in Denmark has decreased over the last years [28], the International Energy Agency predicts [48] that there will be a 50% increase in world energy consumption, by the year 2030. With the rapid increase in demand, terms like “Green Energy” or “Renewable Energy” are becoming more popular. Doing research within the energy industry, particularly within the green energy segment, is therefore highly motivational.

Being able to contribute to ongoing research and development is, by itself, a driving factor. This thesis will contribute within the area of the Danish development of the Smart Grid, and the findings in the prototype might be a part of the final products that are shipped out to the consumers. With the prevalence and increase of distributed energy resources in the common household, this project will not only make an impact on software that will run in the industry, but also positively affect the usability for the consumers, which increases the motivation for doing the thesis.

The cloud platform is becoming very popular within the software industry [27]. The idea of scaling on-demand, allows the underlying servers to dynamically adjust to the workload, and only use “what is needed” for the processes. Developing a solution for the cloud, is therefore a step towards green energy and being able to utilize modern technology, will also keep the motivation high throughout the thesis.

1.2 Project Vision

The vision of this project is to create a proof-of-concept prototype showing, how multiple distributed energy resources can send information data in real-time up into the cloud. The cloud then processes this raw data and generates an information model based on the IEC 61850 data structure. The cloud should now provide an easy access to each distributed energy resource using the IEC 61850 standard. Additionally, the vision is to integrate the iPower Flexibility Interface into the standard, as additional nodes in the IEC 61850-7-420 model.

1.3 Project Description

The number of Distributed Energy Resources (DER) is growing as farmers are installing windmills in their fields [68][34], electric vehicles are gaining on the consumer market [67] and photovoltaic systems are becoming more popular on rooftops [36], due to the investments of photovoltaic systems in the last year [17] being subsidized by the government. With the increase in numbers and their geographic location being spread out, they can become a positive part of the power grid as ancillary services and when multiple DERs are aggregated, they can take part in buying and selling power on the energy markets. A Virtual Power Plant (VPP) is used to control and coordinate a group of DERs and the communication is mapped through information standards, such as the IEC 61850.

Currently the communication model, as illustrated in figure 1.1, is based on the VPP opening the connection to the individual DERs. This approach is suitable when dealing with e.g. centralized power plants and well known networks through proprietary dedicated communication lines. However, as the DERs are placed in many different networks, e.g. at a local farmers house, it becomes difficult to use the current point-to-point communication model. Where the VPP server connects to the DER unit, because of routers, firewalls, ISPs and proxies that may block the communication. End-users often have dynamic IP addresses, which makes it difficult to find the correct IP addresses, to communicate with the users equipment and the addresses may change over time. These are issues that are outside the control of the organization that runs the VPP. Using a cloud-based service, makes it possible for the DERs information to cross network boundaries in a way, which is well supported by today's internet structure.

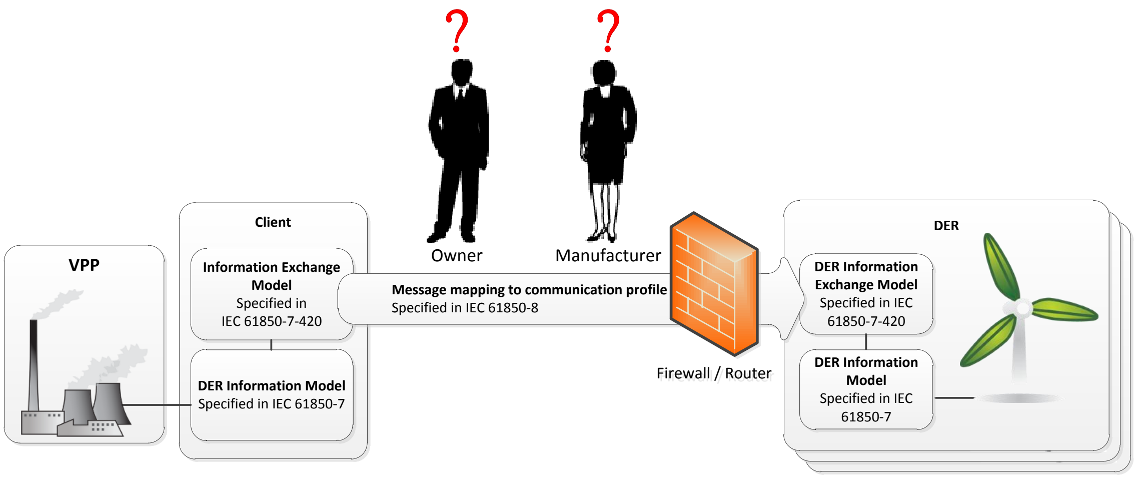


Figure 1.1: The current communication model.

A VPPs responsibility is to aggregate and control a group of DERs and to collect information about the DERs in the group. Different parts of this information can be of interest for e.g. manufacturers and owners. Manufacturers may have hardware on end-users locations that they own or are responsible for. They can then use this information, to be notified about problems in the equipment and to find out what the cause of the problem might be. Other parts of the information can be of interest to the owner, such as energy production, logs, energy overview, controlling the unit etc. In order to access the information, they have to either collect the information themselves, or the VPP has to facilitate access to its information. To collect the information individually, requires communicating with the DER, which includes the same network communication problems, that are present for the VPP.

Solving these issues, using the cloud, see figure 1.2, provides a generic and scalable solution. The cloud makes it possible to handle the communication issues between the VPP and the DERs using modern protocols and technologies. Information could be available both as online web access, which scales perfectly with the could, and data APIs, that can be used by manufacturers to integrate into their systems.

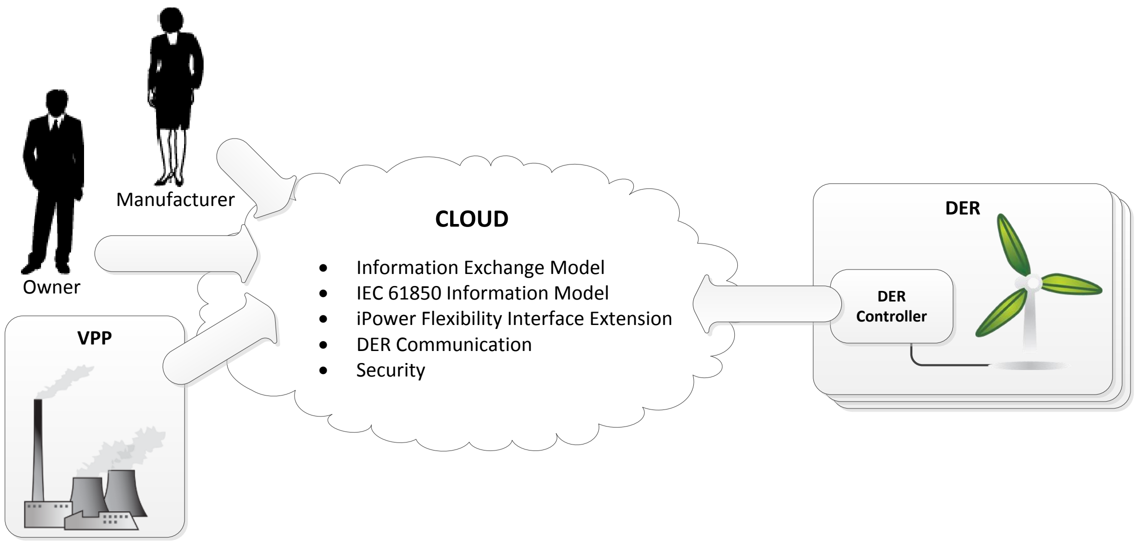

Manufacturer
Owner
VPP
CLOUD
Information Exchange Model
IEC 61850 Information Model
iPower Flexibility Interface Extension
DER Communication
Security
DER
DER Controller

Figure 1.2: The proposed domain.

DERs and the VPP are built by different manufacturers and communicates together using the IEC 61850 standard. Currently each DER controller has to contain the full IEC 61850 server logic with an information model, reporting and logging facilities, the related information exchange model and services. For some DER units, this does not present a problem, outside of the initial cost of developing the standard compliant software, but other units such as embedded PLC[1] systems, may not have the facilities to incorporate necessary IEC 61850 services.

This thesis proposes to relocate most of the IEC 61850 server logic into the cloud. This way the DERs only needs to send raw data to the cloud, which then parses and handles the information according to the standard information model and services. The cloud then acts as a scalable, generic and standards compliant aggregator that collects data from DERs and provides it to external parties such as VPPs, owner and manufacturers. The cloud can support a wide range of DERs of different complexity. DERs may only be able to stream data in a specific format, they may need to be polled or they may support a full IEC 61850 standard etc. By having a large part of the IEC 61850 logic in the cloud, it is possible for manufacturers to build less complex and more cost-effective DERs, which will bring more DER options to the market and further encourage the deployment of DERs.

The DER Information Model structure, described in IEC 61850 [44], is the basis for the generic information model, that are used in the cloud-based service. The iPower Consortium is actively researching a specification for a service and management oriented information model, called the

[1]Programmable Logic Controller

iPower Flexibility Interface [7]. Using the same principles, CENELEC [11] has also defined a flexibility concept, that can be used when communicating with DERs.

For this thesis the iPower Flexibility Interface will be mapped into the IEC 61850-7-420 [46] DER standard structure (as it is a subset of the IEC 61850 standard) as an extension. The information model is going to be generic, in that it can facilitate the IEC 61850-7-420 information model, the extended Flexibility Interface version as well as the basic communication structure IEC 61850-7-4 information model [45].

1.3.1 Stakeholders

During the start-up phase of this project, collaboration with multiple people interested in our project, helped shape the vision of the project. In order to establish a common ground for the project, defining each stakeholder gave a clear definition of each stakeholder's point of view. Each stakeholder has been selected with the purpose, that they have some influence or exposure on the project. In table 1.1 a brief overview of the stakeholders is shown, where each stakeholder is rated according to their importance of this thesis.

The tables of stakeholders consists of columns as follows:

Stakeholder	Name of the stakeholder
Type	Internal or External stakeholder
Exposure	How much the stakeholder will be affected by the project
Power	The amount of influence the stakeholder has on the project and its success
Urgency	How long before the stakeholder's power will have an effect on the project
Importance	How important the stakeholder are for the overall project

Stakeholder	Type	Exposure	Power	Urgency	Importance
Supervisors	Internal	●	●●●	●●●	●●●
iPower-net.dk	Internal	●●●	●●	●●	●●●
Dong Energy	External	●●●	●●	●	●●
Energinet.dk	External	●●●	●●	●	●●
Danish Consumers	External	●●	●	●	●

Table 1.1: Brief overview of the stakeholders for the thesis

To clarify each stakeholder, a more detailed description will be presented in the following.

1.3.1.1 *Supervisors*

As the supervisors are guiding the thesis, supplying materials and are a part of the evaluation, they will be considered an Internal type. They will not be affected by the result of the thesis and therefore the exposure set to low. On the other hand, they have a lot of influence, as they are a part of the evaluation and they are guiding the thesis. On that ground, both the power and urgency of their decisions is set to high. The final importance of the stakeholder are evaluated as high, due to their position and their impact on the thesis.

1.3.1.2 iPower-net.dk

iPower is contributing with a part of the foundation for the thesis with their Flexibility Interface [7]. Therefore as the thesis will build on this, the iPower as a stakeholder will be considered as Internal. Also because of this, their exposure, as they will be able to use the results of the thesis in their production environment and as a part of the Work Package 4 (WP4), will be high. As the Flexibility Interface is only a part of the thesis, their power on how it will succeed is set to medium. The same argument can be used for the urgency. The importance is then, due to the high amount of exposure and the medium of the power and urgency, set to high.

1.3.1.3 Dong Energy

The PowerHub is Dong Energy's concept of implementing a VPP. The PowerHub communicates with the DERs through the OPC-UA protocol [72]. Being able to communicate with their test environment makes Dong Energy an external stakeholder for the thesis. This also set them as a high exposure, because they will have direct communication with the project prototype. Being able to communicate with their PowerHub, which is fully functional, will also bring the thesis to a higher level and therefore the power of the stakeholder, is set to medium. As the stakeholder only has some influence on the thesis result thus, is not the same as saying their decisions will have any immediate effect on the thesis, so the urgency will only be set to a minimum. With the high exposure, medium power and low urgency, the importance of the stakeholder will only result in medium, as they are not critical for the thesis to succeed.

1.3.1.4 Energinet.dk

Energinet.dk and Dansk Energi have both made contributions, to map out the exciting work of intelligent power systems [21] and for the smart grid [20] in Denmark. As both companies have a lot of common ground, they will also be mentioned as a shared stakeholder. Energinet.dk are operating by directly connecting to each DER or through Dong Energys Power Hub. As the thesis will allow the stakeholder to connect to each DER through the cloud, they are considered as an External stakeholder. The same argument can be used when the exposure is determined and it is therefore set to high. Having multiple stakeholders interact with the project prototype of the thesis, means that Energinet.dk have some derived influence and is therefore noted as medium power. A change in Energinet.dk's decisions will not have an immediate effect on the thesis, so the urgency for this stakeholder will be set to minimum. The importance is evaluated to be medium, as they are not critical for the thesis.

1.3.1.5 Danish Consumers

The Danish consumers are set as an stakeholder, because they will be directly interacting with the DERs. They do not have any real control over the internal structure or have anything to say about the communication between the VPP and the DER, therefore they are noted as External. As the owner may need to restart the DER, connect it to a grid location or configure network equipment, their exposure it set to a medium level. They do not have any influence over, and are not directly included in, how this thesis will be written, and as such the power and urgency are set to low. The importance is also set to low, because even though they might be somewhat exposed, it is not enough to make them important for this thesis.

1.3.2 Goals

To clarify the purpose of this thesis, the overall top-goals is to ”Communicate to a DER through a cloud-based solution” as shown in figure 1.3. To achieve this goal, four sub-goals needs to be accomplished.

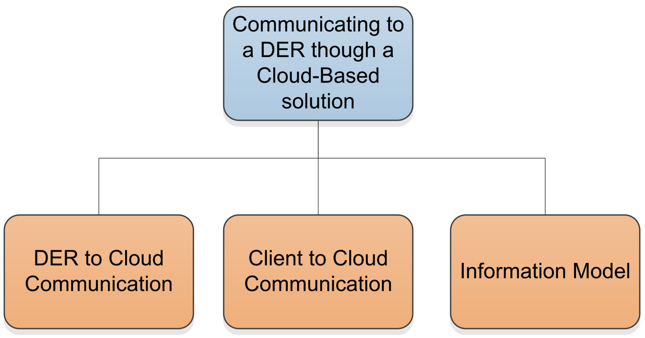

Communicating to a DER though a Cloud-Based solution
DER to Cloud Communication
Client to Cloud Communication
Information Model

Figure 1.3: Top goals of the thesis.

The first sub-goal is to establish communication between a DER and the cloud (see section 1.3.2.1 on page 6). The next sub-goal is regarding the ’other side of the cloud’, more precisely the communication between a client and the cloud (see section 1.3.2.2 on page 7). With the communication related goals defined, the final sub-goals concerns the internal structure of the cloud, which is regarding the information model (see section 1.3.2.3 on page 7) that contains and communicates the information data.

1.3.2.1 DER to Cloud Communication

The goal of establishing communication from DER to the cloud has, by itself, two sub-goals, see figure 1.4.

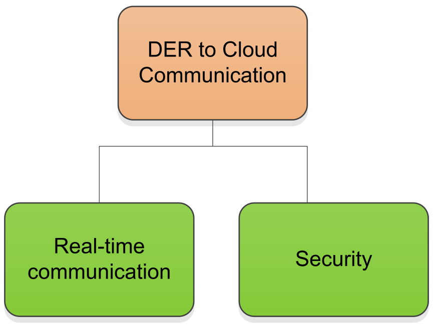


Figure 1.4: Sub-goals for communication between DER and the cloud.

When data are sent from a DER to the cloud, one goal is to do it using real-time communication. This is because, with the use of real-time communication, the exchange of data will be instant or with negligible latency. If an error occur and the operator/VPP needs to e.g. turn off the DER, time can be a valuable factor. Another sub-goal is to establish security in communication between the DER unit and the cloud.

1.3.2.2 *Client to Cloud Communication*

In the goal of establishing communication between a client and the cloud, there are four sub-goals, see figure 1.5.

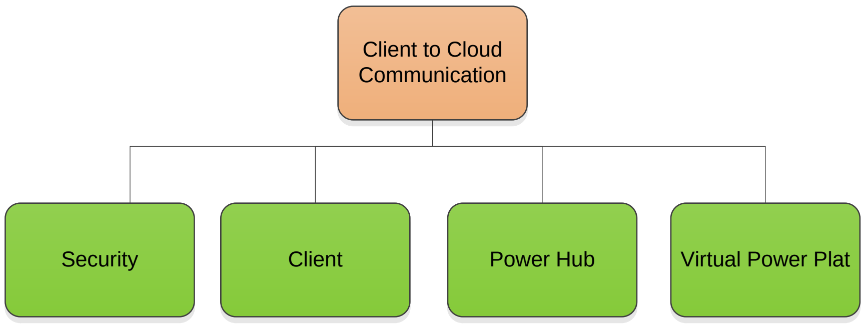


Figure 1.5: Sub-goals for communication between client and the cloud.

As with the communication between the DER and the cloud, establishing security is also a sub-goal for the client to cloud communication. Multiple clients should be able to connect to the cloud. This includes clients such as unit owners, the PowerHub and a Virtual Power Plant.

1.3.2.3 *Information Model*

Building the internal information model also requires additional sub-goals, see figure 1.6.

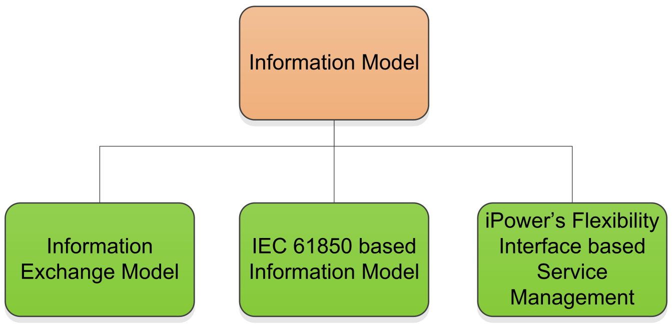

Information Model
Information Exchange Model
IEC 61850 based Information Model
iPower's Flexibility Interface based Service Management

Figure 1.6: Sub-goals for the information model.

One sub-goal is to create an information model, that is based on the IEC 61850 data structures. The information model-goal includes a sub-goal regarding data exchange. A data exchange model is used to communicate the data in the information model. In the communication to each DER, a service management node should be created based on the Flexibility Interface from iPower.

1.4 Project Scope

The focus of this thesis is two-part, the first focus area is on the communication between Distributed Energy Resources (DER) and a Virtual Power Plant (VPP). The second focus area is on how the iPower Flexibility Inferface model can be integrated and used with the IEC 61850-7-420 information model. To explore these two subjects, a prototype will be developed that will facilitate communication with different types of DER units and VPP/aggregator, users and DER operators.

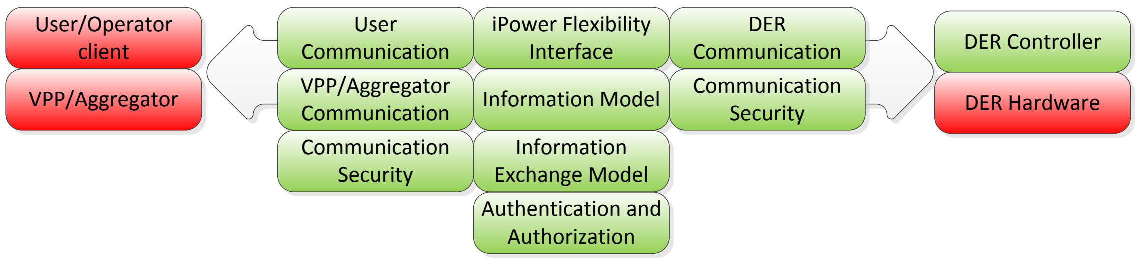


Figure 1.7: Project scope.

As illustrated in figure 1.7, the development of client applications is outside the scope of the project, but the communication interfaces used for programmatic access by the user applications is within the scope. Also a Website that provides some of the functionality expected from an application client along with additional functionality is within the scope. Developing or configuration of an existing VPP/aggregator is outside the scope but as with the application clients, the communication interface for the communication between the VPP/aggregator and the prototype is part of the scope. The communication for these parts will be encrypted and so the interface communication security is within the scope of this project.

This thesis is made in collaboration with the iPower consortium, which is actively researching a method for managing the flexibility of DERs through direct control. The Flexibility Interface and the related IEC 61850 standard (IEC 61850 information model and information exchange model) are important parts of the scope.

Having several external DERs communicate with the cloud will be a part of the project. Handling the DER hardware and I/O is, by itself, outside of the scope, but using multiple DER controllers to send raw data to the cloud is included. When it comes to sending data from the DERs to the cloud, DER units will be authenticated and authorized, but encryption is not applied to the communication. The security model in the project entails authentication and authorization of users which will be part of the scope.

1.5 Software Development Methodology

As the software development methodology acts as a framework for the development of a software product, it is important to have an appropriate methodology that fits. Following an iterative and incremental development approach, is a way to compensate for some of the shortcomings of the traditional waterfall methodology. The iterative method cycles repeatedly through a design, a build and a test phase, until the full functionality of the prototype as been achieved. The incremental model adds functionality to the implementation for each increment of the prototype and the result of each iteration is a working constituent part of the prototype.

This methodology is supported by the Spiral Methodology and Unified Process. The Spiral Methodology, was developed in 1988 and combines the "traditional" waterfall model with prototyping and aims for large projects, where risk analysis is a very important concern. Unified Process is an iterative, risk and use-case driven development model that consists of four phases, see figure figure 1.8:

Inception Estimation of the project vision, time frame and requirements. The inception phase should be short and has the purpose to decide if the project is feasible.

Elaboration The elaboration phase is mostly about determining the risk factors of the project and to establish the system architecture. This is done by creating the necessary use case diagram, along with information models for the system and architecture.

Conception Through the iterations: analysis, design, implementation and test, the system is constructed and as a result the final product should be ready, along with fully specified use cases and diagrams of the system.

Transition The final tests are concluded and the system is being deployed to the users.

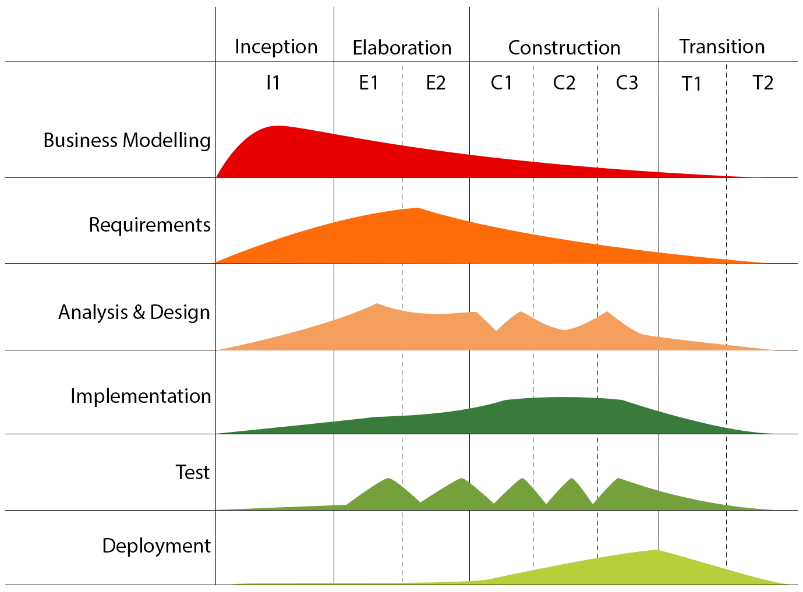


Figure 1.8: Unified Process model over time.

The decision to use Unified Process is based on two clear advantages. The first one being that Unified Process is use-case driven. This fits well in with the aspect of having the user in focus and how this user can benefit from the developed prototype. Another advantage of using Unified Process in regards to The Spiral Methodology, is that, when having a large project, as this thesis, that runs over a long period of time, there can be ongoing reviews throughout project period. This is due to the iterations that it is build upon, but also because of the amount of work, that is required in producing the documentation. This is very fitting for writing a thesis, but could be a disadvantage if working with smaller project in the industry. In order to develop the core functionality of the projet, test-driven development (TDD) will be used. By first creating the underlying tests for a function, a clear definition of what should happen inside the function is established. This allows for continuous integration systems and developers to have ongoing reports of the functionality as it is being developed, through tests.

1.6 Risk Analysis

When planning a project, a risk analysis is an important and vital part as it helps identify, where threats in the project may occur and the estimation of the likelihood, that these threats will materialize into serious problems during the project.

Before determining the potential threats, the project will first be divided into a few important steps and each of theses steps will then be analyzed individually. This helps to avoid generalizing the threats for the whole project, as there is a lot of aspects to take into account, when making the analysis.

The first step is the *analysis step*, where the project description is being developed and analyzed. As this is the "foundation" of the thesis and all further work will rely on this, is it definitely an important step. The second step to look at, is the *development step*, where the prototype is being developed. The importance here lies in, that the development will result in the prototype which is an important part of the project. The last step is the *documentation step*, where the prototype will be developed and the report constructed. As the thesis in the end will be evaluated on this report, this is also an important step.

Moving on to the analysis of each step, the following tables shows the threats in the step and the corresponding description of the threat and its threat estimation on a scale from 1 to 10. The threat estimation is an expression of how likely the risk is to occur and how much of an impact the risk have.

Analysis Step

Threat	Description	Estimate
Human	When analyzing the project, the thesis will depend on other people's input in order to determining the more specific project description. These people will often have completely different perspective and basis for being interested in the project and its findings.	9
Technical	Going into the analysis, a completely new technical topic has to be analyzed in order to write the project description. This may be extensive and exhausting, as it is important to understand the details.	7
Time	The time to read and write the first draft of the thesis might not be essential, but as time always is a risk it is worth considering.	3

Table 1.2: Risk analysis the analysis step

Development Step

Threat	Description	Estimate
Human	During the development of the prototype, only two people will be developing on the prototype. If illness or injury occur, the manpower could be a threat.	7
Operational	Developing the prototype, certain access may be difficult to obtain or be limited. This could for example be the ability to access and develop for the cloud.	9
Financial	When testing the prototype in a cloud, it often cost money to deploy and run. As the thesis is not being financed, this could be a threat for the deployment of the prototype.	4
Time	Being a large step, the time for developing the prototype may be a threat as the thesis of course is on a deadline.	7
Technical	Not knowing how to develop to the cloud and being able to comply with the standard may be a threat.	6
Collaboration	Multiple people developing on a prototype can be done in many different ways. The risk that the people developing the prototype works in different ways, could be threat for the people collaborating on the prototype.	3

Table 1.3: Risk analysis the development step

Documentation Step

Threat	Description	Estimate
Human	As with the development step, having limited manpower can be a threat if illness or injury occur.	7
Time	The documentation step may run parallel with the development step, but when the development step has finished, more time is needed in order to complete the documentation step. Therefore the time threat for the documentation step compared to the development step more important.	10
Technical	Documenting the models and the being able to describe the underlying technologies can be difficult and it is important to get this right.	8
Collaboration	When writing documentation, it is important, even though different people are writing, to carry out a uniform language and writing style. This can be a risk as no one write identically and it can be distracting for the reader.	6
Execution	The expectations for the final documentation and report might be different for each individual. This is a risk for the documentation step as one person might not think this it is “important” to write or simply just do not care.	9

Table 1.4: Risk analysis the documentation step

With the estimate for each step defined, the project can now proceed with each step's threat in mind.

1.7 Report Outline

To give a brief overview of the contents in the report, the following contains a report outline of the chapters along with a small description of each chapter.

Related Work
The thesis is based on ongoing and state-of-the-art research and technologies. The related work chapter will look into some of these concepts and give definitions of some of the more importants concepts. This includes the definition such as CENELEC, Smart Grid, Distributed Energy Resources and Cloud Computing.

Requirements
From the introduction of the chapter and the related work, the requirements chapter will present six use cases that the system will be primarily build to handle. From these the chapter will present functional and non-functional requirements.

Cloud Analysis
The Cloud Analysis chapter elaborates on which service model the prototype should use and some of the concerns regarding using the cloud. Three different providers will be analyzed in order to determine which provider should be used. At last the chapter will analyze the different communication aspects of the cloud and which technology should be used.

Standards Analysis
The Standards Analysis chapter will elaborate on how IEC 61850 standard and the iPower Flexibility Interface can be used together in the cloud application prototype.

Design
The Design chapter presents the design aspects on the prototype. It provides a domain model for the whole prototype and a more detailed system overview for the cloud solution. From this, each part of the system will be designed in different sections. This includes the information model, cloud service design, dynamic modules, storage design, users and the different interfaces.

Implementation
As the design chapter presents the design aspect of the prototype, the Implementation chapter explains exactly how the prototype is build and how it has been implemented.

Testing
The Test chapter elaborates on how the the prototype is tested, what testing methods are used and what benefits were gained from using these testing methods.

Case Studies
Two different case studies will be reviewed to show, that the prototype can handle the requirements and goals that was set in the introduction chapter and requirements chapter.

Conclusion
To round off the thesis, a conclusion will be drawn from the vision of the project and

summaries from each chapter throughout the report. Finally relevant future work will be discussed.

1.8 Additional Activities

During this thesis, the students took part in additional activities directly related to the subject of the thesis.

Paper

A paper titled: 'Utilizing a Flexibility Interface for Distributed Energy Resources Through a Cloud-Based Service' was written and submitted to the IEEE SmartGridComm[2] international conference on smart grid communications in Vancouver, Canada, 21-24 October 2013.

The paper can be found in appendix A. The focus of the paper is on how the IEC 61850 standard [44] can be used with the CENELEC reference architecture [11] to utilize the iPower Flexibility Interface [7] with a cloud solution architecture.

Microsoft ImagineCup

During the development of this thesis, the students participated in the Microsoft ImagineCup[3] competition. As part of the ImagineCup competition, a video presentation of the project was created (see appendix C) and has been uploaded to YouTube[4]:

`http://www.youtube.com/watch?v=lRerrlKCDXI`

The focus of the competition was primarily on the Microsoft Azure Cloud Platform architecture and the communication of the DERs and the cloud.

Additionally an user guide on how to use the prototype was written, see appendix E. This user guide can be followed to try out most of the functionality of the Web Interface and the RESTful WebApi interface.

iPower Research - Work Package 4

The students of this thesis have participated in two meetings concerning the Work Package 4 of the iPower research project. This thesis is created in collaboration with iPower and the subject of the thesis handles multiple objectives of the Work Package 4, such as WP4.9 - Flexibility Interface and WP4.7 - Demonstration.

1.9 Chapter Summary

With a lot of motivation, the project description and project scope has been defined for this thesis. The overall goal of the thesis has been defined to create a prototype, that can "Communicate to a DER through a cloud-based solution". In order to do this, certain sub-goals needs to be achieved. In collaboration with iPower, a flexibility interface is also to be integrated with the IEC 61850 standard, this should be done as additional nodes to the sub-standard IEC 61850-7-420. With this, a clear vision on the final prototype and thesis has been made, along with a risk analysis to specify certain threats during the project phase.

[2]`http://sgc2013.ieee-smartgridcomm.org/content/ieee-smartgridcomm`

[3]Microsoft ImagineCup `http://www.imaginecup.com`

[4]Short URL to the video (Case sensitive): `http://goo.gl/7O4Zf`

CHAPTER 2

Related Work

In the following chapter, some of the concepts related to the thesis and current ongoing research will be investigated. Initially the concept of Smart Grid and Distributed Energy Resources will be explained, following the concept of cloud computing, along with the different types of service models. Lastly the, current state of the art research in standards and in the cloud is investigated.

2.1 Smart Grid

The use of the term Smart Grid, has in the last couple of years been used in a wide variety of contexts, but one definition given by European Technology Platform is

> A Smart Grid is an electricity network that can intelligently integrate the actions of all users connected to it - generators, consumers and those that do both - in order to efficiently deliver sustainable, economical and secure electricity supplies. [32]

This definition says that the Smart Grid is an intelligent "modernized" electrical grid, that are adapted to control renewable energy sources. This can be both sources that are producing electricity, sources that consume electricity or do both. In contrast to an "old fashioned network", a smart grid is a decentralized grid, where multiple distributed energy resources delivers power independently to the grid and is then controlled through a virtual power plant. International research shows that the implementation of Smart Grid, would make for a more reliable power grid, that are both efficient and economical [19]. Some of the functional advantages are [1] [32]

- Increased stability in the power grid
- Reduced utility prices
- Allows for real-time pricing to customers
- Improved load management
- Increased robustness
- Sensing and measurement technologies
- Automated communication between the components in the grid
- Increased utilization without expanding

Smart Grid does not only improve on the functionality, but it also has advantages for the environment, as it is more energy efficient (allows for customers to shift electricity consumption to non-peak hours) and with the use of sustainable energy resources [23] [79]. It would over time make the power market more clean (non-polluting), as the power producers are using sustainable energy resources to power the grid. Smart Grid also supports electric vehicles, where the surplus energy could be supplied back to the grid (V2G).

2.1.1 Distributed Energy Resources

An important part of the Smart Grid is, that it supports a wide-range of distributed energy resources. These can be placed anywhere from a local farm, to the rooftop of private citizens houses in the city. They can be flexible consumption units (e.g. cooling systems or electric vehicles) or production units (e.g. photo voltaic, wind turbines) that can contribute power to the local building or to the power system. One key factor, with the use of distributed energy resources, is that they do not produce or consume power in a constant manner. E.g. a solar panel primarily produce power on a clear day. On a cloudy day, the production level might not be enough to supply a house and certainly not, to provide to the grid. A single unit can therefore not replace a traditional single centralized power plant. By aggregating multiple units, they can each provide a small amount of flexibility, and a virtual power plant can then place a single united bid for all units into the power market. This is necessary as a bid into the danish market requires a minimum guaranty of 300kW [18].

2.1.2 Direct Control

Controlling a distributed energy resource can be done using two approaches called direct control and indirect control [7][66]. Direct control is when an aggregator send controls signals to a DER unit to directly control operation. The aggregator demands that the DER unit must start producing an amount much power and the DER unit does this.

Indirect control is when an aggregator send informational signals, such as price signals, to the DER unit. The DER units itself decides, based on the signals, from the aggregator when, how and if the DER will produce the requested power.

The communication may be one-way or two-way depending on, how the aggregator and DER is able to communicate. In direct control one-way communication would require the aggregator to query the DER unit for its status continuously. In two-way communication it would allow the DER to update the aggregator as needed. In indirect control, when referring to the case where an aggregator primarily sends price signals to the DER unit, two-way communication would be needed to enable the DER to send feedback to the aggregator, which in turn would be able to adjust its DER portfolio accordingly.

An example of indirect control and where the communication is initiated by the DER unit instead of the aggregator, could be an electric car that polls the price signals from all the local power providers. If the electric car was driven to a foreign country and was controlled by direct control, then the controller would have to identify where the car is located, query the power providers in that region and then communicate the price signals to the car. These steps all comes with some probable communication problems. As an alternative, the electric car itself could poll the prices from all the local power providers in the region and figure out the best price.

This thesis builds upon the work of iPower and their Flexibility Interface[7], and will only explore the concept of direct control, between the distributed energy resources and a virtual power plant.

2.2 Cloud Computing

Cloud computing consists of several hardware and software resources, dynamically providing the needed configurations for the servers. The cloud can be a setup of both physical and virtual machines and often includes resources such as a Storage Area Network (SAN) etc.

Within the definition of cloud computing [61], is not only the underlying servers, but also the applications/services that are being provided through the Internet. Depending on the business model of the cloud provider, different services can be available.

2.2.1 Cloud Computing Service Models

Just like a normal computer, the cloud also has three "basic" layers of computing, see figure 2.1 [9]. At the lowest layer, you have access to the hardware of the computer, including its processor(s), memory etc. This layer is called the infrastructure layer. In the middle, an operating system is running and interacting with the computer hardware - this layer is the platform layer. On top, third-party applications are running on the OS and the layer is therefore called the software layer [65].

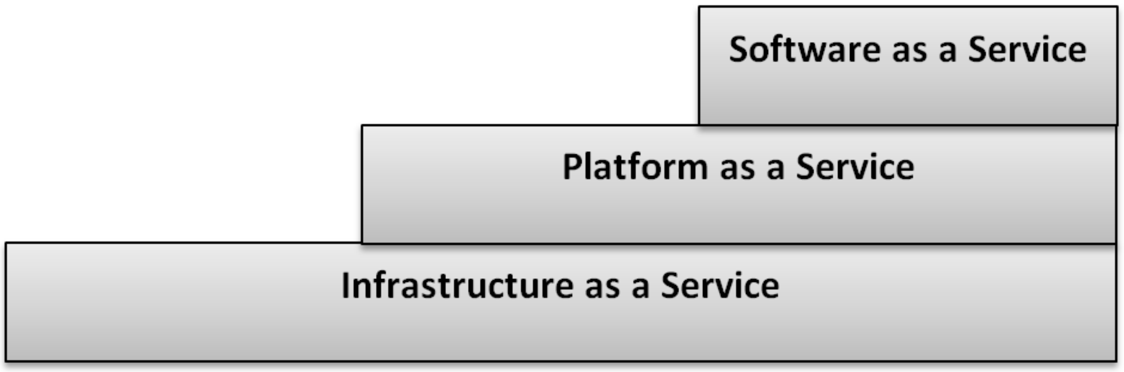


Figure 2.1: Three basic service models for cloud computing.

As mentioned, the cloud uses the same approach, and offers three different types of service models [37]. Each service model has some key differences and "limitations" for the customer.

2.2.2 Software as a Service

The service model shown in figure 2.1 is Software as a Service, also known as SaaS [74]. The service model lets you provide a complete business application to the costumer over the web. As web technologies have advanced dramatically in recent years, applications that normally runs on the costumers desktop, can now be executed directly in a browser. As the application is hosted in the cloud, running in a "foreign" environment from the customers points of view, all maintenance and often configuration are very restricted and should sometimes just be used right out of 'the box'.

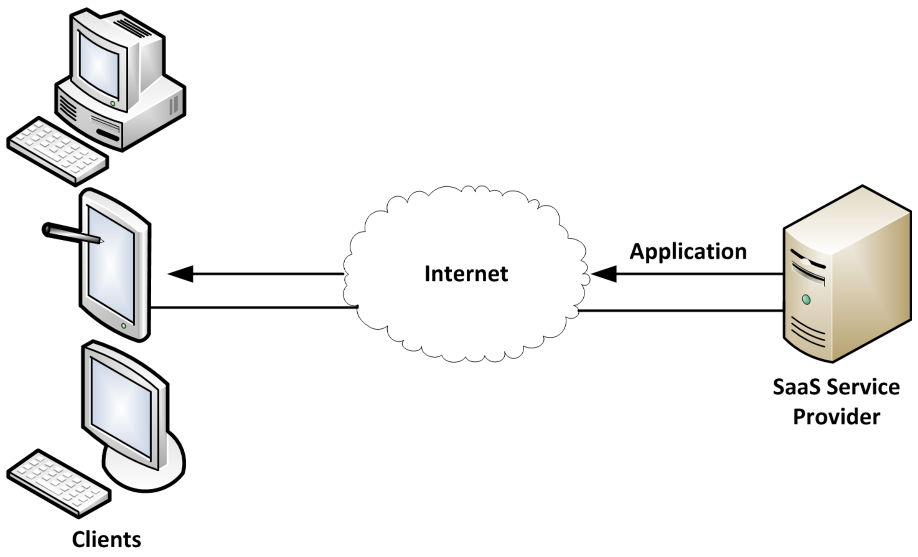


Figure 2.2: Software as a Service Model [74]

Examples of a SaaS "application" could be Microsoft's OneNote[1]. Originally, the application was a part of the Office Suite, but Microsoft created a OneNote Web App[2] where the user can access and edit their OneNote documents, like it was just a normal application.

2.2.3 Platform as a Service

Where SaaS is only for users who should use a provided application, Platform as a Service (PaaS) [74], figure 2.3, goes a bit deeper and provides the users with a stable online environment, where they can build and deploy web application. As the application is deployed to the cloud's operating system, it automatically begin to allocate the required resources such as network resources, memory access or processing power.

[1]Microsoft OneNote: `http://office.microsoft.com/en-us/onenote/`
[2]OneNote Web App: `http://www.microsoft.com/office/rm/onenotewebapp.aspx`

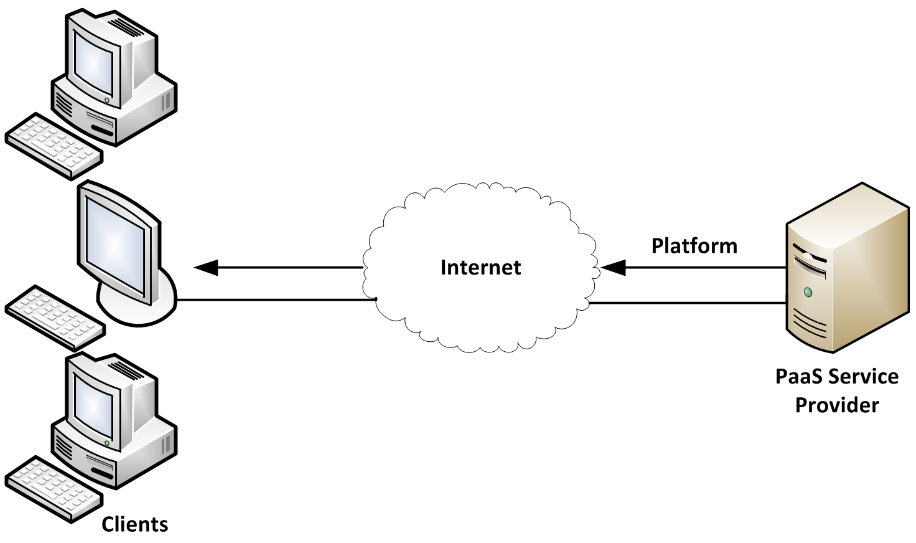


Figure 2.3: Platform as a Service Model [74]

Windows Azure or Amazon EC2 are both examples of PaaS providers, where the costumer can deploy web application, that offer scalability, security and access control.

2.2.4 Infrastructure as a Service

The last service model in cloud computing, is Infrastructure as a Service (IaaS) [74], depicted in figure 2.4. Here the costumer is provided with administrative access to some of the fundamental resources on the system. This could be direct access to processor, memory or network adapters. Granting access to these resources allows the costumer to run other operating systems on the provided hardware, but the customer is not responsible for housing or maintaining the hardware.

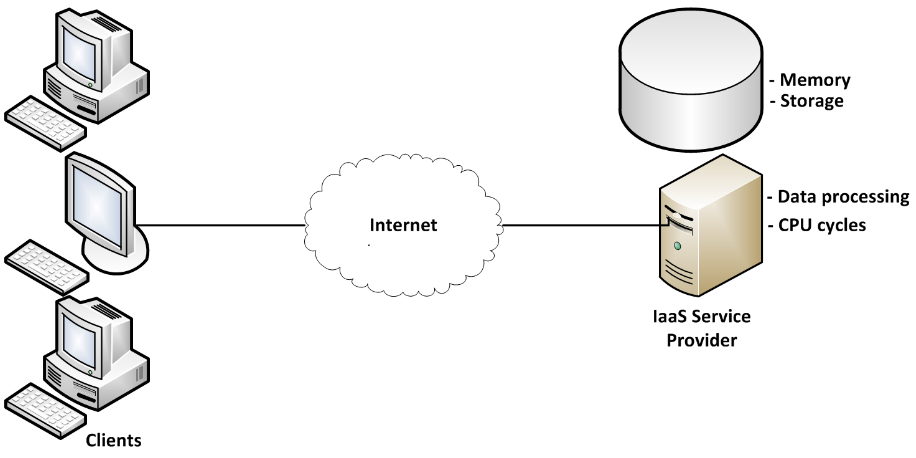


Figure 2.4: Infrastructure as a Service Model [74]

As the customer can manage and change even the operating system running on the hardware, the capabilities of IaaS are extensive. It allows companies to handle and manage servers and with the benefits of the cloud architecture, the servers can dynamically be scaled if more are needed.

2.2.5 Aspects of Using the Cloud

When using cloud computing, the following aspects are core features that defines the basis in cloud computing [10].

2.2.5.1 *Elasticity*

The capabilities of the underlying infrastructure to change and adapt to certain requirements is one of the essentials features of the cloud. Being able to scale and handle multiple numbers of instances running concurrently or dynamically allocate resources to meet a needed workload.

2.2.5.2 *Reliability*

Reliability is not only to guarantee an uptime of nearly 24/7 on the data centers (availability). It also includes running a system with no loss of data, an isolated code execution process etc.

2.2.5.3 *Quality of Service*

As in many business cases, certain quality of services must be a guarantee. For cloud computing this could be response time, throughput or latency.

2.3 Research - State of the Art

As the previous section was a short introduction to some of the topics of this thesis, the following section goes a bit deeper with “State of the Art” research. First it elaborates on the CENELEC rapport for Smart Grid [11]. With this in mind, the iPower-net (the collaborating partner of the thesis) and the work associated with Work Package 4 (WP4) 2.3.2.1 will be briefly looked at, as well as the IEC 61850 standard.

2.3.1 CENELEC

Standardization of Smart Grid [22] includes many committees, pilot- and research-projects, that work both on a national level, but also on a larger international level. In November 2012, CENELEC - a Smart Grid Coordination Group (SG-CG) Reference Architecture Working Group (SG-CG/RA) - had been assembled to focus on certain aspects of the Reference Architecture [12][11]:

- Means to communicate, in a common view and language about a system context, not only in the SG-CG but also with industries, customers and regulators.
- Integration of various existing state-of-the-art approaches into one model with additional European aspects.
- Methods to serve as a basis to analyze and evaluate alternative implementations of an architecture.
- Support for planning the transition from an existing legacy architecture to a new smart grid-driven architecture.
- Criteria for properly assessing conformance with identified standards and given interoperability requirements.

For this thesis, especially the first point of “communicating in a common view and language” is an important part, as the development of a flexible interface into a well known standards.

2.3.1.1 *Conceptual Model*

Before the Reference Architecture can be defined, an underlying conceptual model (domain model) has to be created. CENELEC’s conceptual model is based on previous models such as the National Institute of Standards and Technology (NIST) Conceptual Model [59]. The NIST Conceptual Model is a high-level framework for Smart Grid and it defines seven different domains [12]:

- Bulk Generation
- Transmission
- Distribution
- Customers
- Operations
- Markets
- Service Providers

With notion on how each domain is connected to the others and how the communication and electrical flow are interrelated.

As the trend today is still to become more decentralized, regarding power supplies [78], an extension has been made to the NIST model (see figure 2.5), in order to integrate distributed energy resources. That extension is made due to the following reasons [12]:

- Distributed Energy Resources require a new class of use cases.
- In order to comply to future anticipated regulations and legislations explicit distinction of Distributed Energy Resources will be required.
- Distributed Energy Resources represent the current situation.
- A consistent model requires clear criteria to separate the new DER Domain from the existing Domains, especially from Bulk Generation and the Customer Domains.

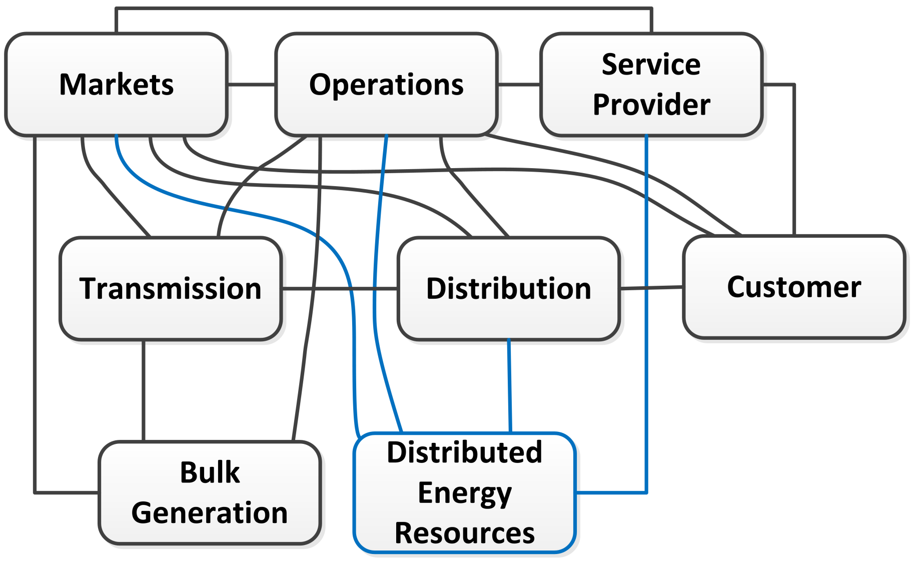


Figure 2.5: NIST Model with DER Extension [11]

As a part of this new integration, the M490 Working Group[3] introduced the concept of flexibility. The flexibility combines the model, consumption, production and storage into one flexibility entity. This allows for more flexibility in the NIST Model, and support future demand-response use cases, as non-IT users would be able to create use cases, that could more easily be implemented. Table 2.1 shows the NIST Domains, where the flexibility concept could be applied.

Domain	Market	Grid	Flexibility
Markets	●		
Bulk Generation			●
DER			●
Customer			●
Transmission		●	
Distribution		●	
Operations	●	●	●
Service Provider	●	●	●

Table 2.1: Flexibility in NIST Domains [11]

2.3.1.2 *Reference Architecture*

An important part of the Reference Architecture, is that it is based on existing materials, such as the NIST Conceptual Model and the GridWise Architecture Council [59][33]. The Smart Grids Architecture Model (SGAM) framework [11] consists of five layers, that have been abstracted from the Joint Working Group[4] recommendations, where the architecture is divided into four layers.

[3]SG-CG/RA and SG-CG/SP
[4]CEN, CENELEC and ETSI Activity

The five layers are as follows:

- Business Layer
- Functional Layer
- Information Layer
- Communication Layer
- Component Layer

The intention of this abstraction model, is to represent the information management zones where interactions are identified.

Implementing a flexibility interface that follows the presented criteria in the specification of Conceptual Model, will be subject to the *Information Layer* and the *Communication Layer*. The *Information Layer* is described as the information, which is being exchanged between the components, using an underlying common data model. It also includes the semantics for functions and services for the communication. The *Communication Layer* is defined as a layer for describing the protocols and the technology for the interoperability between each component.

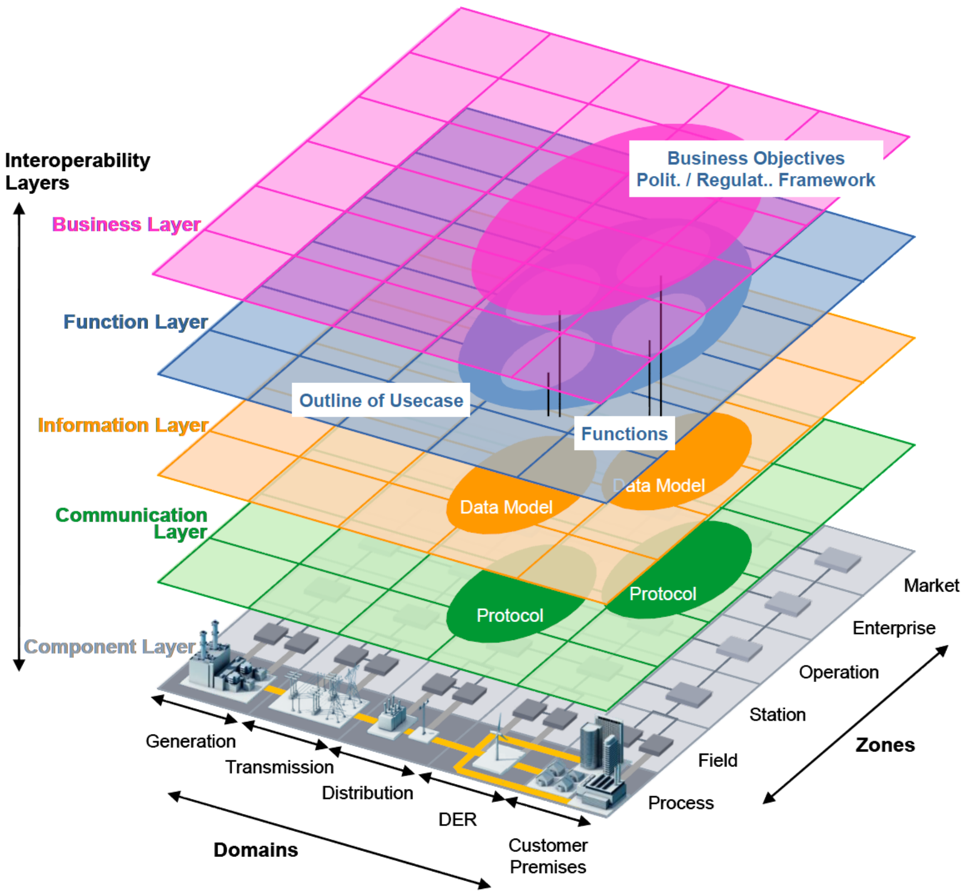


Figure 2.6: SGAM Framework [11]

From these two layers, a formal Information Architecture can be presented, that integrates the data management and the interfaces needed. As far as integration technology concerns, CENELEC prefers a SOA solution [5], as it integrates the technical concepts of enterprise application integration with the organizational embedding. Nevertheless, with more decentralized systems, also a standard for communication is needed, so different vendors can communicate with each other. CENELEC presents the IEC 62541 OPC-UA [47], a SOA-based approach for data exchange between each domain and zones in figure 2.6. CENELEC does not recommend any specific standards for how the data is stored and accessed in the Data Model, but existing standards such as the IEC 61850 [44] are the most prominent data models.

2.3.2 iPower-net.dk

iPower is a collaboration of 32 partners, universities, research institutions and industrial companies. With the department of Electrical Engineering at the Technical University of Denmark being the coordinator, it has focus on research and innovation of Smart Grid. It strives to develop a tool for intelligent control of decentralized power consumption and production, in order to manage lots of flexible consumption units [71]. The project is divided into seven different Work Packages (WP), where each WP is focused on a specific part of Smart Grid:

- WP1 Domestic Demand Response
- WP2 Industrial Demand Response
- WP3 Distribution Grid Operation
- WP4 Control and Market Operation
- WP5 Socio-economic and Investor Evaluation
- WP6 Consumer Behaviour
- WP7 Information Sharing Platform

This thesis contributes to parts of WP4.

2.3.2.1 *WP4 - Control and Market Operation*

Each Work Package by itself is divided into multiple subcategories. WP4, for example, is divided into[71]:

- WP4.0 Control and Market Operation
  Management activities in WP4 and related activities in iPower in general.
- WP4.1 Information platform for simulations IPFORIS
  The purpose of this task is to establish a technical platform where iPower participants can up- and down-load data for analysis and simulations.
- WP4.2 Flexibility clearing house platform FLECH
  Data in IPFORIS is shaped around a DSO market place (a clearing house platform), where the DSOs can request and procure local flexible generation and demand response services.

- WP4.3 Simulation-based evaluation of Control Services
  The purpose of this task is to identify, implement and test methods for validation of control services to be delivered by distributed energy resource (DER), on the request of an aggregator.

- WP4.4 Direct control
  The overall goal is to develop and demonstrate a VPP for control of a large portfolio of flexible consumers towards the current markets and future DSO markets.

- WP4.5 Direct and indirect control in distribution grid
  The purpose of this task is to integrate and evaluate direct control and indirect control policies for demand side management using VPP technology, considering also distribution grid operation constraints.

- WP4.6 Forecasting
  Models and concepts facilitating indirect control policies, included in the deliverable "Flexibility Interface - Indirect Control by Prices" by Tryggvi Jónsson from DTU Compute.

- WP4.7 Demonstration
  The purpose of this task is to demonstrate the solutions developed in WP4 – Control and Market Operation. The purpose is not to demonstrate individual flexibility solutions on a consumer unit level, but rather to demonstrate how the units would respond collectively in an energy system context to provide flexibility services. Larger/full-scale demonstration will often require more competences which exceeds iPowers frame, and in some situation specialized resources and/or systems cannot be committed on a long time horizon. Demonstrations plans are developed along with the new learning's, and coordinated with the WPL.

- WP4.8 Operational scenarios
  Generation and documentation of operational scenarios. An Operational Scenario describes a specific power system's service. Relevant for demand response is connecting existing and new actors in the power system in terms of responsibility, functionality and information flow. This description spans across relevant market rules & regulations and operation procedures. Standard diagrammatic representations are employed to support a formally rigid scenario definition where useful.

- WP4.9 Flexibility interface
  Definition of a common flexibility interface defining the information flow between a set of control policies (indirect or direct) and surrounding entities (i.e., consumer, DSO, TSO, other service providers).

This thesis specifically is a part of the WP4.9 as it defines/implements a flexibility interface previously created by another contributor to WP4. As it also implements a part of the "Demonstration" Work Package (WP4.7).

2.3.3 The IEC 61850 Standard

The IEC 61850 standard [44] is designed to allow for interoperability between different devices in power substations environments, as well as allowing for future extensions and communication technologies.

The standard consists of several sub parts, each covering and explaining its own area in fine detail. The main sub part of focus in this project is the IEC 61850-7-420 standard sub part [46], which the iPower Flexibility Interface [7] will be mapped into. The IEC 61850-7-420 builds upon the basics of the IEC 61850 standard and an analysis of the related sub parts is needed.

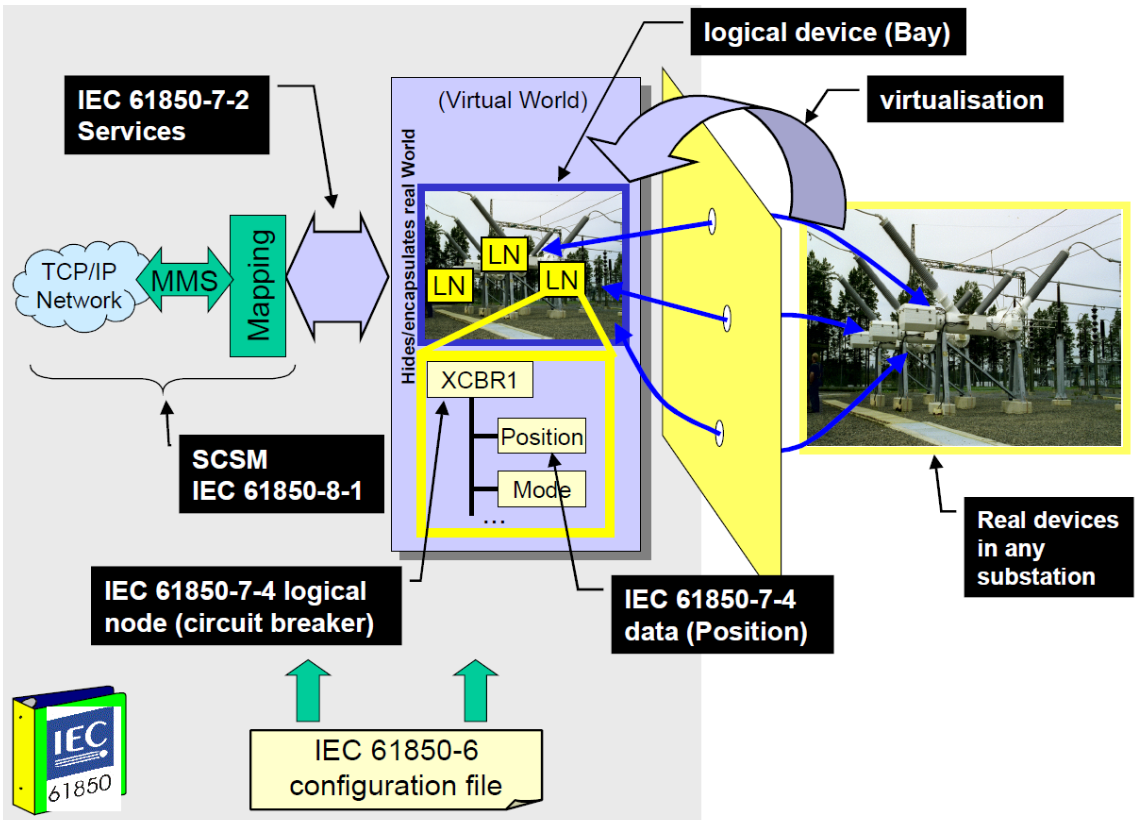

logical device (Bay)
IEC 61850-7-2 Services
(Virtual World)
virtualisation
Hides/encapsulates real World
TCP/IP Network
MMS
Mapping
LN
LN
LN
XCBR1
Position
Mode
...
SCSM IEC 61850-8-1
Real devices in any substation
IEC 61850-7-4 logical node (circuit breaker)
IEC 61850-7-4 data (Position)
IEC 61850
IEC 61850-6 configuration file

Figure 2.7: IEC 61850 Modeling Approach [44]

Figure 2.7 is an illustration of how some of the common parts in the IEC 61850 standard fits together and the areas they cover. The figure depicts a Virtual World of a typical power substation and how the individual components are modeled using the IEC 61850 standard. The yellow plane represents the mapping from the physical substation environment to the logical data model.

The IEC 61850 standard covers many features, some of the more common are [62]:

Key features

- Object oriented data model containing devices, nodes, data, services etc.
- An object naming scheme to allow for easier familiarization
- Predefined names for data-classes

- A common XML-based substation configuration syntax called SCL
- Self-describing devices

Other benefits

- SCL is used for both configuration and for specifying requirements
- Costs are lowered due to easier hardware implementation
- Increased expandability for example by using TCP/IP

In addition to the IEC 61850, there exists other closely related standards such as the IEC 61400-25 standard, which handles wind turbine communication.

2.4 Chapter Summary

The thesis is based on many different concepts, from the electrical world of the Smart Grid and distributed energy resources to modern cloud computing. The Smart Grid is the “goto” concept, when talking about the electrical grid of “tomorrow”. It can intelligently aggregate distributed energy resources, such as wind turbines and electric vehicles through a virtual power plant, whom then can make a bid into the danish market.

The Smart Grid is a state-of-the-art grid, where standardization is still under development. Organizations such as CENELEC defines certain aspects on how the Smart Grid should be built, this includes the architecture, information model and standards, such as IEC 61850. They introduces the concept of flexibility, when communicating with distributed energy resources, but exact standard implementation is still an ongoing topic. A part of the thesis is to collaborate with iPower-net and implement their definition of flexibility into existing standards.

To handle the electrical grid of “tomorrow”, a new concept of communication will also be introduced. Cloud computing is a new concept, where the underlying computational resources are spread out throughout the world. It is a green way to get computational power, that is accessible for all computers with an internet connection. It is a very reliable and elastic platform, that allows scaling of resources when needed and provides a guaranty for quality of service.

CHAPTER 3

Requirements

As the Unified Process methodology is a use-case driven and architecture-centric methodology, analyzing how the system should function, is an important part of the process. In order to do so, a list of functional requirements should describe what the system should do, and a list of non-functional requirements should describe how the system should do it. To get these requirements, a number of use-cases will be created and the requirements will then be derived from these.

3.1 Use Cases

In this section, six use cases will be introduced, a more detailed view is shown in the appendix. Two use cases are for the communication for a farmer into the web module, two use cases are for a customer communicating through a web API and one use case for the connectivity of a distributed energy resource to the cloud.

All of the use cases has been gathered in a use case diagram, see figure 3.1.

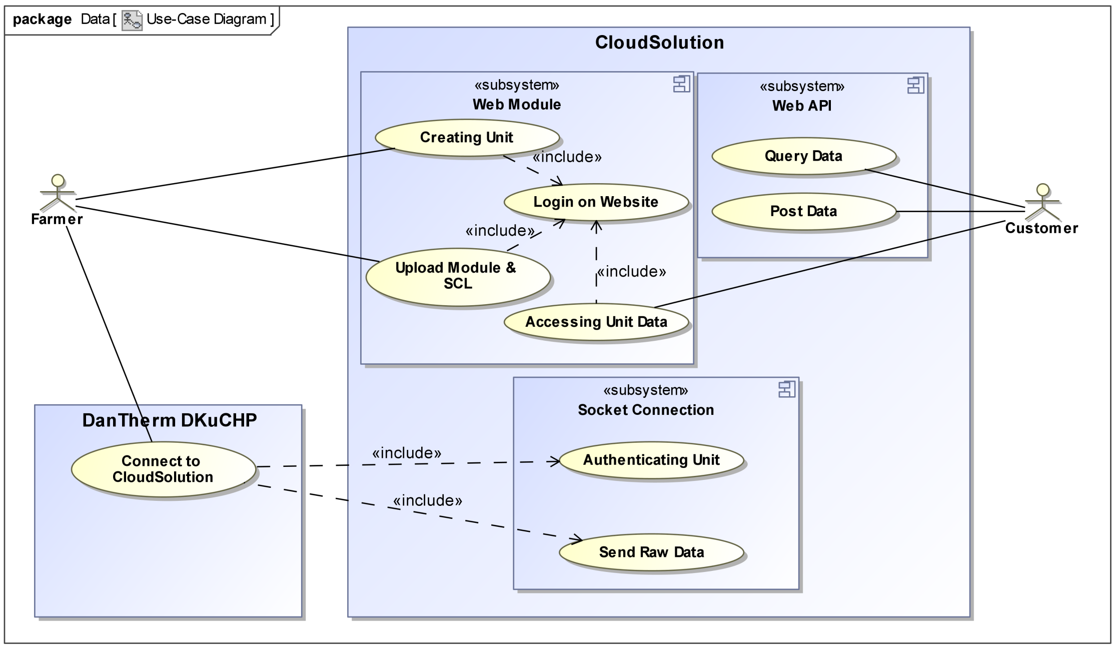


Figure 3.1: Use Case Diagram

The diagram in figure 3.1 shows the CloudSolution as the largest part of the system, with three subsystems.

Web Module
The web module is a web page where the user is presented with a user friendly interface. The site is created for specific purposes, such as creating units, uploading modules etc., where the user friendly interface should make simple operations more easy.

Web API
In the subsystem Web API a user can retrieve and access data based on queries. Contrary to the web module, the web APIs functionally is larger, but more cumbersome to use.

Socket Connection
For communication between units and the CloudSolution, a socket system that handles communication is needed.

The system Dantherm DKµCHP is an actual machine that a user is operating, but while the user is operating the system it interacts and uses certain parts of the system and it is therefore included in the diagram.

Two actors are illustrate on the diagram. First is the Farmer, who connects a unit into the system. On the other side, a Customer is accessing data in the system. Taxonomy for the system is neglected, but the Farmer is inherit from the Customer user.

Each of the primary use cases is described in the following section, a more elaborated description can be found in appendix G.

3.1.1 Use Case : Creating Unit

The first use case (see G.1) is a customer (e.g. a farmer) that logs into the CloudSolution and from here creates a new unit with a default module. As a result of the use case, the farmer retrieves a unit ID and a password associated with the unit.

3.1.2 Use Case : Upload Module & SCL

The second use case (see G.3) is also for a customer (e.g. a farmer), who this time logs into the CloudSolution and then uploads a new module. He is doing this on a specific module page and are also uploading an associated SCL configuration file. At last the farmer locates the unit on the unit-page and changes the module that the unit is using to the new module.

3.1.3 Use Case : Accessing Unit Data

A customer logs into the website and here accesses the units page in this use case (see G.4). The customer can here get an overview of all data posted by the unit into the system.

3.1.4 Use Case : Connect to CloudSolution

In this use case (see G.5) a farmer enters the needed information for a unit. This could be a unit ID and its password. With this, the unit first connects and authenticates with the cloud and then, using a different WebSocket, sends the raw data to the cloud.

3.1.5 Use Case : Query Data

In this use case (see G.8), the customer can access data for a specific unit. Doing so, the customer has to know the units ID and has to have permissions to access the data. Also if these criteria are satisfied, the customer can either access data through a web API or by using the web module.

3.1.6 Use Case : Post Data

The last use case (see G.9) is a customer that has to post data value to a unit. In order to do so, the customer has to know the units ID and have the needed permissions. Using the the web module, the customer can post predefined commands to a unit, such as On, Off or Reboot. But using the web API, and giving a specific path to a variable, the customer can set a value on the unit.

3.2 Functional Requirements

With the use cases describing how the system will be used, the functional requirements can be derived from these and the Project Goals (see section 1.3.2). These are used to describe what the final system should do.

Req 1	Authentication and authorization of users
The cloud application must support multiple users, who can be authenticated and authorized.	
Req 2	Different access level
The cloud application must support different levels of users. These privileges takes the form of user roles.	
Req 3	Generic data handling
Data that are being transferred to the cloud from a DER must be handled in a generic manner.	
Req 4	Query data information
Information from DERs must be available through a query interface.	
Req 5	DER data history
The cloud application must provide a website, where posted data from each DER can be viewed.	
Req 6	DER Management
DER operators must be able to create and manage DER units.	
Req 7	Send Commands
DER operators must be able to send commands through the query interfaces to the DER units.	
Req 8	Communication should use the IEC 61850 standard
The information model used when transferring data must be based on the IEC 61850 standard.	
Req 9	Providing a flexibility interface into the IEC 61850 standard
The IEC 61850 information model must be extended to handle a flexibility interface.	

Table 3.2: Functional Requirements

3.3 Non-Functional Requirements

The Non-functional requirements describes the qualities or constraints of the prototype. Where the functional requirements describes what the prototypes should do, the non-functional requirements describes how the prototype should be. The non-functional requirements is listed as a brief description.

Req #	Non-Functional Requirement
Req 1	All communication with clients/users and the cloud application should use open web standards
Req 2	The communication should be secured with appropriate security for the respective communication interfaces
Req 3	The communication between the cloud and DER units should be near real time and facilitate two-way communication
Req 4	The system must be able to scale to match the hundreds of thousands of potential DER units that will be connected
Req 5	The storage must be able to hold a minimum of two years worth of data for each DER unit

Table 3.3: Non-Functional Requirements

3.4 Requirements Matrix

The requirements matrix correlates the functional requirements with the use cases which helps to ensure that the functional requirements are covered by the use cases and that the use cases are covered by the functional requirements. This ensures that no use case or functional requirement exists without a relation to the other.

Use case	Functional Requirements								
	1	2	3	4	5	6	7	8	9
Creating Unit	•	•				•			
Upload module & SCL	•	•				•			
Connect to CloudSolution			•					•	•
Authenticating Unit	•								
Send Raw Data		•	•					•	•
Query Data	•	•	•	•				•	•
Accessing Unit Data	•	•			•				
Post Data	•	•					•		

Table 3.4: Requirements Matrix

3.5 Chapter Summary

Six use cases has been created in order to define the primary functionality of the prototype. From the use cases and the project goals, functional and non-functional requirements have been created. These requirements defines what the system should do and how it should do it. To ensure that each of the functional requirements are accounted for, a requirement matrix shows the relation between the functional requirements and the use cases.

CHAPTER 4

Cloud Analysis

When using cloud technologies a lot of aspects has to be considered. This is because the cloud does raise some concerns regarding security, data localization, latency, failure and so on. In the following chapter, a look into what kind of service model (as described in chapter 2.2) is needed, for the solution to be deployed.

4.1 Service Models

As previously stated, the three kinds of service models is created using a layered architecture, in the same way as a normal PC is. When choosing a service model, each of them has their benefits and disadvantages.

4.1.1 Infrastructure as a Service

First looking at the "bottom" with IaaS, the immediate benefits are that one has full control over the hardware and can dynamically scale up or down, when the applications needs it. In the Smart Grid world, this could be a very useful thing, especially with the growth in distributed energy resources, the amount of connections it has to handle needs to scale rapidly, e.g. during work hours to handle the extra load. As the connections itself might have a small linear slope in growth, the data transmitted from the units into the cloud has a much larger growth and occur more rapidly as data are produced. As Smart Grid is not only a local grid, but an international grid, it is also important to have an infrastructure that allows communication from all parts of the world, without having to go all the way around the world, in order to communicate with a server. IaaS supports multiple tenants and this allows for multiple access to one cloud instance, but it also allows for connectivity between multiple cloud-farms using private communication channels [74]. This should allow regional connection, so a photo voltaic cell can connect to a regional data center instead of a server being hosted far away. This should help with the communication latency. Some of the disadvantages with the use of IaaS is, that one largely depends on the vendor's capabilities. As all hardware are hosted at the vendor, the customer looses control, and the vendor has the right to turn off and on the server, possibly even without giving notice.

4.1.2 Platform as a Service

Following IaaS is PaaS, where the OS is given. Using PaaS makes the development team relatively vendor locked, as the team has to develop applications for a specific operating system, using a specific technology. If the team had to change cloud provider, the complete application might have to be rewritten. Hosting the application in PaaS does come with some benefits [49]. It do minimizes the maintenance for the application by allowing scalability, configuration and fail-over.

Hosting the application in multiple cloud farms also allows for regional connectivity which is often supported directly from the vendor.

4.1.3 Software as a Service

One of the biggest advantages of SaaS is the use of the scalability [70]. Only running small application in the cloud, moving an instance to a less demanding location does not require as much as it would to relocate a PaaS instance. Using SaaS application is also very user friendly, but does have some of the same issues with application development as PaaS have. Being locked to the vendors technology is not the only disadvantage, because the customer has no control over what else is hosted on the server instance that the application is running on.

4.2 Security & Privacy

Using the cloud does raise many concerns for companies. Especially the security and privacy can be a deciding factor, when the consumers are thinking of moving towards the cloud. Hence it is important to identify which security risk and privacy issues that may be associated with using cloud computing. Many of these issues have already been identified [8] [69], but certain aspects still needs to be considered.

4.2.1 Intruders

Having data stored on a foreign server does raise some security issues with companies, trying to move their data into the cloud, as all security systems data stored in a public cloud can be compromised. In the last couple of years several companies have had major hacker attacks, recently CSC[1] in Denmark had a break-in, where a huge amount of CPR-number were stolen. As the cloud is running in multi-tenanted system, business competitors might share the same underlying hardware and a failure in the security paradigm, might expose vital data to the competitor.

4.2.2 Geographical Data Storage

When storing data in the cloud, the physical location of the machines storing the data, might not be straightforward to localize. Vendors might have certain regional data centers located in different countries and on different continents. So when a company stores data in the cloud storage, the actual data might not be stored in the customers country, but in a foreign country. This raises privacy issues on who can access the stored data. In the US, the USA Patriot Act (the "Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001") makes it possible for the US government to access any data stored within the US [60]. So if an European country stores its data, it has to ensure that the data is being stored within the boundaries of the EU, as the European Data Protection [13] protects European countries with their data's privacy. Microsoft's U.K. managing director Gorden Frazer, says that Microsoft can not guarantee that EU-stored data, would be held within EU-based data centers and not leave the European Economic Area under any circumstances [75]:

> "Microsoft cannot provide those guarantees. Neither can any other company."

[1]CSC: `http://www.csc.com/dk`

This has also been researched [35] and the findings shows [50], that merely avoiding a US based cloud service provider does not solve the problem of avoiding the Patriot Act. Recently leaks from National Security Agency has also indicated that their PRISM program gives the US government access to multiple cloud based services' data without the acknowledgment of the users [76].

These are all great concerns when creating a cloud based solution for Smart Grid. Should the intelligent application be running in a public cloud? or maybe a privately hosted cloud is necessary. For this thesis, this is not of any important concern, as it is only a proof-of-concept, but these facts are important when moving on to a larger scale and production.

4.3 Latency Concerns

When talking about cloud computing, the communication from the local computer to the data centers hosting the cloud, is in general unknown. One just connects to a URL and is then routed to the closest data center (geographically speaking). It might not be so important for a consumer if a site is shown in 0.13 seconds, because it is hosted in a county nearby, or it takes 3.7 seconds, because it is hosted in a country on the other site of the globe. Using real-time communication or when data is required instantly, this might be a concern when choosing to go into the cloud.

4.4 Providers

As previously stated in section 4.1.2, by using and developing to a public cloud, the customer can be “vendor locked”. Using the high level layer of SaaS, the customer can import and export the data between different cloud services if needed. If the customer is developing an application to the PaaS layer of a vendor's cloud, and later has to move the data to another vendor. The customer might have to set a time slot for migration of data resulting in downtime. The application might also be developed for the specific platform/framework that the vendor is using, so a new application has to be rewritten when the applicaion is moved to another vendor. At the lowest layer, IaaS, each virtual server might be able to be moved from one vendor to another, but this does require huge amount of data to be transferred and in a real-life scenario, might not be possible.

As the thesis is to facilitate a prototype, using the IaaS model would require to much work, compared to the value of the solution in the end. Because, as it is a prototype, portability is not a required goal. The SaaS model is a very restricted/small layer and as the SaaS provider has to provide functionality to support communication, data handling etc, the SaaS model is too limited for this solution. This leaves the PaaS model, it allows specific applications to run and the ability to handle communication between the users and the distributed energy resources.

There exists many PaaS providers, but only the three largest cloud providers will be examined.

4.4.1 Amazon

The first company to enter the cloud marked was Amazon in 2006 with the Amazon Web Services (AWS) [4]. As a pioneer in the cloud field, Amazon offers a number of different cloud services, such as [3][2]:

Elastic Compute Cloud (EC2)
EC2 is a web service that provide computational capacity in the cloud. Is is designed for developers and system administrators to get easy access to scalable computational power. Through a web service interface the customer can configure the system and allocate the required resources. It supports both Windows, Linux and other types of virtual machines, that can be requested when needed.

Virtual Private Cloud (VPC)
VPC is a private cloud that is isolated from the rest of the AWS cloud. The customer have control over the private cloud resources, including the virtual network environment.

Simple Storage Service (S3)
S3 is a Amazon's dedicated file storage service, primarily used to host binaries.

DynamoDB
Highly scalable and fault-tolerant managed NoSQL cloud databases.

Relational Database Service (RDS)
Is a managed MySQL and Oracle database that has automated backup and replication.

Simple Queue Service (SQS)
Reliable and scalable queue service, for storing messages, for communication between instances, e.g. EC2.

Simple Notification Service (SNS)
SNS is a web service that makes is possible to send notifications from the cloud. It is highly scalable and very cost efficient to publish messages from an application, running in the cloud, and deliver the message to subscribers.

4.4.2 Google

Google is knows for providing different cloud services to its users. This could be GMail, Google Docs, Google Reader or even YouTube [30]. In 2008 they launched Google App Engine (GAE) [29], which today is leading in the Java and Python PaaS public cloud. Some of the services that are included in GAE is [31]:

Instances
Fully manageable sandboxes (not virtual machines), that can be used as computational resource to host applications.

Backends
For instances that require more computational resources. Up to 1GB of memory and 4.8GHz of processing power more than normal instances. It is primarily used for background processing.

Cloud SQL
Is a fully managed MySQL database with high availability and multiple data replication centers.

Blob store
Used to stores binary file/blobs that can be accessed through a restful API.

4.4.3 Microsoft

In 2010 Microsoft entered that public cloud with Windows Azure [53]. It provides a rich PaaS platform that is mainly focused towards the .NET developers, but also supports JAVA, PHP and Node.JS for API connectivity. Like AWS and GAE, Windows Azure also includes different services [54]:

Azure Compute
Enables the customer to instantiate instances or services as needed. This could be complete virtual machines running Windows Server or Linux OS, or even services as a simple web site or a cloud service. An instance can contains multiple roles for different purposes. E.g. a Web Role for hosting Web Application or Windows Communication Foundation services, or a Worker Role for background long-running computations.

Azure Storage
The Azure storage facilities includes many different kinds of storage capabilities.

Azure Table Storage
A highly scalable distributed cloud database used to store structured data in a NoSQL datastore. It is used when storing TBs of data with datasets, that do not require complex joins or stored procedures.

Azure Blob Storage
Used to store large amount of unstructured data, normally used for storing documents, images etc.

Azure SQL Storage
The SQL Storage uses a full Microsoft SQL Server to host a SQL datastore primarily used when needing a scalable relation database.

Azure Messaging
Azure provides the means for communicating between instances through the Azure Service Bus, or a more general message queue in Azure Queues.

4.4.4 The Chosen Provider

The providers do have many similarities and choosing one over the others can be difficult. There are certain aspects of each provider that are preferable, AWS is well established, documented and has SNS; GAE is free to use; Azure handles the newest .NET framework, Message Queue, is a relatively new player, but as the .NET framework is very preferable to develop in, the GAE is not an option. The choice between AWS and Azure falls on Azure as it is from Microsoft and fully integrated with Visual Studio and the newest .NET framework. Furthermore, developing using Azure allows access to compete in Microsoft ImagineCup 2013[2] and this gives a (limited) 1 year

[2]Microsoft ImagineCup: `http://www.imaginecup.com`

access to the Azure Platform. Regarding the latency concerns, then Microsoft has data centers in three major regions: Asia, Europe & United States. In Europe one is located in Ireland and one in the Netherlands. This should limit the latency as connectivity from each Danish distributed energy resource only has communicate to one of the two countries.

4.5 DER Communication

In order to establish communication between the distributed energy resources and the cloud, certain aspects has to be considered. With the “problems” (see section 1.3) that arise when shifting to the Internet as a communication media, instead of proprietary data lines, the virtual power plant can no longer directly communicate with each unit. This would for example require each user to open ports in their firewall (might even be impossible, if the unit is located inside a corporate network) and the virtual power plant to have an IP-address of unit it has to connect to. In order to handle some of these issues, reversing the communication path, so that each unit connects to the cloud, would solve these issues. The cloud solution has a static IP-address, so every unit, as long as they have Internet connection, can connect to the cloud.

With the units connecting to the cloud, different technologies can be used for this connectivity.

4.5.1 Real-Time Web Technology

There exists many kinds of technologies that can be used for two entities to connects to each other. Using Azure as cloud provider, Windows Communication Foundation[3] (WCF) is a “normal” technology to use when talking about Microsoft technologies. WCF uses web services to establish this connection, unfortunately web services do not support real-time communication. WCF has extended the normal web service to WS-Duplex. This allows for real-time communication, but it also limits the usage to Microsoft clients using the WCF. Azure supports connections for WebSockets, which provides a full-duplex communication channel (allows for communication in both ways simultaneously) through a single TCP socket. Using WebSocket do have some challenges, as it is a very new standard and some operating systems (e.g. Windows 7) do therefore not natively support WebSockets. Fortunately, many implementations exists for WebSockets, and as long as it comply with the RFC 6455 standard [24], the clients can run any operating system or implementation they want to.

4.5.2 Push & Pull Technology

Saying that the connection has been establish between the unit and the cloud, how will the cloud know when the unit has produced new information? To determine so, two different technologies of “request for transaction” will be considered.

The first transaction is ’pull technology’ [73], where the client (the cloud) is sending a request to the server (the unit), requesting for data. The server then make a response to the request in a synchronized fashion. This kind of technology is used when e.g. requesting for a web page or requesting mails from a POP3 account.

The other transaction is ’push technology’ [73]. Here the client (the unit) subscribes to a server (the cloud), and when information is ready at the client side, it pushed the information up to the

[3]WCF: `http://msdn.microsoft.com/en-us/library/ms731082.aspx`

server, which then can act on the inputs. This technology is used in e.g. instant messaging or newsletters, where information is produced and then pushed onto a server.

Both technologies have their use, but in the scenario of communication between a distributed energy resource and the cloud, push technology is preferable. This allows units to send data as soon as it has been generated, maintaining the real-time property for the distributed energy resource communication. It also lines up with the connection path of the unit connecting to the cloud.

4.6 Cloud Communication

Certain possibilities are available for communication into the cloud. For a virtual power plant an API should be provided, so that it can communicate in a programmable fashion. For a normal user, a web site for handling the distributed energy resources might be preferred, as the normal user could be a normal farmer with little to no IT knowledge.

4.7 Chapter Summary

Each of the different service models have different advantages and disadvantages. The IaaS offers full control over the underlying hardware, but requires a lot more technical knowledge/management. PaaS is relatively vendor locked, but it minimizes maintenance and allows for scalability. SaaS is very scalable, but it does limit the development capabilities. The PaaS service model has been chosen as the correct layer that the solution will be developed for. It handles scalability and does not require hardware maintenance, and as it is only a prototype, the vendor lock is not a problem.

The three largest PaaS providers on the market (Amazon, Google and Microsoft) has many similarities, but the Azure platform has been chosen due to the .NET framework and the 1 year access through Microsoft ImagineCup. When communicating with the cloud from a DER, WebSockets with push technology for transactions has been chosen as the ideal solution, as it allows for duplex real-time communication and it is the DER that connects to the cloud.

CHAPTER 5

Standards Analysis

This chapter will analyze the details of the IEC 61850 standard and the iPower Flexibility Interface. The analysis will show, how the two standards can be used together, in order to be used in the cloud application prototype.

5.1 IEC 61850

The IEC 61850 standard [44] has successfully withstood the test of time and is well tested [51]. The Standard provides a data model designed to represent substations and each sub component within, along with functionality to inspect the data model and communicate this data to the clients.

5.1.1 Information Model

The information model in the IEC 61850 standard is designed using a layered hierarchy and is modeled like a tree structure, see figure 5.1.

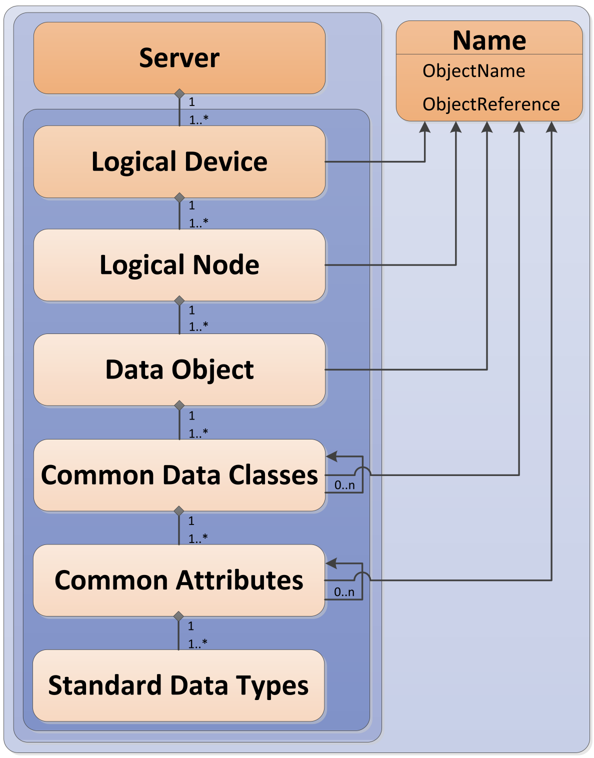


Figure 5.1: Information Model Hierarchy[40]

The logical device (LD) is a sub component of the physical device and is composed of multiple logical nodes. Each logical node (LN) holds a predefined number of data objects and each represents a specific function of the device. In each data object (DO) a predefined name of the associated logical node and their type or format is defined using a common data class. The Common Data Classes (CDC) is a set of standard data types and predefined common attributes. Each predefined common attribute, can be reused in different objects and contains more common attributes or standard data types.

5.1.1.1 Naming Convention

To access a data object within the information model, the model (figure 5.2) can be thought of as a tree like structure, where the root is the logical device. As each of the objects in the structure has an associated `ObjectName`, IEC 61850-7-2 states that the path of the data object can be referenced as follows[40]:

LD⁄LN.DO[.DO[. ...]].DA[.DA[. ...]]

Figure 5.2: Naming Convention [46]

This makes for a clear and unambiguous path to each object in the information model.

5.1.1.2 Functional Constraints

The IEC 61850-7-2 standard [40] describes a filtering mechanism referred to as Functional Constraints (FC). In the standard there are defined 18 different functional constraints. These functional constraints are defined on every Data Attribute and are used to divide the Data Attributes into categories, see Annex B in IEC 61850-7-3 [41] for a table listing of functional constraints. When adding a functional constraint to a data model reference, only the data attributes matching the functional constraint are returned. E.g. adding the functional constraint MX to a reference, would only return Data Attributes containing analogue measurement values.

5.1.2 ASCI Services

Abstract Communication Service Interface (ACSI) is an interface defined in the IEC 61850-7-2 standard sub part and describes how the data model can be accessed, to read and update data, logs and more. The ACSI interface was designed to provide an abstract interface, across any communication protocol used to transport the information.

The ACSI interface consists of a set of methods, that enables a client to retrieve directory listings containing the logical devices, logical nodes and data objects. There are methods to drill into the model and retrieve and update specific sets of data objects and data attributes. Functional constraints can be defined as arguments to these methods, in order to filter the returned list of objects.

When for example a set of data attributes, from a data object of interest, needs to be located and fetched from the data model. A call to *GetServerDirectory* would return a listing of the logical devices on a unit and after inspecting this list, a call to *GetLogicalDeviceDirectory* on a specific logical device node would return the list of logical nodes in that device on the unit.

GetNodeDirectory would then be used to get a list of the data objects, logs, reports etc. contained within a logical node and *GetDataValues* could be used to fetch the set of data attributes.

5.1.2.1 *Configuration*

Substations are configured using an XML file format, defined in the IEC 61850 standard, called Substation Configuration Language (SCL), which describes the various components within the unit. The SCL configuration describes general unit information and how the components communicate etc., but most importantly for this project, is the description of the information model, containing the data of the unit, which is to be communicated to the cloud application.

5.1.3 Other Features

The IEC 61850 standard describes additional features, which are of less interest with regards to this project. The features are briefly described here.

5.1.3.1 *Reporting and Logging*

The IEC 61850 describes an event mechanism for reporting and logging. A client can subscribe to a defined set of events and when the conditions for the event is triggered, a report is sent to the client. Reports comes in two types, a buffered and an unbuffered format. The unbuffered report is not stored anywhere, but is sent as soon as the event is triggered.

The buffered report is a mechanism to filter and group the events and can for example help avoid a large number duplicates being sent to a client, caused by a continuously triggered event. A buffered report is defined by the set of trigger conditions and a time interval, that defines how long the report should be buffered for, before being sent. When the time interval has passed, only the most recent report is then sent to the client.

Logs are similar to reports and uses the same mechanism for event triggering, which then writes a log entry. Logs are not sent to the client but stored locally on the unit.

5.1.3.2 *SCSM*

The ACSI interface is protocol agnostic, it simply defines a set of methods that are used to manipulate the data model. When a client, using any communication protocol, needs to exchange information with the ACSI interface, the data communication needs to be mapped between the protocol and the ASCI interface. A Specific Communication Service Mapping (SCSM) is used to describe the specific mapping of a communication protocol, to support the ACSI interface.

Only a few SCSM mapping exists for the IEC 61850 standard, the best known is the MMS protocol. The MMS protocol and similar protocols are not of interest for use with the prototype in this project, but interesting work is continuously done on more modern protocols, by research groups and industry consortia.

5.1.4 IEC 61850-7-420

As part of the IEC 61850 standard, the IEC 61850-7-420 [46] sub part describes the basic communication structure for distributed energy resources. The standard is specifically designed for DER units such as photovoltaics, fuel cell systems, combined heat and power generation systems and reciprocating engines. The standard allows for communication, not only locally between DER units and the management systems, but also between a complete DER plant and the operators or aggregators. The information model in IEC 61859-7-420 primarily utilizes the existing information model associated with the IEC 61850 standard, but extends it with some DER-specific logical nodes.

5.1.5 IEC 61850-80-3 TR

The IEC 61850-80-3 TR standard sub part [43] is a draft, in which ongoing research efforts are done, to describe how modern communication technologies can be used with the IEC 61850 standard. The draft covers the initial domain of substation automation and includes a new extended scope that has been introduced to the IEC 61850 standard. The new scope covers the following domains [43]:

- PV-Inverters
- Hydro and Thermal generation
- Wind farms
- Combined Heater and Power
- Smart Customers
- E-Mobility
- Virtual Power Plants (VPP) and Micro Grids
- Feeder Automation

The IEC 61850-80-3 TR gives an overview of how modern Web Services technologies can be mapped to the ACSI interface. The technologies discussed are:

- OPC-UA
- RESTful services over WebSockets
- XML Webservices
- XMPP

The draft in its current state describes use cases for the communication exchange with the systems listed in the domain scope and how they interact with the ACSI interface.

5.2 iPower Flexibility Interface

The Smart Grid Reference Architecture [11] as described by CEN-CENELEC-ETSI introduces the flexibility concept where consumption, production and storage are grouped together as an entity and ranked alongside entities, like the power grid and power markets. It is expected, that the flexibility concept will satisfy future requirements to support use-cases, where more flexibility is needed, than what the current rigid standards provide. The flexibility model should enable maximum reuse of the results derived from the models, defined in the current standards. The iPower project [71] has designed an information model of this concept in the form of the iPower Flexibility Interface [7].

The iPower Flexibility Interface is an information model designed for Direct Control of Distributed Energy Resources (DER). The flexibility interface provides a management-oriented approach for communicating the flexibility of DERs to e.g. an Aggregator or a Virtual Power Plant.

DERs consists of power production units such as wind turbines, photovoltaics or flexible consumption units, such as heating and cooling systems or electric vehicles. The flexibility interface addresses the communication of the flexibility, the characteristics, that a DER exposes, between an aggregator or virtual power plant and the DER. An aggregator manages multiple DERs, in order to provide an accumulated response of power production or consumption. This enables an aggregator to buy or sell power on the energy markets or support the power grid as ancillary services.

DERs are equipped with local controllers and are able to operate autonomously. When the DERs are connected to an aggregator, they are able to alter their consumption/production of active/reactive power in response to signals from the aggregator. The flexibility of a DER, describes the capabilities of the unit, operation costs and schedules. For example, a DER may have to prioritize local needs for power, and would be unable to meet a request for power from the aggregator, but other DERs may be able to fulfill this request. The flexibility interface describes how to communicate this information to the aggregator or virtual power plant.

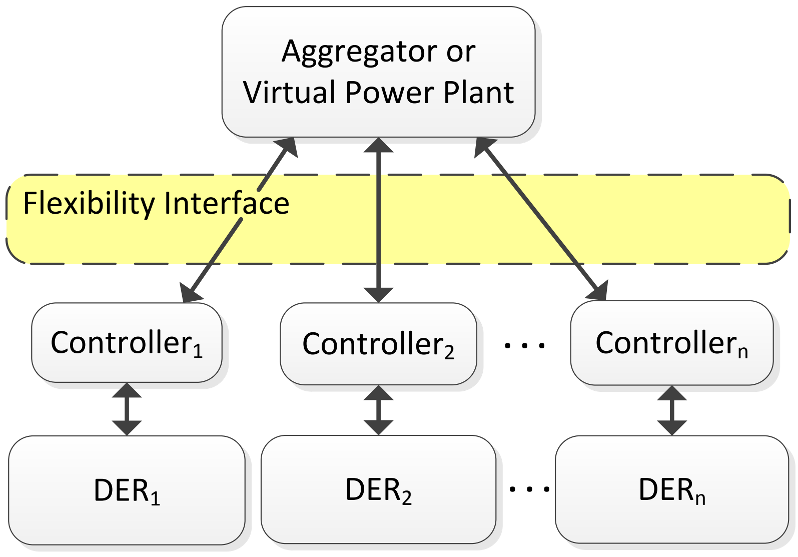


Figure 5.3: The flexibility Interface sits between the aggregator and the DER units[7].

The iPower flexibility interface supports a direct control setup, between the DERs and the aggregator. Direct control refers to a setup where an aggregator dictates the actions of a DER unit for example by telling the unit to produces this much energy and is facilitated by a two-way communication between the aggregator and the DER. The DERs reports their local flexibility to the aggregator and the aggregator controls the DERs based on this information. The flexibility interface, as illustrated in figure 5.3, is situated between the aggregator/virtual power plant and the DERs.

The following is a selection of the characteristics described in the iPower Flexibility Interface:

- **Geographical location**
  The location of a DER in the power grid is important in the case, where the DER supports the grid as an ancillary service.
- **Local control or remote control**
  In the case, where the DER is part of the primary frequency reserve for the power grid, the DER must measure the grid frequency locally and a local control loop, determines if the unit should activate. In the case of the DER being in secondary, tertiary or manual reserves, the DER could be activated using remote signals.
- **Combined deliveries**
  DERs can be either pure consumption units, pure productions units or a combined consumption and productions units. It is therefore important to be able to distinguish between these types of units.
- **Active/reactive power**
  Both the active and reactive power is required to be communicated to the aggregator, in order for the aggregator to properly place bids on the power markets.
- **Limitations**
  Limitations of the DER includes power consumption/production levels and frequency levels. (how many times, given an interval, is it feasible to have the energy level go up and down.)
- **Tracking**
  The ability to track a given remote power reference or to track a planned power schedule.
- **Runtime**
  Minimum and maximum runtime. E.g., a unit must run for at least 30 min when started.
- **Downtime**
  Minimum and maximum downtime. E.g., when a unit is turned off, it must be turned off for at least 1 hour.
- **Cost**
  The aggregator needs to know the cost associated with the active/reactive power production cost, energy level dependent cost, unit start-up cost, unit shutdown cost for discrete power units: cost depending on the number of power level changes and flexible activation time costs. (E.g. the later the activation, the higher the cost).
- **Contracts**
  For an aggregator to be able to manage a DER, some contract or agreement exists between the aggregator and the DER. This is communicated through the flexibility interface.

The information model of the flexibility interface is based on these characteristics, and is divided into a number of flexibility blocks, which form the basis of the flexibility interface. In figure 5.4, a graphical overview of the flexibility blocks is presented.

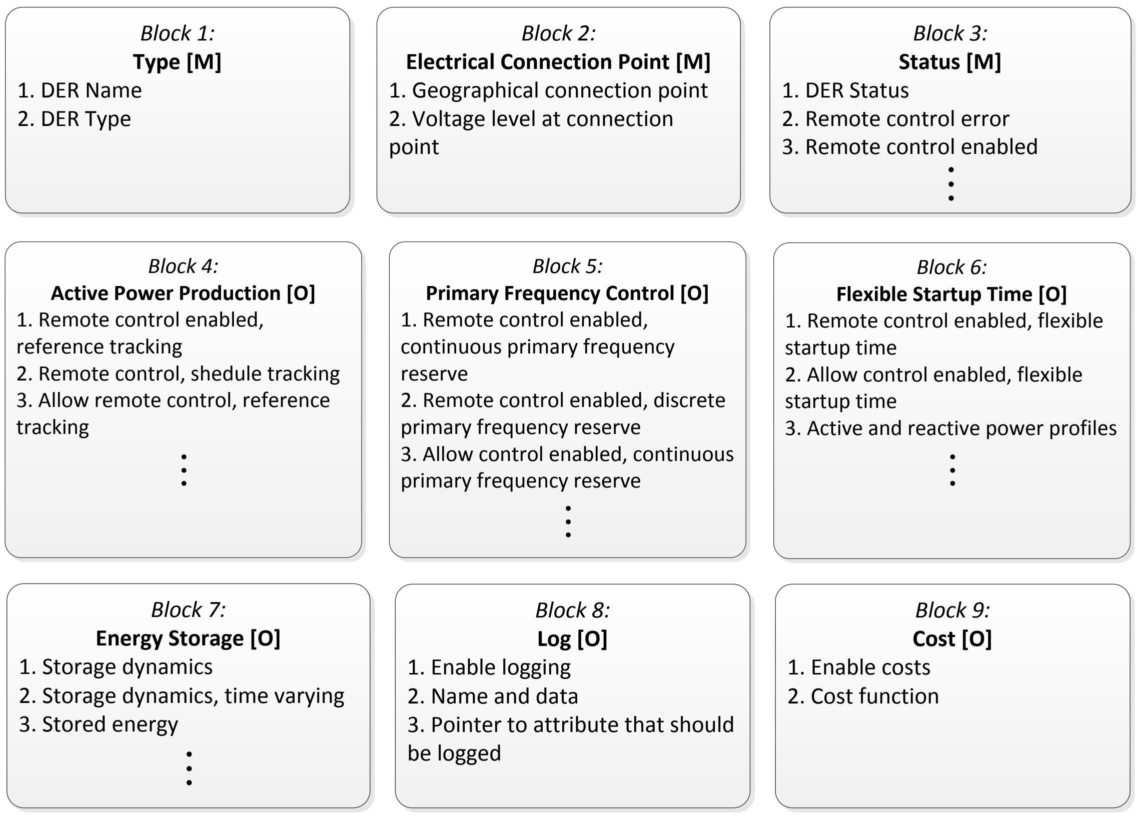

Block 1:
Type [M]
1. DER Name
2. DER Type
Block 2:
Electrical Connection Point [M]
1. Geographical connection point
2. Voltage level at connection point
Block 3:
Status [M]
1. DER Status
2. Remote control error
3. Remote control enabled
⋮
Block 4:
Active Power Production [O]
1. Remote control enabled, reference tracking
2. Remote control, shedule tracking
3. Allow remote control, reference tracking
⋮
Block 5:
Primary Frequency Control [O]
1. Remote control enabled, continuous primary frequency reserve
2. Remote control enabled, discrete primary frequency reserve
3. Allow control enabled, continuous primary frequency reserve
⋮
Block 6:
Flexible Startup Time [O]
1. Remote control enabled, flexible startup time
2. Allow control enabled, flexible startup time
3. Active and reactive power profiles
⋮
Block 7:
Energy Storage [O]
1. Storage dynamics
2. Storage dynamics, time varying
3. Stored energy
⋮
Block 8:
Log [O]
1. Enable logging
2. Name and data
3. Pointer to attribute that should be logged
Block 9:
Cost [O]
1. Enable costs
2. Cost function

Figure 5.4: Flexibility Interface frame blocks[7]

A flexibility model of a specific DER is constructed, by using the relevant blocks to describe its flexibility. This concept is shown in figure 5.5 and illustrates an information model of a single DER unit, constructed by a collection of blocks comprising the DERs flexibility information model. In the figure, the flexibility information model consists of n blocks, each block being comprised by a number of attributes containing the actual data. The specific flexibility information model for a device is known as a flexibility frame.

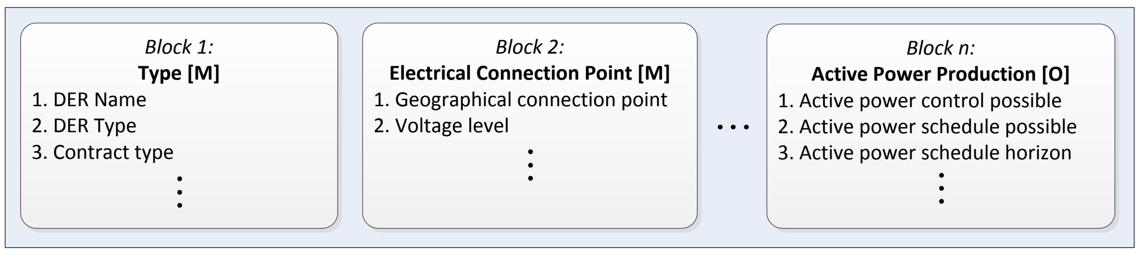

Block 1:
Type [M]
1. DER Name
2. DER Type
3. Contract type
⋮
Block 2:
Electrical Connection Point [M]
1. Geographical connection point
2. Voltage level
⋮
…
Block n:
Active Power Production [O]
1. Active power control possible
2. Active power schedule possible
3. Active power schedule horizon
⋮

Figure 5.5: Example of a flexibility frame[7]

In a flexibility frame, some blocks are mandatory and some are optional, denoted in the figure with [M] for mandatory and [O] for optional.

5.2.1 Flexibility Interface in the Information Model

The work with the Flexibility Interface (FI) done in [7] provides a modeling of the FI concept that are compatible with the IEC 61850 data model. The FI can be suitably inserted into the IEC 61850 standard, as an extension to the IEC 61850-7-420 standard sub part.

As the FI data model is based on the IEC 61850 data model, the information model in the prototype is able to handle the FI with no additional modification. The work done in [7] only concerns the modeling of the FI data model and in order to use the FI in the information model and the prototype, the FI needs to equipped with a base reference and the contents in the FI blocks needs to be mapped to reference names that can be used in the data model, such that the data can be referenced. In figure 5.6, is an example of how the FI Block 1 is mapped to the data model.

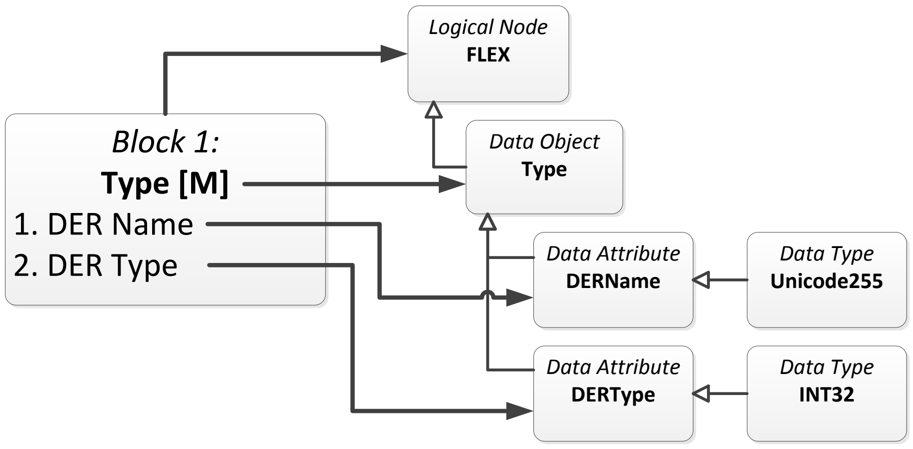


Figure 5.6: Mapping of Flexibility Interface to IEC 61850 Information Model

For the mapping designed in this thesis, the FI Blocks are placed under a Logical Node called FLEX. This design means that FI Blocks can now be referenced using:

<Logical Device> **** FLEX **.** <FI Data Object> **.** <FI Data Attribute>

For example, referencing the FI Block 1 DER Name would look like this:

<LD>****FLEX**.**Type**.**DERName

The contents of the FI Blocks have long names with spaces in them etc. that will not work with the data model. As the FI [7] leaves the specifics of the names open, the names are mapped to abbreviated names in a single word.

Flexibility Interface Block 2 Names	Abbreviated Name Reference
Electrical Connection Point	\FLEX.Ecp
Geographical connection point	\FLEX.Ecp.GeoConPnt
Voltage level at connection point	\FLEX.Ecp.VolConPnt

Table 5.1: Example of mapping of Flexibility Interface Block 2 names to the Information Model

Table 5.1 is an example of how the FI Block 2 is mapped to names with references, that can be used in the Information Model. The name of Block 2 is: Electrical Connection Point and is a Data

Object which means that is contains a number of Data Attributes. The Ecp is a top level Data Object and is referenced directly below the base reference for the FI, which is: \FLEX.Ecp. The contents in the FI Block are then referenced below this Ecp Data Object.

Additionally as with the IEC 61850 data model, Data Attributes such as the 'Voltage level at Connection points' in FI Block 2, may be extended with data attributes for multiple simple types e.g. Integer or Float. In the IEC 61850 standard, basic Data Objects have Data Attributes with multiple simple types exposed as Data Attributes, referenced below the Data Attribute containing the value, as depicted in figure 5.7. (Described in IEC 61850-7-1 [39] section 6.4.3)

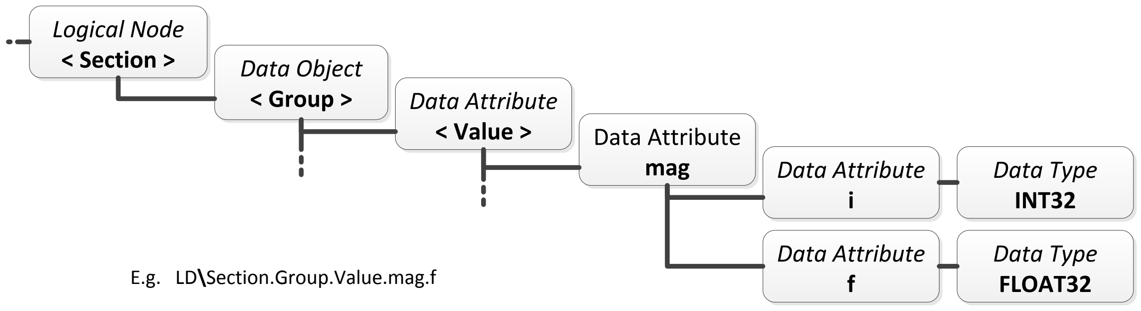


Figure 5.7: Reference with multiple simple types

When the values in the FI are mapped to contain multiple simple types, as in the IEC 61850 data model, the FI Block 2 'Voltage level at Connection Point' would be extended with the references listed in table 5.2.

Type	Reference
Float	\FLEX.Ecp.VolConPnt.mag.f
Integer	\FLEX.Ecp.VolConPnt.mag.i

Table 5.2: References for multiple simple types for FI Block 2 'Voltage level at Connection Point'

The 'mag' part in the reference refers to the way the value is sampled and is relevant for measured values. (The voltage level is a measured value.) There are many other types of values in the IEC 61850 standard, see [39]. Value types related to the 'mag' value includes: instMag for instantly measured, subMag for substituted values and dbMag for deadbanded values. In this example the reference uses 'mag' which is a value that is updated every time the 'instMag' value changes to a different value.

When the IEC 61850 data model and the Flexibility Interface data model is put together in the Information Model, as illustrated on figure 5.8, the result is a tree structure that starts with the Logical Device and then contains a number of Logical Nodes.

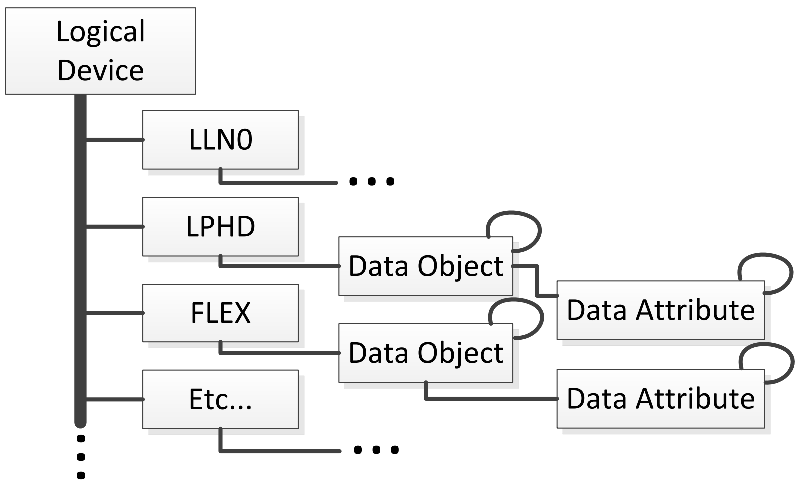


Figure 5.8: Putting the Information Model together

The Logical Node LLN0 is a mandatory node in the IEC 61850, called Logical Node Zero (See [45] section 5.3.4), that represents common information about the logical device itself. The LPHD, called Physical Device Information, (see [45] section 5.3.2) is an example of another node. The LPHD represents information about the physical device. The Logical Node for the Flexibility Interface resides on the same level in the tree as the other Logical Nodes in the IEC 61850 standard.

5.3 Chapter Summary

This chapter has described the IEC 61850 standard and information model. The data in the information model can be referenced using a well defined naming scheme, which allows for a complex data structure. The information model can be inspected using the ASCI interface and the queries can be filtered, using functional constraints.

The iPower Flexibility Interfaces utilizes a service oriented approach, to describe the flexibility provided in DER units. Adding the Flexibility Interface as an additional *Logical Node* to the IEC 61850-7-420 standard, allows the flexibility concept to be exposed through the existing IEC 61850 standard.

CHAPTER 6

Design

This chapter describes the design of the prototype. Based on the findings from the analysis, the design will elaborate from a high level perspective and into the detailed design decisions.

6.1 System Design

An overview of the domain is illustrated in figure 6.1. The contents of the model will be broken down and explained in the following. The prototype solution have three main points, where interaction with the system can occur: the users can access the system via the Web Site running in the cloud, applications such as a VPP can access the system via either a RESTful interface or an XML Webservice and DER controllers can connect to the system through WebSockets. Each entry point is depicted with an actor icon on the domain model. The prototype explores and combines different types of security aspects to authenticate and authorize users with the system and secure the communication. At each of the three entry points, the system is secured with different security measures, that fulfill the requirement for securing the given type of interface. The security is centered around the concept of users and user roles. The security related details are discussed in sections 6.4, 6.7 and 6.8.

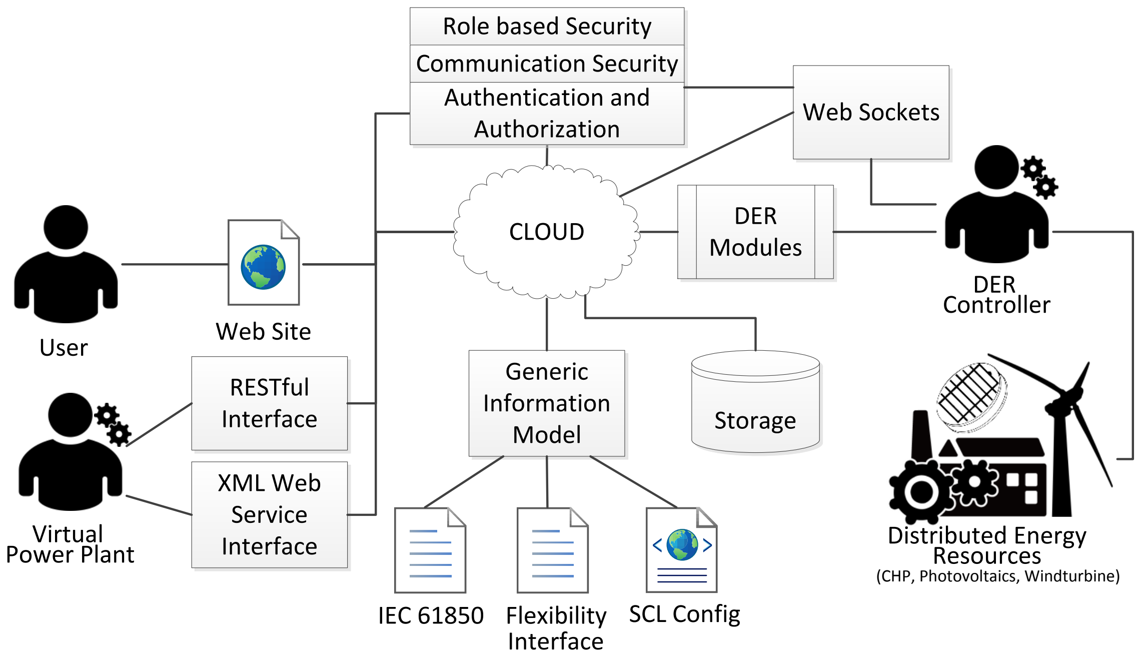


Figure 6.1: Domain model

The concept of dealing with DER-to-cloud communication embraces the idea, that the DER controller should be a basic computing unit, which transmits the raw data that it collects, through lightweight and language neutral Websockets, directly to the cloud service. As a DER unit is created in the system, a custom module must be provided and assigned to the DER unit. The Module is able to handle the raw data from the DER unit and transform it into a generic data structure, that the cloud service can use. The provided module is automatically and securely loaded, as the DER connects to the cloud service. When the module has transformed the received data, it is persisted in the cloud storage, where clients are able to query it. This is described in sections 6.4 and 6.5.

The cloud solution is designed as a multi-tenant cloud application, that is based on the Microsoft Azure cloud computing platform[1]. Using the Website, a user can create and manage DERs, get an overview of the data accumulated on the DERs and perform basic queries. The RESTful service and XML Webservice can be used to perform a wide range of queries, and enables programmatic and automated access from other systems, described in section 6.8.

The central elements for the cloud solution, is the distributed storage and the generic information model. The distributed storage, which is very scalable, enables persistence of large amount of data and can handle the transfer of these data from many distributed DERs simultaneously, see section 6.6. The generic information model, described in section 6.2, is designed to handle the IEC 61850 data model, future additions and extensions to the standard as well as the iPower Flexibility Interface. When an instance of an information model is requested, the model is built using an SCL configuration, which is supplied by the operator of the specific DER unit.

6.1.1 System Coverage

The prototype is developed as a proof of concept, to test the viability of the subject of the thesis. The system prototype, illustrated in figure 6.2, gives an overview of the main components of the system and also the scope that the prototype is designed to cover.

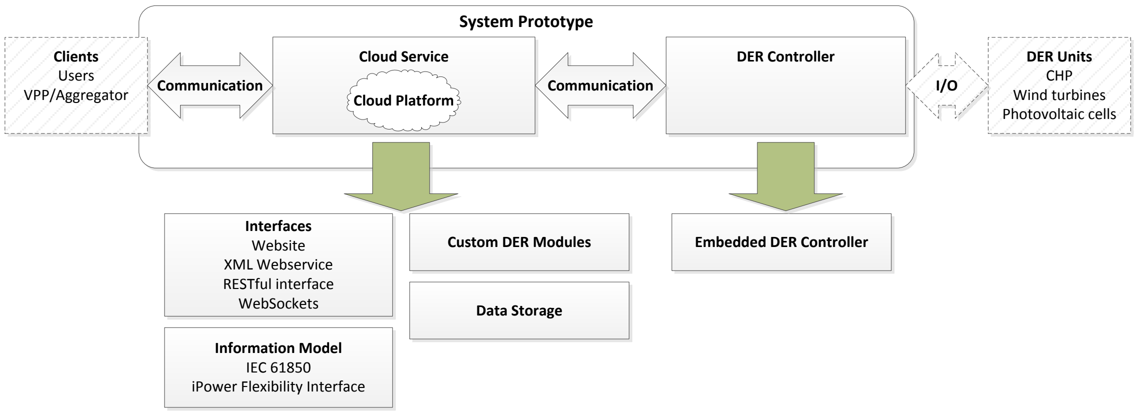


Figure 6.2: System Overview

The system consists of two main components and the communication interfaces they provide. The Cloud Service is a cloud based system and the DER Controller, which is mainly running on an embedded controller close to the DER Units. The Cloud Service can be broken down into

[1]Windows Azure: `http://www.windowsazure.com/en-us/home/features/overview`

its different parts: interfaces for interaction with the service, data model, DER modules and the storage where data from the DER Units is stored.

The components and communication interfaces are described in the following sections.

6.1.2 Mapping of the Standards in the Design

In a substation environment interoperability and communication is needed between the different devices; the IEC 61850 standard was designed to solve this task and handle future changes in technologies. Figure 6.3 gives an overview of how the IEC 61850 standards fits with the design of the prototype. A short description of each standard is given in table 6.1 below. The different parts of the system utilizes different parts of the IEC 61850 standard in order to provide interoperability with other systems. Some of the standards used, are currently in a drafting stage as the usage of modern web technologies are continuously being researched.

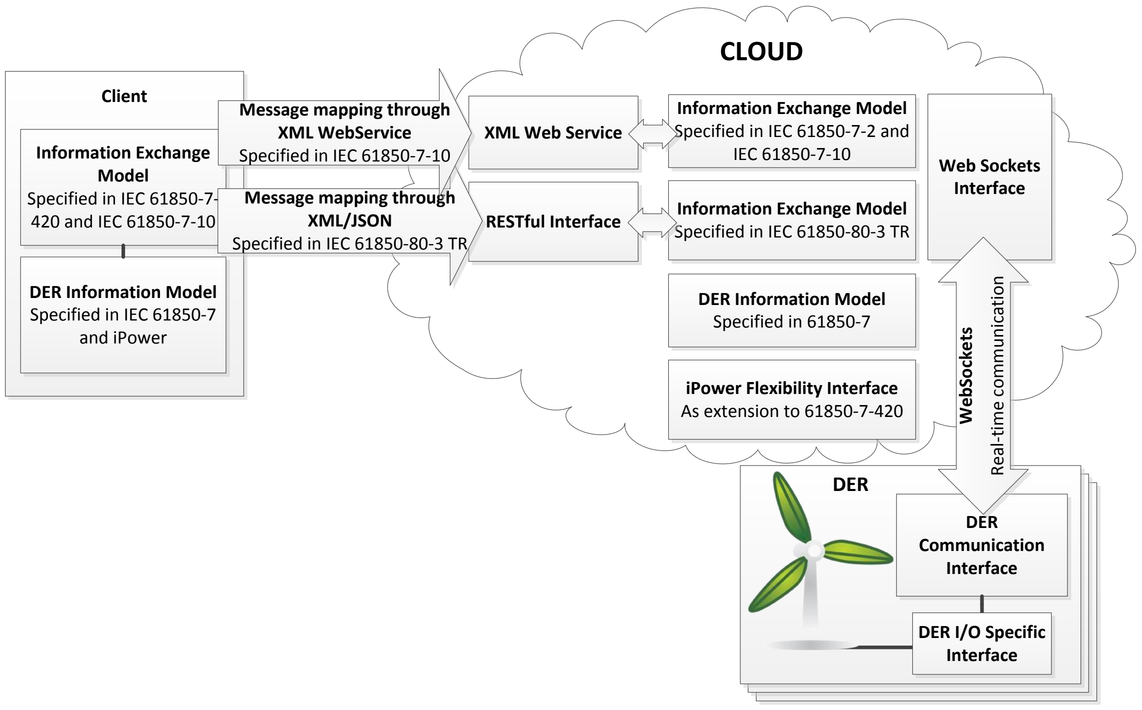

CLOUD
Client
Information Exchange Model
Specified in IEC 61850-7-420 and IEC 61850-7-10
DER Information Model
Specified in IEC 61850-7 and iPower
Message mapping through XML WebService
Specified in IEC 61850-7-10
Message mapping through XML/JSON
Specified in IEC 61850-80-3 TR
XML Web Service
RESTful Interface
Information Exchange Model
Specified in IEC 61850-7-2 and IEC 61850-7-10
Information Exchange Model
Specified in IEC 61850-80-3 TR
DER Information Model
Specified in 61850-7
iPower Flexibility Interface
As extension to 61850-7-420
Web Sockets Interface
WebSockets
Real-time communication
DER
DER Communication Interface
DER I/O Specific Interface

Figure 6.3: Mapping of the standards in the design

A client accessing the system usually has its own implementation of a data model, that can handle the data in which the client is interested. The client would as well contain an information exchange model that can be used to communicate with a DER or the cloud solution. The system has two interfaces facing the client applications: a RESTful interface and an XML Webservice. The standards used to design these interfaces are described in the IEC 61850-7-10, IEC 61850-7-2 and IEC 61850-80-3 TR. Any client that is compatible with these standards and have support for XML Webservices or RESTful API, can communicate with the cloud solution prototype.

The cloud service is based around a generic information model that is designed to handle the IEC 61850 data model. Because the information model is generic, it can handle future extensions to the standards and is able to handle the actively researched iPower Flexibility Interface.

DER-to-cloud is a new concept and is not included in the IEC 61850 standard. There are ongoing efforts to use modern Web technologies such as the cloud and Webservices, with the IEC 61850 standard and this thesis is a contribution to that process. The DER communication in the system supports a method of communicating the data from the DER to the cloud service, where the DER sends its raw data directly to the cloud service, in whatever native format the DER uses. The cloud service then transforms this data, with the help of a custom module, supplied by the DER user, into a lightweight structure and saves the data to the cloud storage.

As this concept is not a part of the IEC 61850 standard, a communication protocol for communication between DERs and the cloud service (see section 6.4.1) has been designed for this prototype.

IEC Standards	**Description**
IEC 61850	Basic communication structure for substation and feeder equipment [44]
IEC 61850-7-2	Abstract communication service interface (ACSI) [40]
IEC 61850-7-420	Basic communication structure - Distributed Energy Resources Logical Nodes [46]
IEC 61850-80-3 TR	Mapping to Web Services – Requirement Analysis and Technology Assessment [43]
IEC 61850-7-10	Requirements for web-based and structured access to the IEC 61850 information models [42]
iPower	**Description**
Flexibility Interface	Flexibility Interface – Information Modeling for Direct Control [7]

Table 6.1: Overview of chosen standards

The overview of the standard parts used in this prototype (Listed in table 6.1) starts with the basic IEC 61850, which is a definition of the data model and structure. This is the basis for the generic information model used in the system, in order to enable clients to query data in a IEC 61850 standard compatible format.

The IEC 61850-7-2 describes the Abstract Communication Service Interface (ACSI) interface, which defines a service interface for clients to inspect the information model and read and set data. The ASCI interface abstracts away the detailed parts of the communication protocol used between clients and the cloud service interface and provides a consistent interface across protocols. The ACSI interface is exposed through the XML Webservice. The requirements for web based access using Webservices is discussed in IEC 61850-7-10.

A different way to represent the data is presented by the RESTful interface, which is a resource based approach [63]. REST [38] is a relatively new technology and is discussed in the draft IEC 61850-80-3 TR.

IEC 61850-7-420 is an extension to IEC 61850 that is concerned with DER units, and in this thesis, the standard is extended with the iPower Flexibility Interface. The purpose of this extension, is to enable the generic information model in the prototype, to handle the IEC 61850 standard and the iPower Flexibility Interface.

6.2 Generic Information Model

The generic information model is a core part of the prototype. It handles the IEC 61850 data model, current and future extensions as well as the iPower Flexibility Interface data model. The information model is based on the previous work in [62] and is designed to be a generic data structure in order to manage every data class, common data classes and data attribute types defined in the IEC 61850 data model specification.

The generic information model closely resembles the IEC 61850 data model seen on the left in figure 6.4 (described in section 5.1.1). The IEC 61850 data model defines a large amount of Common Data Classes that are re-used as much as possible, in order to build up the contents of the Data Classes that makes up the Logical Nodes. The design of the generic information model, seen to the right in figure 6.5, is based on the IEC 61850 data model.

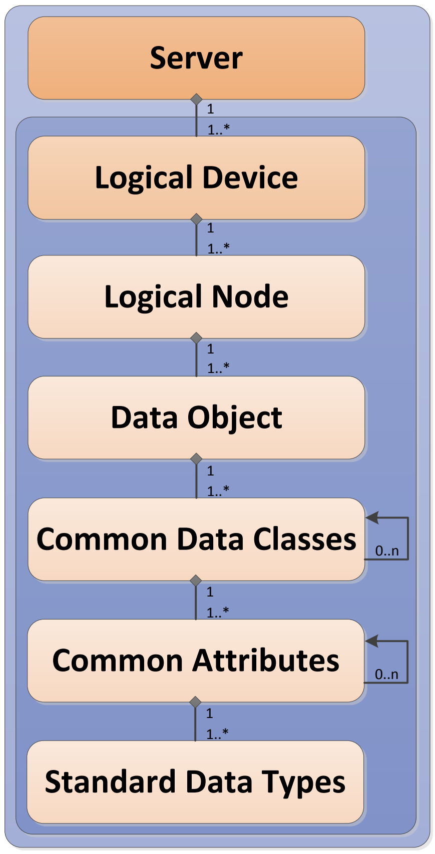


Figure 6.4: IEC 61850 Data Model[40]

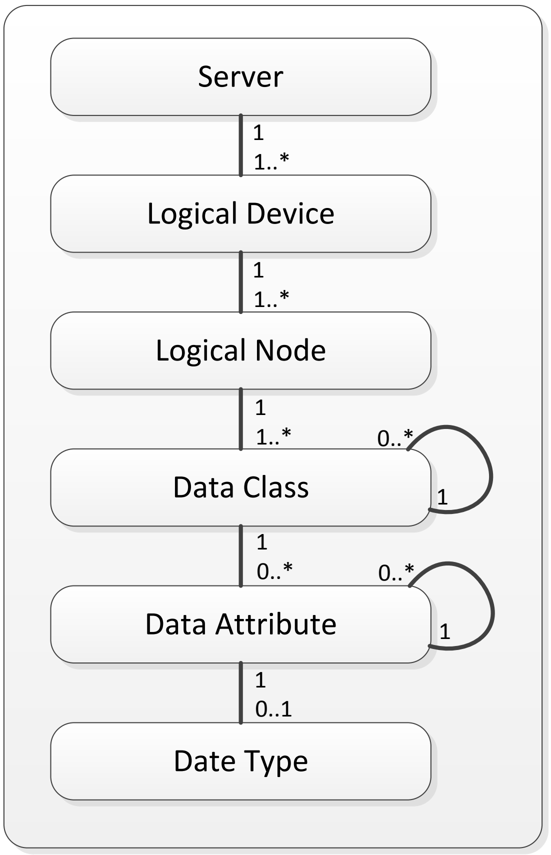


Figure 6.5: Generic Information Model[62]

The generic information model can be seen as a data tree consisting of several nodes. The first node of an information model instance, is the Server Node. The cloud solution is a multi-tenant system, which means that multiple users can own multiple DER units. Therefore the system handles many instances of different information models. Any such instance starts with a Server Node (SN) that represents a physical DER unit.

The Server Node have multiple sub components, called Logical Devices (LD), which can be other sub systems within the DER unit, different computers, terminals or other systems related to the

DER unit. These nodes closely follows the IEC 61850 data model described in section 5.1.1 and so the Logical Devices contains Logical Nodes (LN) that, in the standard are composed of predefined data classes, common data classes and standard data attribute types. However in this generic information model, the predefined data objects and common data classes described in the standard sub part, IEC 61850-7-3 [41], are represented by a generic Data Class (DC) and the common data attributes are represented by a generic Data Attribute (DA).

The Data Class can contain both Data Attributes and/or other Data Classes, which means that the Data Class can be cyclic defined. Likewise the generic Data Attribute can cyclically contain other Data Attributes and/or a Data Type (DT). The last Data Attribute which is meant to hold a data value, (I.e. a DA with no cyclic DA), must then contain a Data Type. A Data Type is a generic class that defines a generic type, this means that a Data Type can contain any data type that are provided by the programming language.

This design allows for a generic information model that can handle all the classes and types defined in the IEC 61850 data model, current and future extensions. The information model can also handle the iPower Flexibility Interface as this is modeled to be compatible with the IEC 61850 data model.

6.3 Cloud Service Design

The Cloud Service application is based on multiple types of cloud computing processes, which supports multiple instances of these computing processes, all running in the Azure Cloud Platform. The design of the Cloud Service is split into different components, see figure 6.6, that each have a separate responsibility. The central component of the Cloud Service is the Core component. The Core handles communication between the other components and is responsible for domain logic, storage and security.

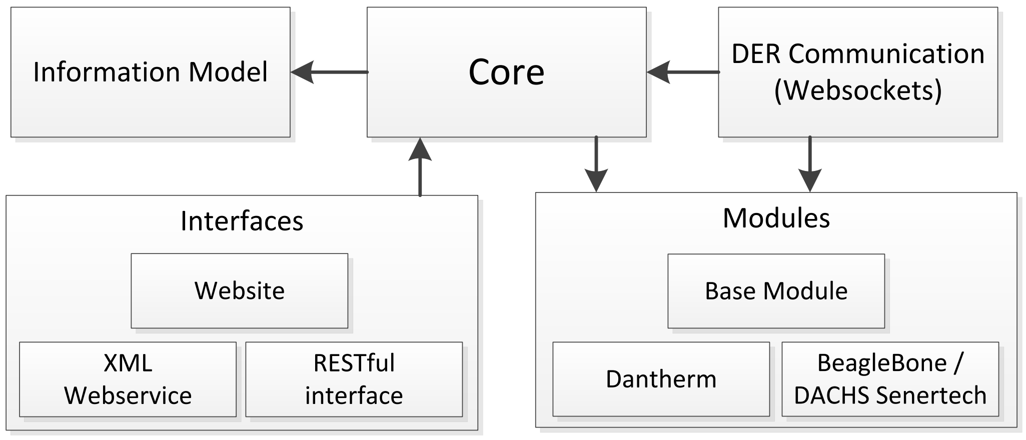


Figure 6.6: Components model

The Information Model component is special, in that in this design, it is completely self contained, with no references to any other component. All the logic that works with the data structure in the Information Model is contained within this component. The goal of this design is to enable complete re-use of the Information Model and the functionality to work with the model in other applications. For example, when building a client application, it is possible to re-use this component and start working with an Information Model that supports the IEC 61850 standard as well as the Flexibility Interface straight away.

The Interfaces component contains the the communication interaction points, where the users and programmatic/automated applications, e.g. a VPP, interfaces with the Cloud Service. The interfaces all communicates with the Core component in order to authorize users, query and update the information model. In the case where a client wants to communicate directly with a DER unit, the communication is passed through the the Core and then to the DER communication component. This in detailed section 6.4.2.

The DER Communication component handles all communication with the DER Units. The communication is based on open web standards using WebSockets that can communicate with WebSocket libraries on a wide range of platforms.

The Modules component is used by both the Core and DER Communication components, however there is no correlation between the two. The DER Communication uses the specific modules, which are based on the base module, to communicate with the DER units and the Core uses the base module to handle the data in a common format.

6.3.1 Layered Architecture

The Cloud Service is logically separated into the layers seen on figure 6.7. The separation of concerns, facilitates the possibility to add or exchange different implementations of any layer at a later time. Each layer presents a well defined interface that supports the separation of the layers and ensures that no layer intrudes on the responsibility of another layer.

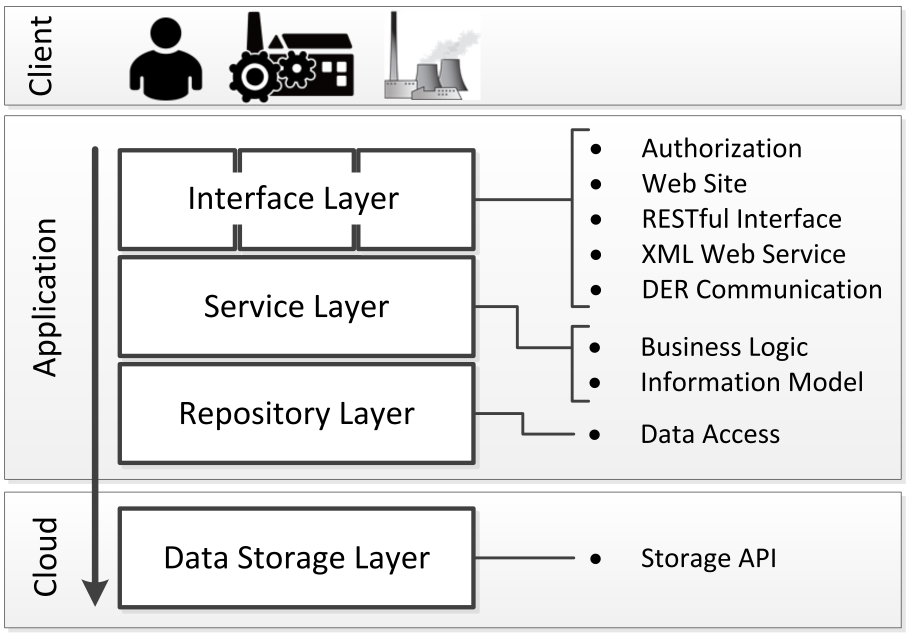


Figure 6.7: Layered Architecture Model

The top layer is the Interface Layer and consists of multiple components, which are the interfaces, that are used to communicate with clients and DER units. The Service Layer is comprised of groups of functionality. The functionality is called “service methods” and provides access to everything in the application: querying and updating the information model, authorization and authentication and communication. The Service Layer is where the business logic of the application is placed. Service methods are able to handle their functionality by communicating with other service methods, communicating with components such as the DER Communication component and by retrieving data from the storage.

When data from the storage is needed, the service methods communicates with the Repository Layer. The Repository Layer is responsible for handling data access to the storage and contains “repositories” which each handles a specific type of data in the storage. A service method can make use of a number of repositories in order to fulfill its functionality. A repository uses the Data Storage Layer to retrieve data from the storage. The repository understands the specifics of how to query the Data Storage Layer where it retrieves the data and maps it from storage entities to domain entities which the service methods works with to perform their functionality. The Data Storage Layer is the specific Storage API provided by the platform of the cloud provider.

6.4 Distributed Energy Resource Design

Figure 6.8 shows an example of a common DER controller (right part) with an embedded IEC 61850 server containing an IEC 61850 data model and exchange model along with the ACSI interface and an implementation of the MMS protocol. This type of DER controller requires a full implementation of the IEC 61850 standard on the controller, which is something that the manufactures, of the different DER units, often cannot provide.

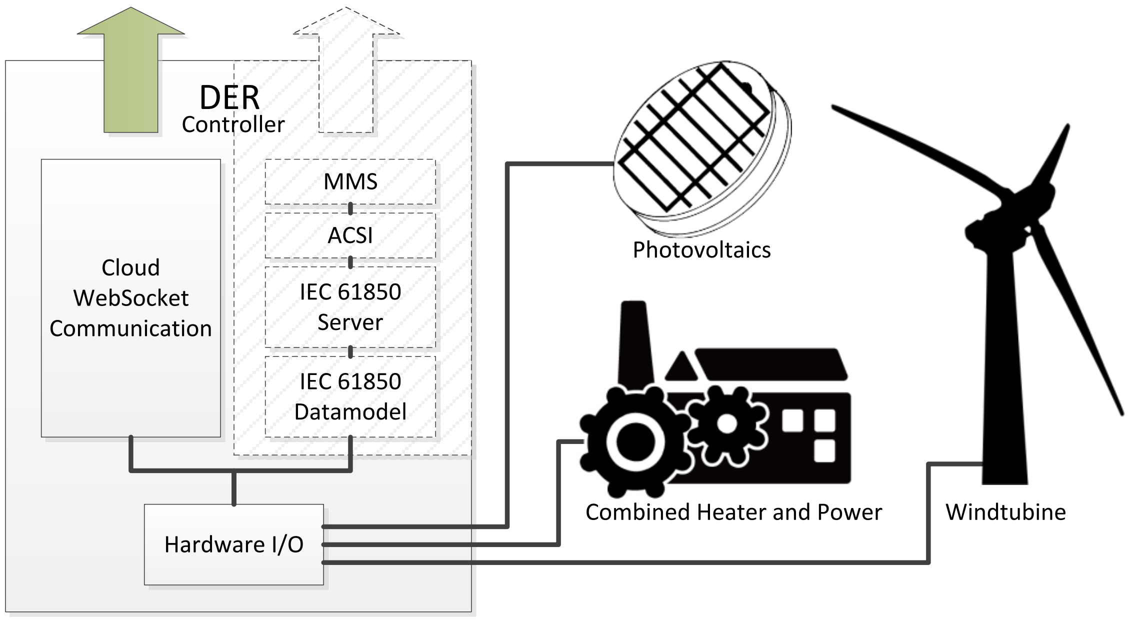


Figure 6.8: DER domain

Utilizing the cloud to handle the IEC 61850 standard and the communication with clients such as owners, operators and VPP/aggregator, means that the design of the DER controller can be more simplistic (left part) as it only needs to be able to communicate with the Cloud Service via internet Websockets.

In this prototype, the DER controllers consists of a communication component, that communicates with the Cloud Service and a hardware I/O component that interacts with the DER unit itself. The DER controller has two-way communication with the Cloud Service via Websockets and can receive commands from the Cloud Service or upload the data from the DER unit.

This design is able to handle many different types of DER units, as the data is uploaded directly from the DER unit to the Cloud Service. The Cloud Service then handles the raw data as described in section 6.5.

Figure 6.9 illustrates the data flow from the DER controller to the Cloud Service, where it is saved in the storage.

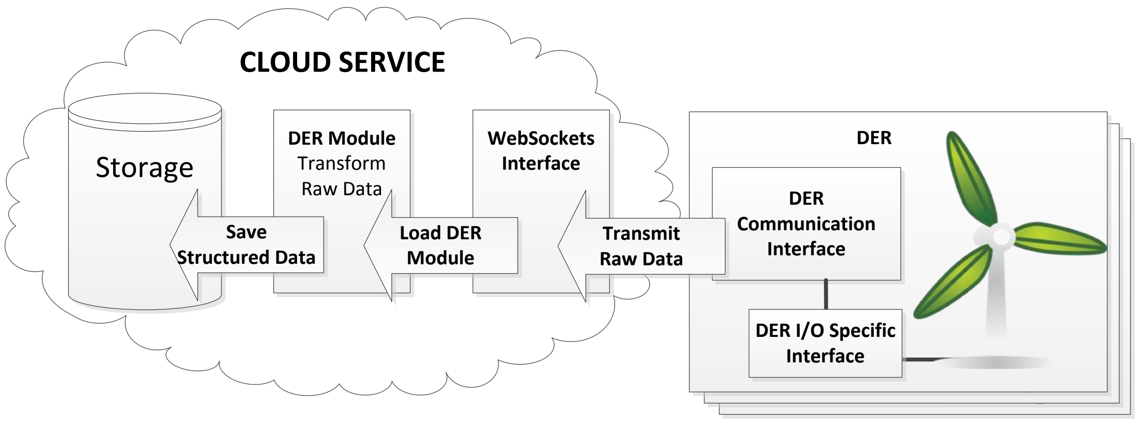


Figure 6.9: DER to Cloud Service Interaction

When the DER controller wants to upload data to the Cloud Service, it opens a WebSocket connection to the WebSocket Interface in the Cloud Service, where it authenticates with the Cloud Service in order to establish the correct user and DER unit subscription to store the data with. As the data is uploaded to the Cloud Service, the specific module for that DER controller is loaded and the data is transformed into a general structure that the Cloud Service can handle. The structured data is finally saved to the cloud storage.

DER Data Structure

Representation of Value	*Mandatory*
Time Stamp	*Mandatory*
Type of Value	*Mandatory*
IEC Reference	*Mandatory*
Description	*Optional*
SI Unit	*Optional*
Multiplier	*Optional*

Table 6.2: DER Data Structure used for storing in the cloud storage

An overview of the general data structure that the raw data is transformed into, is presented in table 6.2. The modules described in section 6.5 is responsible for transforming the raw data uploaded from the DER controller into this structure and the modules contains the knowledge of how to extract the data values from the raw data and designating an IEC reference. The important part of the data structure is the “value representation”, “type of value” and “IEC reference”. The raw value data that is stored in the “value representation” field, is stored as a string representation of the data value and is restored into the correct data type on retrieval based on the “type of value” field. The IEC references are defined within the custom DER modules and stored along with the data in the “IEC reference” field.

6.4.1 DER to Cloud Communication

Transmitting raw data from a DER controller to the Cloud Service is a new concept and in this prototype, a protocol is designed to facilitate two-way communication between the Cloud Service and the DER controller. The protocol described in table 6.3, is centered around a command indicator and a dynamic object that can contain any number of data.

DER Communication Protocol

Command Indicator	
0	Default value of an unknown command
1	Command value for creating a new user
2	Value for deleting an existing user
3	Load a module
4	Log in to the Cloud Service
5	Log out of the Cloud Service
6	Transmit raw data to the Cloud Service
7	Transfer a binary module to the Cloud Service
21	Send a custom command
255	Error message
Dynamic Data	
Data	The data part of the protocol is based on a dynamic object and can contain any number of data.

Table 6.3: DER Communication Protocol

The DER controller communicates with the Cloud Service using a message that contains a command indicator, that describes what kind of action should be taken with the data included in the message. The data part in the message is a dynamic object and can contain any combination of data. The language used in the WebSocket communication is JSON [15] which easily allows for any number of data to be appended to the data part.

6.4.2 Client to DER Communication

A client or VPP can send commands directly, via the cloud, to a DER unit that is connected to the Cloud Service. Figure 6.10 shows the flow of how the command is sent to the DER unit and how the resulting changes can be read by querying the information model afterward.

The client can use either the RESTful interface or the XML Webservice to send commands to a DER controller. The Cloud Service uses two different cloud application processes, one is where the client interfaces resides, called a “Web Role” in Azure Cloud Platform terms. The other process is where the DER WebSocket communication interface is running, called a “Worker Role”. These two types of cloud application processes can have multiple instances, distributed out on multiple machines, which makes up the Cloud Platform. Since the Cloud Service can consist of multiple instances of both the client interfaces and the DER interface, a DER unit may be

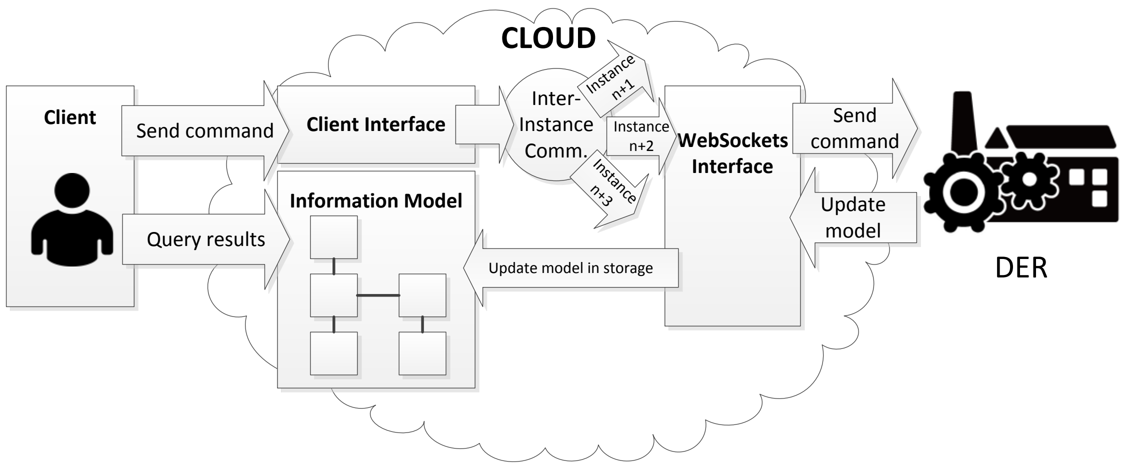


Figure 6.10: Client to DER Communication data flow

connected to any instance of a DER WebSocket interface. Which one, is unknown to the specific client interface, used by the client to send a command message. The client interface needs to find the correct DER WebSocket instance where the DER unit is connected. This is solved using inter-instance communication between the instances, where the command message is published by the client interface and picked up by the correct DER WebSocket interface. The command is then sent to the DER unit.

The design of the components in this prototype, at this point, allows for one-way communication between the clients and the Cloud Service. As such, the clients initiates action, when 'they' are interested in information from the Cloud Service and are given a response at that time. Figure 6.11 illustrates that, as the client requests that a command be sent to a DER unit, the Cloud Service responds with a reply, stating if the message was successfully sent to the DER unit. This means, that a reply, does not contain a response with the results of what changes the command may have caused in the DER unit, but rather whether or not the DER unit has received the message.

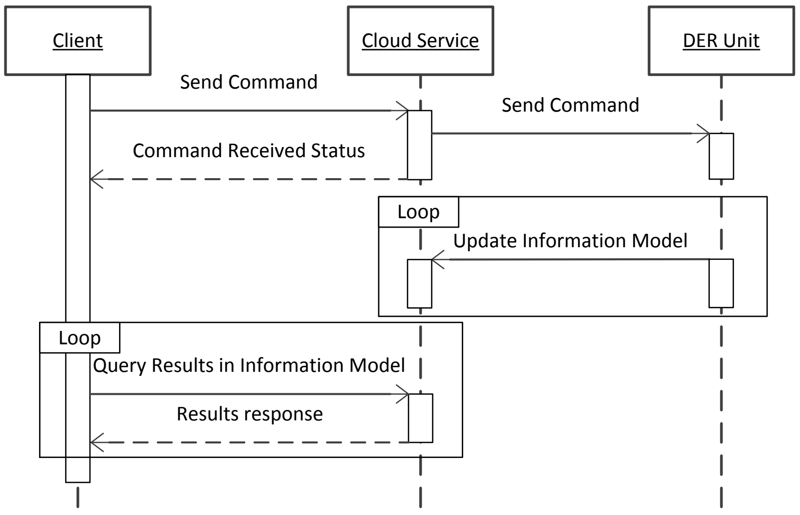


Figure 6.11: Client to DER Communication sequence

It is the responsibility of the DER unit, after receiving a command message, to react to the command and perform any changes that the command may specify. These changes can, depending on the actions performed, be instant or be carried out over time. The expected behavior of the DER unit, in this design, is that the DER unit continuously updates the information model in the Cloud Service, with the status of the DER unit. This means that, when a received command results in changes to the DER unit, these changes are transmitted to the Cloud Service when the DER unit chooses to perform a update. It is at this point, the client will be able to read the results of the command sent to the DER unit, by querying the information model in the Cloud Service.

6.5 Dynamic Modules

To execute the concept of having simple DER controllers, that sends raw data directly to the Cloud Service. A means is needed to handle that raw data and transform it into a general format, that can be used and stored in the Cloud Service.

Custom modules tailored to the specific DER unit is used to handle the raw data, uploaded by the DER unit. As shown in figure 6.12, a specific module is based on a Module Interface, that provides the basic functionality and module API.

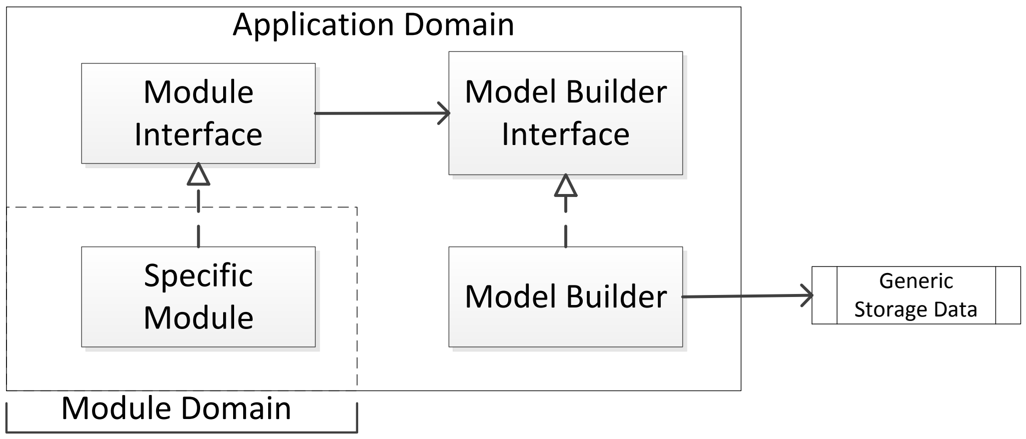


Figure 6.12: Dynamic Module Domain

The individual modules should be able to be developed by a third-party developer. A Builder Pattern [26] has been used to describe the Model Builder. It uses an instance of the Module Interface, in order to produce a data structure that is compliant with the general data structure (see table 6.2). Using the Builder Pattern, the Model Builder disregards how the data is passed into the Model Builder, as long as it inherit the Module Base. In order to avoid exposing the internal Model Builder to a third-part developer, a Factory Pattern [26] is used, as it allows the creation of an object without exposing the internal logic.

To build an instance of the general data structure, in preparation to store the data, the module uses the Model Builder to insert each Data Attribute extracted from the raw data in the structure. The specific module and the Model Builder shares an interface, so that the Module Base and Module Interface can be provided in an external library to developers of custom DER modules. The Model Builder is a part of the Cloud Service and is inserted into the specific module, when it is loaded, in the process of uploading data from the DER unit.

6.5.1 Dynamic Loading

When a DER connects to the DER WebSocket Interface, the specific module for that DER unit is dynamically loaded into memory by the Cloud Service. For security reasons this module is loaded into a different application domain from the Cloud Service itself. It does this using Reflection and Application Domains[2]. The specific modules are loaded into each of their own Application Domain and from here, they can be executed when needed. Placing the modules in each of their own Application Domain provides every module with a sandbox environment and the possibility to load and unload each module. As the Module Builder have access to each of the Application Domains in use by the Specific Modules, it can use the the modules to transform the raw data to the general data structure, and save it in the cloud storage.

6.6 Storage Design

The storage design is based on a key/value storage model, more precisely the Microsoft Azure Table Storage. Access to the data in the storage is based on the Repository Pattern [58] which separates the business logic from the data access logic. The Repository pattern is used by the Repository Layer.

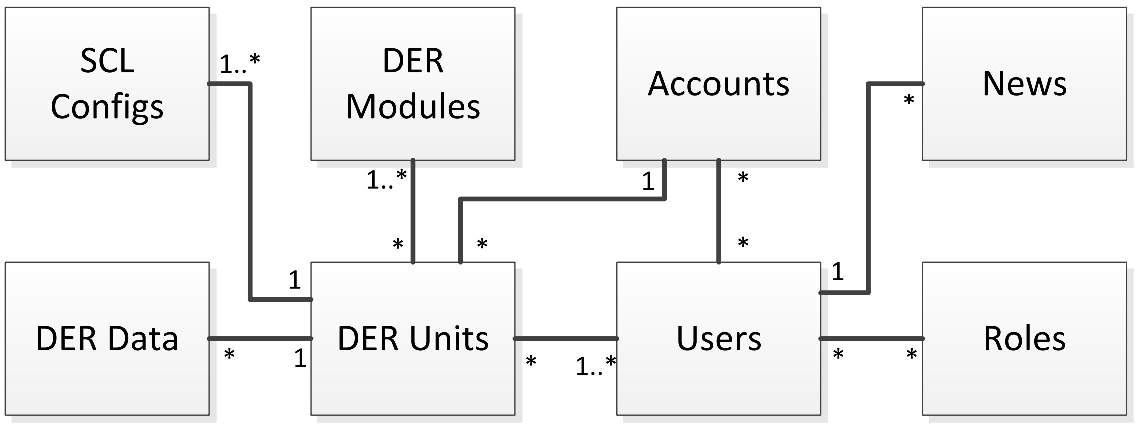


Figure 6.13: Storage design

The entities and their association in the storage is shown in the conceptual model in figure 6.13. The associations in the model are depicted in order to show the relations between the entities to help the reader. The storage is based on a key/value store in which, by the nature of a key/value store, there does not exist any persisted associations between the entities. Any relations between entities are handled by the application logic, created when the entities are retrieved from the storage and then mapped to instances of classes. It is also in the application logic that validation of constraints between entities takes place.

The model shows, that a DER unit can have multiple DER modules associated, however only one module can be set as active for a DER unit at a given time. The same DER module can be used by many DER units, e.g. if multiple DER units of the same type and configuration is connected to the cloud application.

[2]http://www.brad-smith.info/blog/archives/500

The DER data are the data uploaded from the DER units and are specifically tied to one DER unit. An SCL configuration is also tied to one specific DER unit and a DER unit must have one SCL configuration defined as active; but can have multiple associated.

An account is used to associate users with the DER units they have access to. A user can be associated with many accounts and an account can naturally have multiple users. Users can have many roles and roles can be associated with many users.

News items, shown on the landing page of the website, is associated with the user that created the news item.

6.6.1 Model Generated Entities

The storage model in the prototype is designed as a model by using a graphical editor, provided by Visual Studio, which is then transformed into the entities used in the code.

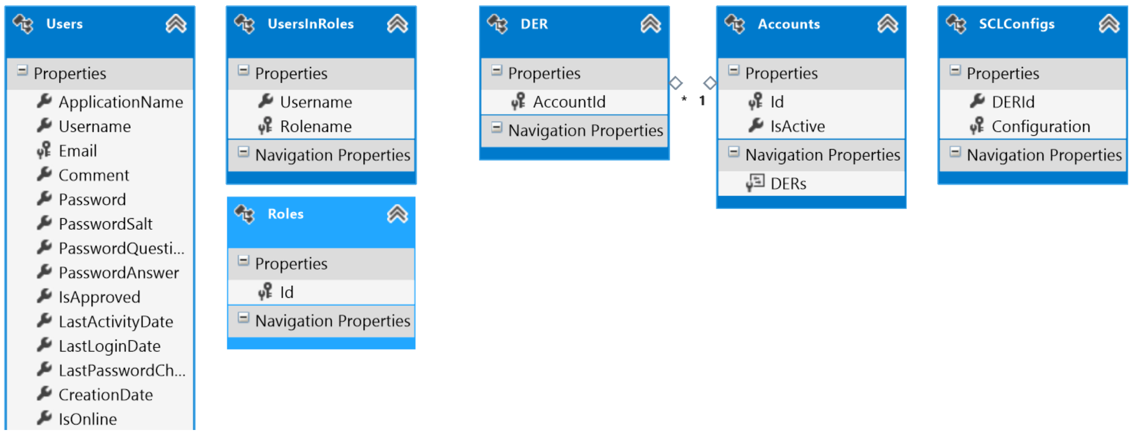


Figure 6.14: Snippet of the model used to generate the storage entities

Figure 6.14 is a snippet of the model from the development tool, Microsoft Visual Studio 2012, showing a selection of the entities used with the storage.

The Microsoft Azure Table Storage is, at the time of writing, not supported by the Visual Studio modeling tools and so for this project, customized code generation templates was created. The templates enables the transformation of the entity model into actual code which can be used with the Azure Table Storage.

As seen on the model snippet in figure 6.14, which shows that in contrast to the storage design in figure 6.13, there are no associations modeled. These are not supported and persisted in the storage and the association between DER and Accounts is only used after the entities have been retrieved.

6.7 Users

The interfaces in this prototype all makes use of different security schemes to secure communication and authenticate users, however all the schemes draws on the same central user and role based system. The user and role based system is based on the Microsoft .Net Membership framework [55] with custom extensions to enable the persistence of users and roles in the Azure Table Storage, which is not directly supported by the .Net Membership framework.

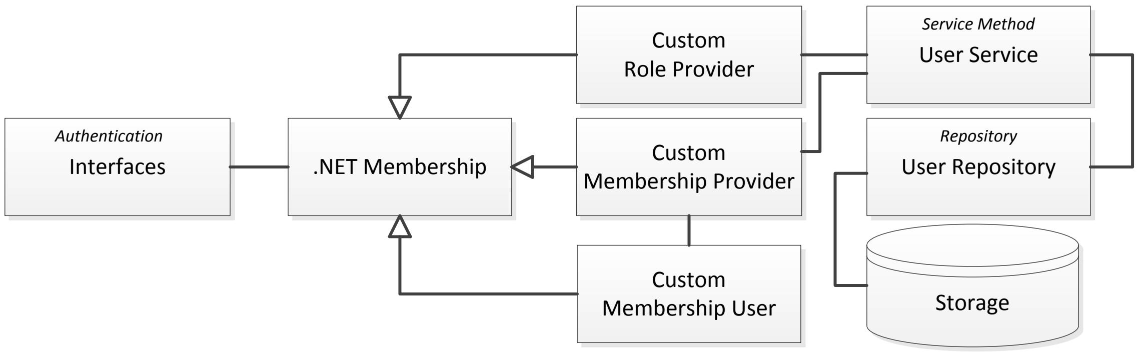


Figure 6.15: Membership components

Figure 6.15 shows the components involved with the authentication and authorization of users and their roles. The different interfaces uses the Membership API to validate and manage users and roles and the Membership framework in turn uses the Service Layer methods to load and save user/role data. The Service Methods uses the User- and Role-repository to access the storage. By using custom provider extensions for the Membership framework, it is possible to use the Azure Table Storage through the Service Methods and Repositories. Additionally the user entity that represents the user in the Membership framework can be extended with custom properties.

Three roles are defined in the system, which are assigned different privileges. The privileges are described in table 6.4.

Privilege	Customer	Operator	Administrator
Create DER unit		•	•
Assign DER unit to user		•	•
Upload DER module		•	•
Assign DER module to DER unit	•	•	•
Create and manage users			•
View data on DER units	•	•	•
Upload data via Websockets interface	•	•	•
Query the information model	•	•	•
Update the information model		•	•
Send commands to the DER unit		•	•

Table 6.4: User privileges and roles

The role Customer represents the most basic user of the system. The Customer is only allowed to look up information and query the information model, from DER units that have been assigned to the Customer. The Customer is also allowed upload data via the WebSocket interface as this role is also assigned to the DER units to use for authorization when authenticating during transmission of the DER data.

The Operator role has more privileges, which would suit a DER owner or VPP/aggregator that needs to: query the information model, update the information model and send commands to the DER unit. The Operator is responsible for assigning the DER units to the different users and can create the DER units in the system.

The Administrator creates and manages users and have full across all DER units. The administrator role is meant for the operators of the Cloud Service to assist Operators, Customers and create new users.

6.8 Interfaces

The RESTful interface and the XML Webservice both uses the same service methods to query and update the information model. The Website additionally uses more of the service methods available in the service layer to provide additional information. Figure 6.16 illustrates how a request coming in on the interfaces is received by the Cloud Service, where it is authenticated, authorized and accepted by the Cloud Service.

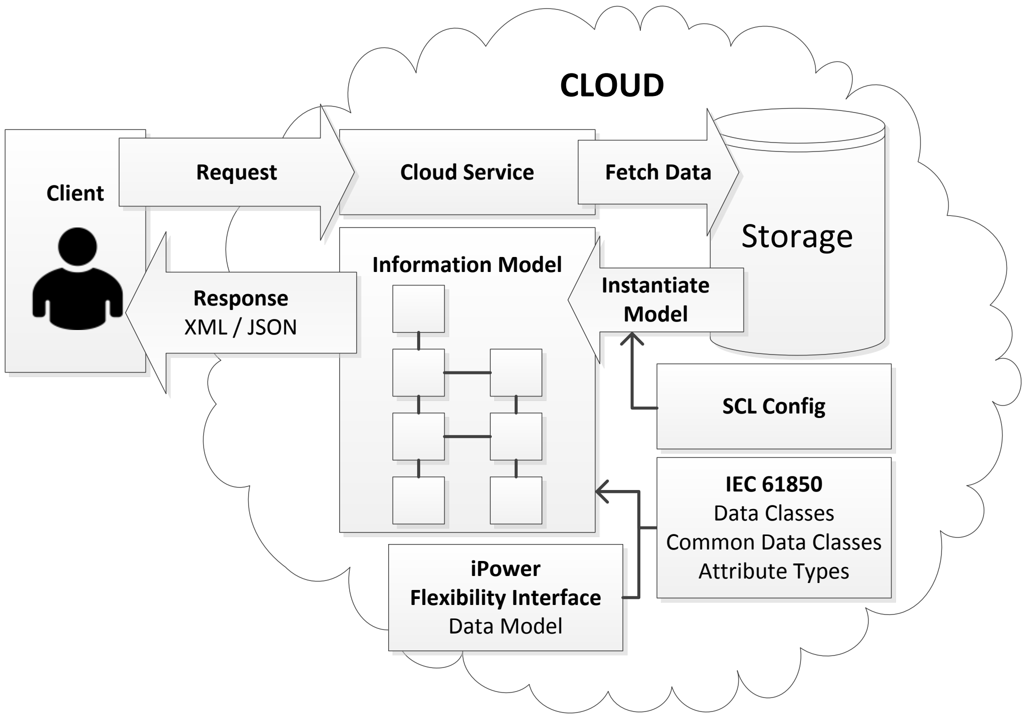


Figure 6.16: Correlation between client and cloud service

In the case where a request for example is a query on the information model, an instance of the information model is created and filled with data retrieved from the storage. The information model is built using an SCL (Substation Configuration Language) configuration file (see [62]) that is supplied when the DER unit is created. The SCL configuration file describes the logical contents of the DER unit. The information model is then queried according to the request and the result is sent back to the user through the interface used.

6.8.1 Web Site

The Website is designed using the MVC 4 (Model View Control) pattern [56]. The MVC 4 pattern separates the Web application into three components: the Model, the View and the Controller. The Model contains data used in the website and logic to work with this data, the Controller handles user interaction and uses the Model in response to the user and the View displays the website's user interface. In this prototype, the Website has a Model, however this model is only concerned with the Website logic in order to make the Website work and move data between the pages. The business logic of this prototype resides in the Service Layer in the Core component (see section 6.3.1).

Figure 6.17 gives an overview of the main parts that make up the Website.

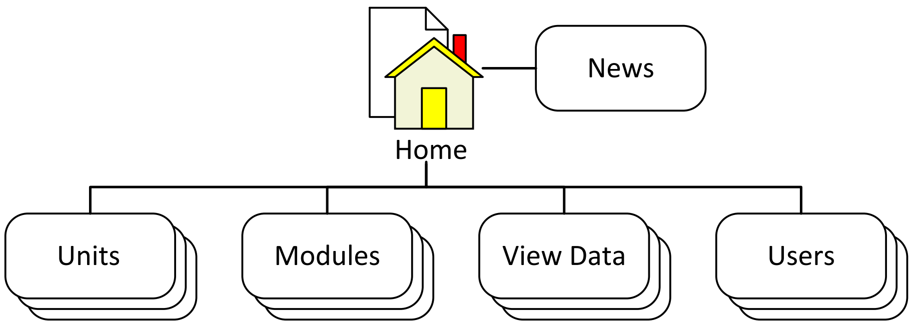


Figure 6.17: Website overview

- The landing page of the Website is the Home section. The Home section provides a news stream with information about the site.
- The section Units handles the creation and managing of DER units.
- The section Modules is responsible for updating, assigning and managing the custom DER Modules for the DER Units.
- The section View Data allows the users of the Website to view detailed information about the data the DER units have uploaded to the Cloud Service.
- The section Users handles the cration and managing of users that have access to the Website, RESTful interface, XML Webservice and DER WebSocket interface.

From these design concepts a simple web site can be created to handle the needed functionality.

6.8.2 RESTful Interface

The RESTful interface [38] is based on a concept of resource-based access, the object references in the information model is mapped to be compliant with an URL. Each resource in the data model is accessed via URL nodes and updated the same way. As the naming convention of the IEC 61850 standard resembles a URL, changing the . (dot) to a / (slash) would satisfy the URL requirement[63]. RESTful communication uses the standard HTTP methods such as GET, PUT, POST and DELETE to manipulate the resources that are exposed through the interface.

This is an example of an object reference for the flexiblity interface node: ECP Voltage Connection Point.

LogicalDevice\FLEX.Ecp.VolConPnt.mag.f

When the reference is mapped into an URL, it looks like this:

http://hostname/api/unit-id/LogicalDevice/FLEX/Ecp/VolConPnt/mag/f

To retrieve the data using the URL in the example above, a HTTP GET request would be used and the server would respond with the data object tree, starting from the specified resource in the URL. To set data, the same URL would be used, but by using a HTTP POST request and submitting the data to set in the request body.

The RESTful architecture is resource based with an URL scheme to address the data whereas the ACSI interface uses a set of defined methods to access the data. As the approach to access the data with the RESTful interface is a different paradigm from the ACSI interface, the ACSI interface is replaced by the RESTful interface.

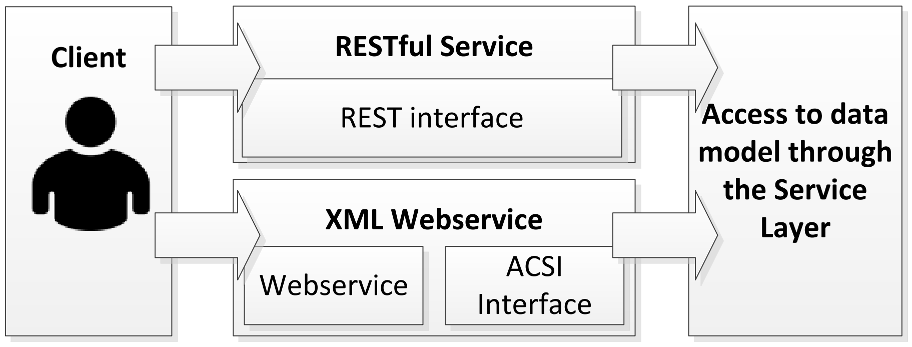


Figure 6.18: ACSI in RESTful and XML Webservice interfaces

Figure 6.18 shows how the RESTful interface replaces the ACSI interface from the IEC 61850 standard and how the XML Webservice Interface, described in section 6.8.3, makes use of the ACSI interface.

The approach to map the ACSI method to the RESTful resource based architecture is based on the ideas described in [63] of which a summary is shown in table 6.5.

URL	Output	ACSI equivalent
/	List or hierarchy of all logical devices in the system	GetServerDirectory (GetAllDataValues) (GetDataValues)
/[LD]/	List or hierarchy of all logical nodes within a device	GetLDDirectory (GetAllDataValues) (GetDataValues)
/[LD]/[LN]/	List or hierarchy of all data objects within a logical node	GetLNDirectory (GetAllDataValues) (GetDataValues)
/[LD]/[LN]/[DO]/	List or hierarchy of all data attributes and child-data objects within a data object	GetDataDirectory GetDataDefinition (GetAllDataValues) (GetDataValues)
/[LD]/[LN]/[DO]/[DA]/	Hierarchy of all child-attributes within a data attribute	(GetAllDataValues) (GetDataValues)

Table 6.5: Summary of RESTful mapping from ACSI [63]

6.8.2.1 *Security in the RESTful Interface*

The RESTful interface is secured using two methods that are not necessarily used at the same time. Figure 6.19 shows the two options, which are HTTPS between the client and the cloud application and using public/private key in the form of RSA [14] encryption to secure the login credentials in the client request.

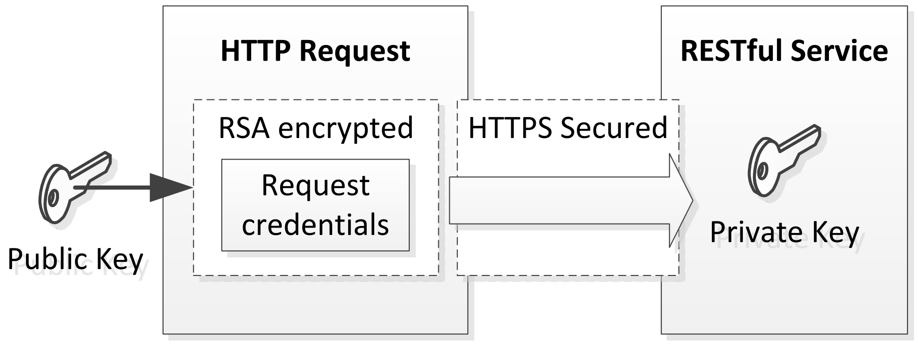


Figure 6.19: RESTful interface security

The first option is to secure the communication using HTTPS between the client and the cloud application where the identity is verified by using HTTPS certificates. This is straight forward and provides adequate security for the communication.

In the case where HTTPS transport security is unwanted or unavailable[3], this project also explores the option of using asymmetric public/private key encryption using the RSA algorithm. The design is to encrypt the user credentials in the HTTP request using the public key which is then decrypted

[3]An asymmetric public/private key encryption scheme was used in this project, resulting from both an interest in the subject and the fact that a valid HTTPS certificate was not available; which leads to a self-singed certificate which immensely complicates development.

in the cloud application using the private key. Using a public key for the request means that anyone can encrypt requests for the cloud application, but only the cloud application which hold the private key can decrypt the request.

When only encrypting the user credentials, the communication is susceptible to a form of attack called Replay-Attack[4]. An example of a Replay-Attack is when an attacker captures and repeats a valid data transmission where the encrypted identity is kept but the data content is changed. There are different methods to safeguard against this, such as using session tokens, random numbers, one-time passwords, message authentication code (MAC) and timestamping. In this prototype, timestamping is used to prevent Replay-Attacks.

A timestamp is added to the encrypted credentials and when the request is sent to the cloud application, the request is only accepted, if the timestamp of the request is within a reasonable time frame of when it was received.

When using the RSA to encrypt the user credentials without using HTTPS, the body of the request and the response is sent in clear text. If these needs to be secured, this could be done by for example providing a symmetric encryption key along with the encrypted credentials which can be used to encrypt the request body and the response.

6.8.3 XML Web Service

The XML Webservice uses the Simple Object Access Protocol [77] (SOAP) to send and receive messages over HTTP. The XML Webservice is based on the ACSI interface, as illustrated earlier in figure 6.18, and defined in the IEC 61850-7-2 standard sub-part [40].

ACSI methods	**Description**
GetServerDirectory	A list of all the logical devices available on the DER unit
GetLDDirectory	A list of all the Logical Nodes within the Logical Device
GetLNDirectory	A list of all the Data Objects within a Logical Node
GetDataDirectory	A list of all the Data Attributes and child-Data Objects within a Data Object
GetDataDefinition	A list of all the Data Attributes definitions within a Logical Node
GetAllDataValues	Returns all Data Attributes, having the same Functional Constraint, within a Logical Node
GetDataValues	Returns the values of the Data Attribute

Table 6.6: Summary of ACSI methods used in the prototype

The table 6.6 is a summary of the most basic ACSI methods used in the XML Webservice and a description of the data they return. As the cloud solution is multi-tenant based and provides access to many DER units, the process of using the ACSI methods requires the client to additionally inform which DER unit the ACSI queries are performed on.

[4]Replay-Attack: `http://msdn.microsoft.com/en-us/library/aa738652.aspx`

6.8.3.1 *Security in the XML Webservice Interface*

The communication in the XML Webservice is secured using the standard security capabilities provided by the cloud platform and chosen development platform. In this prototype, which is running on the Microsoft Azure Cloud Platform using .NET and Windows Communication Foundation, the XML Webservice is secured using per message security [64] [52]. Per message security uses certificates to ensure the server identification and encrypts each SOAP message, between the client and the XML Webservice individually, which additionally enables the use of customized user authentication and authorization (described in section 6.7).

6.9 Chapter Summary

Moving to a cloud-based solution changes a lot of the design aspects, in the existing domain model of the smart grid. To handle multiple distributed energy resources, each DER connects (using a controller) to the cloud solution through a WebSocket, allowing for near real-time duplex communication. Each DER sends the raw data directly up to the cloud, depending on the time intervals defined in the DER unit. To handle the different raw data structures in the cloud, each DER has an associated module. The modules could be developed by the DER manufacturers, as long as they comply with an interface and a general data API for storing the data. The raw data is processed by the module and then stored in the cloud storage.

The cloud solution architecture is designed as a layered architecture, limiting each layer for passing layer-specific values, but also provides the means for changes e.g. changing storage type. When accessing the stored data, a RESTful interface should allow programmable access to query or update data. The information model of the data is based on the information model found in the IEC 61850 standard. When making a query in the RESTful interface, the object reference in the IEC 61850 has been mapped into an URL, by changing the . (dot) to a / (slash). This allows for easy access to each of the nodes in the information model, and with functional constraints, it also provides the functionality for making a filtered query. When the data are retrieved from the storage, it should then (using the SCL associated with the DER module) transform from the general data structure into an IEC 61850 compliant information model. To handle a more user friendly access to the stored data, a web site has been designed to handle the user-, DER- and module-management.

CHAPTER 7

Implementation

This chapter describes how the prototype is implemented based on the design. In favor of condensing the implementation description, at most one behavior model will be used to illustrate each section. It will describe the most interesting behavior, which best covers the application logic. Class diagrams will be minimized and larger versions can be viewed in appendix F.

7.1 System Structure

Figure 7.1 shows the components that makes up the cloud application. The implementation of the components separates the components into sub projects, mostly class libraries in the form of DLL files. The components centers around the *Core* component which provides access to the Azure Table Storage and enables the components to communicate with each other.

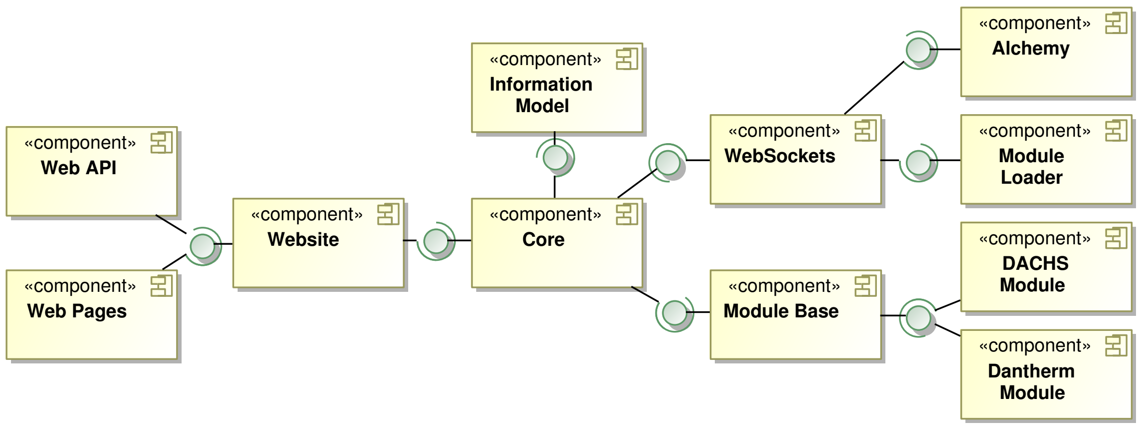


Figure 7.1: Cloud Application Components

The Website is implemented as an Microsoft Azure Web Role project using the Microsoft MVC v4 Website template and contains both the RESTful and the XML Webservice interfaces.

The Information Model, as described in the design, is realized as a component with no dependencies on any other component and the resulting DLL file can be imported into and used by other .NET project. The Information Model component also contains logic to traverse the data tree model which means the Information Model component provides both the data model supporting the IEC 61850 standard and the iPower Flexibility Interface and application logic to work with the model.

7.2 Implementation of Layered Architecture

The implementation of the service layer and repository layer parts of the layered architecture, described in the design (section 6.3.1) are illustrated in figure 7.2.

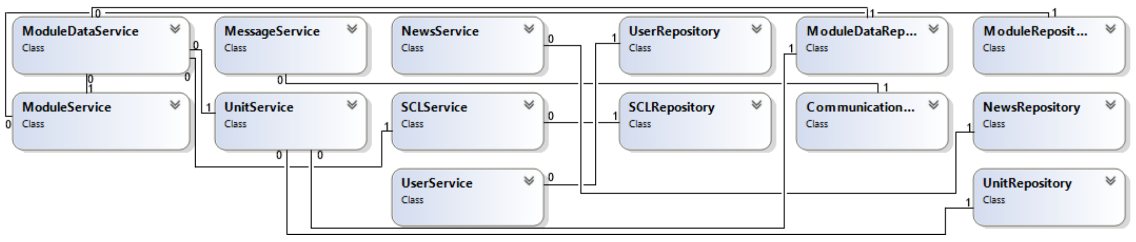


Figure 7.2: Class diagram: Service Layer and Repository Layer Combined

The classes are collapsed in the figure, to provide an overview of the service classes and repository classes that have been implemented, in order to fulfill the functionality described in the requirements. However a larger version, with all the details can be seen in the appendix figure F.1 and F.2

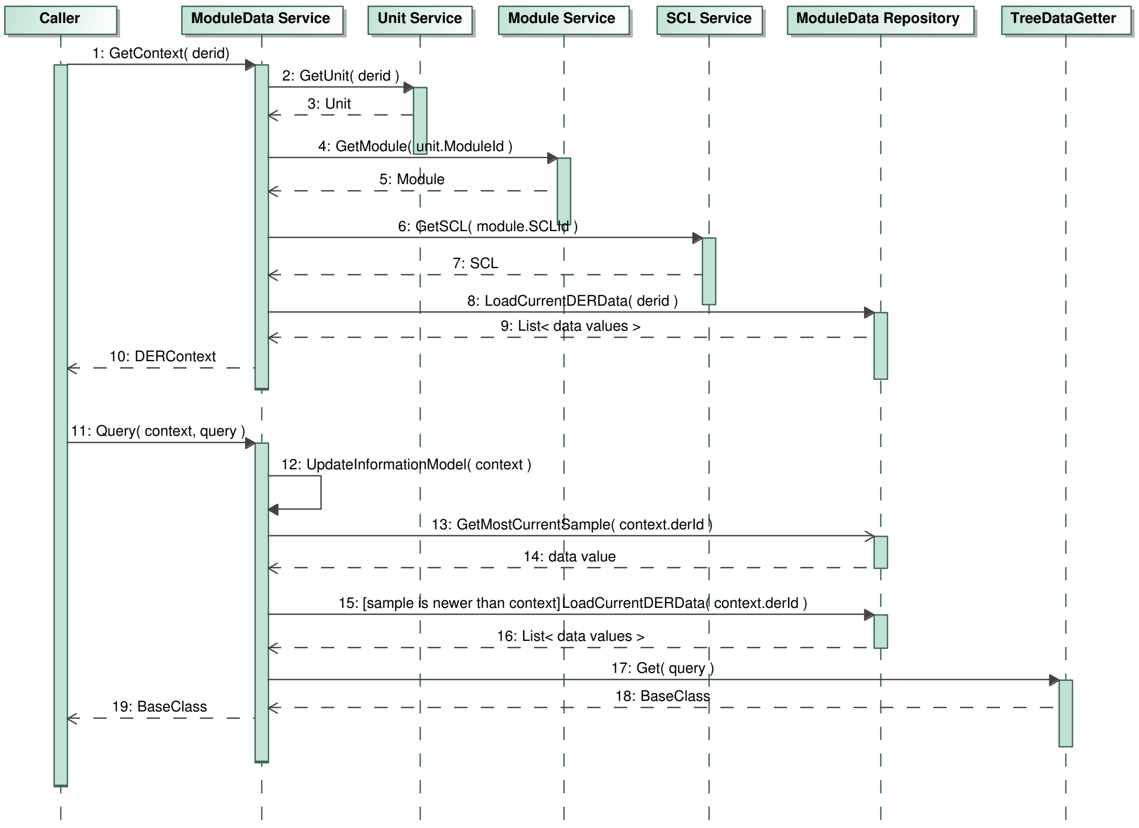


Figure 7.3: Behavior: Query Information Model using the Service Layer

The behavior diagram in figure 7.3, shows an example of the service methods and repository methods working together to query the information model and retrieve a cloned model of the most current model stored in the table storage.

The sequence is a two-part process in order to query the information model. The first part retrieves a *DER Context* for a given DER unit, specified by providing the *GetContext* method with an ID of the DER unit. The *DER Context* is used to query the information model. The context itself is an object meant to be used with the service methods and contains the data model of the information model and the values retrieved from the storage at the time the context was created.

The design idea is, that the context is passed to service methods, which in turn manipulates the context. Using the context avoids the need to build the data model, and retrieve the full set of values every time a query is performed.

In the second part of the behavior diagram, where the context is passed to the *Query* method, the values are checked against the table storage, and if newer values are found in the storage, the values are updated to the most current values. When the values have been updated, the query is performed and the resulting model of the query, is returned as a clone of the data model contained in the context. Using a clone ensures that the data model in the context will not be changed by manipulating the returned model.

7.3 Storage

The storage namespace of the cloud application is based on using Entity Contexts (shown in figure 7.4) to access the data entities in the storage. The *QueueContext* and *BlobContext* are manually coded and provides access to Blobs and Queues in the Azure Storage. The *ModelContext* is auto-generated using the model generated entities described in section 6.6.1.

The *ModelContext* provides auto-generated helper methods to create strongly typed *Query* objects for querying the storage and some generic methods to perform basic CRUD operations on the data entities. Any advanced manipulation of the data entities are performed in the repositories, but the *ModelContext* provides re-useable methods of the most basic operations. These methods were developed along with the customized model-to-code translation templates.

All the data entities, depicted to the right in the figure, are designed in the graphical model editor in Visual Studio and then auto-generated into code. In the design, section 6.6.1, a snippet was shown of how the data entities are modeled graphically. A full version of the storage data model used for auto-generation of data entities can be viewed in the appendix in figure F.4. Additionally the full version of the class diagram, figure 7.4, can be viewed in figure F.3.

Access to the storage starts with the *StorageConfiguration* class. This class is a static class that initializes the Azure Cloud Platform framework and sets up auto-cycling of the cloud application, should the environment variables for the cloud application change. When the *StorageConfiguration* has initialized the storage it can be used by the Entity Contexts and repository classes to access the storage.

The *UniqueId* class is a class that was implemented to generate unique IDs for use with the Azure Table Storage. This is described in section 7.3.1.

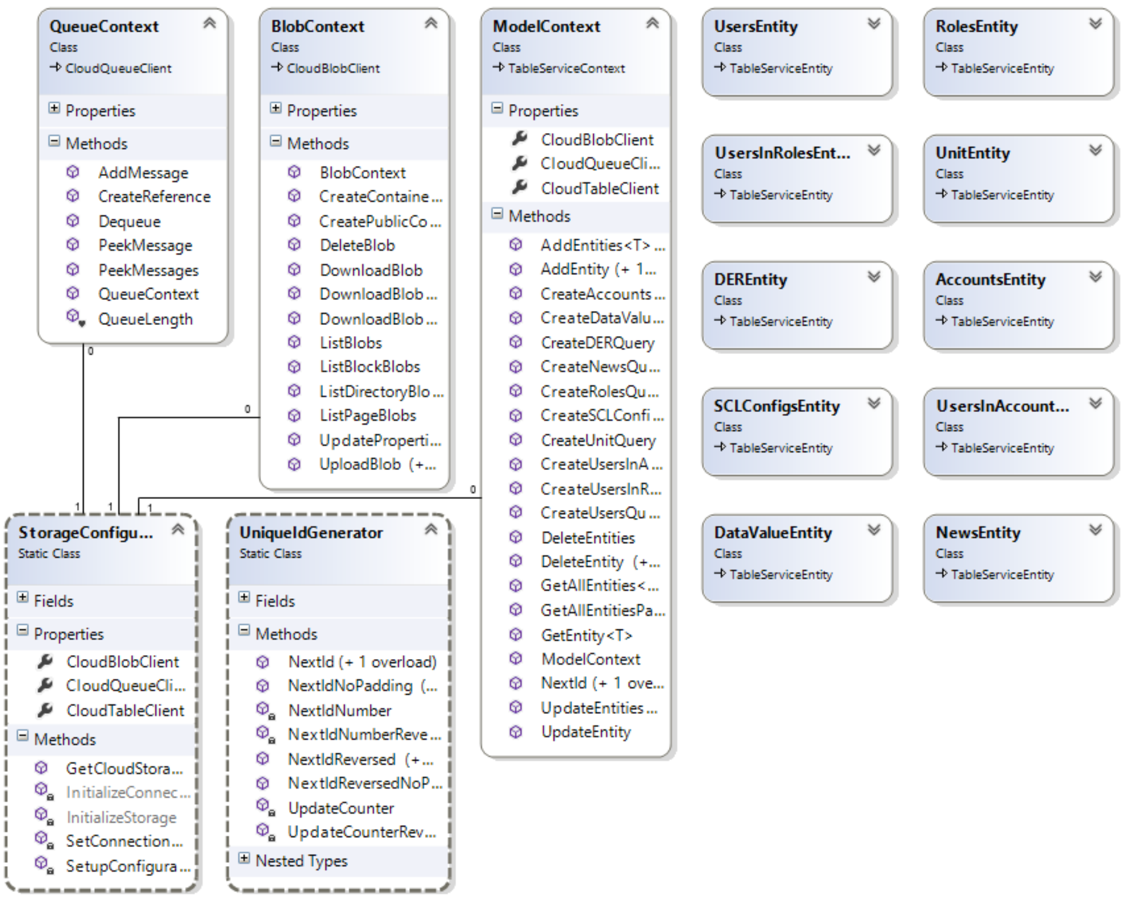


Figure 7.4: Class diagram: Storage

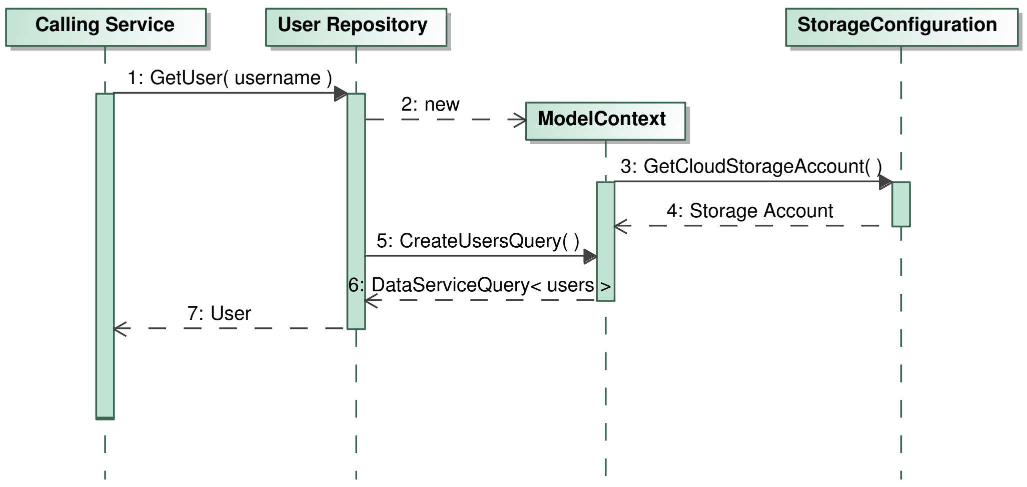


Figure 7.5: Behavior: Query a user in the storage

The behavior illustrated in figure 7.5, is an example of how to use the *User Repository* to retrieve a user, specified by user name, from the storage. The sequence diagram depicts the details down to the platform API. The *GetUser* method is called on the *User repository*, which would normally happen from a service method. The *User Repository* then creates an instance of the *ModelContext* which uses the *StorageConfiguration* to initialize the necessary parameters of the *ModelContext* in order to access the storage. The *User Repository* then uses the strongly typed query method of the *ModelContext* to perform a query on the storage and finally returns the user if found.

7.3.1 Unique ID

The Azure Table Storage does not support auto incrementing primary keys, so for this prototype the *UniqueId* class was implemented. The *UniqueId* class is able to generate unique IDs globally for the application and specifically unique IDs for each single table.

The *UniqueId* can generate reversed chronologically IDs which can be used to add and more importantly, retrieve entities from the storage in a LIFO manner. This means that entities are sorted in the storage so that the newest entities are retrieved first. This feature is not supported by the Azure Table Storage[1], but helps to efficiently retrieve the most current DER data when responding to a query.

The Azure Table Storage is a distributed storage and the application processes that runs in the cloud are also distributed and can be scaled out to many instances. This is relevant for the *UniqueId* class as it uses the storage to centrally keep a count of the unique IDs that have been used. In order to optimize access to the counter and avoid congestion and deadlocking on the counter, the *UniqueId* class reserves itself a range of IDs to use instead of taking a single ID at a time. This drastically reduces the amount of times the *UniqueId* has to lock the counter in the Table Storage as it only needs to do this when it runs out of the IDs it has reserved.

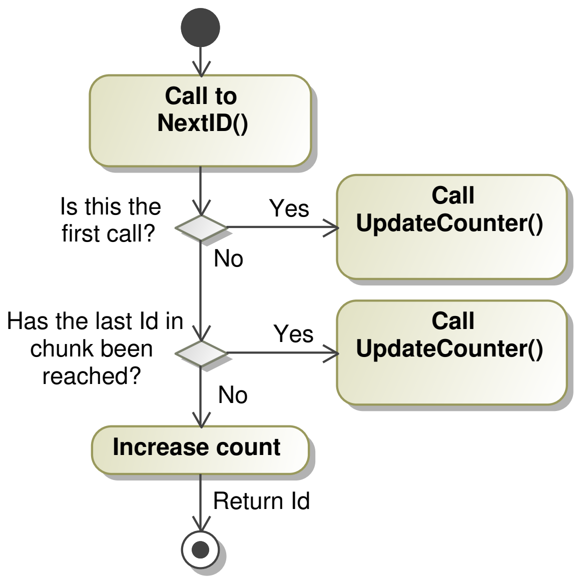


Figure 7.6: Flow diagram: Generate a unique Id for use with table storage - part 1

[1] http://msdn.microsoft.com/en-us/library/windowsazure/dd135725.aspx

As the *UniqueId* class uses blocks of IDs, this means that the IDs are unique but not continuous. This is a compromise, made in order to ensure good performance of the *UniqueID* class.

The operation of the *UniqueId* class is illustrated in the flow diagrams in figures 7.6 and 7.7. The flow is covered by two parts as the second part can be called multiple times from the first part but from different places in the flow.

When a call for a unique id comes, the first check is to see whether this is the first time the *UniqueId* class has been used, in which case it has to retrieve the count from the storage and reserve a new range of IDs. Next is a check for whether the range of IDs assigned to the *UniqueId* class has been used up, in which case, a new range must be computed.

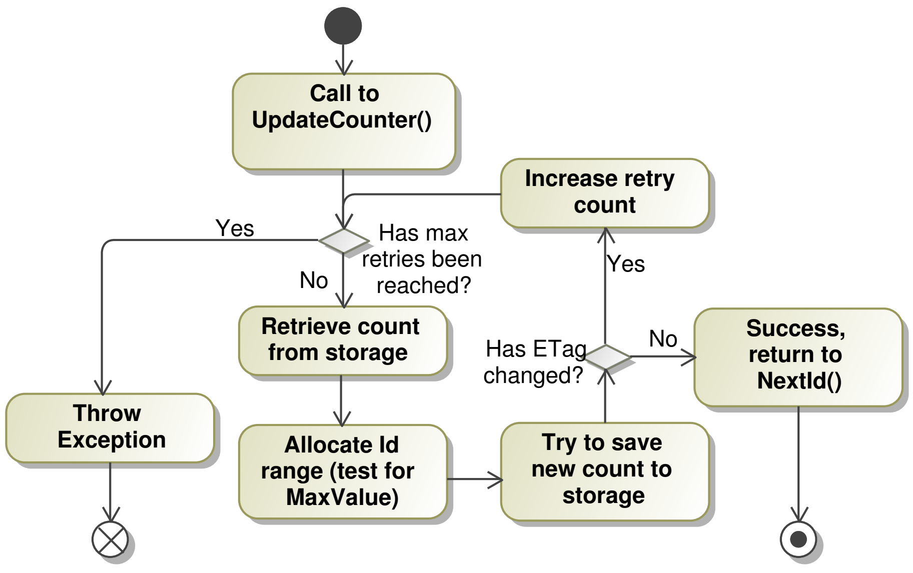

Call to UpdateCounter()
Increase retry count
Yes
Has max retries been reached?
No
Yes
Retrieve count from storage
Has ETag changed?
No
Success, return to NextId()
Throw Exception
Allocate Id range (test for MaxValue)
Try to save new count to storage

Figure 7.7: Flow diagram: Generate a unique Id for use with table storage - part 2

The flow to retrieve a new range of IDs is described in the second flow diagram and contains a retry loop to continue to try an get a new range of IDs. The process of getting a new range of IDs is based on the concept of Optimistic Concurrency [57] as the Azure Table Storage uses Optimistic Concurrency to manage concurrent access to the storage.

When the current count is fetched from the storage, a new range of IDs is chosen, the count is updated to match this and the the count is saved back to the storage. If another process have managed to save their count back to the storage in between, the ETag of the count would have changed, and saving the new count fails. An ETag is a form of checksum for the collection of entities that have been fetched from the storage and when the entities are saved back to the storage, the ETag is compared to the ETag of the same entities in the storage. If the entities in the storage have been changed, the ETags will not match up. In that case, the new count must be fetched from the storage and the procedure tried again. This is where the use of blocks of IDs will help keeping the misses down.

7.4 Information Model

The implementation of the Information Model builds upon and further extends the work done in [62]. The Information Model (figure 7.8) is a tree structured data model that has been closely modeled after the hierarchical data model defined in the IEC 61850 standard.

The IEC 61850 standard defines a large set of predefined logical nodes and data-classes. All these classes could have been implemented explicitly, but that would make the system hard to maintain and develop further as the data model must be extended or changed every time the standard receives updates and extensions. A generic approach has been taken to develop the data model as this offers more flexibility and makes is easier to include future extensions to the standard [62].

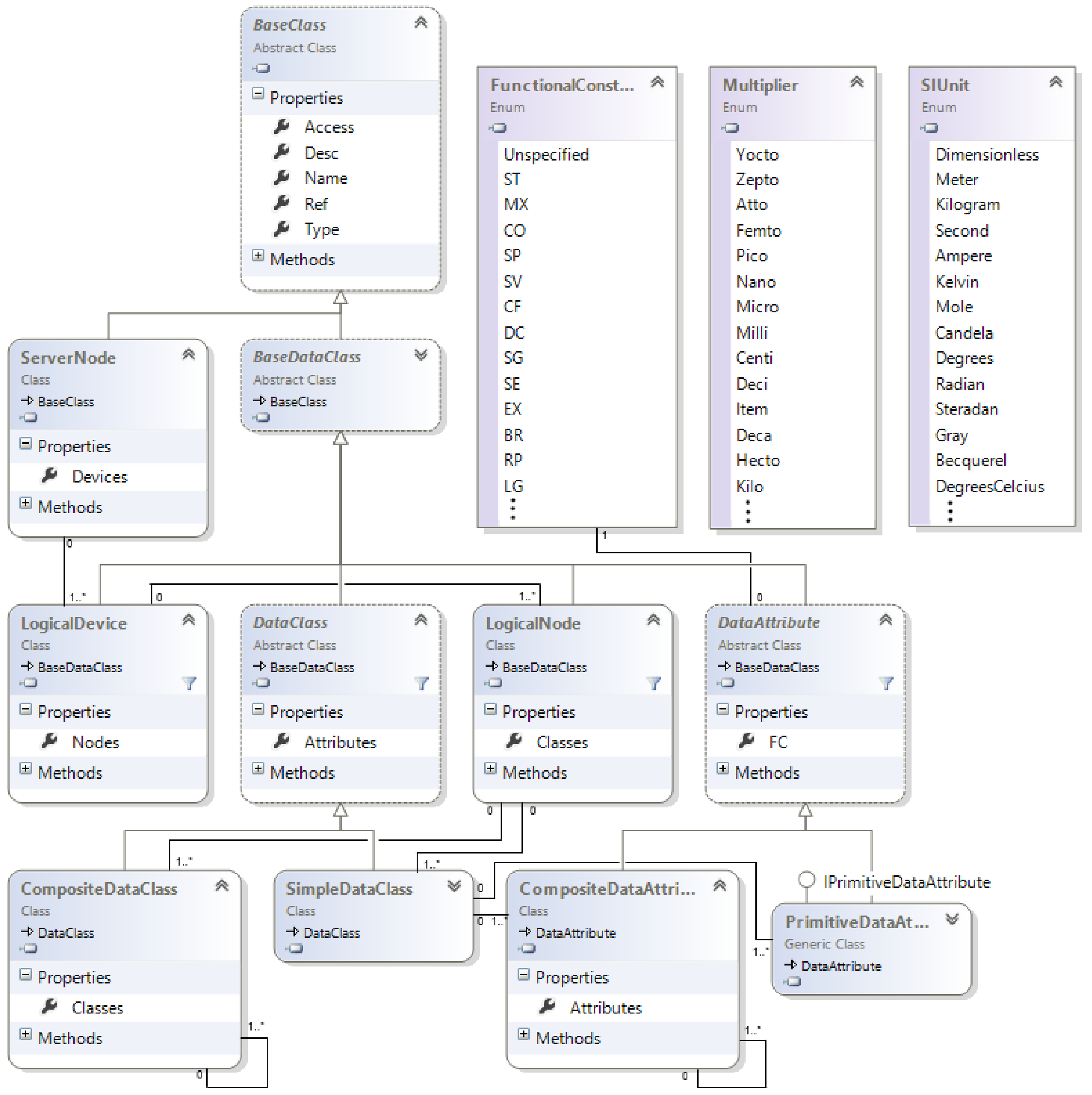


Figure 7.8: Class diagram: Information Model

The implementation of the generic classes, that makes up the Information Model, starts with the *BaseClass* which contains a number of attributes that are common to and inherited by all the child classes. The *ServerNode* defines the DER unit and contains a number of *LogicalDevices* which are the sub-systems of the DER unit and consists of logical systems and physical computing devices etc. Each device then contains a number of *LogicalNodes* and *DataAttributes*.

The details of the Information Model and the features of the classes are elaborated in the Design, section 6.2.

7.4.1 Working with the Data Tree Structure

The common ancestral *BaseClass* enables the system to traverse the data model in a generic way, from any point in the data model, as all the classes are of the type *BaseClass*. The prototype is designed to handle the *BaseClass* and can therefore handle any sub-set of the data model in the same way.

The classes *TreeDataGetter* (mentioned in figure 7.3), *TreePopulator* and additional classes, located in the Information Model component, which are based on the work in [62], are used to manipulate the data model. The *TreePopulator* is the main class that performs the actual recursive transversal of the data model. When invoked, it returns a clone of the data model, containing the elements of interest defined by the supplied arguments. The additional classes used to work on the data model either uses or are used by this class.

7.4.2 SCL Parser

The SCL Parser class is a recursive descent parser and is used to parse the SCL configurations that are supplied by the operators of the DER units. The SCL Parser is based on the work done in [62].

The SCL Parser is used to instantiate the data model and does this by parsing an SCL configuration of a DER unit. An SCL configuration describes the logical devices, logical nodes and data attributes along with the reference definitions of where the data is located. The SCL additionally contains a description of the data, multiplies and IEC 61850 data type definitions.

The SCL Parser is used in the prototype to instantiate a data model when a query from a client is received. This is described in the Design, section 6.8.

7.5 Users

The authentication and authorization of users in this prototype is based on the .NET Membership framework [55]. It is based on users and roles. Users can have multiple roles and roles can be used by multiple users. The Membership framework can be used throughout the the API's provided by the .NET framework, which includes: Websites, WebServices etc. and programatically everywhere else.

The Membership authorization can be made declaratively with many of the .NET API's and programatically as needed. Listing 7.1 is an example of how to declaratively define what roles a user needs to be in, in order to query the information model via the RESTful interface.

```
[HttpGet]
[MembershipAuthorize(Roles = "Administrator, Operator, Customer")]
public BaseClass ModelQuery([BindCatchAllRoute('/')]string[] path, string level = "",
    string fc = "")
{ .....
```

Listing 7.1: Example of declarative security

The *MembershipAuthorize* attribute is actually a custom extension of the *Authorize* attribute provided by the .NET framework. The MVC4 Website template is not directly compatible with the old Membership framework, as Microsoft has created a new simpler and lighter version of the .NET Membership. However, the new version has been created mainly for the MVC4 Website template and does not support as many of the .NET framework API's as the old Membership framework does. As the prototype uses some of the functionality provided by the older Membership framework such as XML Webservices, the older Membership framework has been chosen and extended.

Listing 7.2 is an example of how to use the .NET Membership framework programatically to create a new user, test if the users is assigned to a role, add the user to a role and validate the user.

```
// Create user
var password = Membership.GeneratePassword(10, 2);
var name = "Test User";
var newuser = Membership.CreateUser(name, password);

// Add user to role
if (!Roles.RoleExists("Customer"))
    Roles.CreateRole("Customer");
if (!Roles.IsUserInRole(name, "Customer"))
    Roles.AddUserToRole(name, "Customer");

// Retrieve user and validate login
var user = Membership.GetUser(name);
Membership.ValidateUser(name, password);
```

Listing 7.2: Example of programatic security

The .NET Membership framework uses a series of *Providers* to store and retrieve the users, roles and additional information from different storage solutions. E.g. SQL Database, DB2, ODBC etc. The Azure Table Storage is not, at this time of writing, supported by the Membership framework and so custom Membership providers was implemented to support the Azure Table Storage.

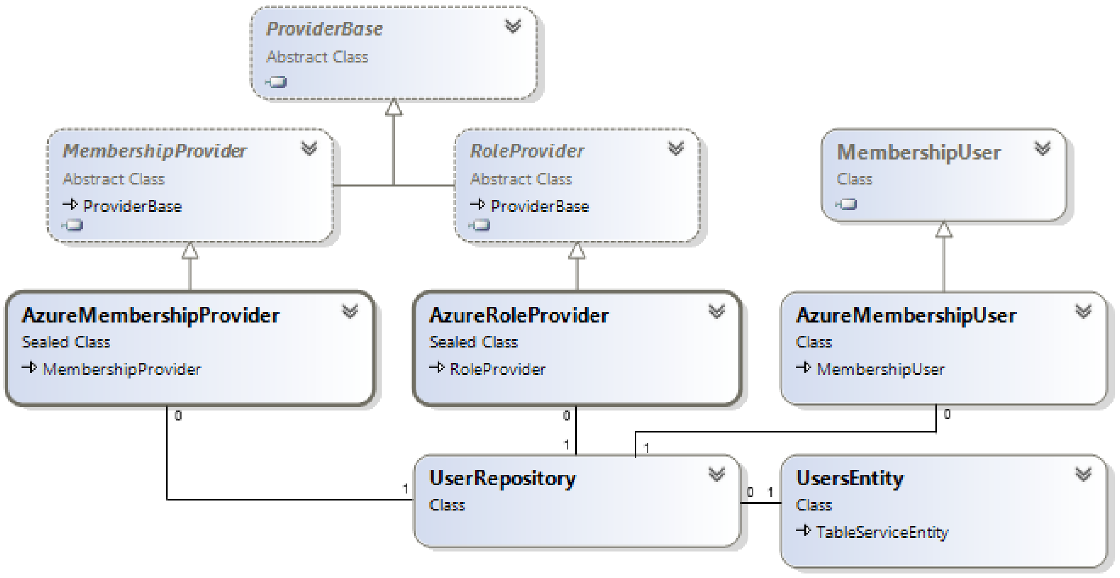


Figure 7.9: Class diagram: Users and Membership Providers

The custom membership providers implemented (see figure 7.9) are the *AzureMembershipProvider* which handles the users and the *AzureRoleProvider* that handles the roles. The *AzureMembershipUser* is an extension of the user entity provided by the framework and can be used to add custom attributes to the Membership user which in turn is passed around by the Membership framework and enables access to the custom attributes using the standard API'methods.

The custom providers uses the *UserRepository* to store and retrieve the user- and roles-entities. A fully detailed version of the class diagram can be viewed in the appendix in figure F.5.

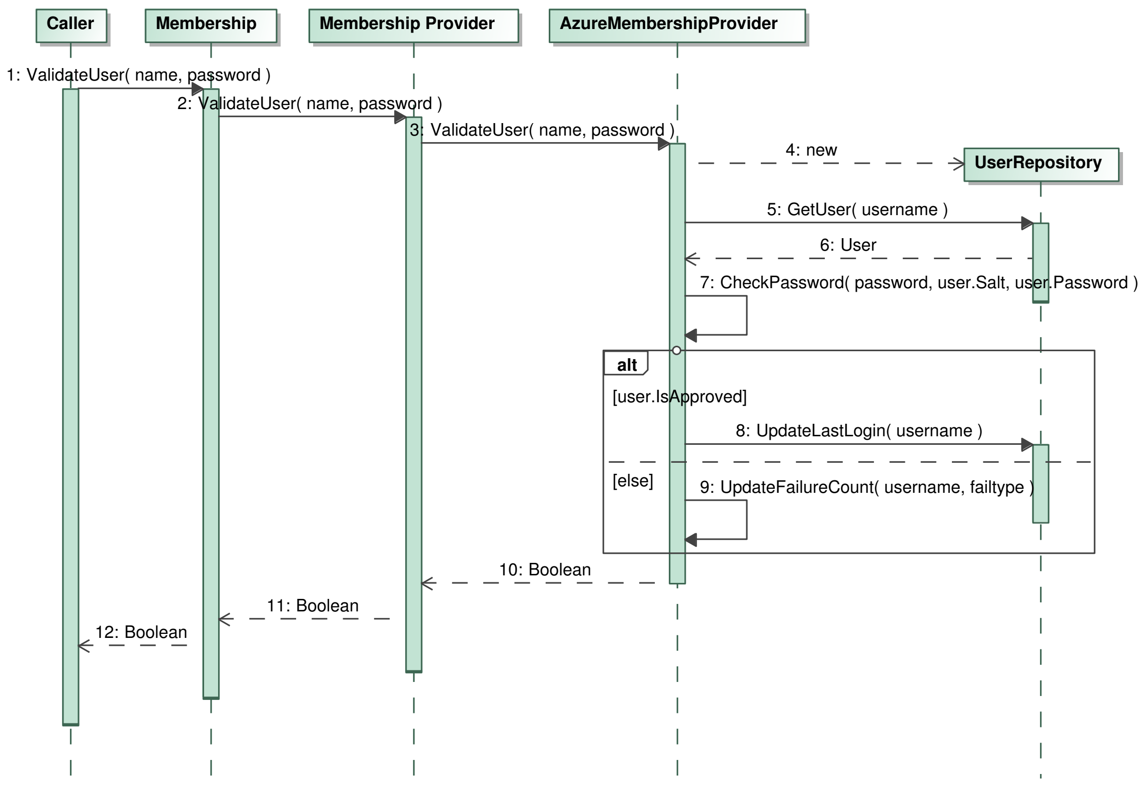

Caller
Membership
Membership Provider
AzureMembershipProvider
1: ValidateUser(name, password)
2: ValidateUser(name, password)
3: ValidateUser(name, password)
4: new
UserRepository
5: GetUser(username)
6: User
7: CheckPassword(password, user.Salt, user.Password)
alt
[user.IsApproved]
8: UpdateLastLogin(username)
[else]
9: UpdateFailureCount(username, failtype)
10: Boolean
11: Boolean
12: Boolean

Figure 7.10: Behavior: Using the custom Membership Provider

The behavior diagram 7.10 illustrates the process of how the .NET Membership validates a user. The caller uses the Membership method *Membership.ValidateUser(name, password)* which checks the application configuration for which provider to use and then delegates the work to this provider. The provider in this case is the *AzureMembershipProvider*, which uses the *UserRepository* to retrieve the user information from the storage.

The diagram in figure 7.10 ends at the *UserRepository*, but the rest of the behavior from that point can be seen in section 7.3 figure 7.5.

7.6 Modules

As described in the Design section 6.5, the custom DER modules is loaded dynamically into memory as a DER unit connects to the cloud application. In .NET this is achieved by using an Application Domain to load the module into. The module is running in its own application domain which is separate from the application domain that the cloud application is running in, for security reasons and this is dictated by the .NET framework.

For the cloud application to communicate with the module, the module must implement an interface shared between the two applications. This interface is the *IModule* (shown on figure 7.11) and it enables the cloud application to call the methods in the module.

These methods, as illustrated in the behavior diagram 7.12, in turn needs to communicate the DER data back to the cloud application (as described in section 6.5). For this, the interface *IStorageBuilder* is used by the cloud application to inject an instance reference of the

ModelBuilder from the cloud application domain into the module in the module application domain. The injection of the *ModelBuilder* reference is done using the Dependency Injection pattern [25], through Setter Injection.

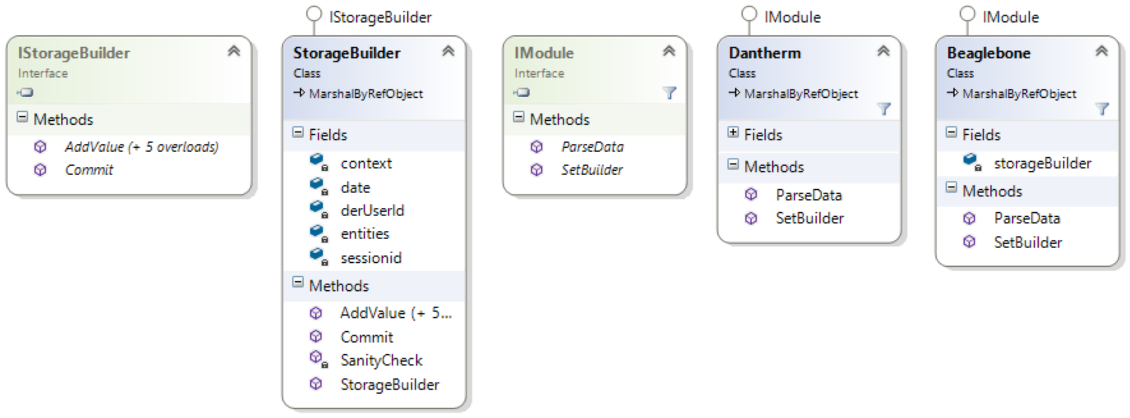


Figure 7.11: Class diagram: Modules

The *StorageBuilder* contains two methods: The *AddValue* is used to add one primitive data attribute consisting of a data value, data model reference, sessionid and additional information. The *Commit* method is used after all the values has been added to commit the data collection to the storage.

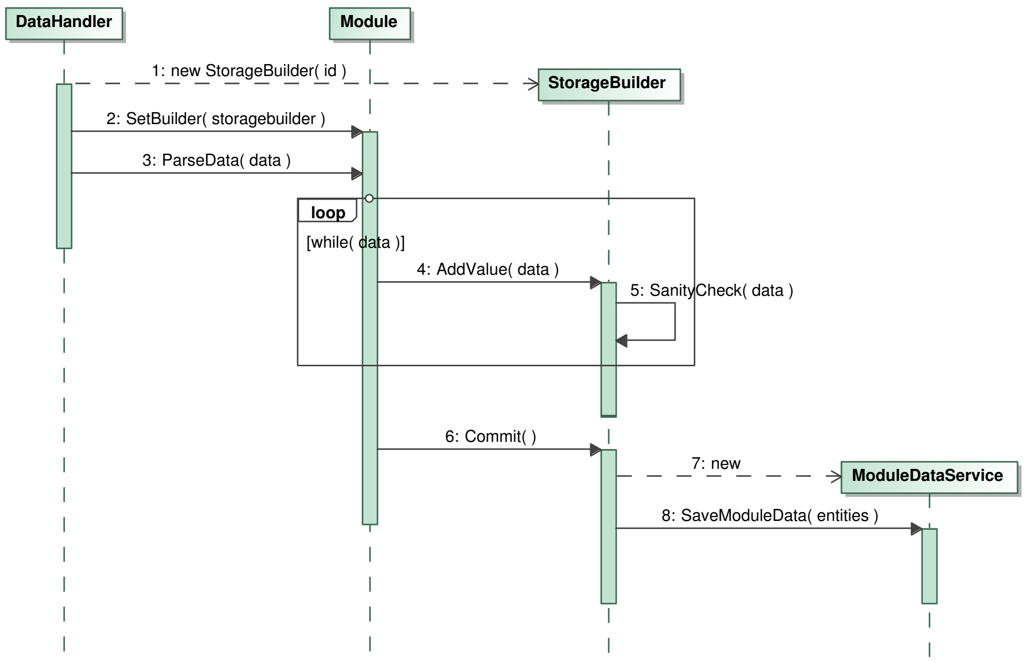


Figure 7.12: Behavior: Receive data in module and save to storage

The behavior diagram 7.12 describes how the *DataHandler*, running in the cloud application, interacts with a module running in the module application domain. The *DataHandler* is a class in the DER Websockets interface that handles the communication with the DER Units.

When the *DataHandler* loads a module it create an instance of the *ModelBuilder* and injects it into the module. It then passes the raw data to the module which in turn parses the raw data and adds all the data values to the *StorageBuilder*. Finally the module calls *Commit* on the *StorageBuilder* and the now structured data is sent to the *ModuleData Service* which uses the *ModuleData Repository* to save the data in the storage.

7.7 DER Websocket Interface

The DER Websocket interface communicates with the DER units through the use of open Web standards WebSockets [24]. The DER WebSocket interface (depicted in figure 7.13) is based on an Azure Worker Role, which is a cloud process without an User Interface.

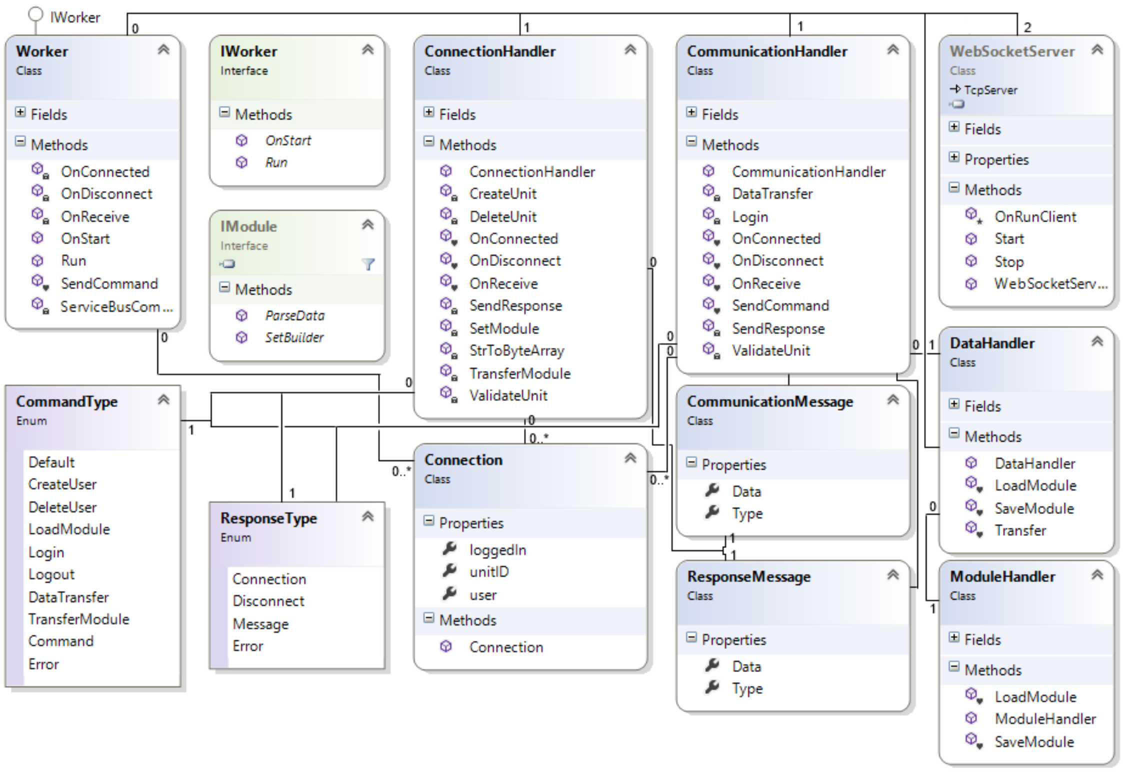


Figure 7.13: Class diagram: DER WebSocket

The WebSockets are based on the Alchemy WebSocket Library[2]. The Alchemy library provides WebSockets server and client for .NET and a client for JavaScript. The library officially supports hybi-00, hybi-10, hybi-17, and the official RFC6455 [24], which means that it possible to communicate with the WebSockets from any language and platform.

[2]Alchemy WebSockets: `http://alchemywebsockets.net/`

The DER WebSocket Interface creates two WebSockets: *ConnectionHandler* and *Commnication-Handler* that each handles a separate concern, as described in table 7.1.

Request Path	Description
/connection	Handles creation of DER users and uploading of DER Modules
/communication	Handles uploading of DER data from DER units.

Table 7.1: DER Websockets Interface Request Paths

A WebSocket has a "Request Path" that indicates to the WebSocket what the intention of client is. The statemachine in figure 7.14 illustrates how the DER WebSocket Interface handles incoming data from the DER units.

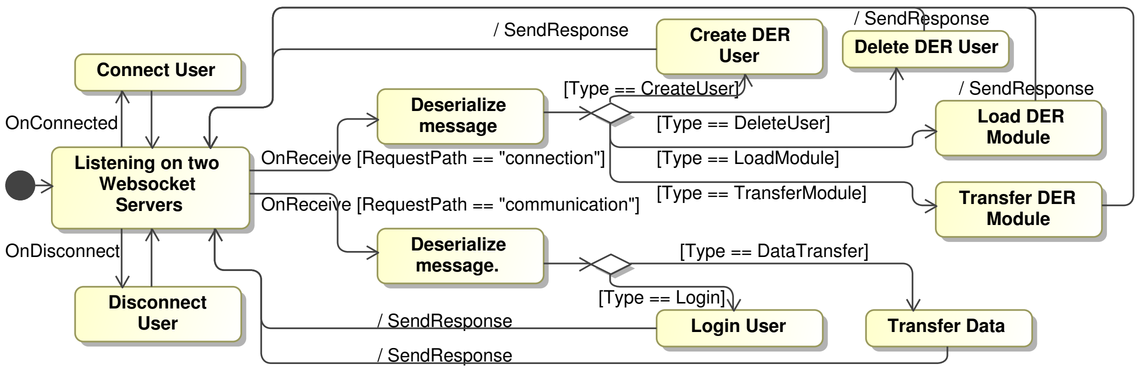


Figure 7.14: Statemachine: DER WebSocket

Based on the request path the interface breaks out into separate paths. If the request path is *connection* the client can now create new DER users, delete DER users, upload DER modules to the cloud application and load modules into memory.

If the request path is *communication* the client must first authenticate with the cloud application in order to tell the interface which DER user is connecting and which DER unit to store the data with. The client can then upload the data from the DER unit to the cloud application.

7.7.1 Inter-process Communication

The RESTful and the XML Webservice interfaces are implemented as a distributed Azure Web role process and the DER WebSocket interface is an Azure Worker role process. The two roles runs in different processes and can consist of many instances. This consequence is that the RESTful/XML Webservice interface process does not know which instance of the DER WebSocket interface process a given DER unit may be connected to.

The solution, illustrated in figure 7.15, is to use the Azure Service Bus to relay the communication from the RESTful/XML Webservice interfaces to the DER WebSocket interface.

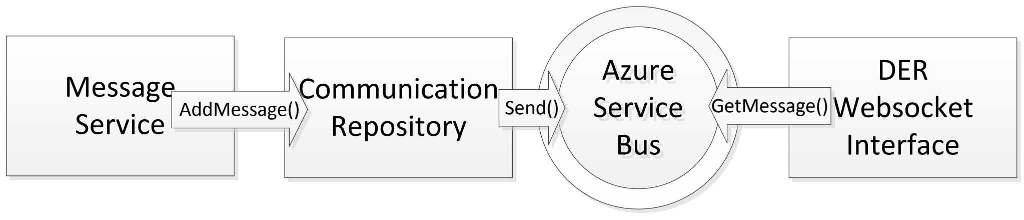


Figure 7.15: Inter-process communication via the Service Bus

When for example, a command to a DER is received on the RESTful interface, the message is passed to the *Message Service*, which in turn uses the *Communication Repository*. The message is handed to a repository because the *Service Bus* and the Azure Queues that are used are part of the Azure Storage namespace. The *Communication Repository* sends the message to the *Service Bus*.

Every instance of the DER WebSocket interface processes is subscribed to the *Service Bus* and when a message arrives for a DER unit, that is connected to a given DER WebSocket interface, the message is fetched from the *Service Bus* and sent to the DER unit.

7.8 Website

The Website enables DER owners and DER Operators to configure their DER units, add new DER units, upload different DER Modules and configurations, select the active configuration and view information about the DER data their DER units have uploaded to the cloud application so far.

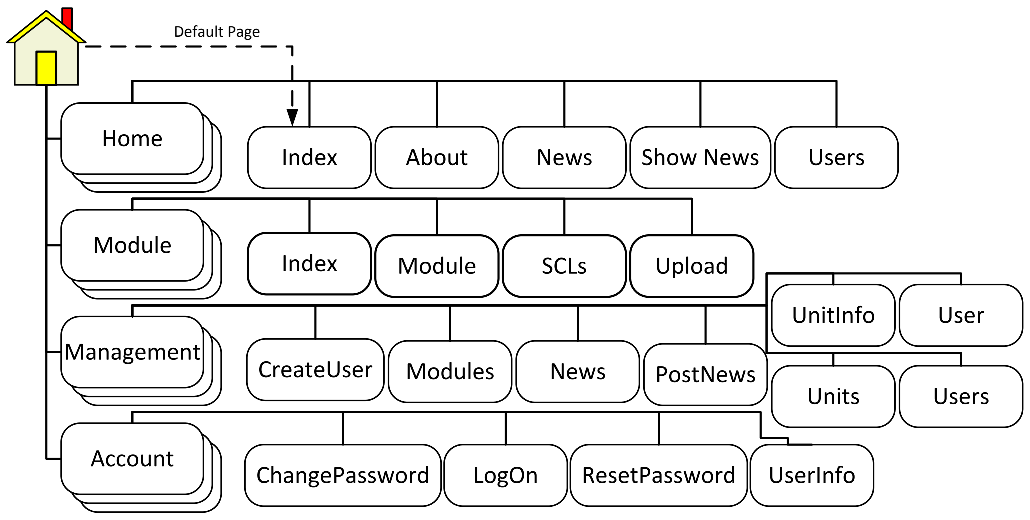


Figure 7.16: Website Web Pages Diagram

Figure 7.16 shows the Web page hierarchy of the Website. The details of the functionality of the Website has been described in the Design section 6.8.1.

The Website is built as an Azure Web role process, which is similar to a worker role except that the web role has a user interface. The user interface of the Website Web role is based on the Microsoft MVC v4 website template and provides the new Microsoft Web API which makes it straight forward to build RESTful services. The RESTful interface in this prototype is based on the Web API.

Screenshots of the Website can be enjoyed in appendix D.

A user guide has been written, that walks through the usage and functionality of the Website. The user guide can be viewed in appendix E.

7.9 RESTful and XML Webservice Interfaces

The RESTful and XML Webservice interfaces are both implemented in the Website component. The RESTful interface is built on the Web API which is a part of the ASP .NET Web framework and the implementation follows the description of the design in section 6.8.2.

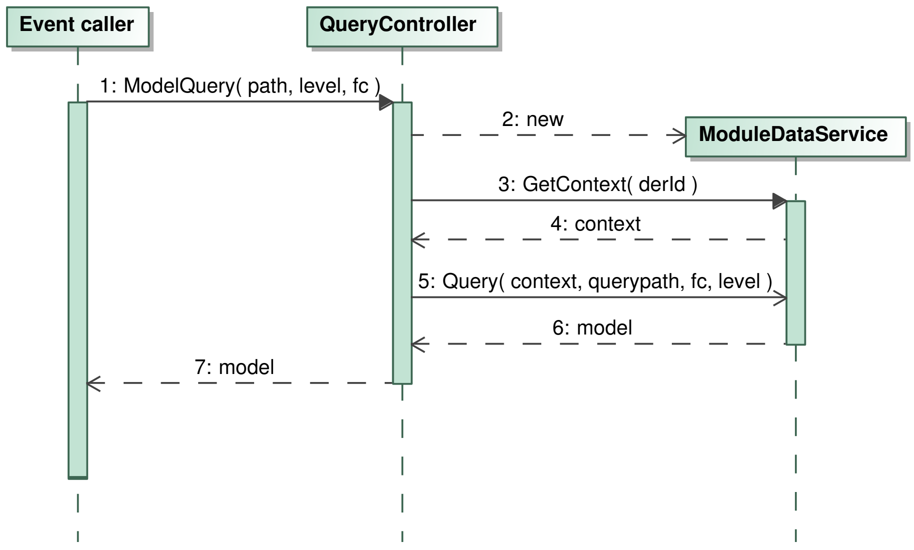


Figure 7.17: Behavior: Query the information model via the RESTful interface

Figure 7.17 illustrates the behavior of the RESTful interface when a query request is received. After the request has been authenticated and authorized, the request details are sent to the *QueryController* which build a query object that can be sent to the *ModuleData Service*. As with the behavior described in figure 7.3, a *DER Context* is retrieved and then the query is performed. The resulting cloned model containing the requested data is returned to the client.

The response is output in either XML or JSON based on the request header in the client request. The client can specify whether the response should be in XML or JSON. For the prototype the XML output has been customized with the use of a custom XML Serializer. This way the XML output is structured in a layout that closely resembles the data model specified in the IEC 61850 standard.

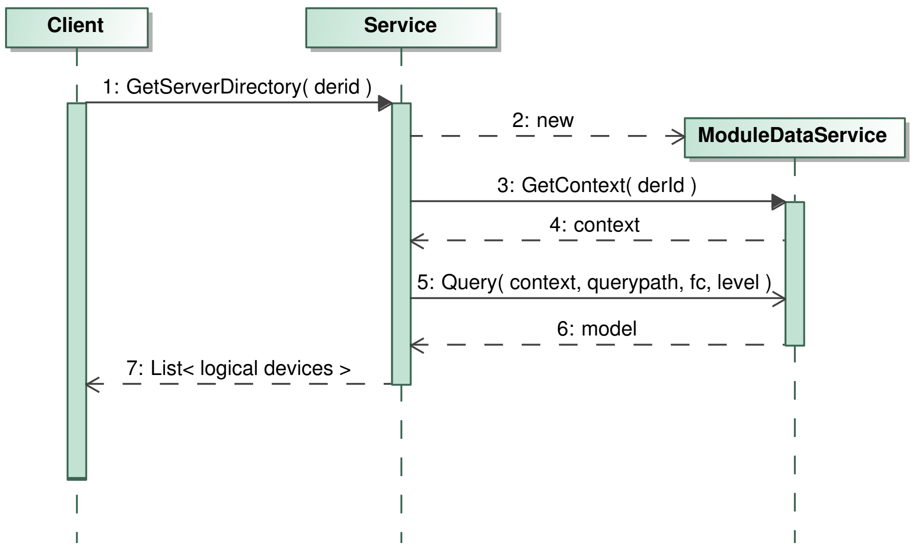

Client
Service
1: GetServerDirectory(derid)
2: new
ModuleDataService
3: GetContext(derId)
4: context
5: Query(context, querypath, fc, level)
6: model
7: List< logical devices >

Figure 7.18: Behavior: Query the ACSI method GetServerDirectory via the XML Webservice

Figure 7.18 shows the behavior of the XML Webservice when the ACSI method *GetServerDirectory* is called. The method is called on the class called *Service* which implements the basic functionality of the ACSI interface, which in turn build a query object and uses the *ModuleData Service* to perform the query. The resulting model is returned to the *Service* (ACSI interface) and then returned to the client.

7.9.1 Asymmetric Public Key Encryption

The RESTful interface supports asymmetric private/public key encryption of the user credentials in query requests using the RSA algorithm. An overview of the usage of the RSA functionality is described in the Design section 6.8.2.1, this section only describes the implementation details.

When creating a query that requires authentication, the RESTful interface expects the user credentials to be sent via Basic HTTP Authentication where the password is encrypted using the public key. The RESTful service then decrypts the password and validates the user.

To use Basic HTTP Authentication, the request must contain the following HTTP header:

```
Authorization:  Basic username:password
```

The Replay-Attack prevention is achieved by packing a timestamp and the password into the password part and encrypting it with the public key:

```
Authorization:  Basic username:[timestamp|password]
```

The timestamp and password are base64 encoded, then encrypted with the public key and hex encoded. Finally the username and packed-password are base 64 encoded. This block is illustrated in figure 7.19.

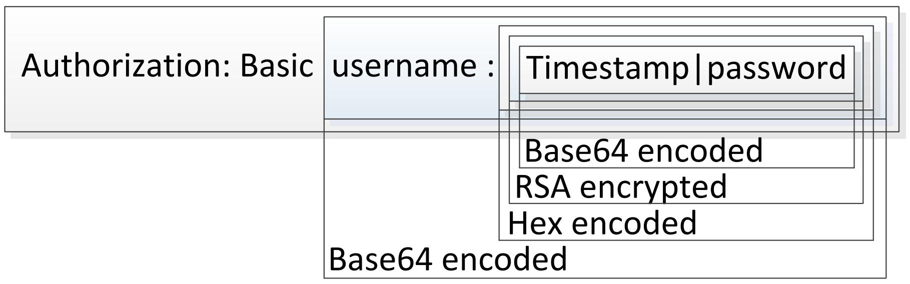


Figure 7.19: RSA user credentials block

When the username and packed-password block arrives at the cloud application the packed-password part is decrypted and separated. The timestamp is checked for whether it is current enough to be accepted and then the user and unpacked password is validated with the Membership framework. This security scheme can be used by any type of client, as long as the username, password and timestamp are packed and encoded in the correct order. For this prototype a JavaScript client was developed to test the RSA encrypted user credentials.

In this implementation, only the timestamp and the password are encrypted and that is enough to securely send the user credentials to the cloud application and prevent Replay-Attacks. It would be possible to additionally secure the body content of the request, using the same encryption approach. An alternative idea would be, to encrypt the body with a symmetric algorithm such as AES and pack the key with the timestamp and password.

7.10 Chapter Summary

From the initial design, each of the components have been implemented in the .NET framework. The final solution consists of a Web Role, running the MVC 4 based web site, an XML Webservice and the RESTful interface, through the Web API in the .NET framework. The DER WebSocket Interface is implemented as a Worker Role. The solution is centered around the *Core* that is built using multiple components to implement the layered architecture design.

To handle the raw data from the DERs, the Azure Table Storage is used, as this allows for a distributed storage solution. It raised some problems with querying data, as the Azure Table Storage does not support auto incrementing primary keys and sorting when querying, but this was solved by implementing the *UniqueId* class. When querying data, the transformation from raw data into an IEC 61850 compliant information model is accomplished by using an SCL parser. The parser instantiates a data model based on the provided SCL configuration file and then populates and returns a *DER Context* with the data model and with the most recent DER data, fetched from the storage.

Saving DER data to the storage, using a general data structure, is accomplished by providing two interfaces. The module uses *IModule* as base and gets a reference to an *IStorageBuilder* instance injected by the cloud application. Using the provided methods from the storage builder, the modules builds a list of generic values, using a string representation of the values and a description of the data type that should be used to restore the data value.

To transfer the DER data to the cloud, the Alchemy WebSocket library has been used to handle the connectivity. Besides having support for the official RFC6455 standard for WebSockets, the library handles issues, such as Windows 7 not having native support for WebSockets. Two sockets is used in the prototype, one provides functionality to create DER users and upload DER modules. The other provides functionality for transmitting the raw DER data to the cloud application. Both sockets have functionality to authenticate and authorize the user before the rest of the functionality can be used. When communicating from a client to a DER unit, a message is sent through the Azure Service Bus, and using the two-way communication provided by the WebSocket, it is passed to the DER unit.

CHAPTER 8

Testing

This chapter describes the testing methods used for the development of the prototype to ensure the correctness of the implementation of the design. The validation of the functionality described in the requirements is described in the Case Studies chapter.

8.1 Unit Testing

Unit testing is used to individually test the smallest possible parts of program components together with their associated data. Unit tests are isolated tests meant to show that the individual code parts are behaving correctly.

In this project unit tests have been used for the following purposes:

Quick debug turnaround

When developing applications for the Azure cloud, the Azure SDK is used to emulate the cloud platform where the application processes run and the Azure Storage where the data is stored. The turnaround for debugging code with the Azure SDK can be lengthy as the code must be re-deployed every time a debugging session is started. By utilizing a unit test framework, it is possible to write test cases to test an isolated unit of code, and then develop the actual code. When using the unit test framework to run and debug the code, the re-deployment can be avoided in many cases, providing a quick turnaround with regards to debugging time and the unit test can further be used as an integration test in the testing phase. This approach to writing code is also a concept used with Test-Driven Development[6].

Find problems early

As mentioned above, multiple unit tests have been written using the Test-Driven Development (TDD) approach. In TDD unit tests are created before the code logic is written, the tests contains assertions that describes what the code is expected to do. The code is written and then, when the tests pass, the code is considered complete. As the code base grows, the same unit tests are run against the code continuously and if the unit tests fails, it is considered a bug in the changed code.

Facilitates changes

One of the most cherished consequences of unit testing is the ability to refactor code at a later date, with confidence that the code still functions correctly. When changes to the code base is introduced during development, the unit tests can be run to ensure the existing code still works as intended and during refactoring, the unit tests will show if changed methods still produces the expected results.

Simplifies integration

Unit testing simplifies integration testing with external components. When changes are made to external components, such as the DER Controllers, it is important to test whether the application code is still able to communicate with the external components correctly. This would normally be a manual process, however testing the integration of these components can be simplified by automatic unit testing.

Unit testing comes in different frameworks, the unit testing framework used in this project is the NUnit Unit-Testing framework for .NET[1]. This framework was chosen as it provides a preferred assertion scheme over the built in test framework provided by Visual Studio and also provides extra functionality used in section 8.4.

Unit testing typically follows these three steps:

1. Initialize the test by preparing associated data and dependencies.
2. Run the section of code that is to be tested.
3. Assert the properties of the results.

Unit tests should be short, isolated and run quickly, which is often achieved by *mocking* the data and dependencies that the tested code uses. A mocking framework, such as moq[2], can be used to substitute classes with dummy classes, called *mocks*. When mocking classes, they are separated from the rest of the code and supplies the tested code with dummy data. It can also be used to simulate data from parts of the application that has not been developed yet or data from external components, such as databases, which can be slow or may not be available locally.

8.2 Integration Testing

Integration testing is used to verify functionality- and performance-requirements of major components and identify problems that arises when these components are coupled together. The major components of this project that must integrate consists of:

Cloud Application

The cloud application is the main component and interacts with both clients through the RESTful interface, XML Webservice interface, the Azure Cloud Platform and the DER Controllers. Of the many points in the code where integration testing takes place, the major ones are where the cloud application communicates with the other components and each time data from the Azure Storage is used.

DER Controllers

The DER Controllers integrates with the cloud application at one end and the DER hardware unit at the other end. They may also need to integrate with a storage for configuration and application data.

[1]NUnit: `http://www.nunit.org`
[2]moq: `http://code.google.com/p/moq`

Azure Cloud Platform

The Azure Cloud Platform is a component in the form of the platform APIs and the Azure Storage which contains the data of the cloud application. The Azure Storage is heavily integrated in the cloud application as much of the functionality operates on the data stored within.

When creating a unit test with NUnit in .NET the class must simply be marked with the *TestFixture* attribute and the test case marked with *Test* in order to let the testing framework know about the unit tests.

```
[Test, Category("Integration")]
public void InstantiateModelAndPopulateFromTableStorage()
{ .....
```

Listing 8.1: Example of an Integration Test Declaration

The unit test can be categorized by appending a category to the *Test* attribute as shown in listing 8.1. This was be used to separate the normal unit tests from the integration tests.

Whereas the unit testing verifies the correctness of the code in each component the integration test verifies that the components functions correctly when put together. The integration tests sums up the unit tests and verifies the functionality of the system as a whole and enables the next step of validating the functionality of the requirements specification. For this project the validation testing is done in the form of case studies, described in chapter 9.

8.3 Build System

To enable automated testing for this project (section 8.4) and the ability to generate additional artifacts such as code documentation and code metrics, a build system was implemented. The backbone of the build system are the unit tests and the build framework. The unit tests provides the input for large parts of the generated results and the build framework brings the operations together in a component where they are easily accessed and can be automated.

In this project, the build system is based on the NAnt build tool for .NET[3] with some dependencies on the built-in build framework MSBuild, provided with Visual Studio. The model in figure 8.1, shows the basic build files that make up the build system along with a selection of the main build targets that the build system provides. A build target describes the operation that should be performed on the code base.

Default

This is the starting point of the build system, it is the *default* build file and it includes and enables the usage of all the other build files.

Settings

The *settings* contains the global settings for the build system such as the base directory, source code directory, build configuration and tools directory. These settings are used extensively throughout the build system to reference the directories containing the software and code. The tools directory is where all the tools needed are found, the build system makes use of a large collection of external tools to provide unit testing, code coverage, code documentation etc.

[3]NAnt: `http://nant.sourceforge.net`

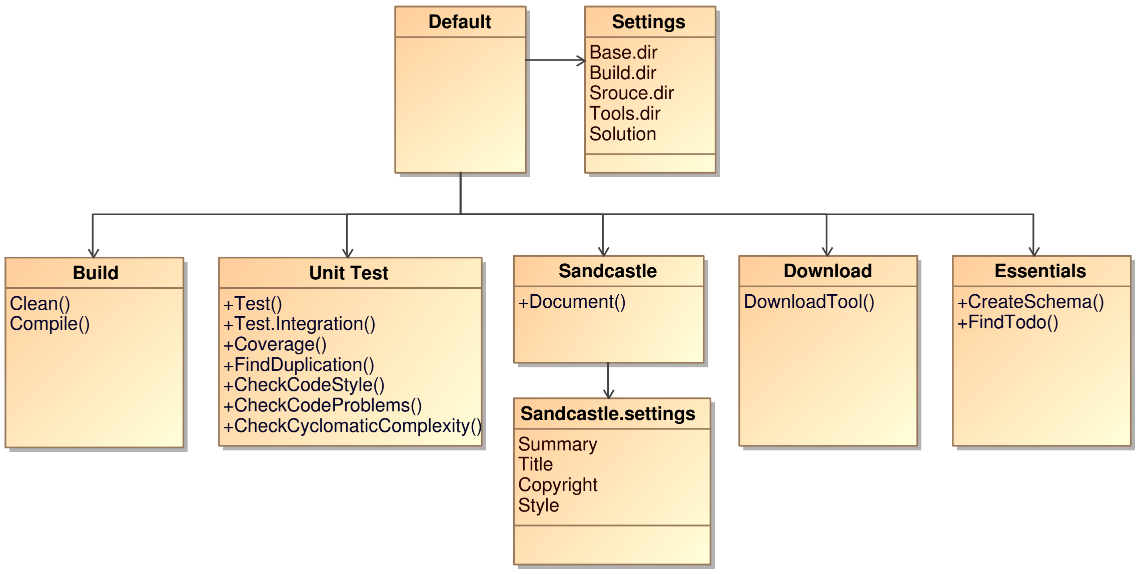


Figure 8.1: Build System

Build

The *build* file is responsible for building the project solution and cleaning the source code directories both before and after a build. It is important to clean output directories between builds depending on the operation to avoid cross pollution between builds.

Unit Test

The *unit test* build file handles unit testing, integration testing, code coverage and generation of various reports etc.

Sandcastle

Sandcastle is a rather large tool that provides API documentation of the code. It uses the XML comments in the code to create the documentation. Examples of the API documentation can be seen in the appendix figure D.37 and D.38.

Download

The collection of tools in the build system takes up well above 800 MB of space when unpacked. The idea of a build system is to commit it to an version control system alongside with the source code, so the build system is always available when the code is checked out. This build file handles automatic downloading of the tools when an operation is called the first time.

Essentials

The *essentials* provides ancillary functionality to the build system, such as generating a report of Todo's found in the source code and generation of an IntelliSense schema for visual studio to enable editing of the build system files etc.

The primary way to use the build system is through the commandline. This is an example of how to compile and run unit tests on the project:

```
nant compile test
```

This command cleans the output directories, compiles the project defined in the settings and uses NUnit to run the unit tests on the project. (`nant test` can be used for shorthand) An overview of the targets available along with a summary can be viewed by running: `nant` with no arguments.

8.4 Continuous Integration

When using TDD and having multiple people developing on the same project, it can often lead into integration problems. To handle this, a continuous integration (CI) environment can be setup. CI is used to streamline the building process of a development solution. Whenever a developer has checked in code to the version control system, the CI server pulls down the latest version and runs a set of scripts. This can include the ability to run the unit tests created using TDD or even create deployment ready executables.

The build system, described in section 8.3, is used to facilitate automated testing of the project and for the development of this project, a Jenkins CI[4] server has been setup in an Azure Server instance. The Jenkins CI provide a user friendly environment, with a web site, to help manage and show the project solutions added on the server, this can be seen in figure D.34. When the Jenkins CI checks out any newly committed code, the operations provided by the build system, is used to run unit tests and integrations tests. Jenkins CI also has a local application, that runs on e.g. the developers machine and can track the build status of any given solution, shown in figure D.36. Using CI with the build system it is possible to generate a variety of reports, for example a code coverage report to show the amount of code covered by unit testing. As figure D.35 shows, the code coverage for the assembly *Core*, which is part of the cloud application project, is at 26.6%.

8.5 Chapter Summary

This chapter described how the system has been tested throughout project and the methods used. The validation of the requirements is conducted through the usage of case studies, which is described in chapter 9.

The unit tests and integration tests have been used to make the development process more effective and ensure the reliability of existing functionality, as changes are introduced into the code. The build system has enabled automatic testing with the use of continuous integration and can generate of a variety of reports.

[4]Jenkins CI: `http://jenkins-ci.org`

CHAPTER 9

Case Studies

Throughout the project, the use cases, the goals and the requirements have been the foundation of the development of the prototype. As the previous chapter handled the testing of the solution, the following chapter will show how two implemented case studies are established, each containing one or more of the defined use cases.

9.1 Case study : Dantherm DKµCHP

In the first case study, a demonstration of how to utilize a running Dantherm DKµCHP with the Cloud Service will be explained. This will include the setup, configuration and the development of the module for the µCHP. The Dantherm DKµCHP is a distributed energy resource located on an unknown address and only delivers an XML feed with the most current values. This makes for some limitation, such as direct control of the DER. To handle this, a DER Intermediary is introduced for legacy units.

9.1.1 DER Intermediary

The DER Intermediator is a separate cloud application that opens the communication with the DER unit, as it is done using the current convention (see figure 1.1) for DER communication. It then transmit the data to the Cloud Service via the WebSocket Interface in the Cloud Service. The application also runs in the cloud, but is a separate application from the Cloud Service.

The DER Intermediator is able to support a wide range of communication methods in order to fetch the data from any DER unit. It could be via FTP to fetch log data uploaded from a simple DER controller, using a proprietary communication protocol supported by a specific brand of DER units or via a Web server where a DER unit has exposed its data via an XML feed.

9.1.2 System setup

As mentioned, the Dantherm DKµCHP is a fully running machine, located on a remote location, so there is no physical access to the machine. All that is provided is an XML feed through a web page, that is continuously updated with data from the machine. As no physical contact is possible, a local DER controller cannot be attached to the machine. Therefore, a “simulated” controller or “intermediator” is created, as shown on figure 9.1.

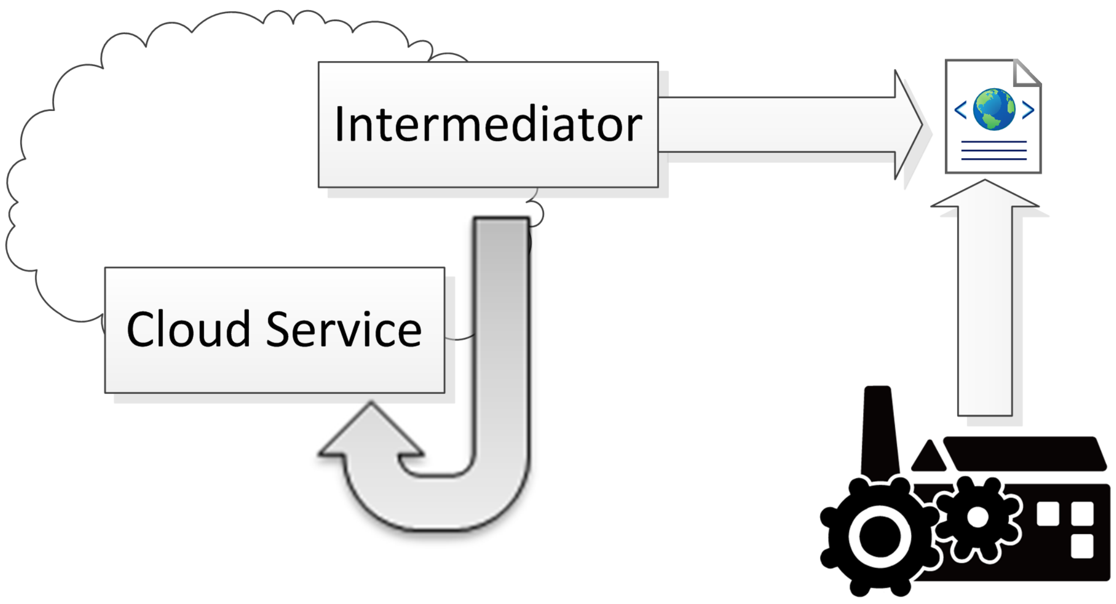


Figure 9.1: Setup for case study : Dantherm DKμCHP

The intermediator first connects and authenticate the associated unit with the Cloud Service. When it is authenticated, it requests the XML feed from the unit and sends it directly into the cloud using the communication WebSocket. When the data is sent, it waits for a confirmation that the data has been retrieved from the Cloud, then waits for 10 seconds to redo the procedure by requesting the XML feed again and sending it. In this setup, the intermediator represents a "normal" DER controller, it is just located elsewhere.

9.1.3 Output, Configuration and Module

The XML feed from the unit consists of 8 different values:

```
<Response>
 <Terminal>
   <Name>Gas input power (W)</Name>
         <Value>2750.134491</Value>
        </Terminal>
        <Terminal>
         <Name>Electrical input power (W)</Name>
         <Value>113.500000</Value>
        </Terminal>
        <Terminal>
         <Name>Thermal output power (W)</Name>
         <Value>1722.240000</Value>
        </Terminal>
        <Terminal>
         <Name>Electrical output power (W)</Name>
         <Value>951.200000</Value>
        </Terminal>
```

```
          <Terminal>
           <Name>System State</Name>
           <Value>Operation</Value>
          </Terminal>
          <Terminal>
           <Name>Control Mode</Name>
           <Value>4</Value>
          </Terminal>
          <Terminal>
           <Name>Load point (\%)</Name>
           <Value>100</Value>
          </Terminal>
          <Terminal>
           <Name>System Running?</Name>
           <Value>1</Value>
          </Terminal>
</Response>
```

From this, the SCL can be constructed, using the logical nodes definitions found in IEC 61850-7-420 standard and the IEC 61850-7-4 standard.

```
<IED name="DNTHRM1">
 <AccessPoint name="S1">
  <Server>
   <Authentication/>
   <LDevice desc="Dantherm DER unit" inst="DNTHRM1">
    <LN0 lnClass="LLN0" inst="" lnType="LLN0a" />
    <LN lnClass="Terminal" inst="" lnType="TERM">
     <DOI name="GasInput"/>
     <DOI name="ElectricalInput"/>
     <DOI name="ThermalOutput"/>
     <DOI name="ElectricalOutput"/>
     <DOI name="SystemState"/>
     <DOI name="ControlMode"/>
     <DOI name="LoadPoint"/>
     <DOI name="SystemRunning"/>
    </LN>
   </LDevice>
  </Server>
 </AccessPoint>
 <DataTypeTemplates>
  <LNodeType id="TERM" lnClass="Terminal">
   <DO name="GasInput" type="GasInput" />
   <DO name="ElectricalInput" type="ElectricalInput" />
   <DO name="ThermalOutput" type="ThermalOutput" />
   <DO name="ElectricalOutput" type="ElectricalOutput" />
   <DO name="SystemState" type="SystemState" />
   <DO name="ControlMode" type="ControlMode" />
   <DO name="LoadPoint" type="LoadPoint" />
```

```
    <DO name="SystemRunning" type="SystemRunning" />
   </LNodeType>
   <DOType id="GasInput" cdc="GasInput">
    <DA name="instMag.f" fc="MX" bType="FLOAT32" />
   </DOType>
   <DOType id="ElectricalInput" cdc="ElectricalInput">
    <DA name="instMag.f" fc="MX" bType="FLOAT32" />
   </DOType>
   <DOType id="ThermalOutput" cdc="ThermalOutput">
    <DA name="instMag.f" fc="MX" bType="FLOAT32" />
   </DOType>
   <DOType id="ElectricalOutput" cdc="ElectricalOutput">
    <DA name="instMag.f" fc="MX" bType="FLOAT32" />
   </DOType>
   <DOType id="SystemState" cdc="SystemState">
    <DA name="stVal" fc="ST" bType="Unicode255" />
   </DOType>
   <DOType id="ControlMode" cdc="ControlMode">
    <DA name="stVal" fc="ST" bType="INT32" />
   </DOType>
   <DOType id="LoadPoint" cdc="LoadPoint">
    <DA name="stVal" fc="MX" bType="INT32" />
   </DOType>
   <DOType id="SystemRunning" cdc="SystemRunning">
    <DA name="stVal" fc="ST" bType="INT32" />
   </DOType>
  </DataTypeTemplates>
 </IED>
```

With the SCL describing the structure of the unit (seen from the XML feeds point of view), the module to handle the raw data, also needs to be created. To do so, an assembly based on the *ModuleBase* class library project, with a class inheriting the *IModule* interface is created. The function *ParseData* (see section 7.6) takes in the raw string, and using the built-in *XDocument* class, it parses through each node. For each node it calls the *AddValue* on the instantiated *StorageBuilder*. This creates an internal dictionary with a collection of values in the cloud solution, that can then be passed into the Table Storage. As an example for adding data to the storage builder, the value of the “Gas input power” is added with the following line:

```
storageBuilder.AddValue(node.Element("Value").Value,
   DataTypeEnum.FLOAT32, "Terminal.GasInput", "Terminal.GasInput
   .instMag.f", "Gas input power", SIUnit.Watt, Multiplier.Kilo)
   ;
```

9.1.4 Unit Configuration

When the module and the SCL configuration file have been created, the files can be uploaded into the cloud solution and a unit can be created. This is done through the web page (figure D.1), by authenticating with a user and navigating to the "Upload Module" page (figure D.18). The module is given the name "Standard.DKuCHP" and the version number 1.0. Then the selected SCL configuration file is input with the version number 1.0.

When the information is posted to the web site, the unit can be created on the "Create Unit" page (figure D.13). "Dantherm Unit 2" is used as the unit name.

The module is selected from the drop down box, and the information is then posted. This allows the user to view the new ID and password associated with the unit (figure D.8):

Unit ID: 9223372036854775607
Password: 11Tv9q|uWx)Hq64:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ

These values are then configured in the intermediator, which then connects to the cloud, authenticates, retrieves the feed from the DER unit and sends it to the cloud.

9.1.5 Data Retrieval

With the unit and the intermediator running and posting data to the cloud, the data can be retrieved through different interfaces. A user with the following credentials can be used: Username: `user1` Password: `password1`

9.1.5.1 *Using the Website*

To view the posted data on a web page, the unit is simply found in the user's list of units (figure D.7). On the right side the "View Data" is click-able and navigates to a new page (figure D.29), where all the posted data can be viewed by session ID listed after most current time stamp. Accessing one of the sessions (figure D.30), shows all of the stored information in a simple view.

9.1.5.2 *Query Interface*

Another way is for the user to go through the Web API and query data using a RESTful interface. To do this, the unit is queried by following the notation described in section 6.1.2. As an example, a query to get the complete information model, would look like this:

http://cloudsolution39.cloudapp.net/api/query/9223372036854775607

The query returns an IEC 61850 compliant information model in XML:

```xml
<Server>
 <LD Name="DNTHRM1" Ref="DNTHRM1">
  <LN Name="LLN0" Type="LLN0" Ref="DNTHRM1/LLN0" />
  <LN Name="Terminal" Type="Terminal" Ref="DNTHRM1/Terminal">
   <DO Name="GasInput" Type="GasInput" Ref="DNTHRM1/Terminal.
      GasInput">
      <DA FC="DC" Name="instMag.f" Type="FLOAT32" Ref="DNTHRM1/
         Terminal.GasInput.instMag.f">0.000000</DA>
    </DO>
    <DO Name="ElectricalInput" Type="ElectricalInput" Ref="
       DNTHRM1/Terminal.ElectricalInput">
      <DA FC="DC" Name="instMag.f" Type="FLOAT32" Ref="DNTHRM1/
         Terminal.ElectricalInput.instMag.f">47.500000</DA>
    </DO>
    <DO Name="ThermalOutput" Type="ThermalOutput" Ref="DNTHRM1/
       Terminal.ThermalOutput">
      <DA FC="DC" Name="instMag.f" Type="FLOAT32" Ref="DNTHRM1/
         Terminal.ThermalOutput.instMag.f">0.000000</DA>
    </DO>
    <DO Name="ElectricalOutput" Type="ElectricalOutput" Ref="
       DNTHRM1/Terminal.ElectricalOutput">
      <DA FC="DC" Name="instMag.f" Type="FLOAT32" Ref="DNTHRM1/
         Terminal.ElectricalOutput.instMag.f">-34.500000</DA>
    </DO>
    <DO Name="SystemState" Type="SystemState" Ref="DNTHRM1/
       Terminal.SystemState">
      <DA FC="DC" Name="stVal" Type="Unicode255" Ref="DNTHRM1/
         Terminal.SystemState.stVal">Safelock</DA>
    </DO>
    <DO Name="ControlMode" Type="ControlMode" Ref="DNTHRM1/
       Terminal.ControlMode">
      <DA FC="DC" Name="stVal" Type="INT32" Ref="DNTHRM1/Terminal.
         ControlMode.stVal">4</DA>
    </DO>
    <DO Name="LoadPoint" Type="LoadPoint" Ref="DNTHRM1/Terminal.
       LoadPoint">
      <DA FC="DC" Name="stVal" Type="INT32" Ref="DNTHRM1/Terminal.
         LoadPoint.stVal">0</DA>
    </DO>
    <DO Name="SystemRunning" Type="SystemRunning" Ref="DNTHRM1/
       Terminal.SystemRunning">
      <DA FC="DC" Name="stVal" Type="INT32" Ref="DNTHRM1/Terminal.
         SystemRunning.stVal">0</DA>
    </DO>
   </LN>
  </LD>
</Server>
```

9.1.6 Summary

The previous case study includes multiple use cases defined in the requirements chapter. It shows how to create a unit, upload a module and SCL configuration file , accessing unit data, connecting to the cloud and how to query data. Moreover, it shows that the system fulfills the functional requirements.

As only 8 data point was available for the DER, the case study do not show how to view the flexibility interface as it is not supported by the DER. Furthermore, it is not possible to post data to the DER, as it is only an intermediator and not a real DER controller.

When querying the data in the cloud, it goes directly into the Azure Table Storage and retrieves the latest data set. It is currently not supported to make a “direct retrieval” from the unit, but as the unit is posting data in real-time to the cloud, this should not be a problem.

9.2 Case study : BeagleBone Communication

The second case study will show, how a BeagleBone[1], running Angstrom Linux on an embedded micro-controller, can be connected to the cloud acting as a DER Controller. Showing that an embedded micro-controller can connect to the Windows Azure platform, and communicate with a Microsoft technology using a Linux based system, is important, as is it demonstrate that there are no limits as to who can connect. A micro-controller based DER controller is a common scenario, with benefits like being able to run with a very low power consumption.

9.2.1 System Setup

The BeagleBone has capabilities to attach boards such as serial board, in order to establish communication between the embedded micro-controller and a foreign unit (e.g. a DER). In the following setup, the BeagleBone will only generate simple values at random, to illustrate a connected DER unit. The programming language used will be Python, as it can run natively on the BeagleBone and is a widely used programming language.

The BeagleBone DER Controller has been created to support the iPower Flexibility Interface, and relevant parts of the information model simulated in the Beaglebone, are exposed through the iPower Flexibility Interface.

9.2.2 BeagleBone

The BeagleBone consists of two boards:

Top Board
On the top board two light bulbs are located, along with two buttons.

Bottom Board
On the button board, three small light bulbs are located.

This can be seen in figure 9.2, where the actual BeagleBone is shown.

[1]BeagleBone: `http://beagleboard.org/bone`

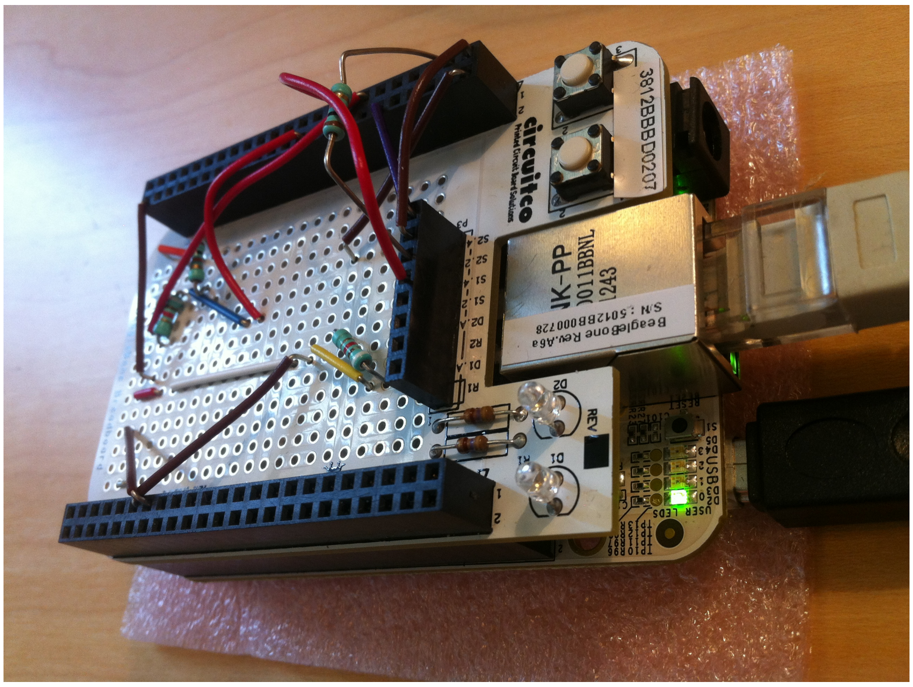


Figure 9.2: BeagleBone

9.2.2.1 *BeagleBone Client*

To establish the communication from the BeagleBone to the cloud, four Python modules have been created:

Main.py
This is the main executable of the Python application. It setup the environment using the remaining python modules.

HardwareIO.py Handles the light state of each of the light bulbs on the board. It does so through direct communication with the system path, for example:

open("/sys/devices/platform/leds-gpio/leds/beaglebone::usr1/brightness", 'w').write("1")

would turn on the light and one of the LED's on the bottom board. The modules also starts a thread running the `pollButtons` function in ButtonsAndLights.py.

WebsocketClient.py
Establishes the connection to the cloud through an imported WebSocket library. It opens a WebSocket and runs in a continuous loop, posting data to the cloud. To connect the BeagleBone to the Cloud Service, only a little Python code is needed. Using PyPI WebSocket client[2] a connection is established, by entering the following code:

```
ws = websocket.WebSocketApp(ws://cloudsolution39.cloudapp.net
    :10101/communication)
ws.run_forever()
```

[2]Python WebSocket library: `http://pypi.python.org/pypi/websocket-client`

With the connection established, posting data is accomplished by the sending a DataTransfer command (Type:6) on the connection, along with the ID and the raw data in a JSON query.

```
ws.send('{"Type":6,"Data":{"ID":"cf6cc8e9-df8b-4f05-8bf2-405
   c06a15a44","Raw":"test data"}}')
```

ButtonsAndLights.py

The modules[3] helps establish communication with the button on the top board and for turning on the lights on the top board.

With the established connection, the raw data is transmitted using the following encoding:

```
dataid:value | dataid:value | ...  | dataid:value | crc(md5)
```

The first part of the raw data string, consists of multiple sets of an unique data ID and its value, separated by a ':'. Each of the data sets is separated with a '|', until the end where an MD5 hash of the data sets in the data string is added.

9.2.2.2 *BeagleBone Module & SCL*

To handle the raw string, the module first verifies that the data string (until the last '|') has the correct MD5 hash. This will verify, that the data has been transmitted correctly and the parsing can continue. The parser splits the raw data string and using the `StorageBuilder` and the data are passed into the *AddValue* function. At last, the added values are committed to the storage and the parser has completed its job.

To interpret the data from the BeagleBone, the following SCL configuration file has been created:

```
<IED name="BBDERC1">
 <AccessPoint name="S1">
  <Server>
   <Authentication/>
   <LDevice desc="Flexibility DER unit" inst="BBDERC1">
    <LN0 lnClass="LLN0" inst="" lnType="LLN0a" />
    <LN lnClass="FLEX" inst="" lnType="FLEXa">
         <DOI name="Type"/>
     <DOI name="ECP"/>
     <DOI name="Status">
      <DAI name="AllowRemCtrl">
       <Val>1</Val>
      </DAI>
      <DAI name="ActivateRemCtrl">
       <Val></Val>
      </DAI>
     </DOI>
    </LN>
    <LN lnClass="Light" inst="" lnType="LIGHT">
     <DOI name="TopBoard" />
```

[3] http://www.gigamegablog.com/2012/03/16/beaglebone-coding-101-buttons-and-pwm/

```
     <DOI name="BottomBoard" />
    </LN>
   </LDevice>
  </Server>
 </AccessPoint>
</IED>
```

With the data types:

```
<DataTypeTemplates>
 <LNodeType id="LIGHT" lnClass="Light">
  <DO name="TopBoard"    type="TopBoard" />
  <DO name="BottomBoard" type="BottomBoard" />
 </LNodeType>
 <DOType id="TopBoard" cdc="TopBoard">
  <SDO name="Left"  type="StatusValue" />
  <SDO name="Right" type="StatusValue" />
 </DOType>
 <DOType id="BottomBoard" cdc="BottomBoard">
  <SDO name="Left"   type="StatusValue" />
  <SDO name="Middle" type="StatusValue" />
  <SDO name="Right"  type="StatusValue" />
 </DOType>
 <DOType id="StatusValue" cdc="StatusValue">
  <DA name="StVal"  fc="ST" bType="INT32" />
 </DOType>
 <LNodeType id="FLEXa" lnClass="FLEX">
  <DO name="Type" type="FLEXType"/>
  <DO name="ECP" type="ECP"/>
  <DO name="Status" type="FLEXStatus"/>
 </LNodeType>
 <DOType id="FLEXType" cdc="FLEXType">
  <DA name="DERName"   fc="DC" bType="Unicode255"/>
  <DA name="DERType"   fc="DC" bType="Unicode255"/>
 </DOType>
 <DOType id="ECP" cdc="ECP">
  <DA name="GeoConPnt"   fc="CF" bType="Unicode255"/>
  <SDO name="VolConPnt"   type="VolConPnt"/>
 </DOType>
 <DOType id="VolConPnt" cdc="VolConPnt">
  <DA name="mag.f"         fc="MX" bType="FLOAT32"/>
  <DA name="mag.i"         fc="MX" bType="INT32"/>
 </DOType>
 <DOType id="FLEXStatus" cdc="FLEXStatus">
  <SDO name="DERStatus"       type="StatusValue"/>
  <SDO name="RemCtrlErr"      type="StatusValue"/>
  <SDO name="RemCtrlEnabled"  type="StatusValue"/>
  <DA name="AllowRemCtrl"    fc="CO" bType="INT32"/>
  <DA name="ActivateRemCtrl" fc="CO" bType="INT32"/>
 </DOType>
</DataTypeTemplates>
```

The important parts in the SCL configuration file, are the three flexibility nodes, configured under the FLEX logical node. The FLEX node includes: Type, ECP (Electrical Connection Point) and Status. If more information was required, the FLEX logical node would simply be extended with more data objects. In addition to the FLEX logical node, the logical node Light has been configured. The Light node consists of a data object for each of the two boards and contains the status values of each led on the board.

9.2.3 Data Retrieval

As with the previous case study, information from the BeagleBone DER Controller is stored in the Azure Table Storage. Accessing the data is therefore retrieved in the same way, through the web page or the RESTful interface. An example of a response for a RESTful query, can be seen in the following listing:

```
<Server>
 <LD Name="BBDERC1" Ref="BBDERC1">
  <LN Name="LLN0" Type="LLN0" Ref="BBDERC1/LLN0"/>
  <LN Name="FLEX" Type="FLEX" Ref="BBDERC1/FLEX">
   <DO Name="Type" Type="FLEXType" Ref="BBDERC1/FLEX.Type">
    <DA FC="DC" Name="DERName" Type="Unicode255" Ref="BBDERC1/FLEX.Type.DERName">BeagleBone DER Controller</DA>
    <DA FC="DC" Name="DERType" Type="Unicode255" Ref="BBDERC1/FLEX.Type.DERType">Python based prototype DER Controller</DA>
   </DO>
   <DO Name="ECP" Type="ECP" Ref="BBDERC1/FLEX.ECP">
    <DO Name="VolConPnt" Type="VolConPnt" Ref="BBDERC1/FLEX.ECP.VolConPnt">
     <DA FC="MX" Name="mag.f" Type="FLOAT32" Ref="BBDERC1/FLEX.ECP.VolConPnt.mag.f">413</DA>
     <DA FC="MX" Name="mag.i" Type="INT32" Ref="BBDERC1/FLEX.ECP.VolConPnt.mag.i">413</DA>
    </DO>
    <DA FC="CF" Name="GeoConPnt" Type="Unicode255" Ref="BBDERC1/FLEX.ECP.GeoConPnt">0000000x11111111/abcdefg</DA>
   </DO>
   <DO Name="Status" Type="FLEXStatus" Ref="BBDERC1/FLEX.Status">
    <DO Name="DERStatus" Type="StatusValue" Ref="BBDERC1/FLEX.Status.DERStatus">
     <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/FLEX.Status.DERStatus.StVal">1</DA>
    </DO>
    <DO Name="RemCtrlErr" Type="StatusValue" Ref="BBDERC1/FLEX.Status.RemCtrlErr">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/FLEX.Status.RemCtrlErr.StVal">0</DA>
   </DO>
   <DO Name="RemCtrlEnabled" Type="StatusValue" Ref="BBDERC1/FLEX.Status.RemCtrlEnabled">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/FLEX.Status.RemCtrlEnabled.StVal">1</DA>
   </DO>
   <DA FC="CO" Name="AllowRemCtrl" Type="INT32" Ref="BBDERC1/FLEX.Status.AllowRemCtrl">1</DA>
   <DA FC="CO" Name="ActivateRemCtrl" Type="INT32" Ref="BBDERC1/FLEX.Status.ActivateRemCtrl">0</DA>
  </DO>
 </LN>
 <LN Name="Light" Type="Light" Ref="BBDERC1/Light">
  <DO Name="TopBoard" Type="TopBoard" Ref="BBDERC1/Light.TopBoard">
   <DO Name="Left" Type="StatusValue" Ref="BBDERC1/Light.TopBoard.Left">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/Light.TopBoard.Left.StVal">0</DA>
   </DO>
   <DO Name="Right" Type="StatusValue" Ref="BBDERC1/Light.TopBoard.Right">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/Light.TopBoard.Right.StVal">1</DA>
   </DO>
  </DO>
  <DO Name="BottomBoard" Type="BottomBoard" Ref="BBDERC1/Light.BottomBoard">
   <DO Name="Left" Type="StatusValue" Ref="BBDERC1/Light.BottomBoard.Left">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/Light.BottomBoard.Left.StVal">1</DA>
   </DO>
   <DO Name="Middle" Type="StatusValue" Ref="BBDERC1/Light.BottomBoard.Middle">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/Light.BottomBoard.Middle.StVal">0</DA>
   </DO>
   <DO Name="Right" Type="StatusValue" Ref="BBDERC1/Light.BottomBoard.Right">
    <DA FC="ST" Name="StVal" Type="INT32" Ref="BBDERC1/Light.BottomBoard.Right.StVal">1</DA>
   </DO>
  </DO>
 </LN>
</LD>
```

```
52 </Server>
```

As it can be seen, the two boards on the BeagleBone each have their Data Object. The first (TopBoard) contains two lights, where the status of the left light bulb is 1 and the right is 0. The second board contains three light bulbs, where the left is 0 and the other two are 1. The flexibility interface has its own logical node, where three Data Objects are available.

9.2.4 Posting Data

In the case where e.g. a VPP wants to send a control command via the RESTful interface, the VPP would issue a HTTP POST request to the cloud service. As the cloud service is a distributed application running in multiple data centers, the cloud service has to route the message to the correct server instance, where the DER is located. As figure 9.3 shows, the POST command is routed through the Inter-instance communication and then through the duplex WebSocket connection, down to the BeagleBone DER Controller.

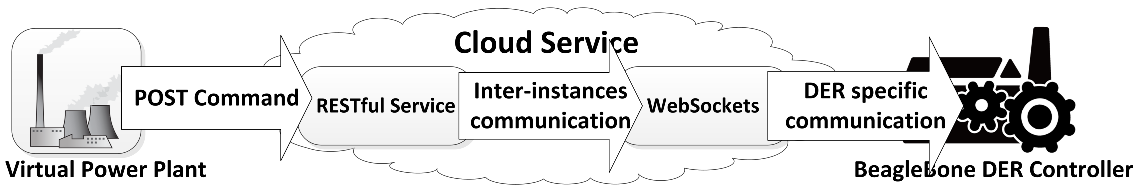

Cloud Service
POST Command
RESTful Service
Inter-instances communication
WebSockets
DER specific communication
Virtual Power Plant
BeagleBone DER Controller

Figure 9.3: Communication flow from a VPP to BeagleBone DER Controller

Sending a post request to turn on the light at *Light.BottomBoard.Left*, would be as follows:

```
POST /api/query/9223372036854775207/BBDERC1/Light/BottomBoard/Left
Host: 168.63.148.192
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 25
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

{ "value" : "1" }
```

The post is transmitted through the cloud service, directed into the correct server instance and then sent the to BeagleBone DER Controller. The response given to the post request, is a confirmation that the message has been received by the DER Controller. To verify that the control value has been set, the change can be found in the next update of the information model in the cloud, as described in section 6.4.2.

9.2.5 Summary

The previous section showed how an embedded micro-controller, running a Linux operating system can connect and use the cloud solution. Using a BeagleBone to represent a DER Controller, simulating connectivity with a DER, the flexibility interface has been implemented as an extension to the IEC 61850 standard. Running an application (written in Python) on the BeagleBone, a connection has been established to the cloud through a WebSocket and (randomized) data is transmitted. With the connection open, it is possible to send a HTTP Post command, through the cloud, down to the DER Controller, in order to turn on/off a LED on the BeagleBone.

9.3 Chapter Summary

With two case studies, each showing different perspective of the implementation, all of the use cases, described in section 3.1, have been carried out.

For the first case study of a Dantherm DKμCHP, the use cases “Creating Unit”, “Upload Module & SCL”, “Accessing Unit Data” and “Connect to CloudSolution” have been the primary focus. Showing how a legacy DER can be connected with the cloud and then present its content. Either through a website or through a RESTful interface.

The second case study focuses on how to implement a DER Controller, based on the new DER-to-cloud concept, on an embedded micro-controller. Using a BeagleBone, it is shown how to simulate a DER connected to a DER hardware unit, posting data to the cloud service, accessing the data as a client and sending commands to the DER Controller. The focus of this case study was primarily the use case “Query Data” and “Post Data”, including the implementation of the iPower Flexibility Interface.

CHAPTER 10

Conclusion

In the following, the thesis will be concluded. The project vision will be revisited from section 1.2 and the goals from section 1.3.2, in order to conclude if the goals have been fulfilled. From this, the implementation of the prototype, results of the case studies and some of the future work and ideas that could be implemented in the project at a later date, will be discussed.

The chapter then completes the thesis by looking into the work that have been conducted concurrently with the thesis and the remarks given by iPower, regarding the thesis contribution to Work Package 4.

10.1 A Revisit of the Project Vision

The vision of the thesis was to create a proof-of-concept prototype, that would show how multiple distributed energy resources could communicate in real-time up to a cloud. The cloud then processes the data from the distributed energy resources and provides easy access to the data, using the IEC 61850 standard as an information model, which includes the iPower Flexibility Interface implemented as additional nodes to the IEC 61850-7-420 standard sub part.

10.2 Summary of Chapters

The overall goal of the thesis has been defined in the project description and is to create a prototype, that can "Communicate to a DER through a cloud-based solution". In order to do this, certain sub-goals needs to be achieved. In collaboration with iPower, a flexibility interface is also to be integrated with the IEC 61850 standard, this should be done as additional nodes to the standard sub part described in IEC 61850-7-420. With this, a clear vision on the final prototype and thesis have been made, along with a risk analysis to specify certain threats during the project phase.

The thesis is based on many different concepts, from the electrical world of the Smart Grid and distributed energy resources to modern cloud computing. The Smart Grid is the "goto" concept, when talking about the electrical grid of "tomorrow". It can intelligently aggregate distributed energy resources, such as wind turbines and electric vehicles through a virtual power plant, which then can make a bid into the danish market.

The Smart Grid is a state-of-the-art grid, where standardization is still under development. Organizations such as CENELEC defines certain aspects on how the Smart Grid should be built, including the architecture, information model and standards, such as IEC 61850. They introduces the concept of flexibility, when communicating with distributed energy resources, but the exact standard implementation is still an ongoing topic. A part of the thesis is to collaborate with iPower-net and implement their definition of flexibility into existing standards.

To handle the electrical grid of "tomorrow", a new concept of communication will also be introduced. Cloud computing is a new concept, where the underlying computational resources are spread out throughout the world. It is a green way to get computational power, that is accessible for all computers with an Internet connection. It is a very reliable and elastic platform, that allows scaling of resources when needed and provides a guarantee for quality of service.

Six use cases have been created in order to define the primary functionality of the prototype. From the use cases and the project goals, functional and non-functional requirements have been created. These requirements defines what the system should do and how it should do it. To ensure that each of the functional requirements are accounted for, a requirement matrix shows the relation between the functional requirements and the use cases.

The analysis of the cloud showed that each of the different service models have different advantages and disadvantages. The IaaS offers full control over the underlying hardware, but requires a lot more technical knowledge/management. PaaS is relatively vendor locked, but it minimizes maintenance and allows for scalability. SaaS is very scalable, but it limits the development capabilities. The PaaS service model has been chosen as the correct layer that the solution will be developed for. It handles scalability and does not require hardware maintenance, and as it is only a prototype, the vendor lock is not a problem.

The three largest PaaS providers on the market (Amazon, Google and Microsoft) have many similarities, but the Azure platform has been chosen due to the .NET framework and the 1 year access through Microsoft ImagineCup. When communicating with the cloud from a DER, WebSockets with push technology for transactions have been chosen as the ideal solution, as it allows for duplex real-time communication and it is the DER that connects to the cloud.

The analysis of standards described the IEC 61850 standard and the information model. It was shown how the data, in the information model, can be referenced using a well defined naming scheme, which allows for a complex data structure. The information model can be inspected using the ASCI interface and the queries can be filtered, using functional constraints.

The iPower Flexibility Interfaces utilizes a service oriented approach, to describe the flexibility provided in DER units. Through the analysis of the standard and the proposed flexibility concept, the Flexibility Interface was added as an additional *Logical Node* to the IEC 61850-7-420 standard, allowing the flexibility concept to be exposed through the existing IEC 61850 standard.

Moving to a cloud-based solution changes a lot of the design aspects, in the existing domain model of the smart grid. To handle multiple distributed energy resources, each DER connects (using a controller) to the cloud solution through a WebSocket, allowing for near real-time duplex communication. Each DER sends the raw data directly up to the cloud, depending on the time intervals defined in the DER unit. To handle the different raw data structures in the cloud, each DER has an associated module. The modules could be developed by the DER manufacturers, as long as they comply with an interface and a general data API for storing the data. The raw data is processed by the module and then stored in the cloud storage.

The cloud solution architecture is designed as a layered architecture, limiting each layer for passing layer-specific values, but also providing the means for changes e.g. changing storage type. When accessing the stored data, a RESTful interface should allow programmable access to query or update data. The information model of the data is based on the information model found in the IEC 61850 standard. When making a query in the RESTful interface, the object reference in the IEC 61850 has been mapped into an URL, by changing the . (dot) to a / (slash). This allows for easy access to each of the nodes in the information model, and with functional constraints, it also provides the functionality for making a filtered query. When the data are retrieved from the storage, it will then (using the SCL associated with the DER module) transform from the general data structure into an IEC 61850 compliant information model. To handle a more user friendly access to the stored data, a web site has been designed to handle the user-, DER- and module-management.

From the initial design, each of the components have been implemented in the .NET framework. The final solution consists of a Web Role, running the MVC 4 based web site, an XML Webservice and the RESTful interface, through the Web API in the .NET framework. The DER WebSocket Interface is implemented as a Worker Role. The solution is centered around the *Core* that is built using multiple components to implement the layered architecture design.

To handle the raw data from the DERs, the Azure Table Storage is used, as this allows for a distributed storage solution. It raised some problems with querying data, as the Azure Table Storage does not support auto incrementing primary keys and sorting when querying, but this was solved by implementing the *UniqueId* class. When querying data, the transformation from raw data into an IEC 61850 compliant information model is accomplished by using an SCL parser. The parser instantiates a data model based on the provided SCL configuration file and then populates and returns a *DER Context* with the data model and with the most recent DER data, fetched from the storage.

Saving DER data to the storage, using a general data structure, is accomplished by providing two interfaces. The module uses *IModule* as base and gets a reference to an *IStorageBuilder* instance injected by the cloud application. Using the provided methods from the storage builder, the modules builds a list of generic values, using a string representation of the values and a description of the data type that should be used to restore the data value.

To transfer the DER data to the cloud, the Alchemy WebSocket library has been used to handle the connectivity. Besides having support for the official RFC6455 standard for WebSockets, the library handles issues, such as Windows 7 not having native support for WebSockets. Two sockets is used in the prototype, one provides functionality to create DER users and upload DER modules. The other provides functionality for transmitting the raw DER data to the cloud application. Both sockets have functionality to authenticate and authorize the user before the rest of the functionality can be used. When communicating from a client to a DER unit, a message is sent through the Azure Service Bus, and using the two-way communication provided by the WebSocket, it is passed to the DER unit.

To test the system, unit tests and integration tests have been used to make the development process more effective and ensure the reliability of existing functionality, as changes are introduced into the code. The build system has enabled automatic testing with the use of continuous integration and can generate of a variety of reports.

With two case studies, each showing different perspective of the implementation, all of the use cases, described in section 3.1, have been carried out.

For the first case study of a Dantherm DKµCHP, the use cases “Creating Unit”, “Upload Module & SCL”, “Accessing Unit Data” and “Connect to CloudSolution” have been the primary focus. Showing how a legacy DER can be connected with the cloud and then present its content. Either through a website or through a RESTful interface.

The second case study focuses on how to implement a DER Controller, based on the new DER-to-cloud concept, on an embedded micro-controller. Using a BeagleBone, it is shown how to simulate a DER connected to a DER hardware unit, posting data to the cloud service, accessing the data as a client and sending commands to the DER Controller. The focus of this case study was primarily the use case “Query Data” and “Post Data”, including the implementation of the iPower Flexibility Interface.

10.3 Achieved Goals

To conclude on the project vision and the project description, the goal “**Communicating to a DER through a Cloud-Based solution**” was created as a top goal for the thesis. To achieve this goal, three different sub goals first had to be achieved.

The first sub goal is “**DER to Cloud Communication**” and has been achieved by establishing real-time communication and security in the communication. Using a WebSocket, as a connection method between the distributed energy resource and the cloud, providing near real-time communication and following the RFC 6455 standard for the WebSocket protocol, allows for OS independent units to connect to the cloud. Basic security in the communication has been implemented, where a unit authenticates using a simple handshake and is then authorized to transfer raw data to the cloud.

The second sub goal is “**Client to Cloud Communication**” and it has four sub goals. Two of these goals are client specific, where special implementation was needed for each client. Connectivity for a Virtual Power Plant requires a programmable API, that can be queried whenever the virtual power plant needs to. To handle this, a RESTful interface and an XML Webservice was implemented. In the RESTful interface an object reference in the information model, is accessed by mapping the reference into an URL. By adding functional constraints to the reference, it is possible to add a filter to the query. Using the XML Webservice provides access to a basic ACSI interface. The sub goal of establishing a connection with the Dong PowerHub has not been met, but as the importance of the stakeholder, Dong Energy, is only medium, this is not critical for the overall project. For handling the sub goal of generic client access, a web site has been created, that enables the clients to create units, upload modules and even view the raw data. All of the communication is based on a user and role-based authentication system, the RESTful interface has support for RSA public key encryption and the XML Webservice uses ’per message encryption’, thus the sub goal of communication security has also been met.

The third and last sub goal is "**Information Model**" which specifies that each distributed energy resource should be able to provide an information exchange model that is based on the IEC 61850 model. More than that, it should also include a notion of flexibility as described in the iPower Flexibility Interface. To handle multiple distributed energy resources, that sends different types of raw data to the cloud, a generic module based system has been implemented, allowing each unit to transform the raw data and save it in the cloud storage. When querying the data from the cloud storage, each module has an associated SCL configuration, that represents the unit's internal components. Using the IEC 61850 information model with the flexibility concept, from iPower, which is added as an additional logical node, a response to a query is formatted in JSON or XML is created and returned as the reply.

Although one goal was not satisfied, the overall goal of "**Communicating a DER through a Cloud-Based solution**" has been fulfilled and the implementation of the prototype was created successfully. To test the implementation of the prototype and to ensure that the use cases in section 3.1 and requirements in section 3.2 have been satisfied, two case studies have been carried out. The first case study enables a Dantherm DKµCHP, as a legacy unit, to connect to the cloud. Using an intermediator, it connects to the cloud service, then it pulls data from the µCHP unit and transmits it to the cloud. The case study's primary focus is on four of the use cases and thereby also on the majority of the functional requirements. The second case study focuses on how to implement a DER Controller, based on the new DER-to-cloud communication concept, on an embedded micro-controller. Using a BeagleBone, it is shown how to simulate a DER connected to a DER hardware unit, posting data to the cloud service, accessing the data as a client and sending commands to the DER Controller. The focus of the last case study was primarily on the last two use cases and thereby fulfilling the remaining functional requirements, including the implementation of the iPower Flexibility Interface.

10.4 Future Work

The project has fulfilled the main goal, that was set forth at the beginning of the thesis. One of the sub goals, the Dong PowerHub integration and several interesting ideas, that were discovered along the way, have been left incomplete. The sub goal and the new ideas includes:

Reporting and logging

Reporting and logging is an interesting part of the IEC 61850 and covers buffered/un-buffered reports and logs (logs are stored on the local unit). Bringing the reports into the communication with the cloud, and adding support for the data structures needed to handle the reports, is definitely doable with the current architecture in the cloud application. When moving forward from the prototype, being able to support the reporting mechanism, would be an expected aspect of a full implementation of the IEC 61850 standard. If the logs stored locally on the DER units is brought into the cloud, it would facilitate a new way of using and accessing the log data. As DER units often have limited storage capabilities, logs are only kept for a limited time. Moving to the cloud, the scalable storage offered by the cloud providers, enables extended storage of the logs for a longer period of time.

Larger scale distributed testing

Expanded testing would help test the feasibility of the cloud service and clarify the limitations with regards to latency and throughput. The selection of DER units used during the development of the prototype was fairly limited, and testing on a larger scale, with units geographically distributed around the world would give a realistic perspective of communication latencies.

Utilization of new services made available with the use of the cloud

By making the DER data accessible in the cloud, it is possible to use the data in new ways, other than simply controlling the DER units from an VPP/aggregator. As the cloud holds data from many DER units, a variety of statistics and behavior patterns can be drawn from the data. Manufactures could pull operating/fault statistics from all of their units, DER owners could compare their DER's efficiency with the general DER in the power grid and energy production patterns of DER units could be used by manufactures to improve their DER design. There are a lot of possibilities and innovative ways to make use of the collected data, which would be valuable for future work.

Dong PowerHub

A sub goal for the project was to interface with the Dong PowerHub, with the time allotted for this project, this integration has been left for future work. The Dong PowerHub supports OPC-UA as the communication protocol, and the design and implementation of a new OPC-UA communication interface is needed. Implementing support for the Dong PowerHub would enable the DER units to be connected to and managed by a real world virtual power plant.

Communication security between DER and cloud

In this project, the communication between the DERs and the cloud service is secured using the user and role-based system and the connection is done through lightweight WebSockets. Based on the fact that many types of DER units are small computing units, an encryption scheme was not applied to the communication. Future work should explore the possibility to apply encryption to the communication, for example using public and private keys or a token based system.

Extended response when communicating from a client to a DER

Currently when a command is sent to a DER, the response only confirms that the message has been received, as described in section 6.4.2. An useful extension to this, would be to enable the DER to reply to a command with a detailed response, for example containing the direct results of the command. The protocol described in section 6.4.1 could be extended to support this.

Two-way communication between the *client* and DER

As mentioned above, a future use case would be to enable a more detailed communication with a DER from a client via the cloud service. Further expanding on this, a future idea, would be to enable two-way communication with the DER unit to and from the client via the cloud service. The framework and DER communication protocols used by the prototype was designed with this future ind mind, and could be developed further to support this.

Communication initiated from the DER to a client

Facilitating the detailed response and two-way communication is based on the assumption, that the clients will initiate the connection to the DER units, as the DER units are usually connected to the cloud service all the time and the clients only connects to the cloud service as needed. An idea for future work, would be to allow the DER units to initiate the connection and send information to the clients trough the cloud service. The DERs are currently connected to the cloud service using a two-way connection, providing near real-time communication. Sending a message from the DER unit, via the cloud service to a client, that is not connected to the cloud most of the time, would require some form of push or extended poll technology to bridge the connection gap between the client and the cloud service.

10.5 Additional Activities

During the project period, the concept of the thesis has drawn a lot of attention. Through Microsoft ImagineCup, the thesis has reached a regional final in the Windows Azure competition. A paper has been written and submitted to the IEEE SmartGridComm 2013 in Vancouver and is already referenced by the iPower platform manager Rasmus Villefrance and the Danish Technological Instutute, by Søren Østergaard Jensen, who is researching how to add the flexibility concept into building construction, rooted in the Smart Grid. Two presentations of the thesis and its implementation have been presented to the iPower consortium, one at DTU and one at RISØ. The external supervisor iPower and DONG Energy, Lars Henrik Hansen, has made the following endorsement (next page) about the thesis, regarding the paper presentation at the IEEE SmartGridComm.

DONG
energy

Til hvem det måtte vedrøre

DONG Energy Wind Power A/S
Nesa Allé 1
2820 Gentofte
Danmark

Tlf. +45 99 55 11 11
Fax +45 99 55 10 45

www.dongenergy.dk
CVR-nr. 18 93 66 74

Anbefaling af Jesper Bach og Lasse Ordas kandidatarbejde

24. maj 2013

Vores ref. Lars Henrik Hansen
Dok. ansvarlig LARHA

larha@dongenergy.dk
Tlf. 9955 8168

Jesper og Lasse deltager i en aktivitet, som har stor betydning for dansk industri. Gennem deres kandidatarbejde er de blandt andet aktive i det store nationale projekt *iPower*, hvor de eksempelvis bidrager med vigtigt arbejde inden for udvikling og opbygning af et kommende standardiseret interface til integration af DER (distribuerede energi ressourcer) enheder, dvs. energiforbrugende/producerende apparater og anlæg ude i elnettet, hvis driftsfleksibilitet skal gøres anvendelig som en fleksibel ressource i det fremtidige elsystem. Omtalte interface skulle gerne danne grundlag for en de facto standard for integration af DER enhederne og dermed give dansk industri en konkurrencemæssig fordel.

Som opgavestiller til Jesper og Lasse kandidatarbejde er undertegnede meget tilfreds med de løsninger, som de har valgt at benytte i deres kandidatarbejde. Deres løsning indeholder en del attraktive muligheder eksempelvis med hensyn til skalerbarhed og allokering af ressourcer. De har 2 gange deltaget i iPower arbejdspakkemøder, hvor de har præsenteret fremdriften i deres arbejde. Endvidere har de skrevet artiklen *"Utilizing a flexibility interface for distributed energy resources through a cloud-based service"*, med henblik på at få den optaget i IEEE SmartgridComm 2013 proceedings, hvilket rækker et godt stykke ud over, hvad man kan forvente i et sædvanligt kandidatarbejde.

Hverken iPower projektet eller DONG Energy har midler til at kunne støtte Jesper og Lasse i at forelægge resultaterne på *SmartGridComm* konferencen. Undertegnede håber derfor, at deres ansøgning bliver vel modtaget.

Med venlig hilsen

Lars Hansen
Lars Henrik Hansen
Senioringeniør, Ph.D.
DONG Energy

Hovedkontor:
Kraftværksvej 53
7000 Fredericia, Danmark

Appendix

Paper

This paper was submitted to the IEEE SmartGridComm international conference on smart grid communications in Vancouver, Canada, 21-24 October 2013.

The focus of the paper is on how the IEC 61850 standard [44] can be used with the CENELEC reference architecture [11] to utilize the iPower Flexibility Interface [7] with a cloud solution architecture.

Conference web page:
`http://sgc2013.ieee-smartgridcomm.org/content/ieee-smartgridcomm`

Utilizing a Flexibility Interface for Distributed Energy Resources Through a Cloud-Based Service

Lasse Dreisig Orda
M.Sc.
Technical University of Denmark

Jesper Bach
M.Sc.
Technical University of Denmark

Anders Bro Pedersen
Ph.d.
Technical University of Denmark

Bjarne Poulsen
Associate Professor, Ph.d.
Technical University of Denmark

Lars Henrik Hansen
Senior Engineer, Ph.d.
DONG Energy

***Abstract*—With governments around the world pushing for an ever increasing shift towards renewable energy production, large numbers of controllable distributed energy resources are starting to appear. Already a multitude of proposed control solutions have seen the light of day, but most are focused solely on the control itself and not the more practical network- and data management issues that follows trying to handle such huge portfolios. This papers covers a cloud based solution to the aforementioned issues, greatly aiding aggregators scale to meet future demands. It also includes a flexibility interface that are currently being researched by iPower, that is mapped to the well tested standard of IEC 61850 as additional sub-nodes. By mapping to existing standards, no major changes would be needed to adapt existing systems.**

I. Introduction

Power production has traditionally been very centralized, with large fossil fired combined heat and power plants carrying the vast majority of the load. In recent years, however, the interest in renewable energy has spiked and more and more dispersed energy resources are starting to appear. In Denmark the government has set an ambitious goal of 50% wind power penetration by 2020 [1], and a complete out-phasing of fossil fuels by 2050.

Because the power system has to be kept in balance, a multitude of new control schemes are being researched in order to properly manage the new resources [2].

From a network-technical perspective, handling connections to a multitude of units constitutes a problem in itself. Not only does it require a significant amount of server capacity, but obtaining access to all DERs through firewalls and the like, is a rather daunting task.

By shifting these more tedious tasks to the cloud, the aggregator not only takes a great step towards true scalability, but also towards increased security. In connecting directly to the cloud, the aggregator no longer needs to keep sockets open for connecting DERs, effectively distancing the critical control algorithms from the outside world, leaving this to the cloud.

This paper covers the development of a cloud based framework, for receiving incoming connections from DERs and mapping their often varying data model into a generic one based on the well tested IEC 61850. It also includes the notion of flexibility into IEC 61850 as additional sub-nodes when communicating with DERs.

II. Cenelec Reference Architecture

Standardization of Smart Grid [3] includes many committees, pilot and research projects, that works both on a national level, but also on a larger international level. At CENELEC [4] a Smart Grid Coordination Group (SG-CG) Reference Architecture Working Group (SG-CG/RA) [5] has been assembled to focus on certain aspects of the Reference Architecture. This includes the means to communicate in a more common view and for integrating various state-of-the-art approaches. Before visiting the underlying Reference Architecture, the Conceptual Model is a top layer model that act as a bridge between the different viewpoints of the Reference Architecture [5].

A. Conceptual Model

More than a century ago, all electrical power supply was decentralized isolated networks, which developed into a mixed European centralized network. Within the past fifteen years, the emergence of Distributed Energy Resources (DER) [6] [7] has increased and with this, the energy network has become increasingly more decentralized [5]. Today the trend is still to become more decentralized [8].

With the integration of DERs into the existing NIST Model [9], a new "DER" Domain has to be integrated into the existing model.

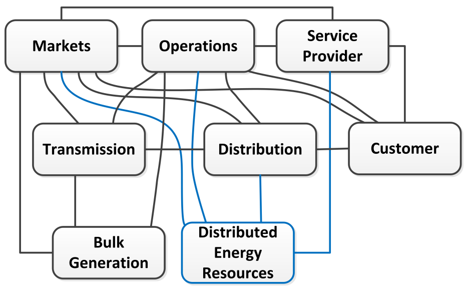


Fig. 1. NIST Model with DER Extension [5]

As a part of this new integration, the M490 Working Group[1] introduced the concept of flexibility. The flexibility combines the model, consumption, production and storage into one flexibility entity. This should allow for more flexibility in the NIST Model, and support future *demand-response* use cases, as non-IT knowledgeable users would be able to create use cases, that more easily could be implemented. Table I shows the NIST Domains, where the flexibility concept could be used.

[1]SG-CG/RA and SG-CG/SP

Domain	Market	Grid	Flexibility
Markets	X		
Bulk Generation			X
DER			X
Customer			X
Transmission		X	
Distribution		X	
Operations	X	X	X
Service Provider	X	X	X

TABLE I. FLEXIBILITY IN NIST DOMAINS [5]

B. *Reference Architecture*

An important part of the Reference Architecture, is that it is based on existing materials, such as the NIST Conceptual Model [9] and the GridWise Architecture Council [10]. The Smart Grids Architecture Model (SGAM) framework [5] consists of five layers, that have been abstracted from the Joint Working Group[2] recommendations, where the architecture is divided into four layers. The five layers are as follows:

- Business Layer
- Functional Layer
- Information Layer
- Communication Layer
- Component Layer

The intention of this abstraction model, is to represent the information management zones where interactions is identified.

Implementing a flexibility interface that follows the presented criteria in the Conceptual Model, will be a subject to the *Information Layer* and the *Communication Layer*. The *Information Layer* is described as the information, which is being exchanged between the components using an underlying common data model. It also includes the semantics for functions and services for the communication. The *Communication Layer* is defined as a layer for describing the protocols and the technology for the interoperability between each component.

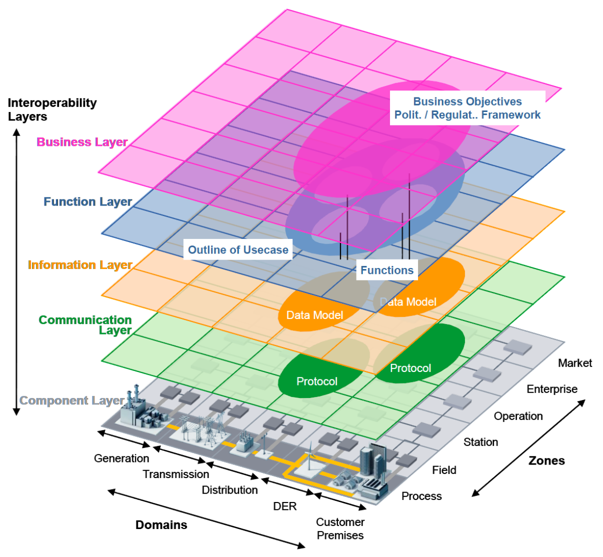


Fig. 2. SGAM Framework [5]

[2]CEN, CENELEC and ETSI Activity

From the two layers, a formal Information Architecture can be presented, that integrates the data management and the interfaces needed. As integration technology a SOA solution [11] is preferred, as it integrates the technical concepts of enterprise application integration with the organizational embedding. Nevertheless, with the more decentralized systems, standards for communications is needed. CENELEC presents the IEC 62541 OPC UA [12], a SOA-based approach for data exchange between each domain and zones in figure 2. CENELEC does not recommend any specific standards for how the data is stored and accessed in the Data Model, but existing standards such as the IEC 61850 [13] are the most prominent data models.

III. IEC 61850-7-420 STANDARD

The IEC 61850 standard [13] is a tested standard [14], that offers a structural data model of a unit and each of its sub component. As part of the standard the IEC 61850, the IEC 61850-7-420 [15] describes the basic communication structure for distributed energy resources logical nodes. This allows for communication not only locally between DER units and between the management systems, but also between a complete DER plant and the operators or aggregators. The information model of IEC 61859-7-420 primarily utilizes the existing information model associated with the IEC 61850 standard, but extends it with some DER-specific logical nodes where it is needed.

A. *Information Model*

To represent a physical substation environment, figure 3 illustrates the IEC 61850 information model using a layered hierarchy.

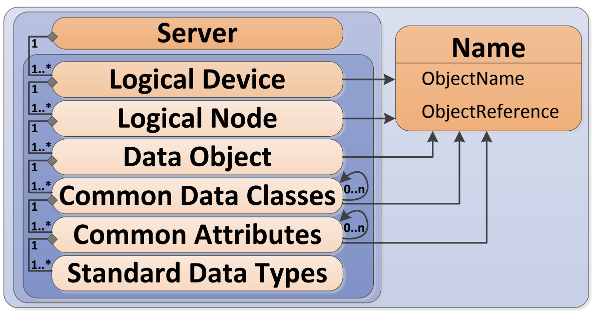


Fig. 3. Information Model Hierarchy [16]

The logical device (LD) is a sub component of the physical device and is composed of multiple logical nodes. Each logical node (LN) holds a predefined number of data objects and each represents a specific function of the device. In each data object (DO) a predefined name of the associated logical node and their type or format is defined using a common data class. Common Data Classes (CDC) is a set of standard data types and predefined common attributes. Each predefined common attribute, can be reused in different objects and contains more common attributes or standard data types.

B. Naming Convention

To access a data object within the information model, the model (as seen in figure 3) can be thought of as a tree like structure, where the base root is the logical device. As each of the objects in the structure has an associated `ObjectName`, IEC 61850-7-2 states that the path of the data object can be referenced as follows [16]:

LD/LN.DO[.DO[. ...]].DA[.DA[. ...]]

Fig. 4. Naming Convention [15]

This makes a clear and unambiguous path reference for each object in the information model.

C. New Logical node

For a DER unit or a DER plant, an Electrical Connection Points (ECP) is needed for defining the properties between one or more DER units and the electric power system. With the IEC 61850-7-420 standard, a specific logical node is required for communication with each ECP. This provides a means for operating the connectivity for each DER unit, but it also sets requirements for the communication. The DPST node [15] provides real-time status and measurements at the ECP, and communication to and from the ECP, would therefore be required to be in real-time as well, in order to not loose functionality in the communication layer.

As an extension to the IEC 61850-7-420, iPower has created an information model to handle the flexibility requested from CENELEC.

IV. iPower Flexibility Interface

The iPower project is a collaboration of 32 partners including universities, research institutions and companies in the industry. The project has been granted support from SPIR [17] - Strategic Platform for Innovation and Research and aims to develop and grow the use of Smart Grid in the Danish industry.

The Smart Grid Reference Architecture [5] as described by CEN-CENELEC-ETSI introduces the flexibility concept where consumption, production and storage are grouped together as an entity and ranked alongside entities, like the power grid and power markets. It is expected, that the flexibility concept will satisfy future requirements to support use-cases, where more flexibility is needed, than what the current rigid standards provide. The flexibility model should enable maximum reuse of the results derived from the models, defined in the current standards. The iPower project [18] has designed an information model of this concept in the form of the iPower Flexibility Interface [19].

The iPower Flexibility Interface is an information model designed for Direct Control of Distributed Energy Resources (DER). The flexibility interface provides a management-oriented approach for communicating the flexibility of DERs to e.g. an Aggregator or a Virtual Power Plant.

DERs consists of power production units such as wind turbines, photovoltaics or flexible consumption units, such as heating and cooling systems or electric vehicles. The flexibility interface addresses the communication of the flexibility, the characteristics, that a DER exposes, between an aggregator or virtual power plant and the DER. An aggregator manages multiple DERs, in order to provide an accumulated response of power production or consumption. This enables an aggregator to buy or sell power on the energy markets or support the power grid as ancillary services. DERs are equipped with local controllers and are able to operate autonomously. When the DERs are connected to an aggregator, they are able to alter their consumption/production of active/reactive power in response to signals from the aggregator. The flexibility of a DER, describes the capabilities of the unit, operating costs and schedules. For example, a DER may have to prioritize local needs for power production, and would be unable to meet a request for power from the aggregator, but other DERs may be able to fulfill this request. The flexibility interface describes how to communicate this information to the aggregator or virtual power plant.

The iPower flexibility interface supports a direct control setup, between the DERs and the aggregator. Direct control refers to a setup, where a two-way communication exists between aggregator and DER. The DERs reports their local flexibility to the aggregator and the aggregator controls the DERs based on this information. The flexibility interface, as illustrated in figure 5, is situated between the aggregator/virtual power plant and the DERs.

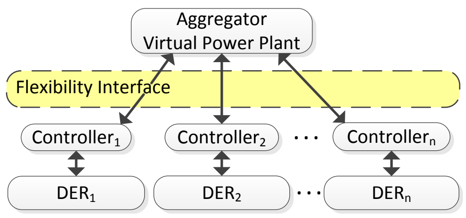


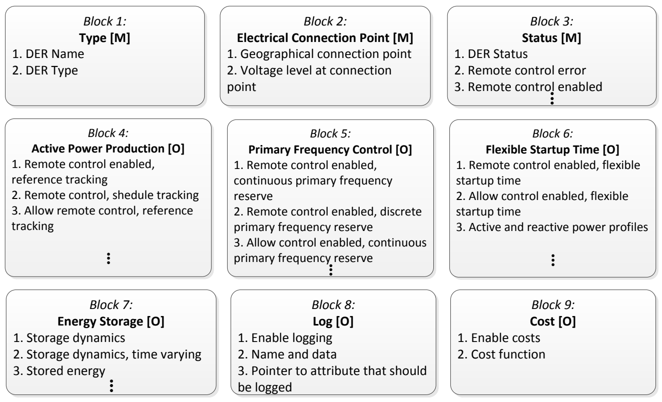
Fig. 5. The flexibility Interface sits between the aggregator and the DER units [19].

The following is a selection of the characteristics described in the iPower Flexibility Interface:

- **Geographical location** The location of a DER in the power grid.
- **Combined deliveries** DERs can be either pure consumption units, pure productions units or a combined consumption and productions units.
- **Active/reactive power** Both the active and reactive power is required to be communicated to the aggregator, in order for the aggregator to properly place bids on the power markets.
- **Tracking** The ability to track a given remote power reference or to track a planned power schedule.
- **Runtime** Minimum and maximum runtime. E.g., a unit must run for at least 30 min when started.
- **Downtime** Minimum and maximum downtime. E.g., when a unit is turned off, it must be turned off for at least 1 hour.
- **Cost** The aggregator needs to know the cost associated with the active/reactive power production, energy level dependent, unit start-up, unit shutdown for discrete power units.

- **Contracts** For an aggregator to be able to manage a DER, some contract or agreement must exist between the aggregator and the DER. This is communicated through the flexibility interface.

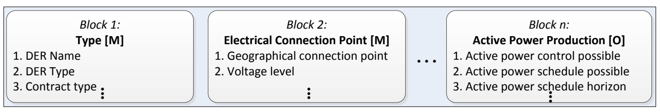
The information model of the flexibility interface is based on these characteristics, and is divided into a number of flexibility blocks, which form the basis of the flexibility interface. In figure 6, a graphical overview of the flexibility blocks is presented.



Block 1:
Type [M]
1. DER Name
2. DER Type
Block 2:
Electrical Connection Point [M]
1. Geographical connection point
2. Voltage level at connection point
Block 3:
Status [M]
1. DER Status
2. Remote control error
3. Remote control enabled
Block 4:
Active Power Production [O]
1. Remote control enabled, reference tracking
2. Remote control, shedule tracking
3. Allow remote control, reference tracking
Block 5:
Primary Frequency Control [O]
1. Remote control enabled, continuous primary frequency reserve
2. Remote control enabled, discrete primary frequency reserve
3. Allow control enabled, continuous primary frequency reserve
Block 6:
Flexible Startup Time [O]
1. Remote control enabled, flexible startup time
2. Allow control enabled, flexible startup time
3. Active and reactive power profiles
Block 7:
Energy Storage [O]
1. Storage dynamics
2. Storage dynamics, time varying
3. Stored energy
Block 8:
Log [O]
1. Enable logging
2. Name and data
3. Pointer to attribute that should be logged
Block 9:
Cost [O]
1. Enable costs
2. Cost function

Fig. 6. Flexibility Interface frame blocks [19]

A flexibility model of a specific DER is constructed, using the relevant blocks to describe its flexibility. This concept is shown in figure 7 and illustrates the information model of a single DER unit, constructed of a collection of blocks comprising the DERs flexibility information model. In the figure, the flexibility information model consists of n blocks, each block being comprised by a number of attributes containing the actual data. The specific flexibility information model for a device is known as a flexibility frame.



Block 1:
Type [M]
1. DER Name
2. DER Type
3. Contract type
Block 2:
Electrical Connection Point [M]
1. Geographical connection point
2. Voltage level
Block n:
Active Power Production [O]
1. Active power control possible
2. Active power schedule possible
3. Active power schedule horizon

Fig. 7. Example of a flexibility frame [19]

In a flexibility frame, some blocks are mandatory and some are optional, denoted in the figure with [M] for mandatory and [O] for optional.

A. Flexibility Interface as an extension to IEC 61850-7-420

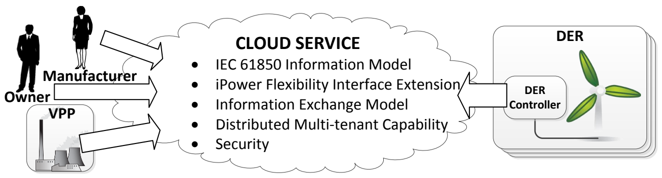
In order to use the flexibility interface with virtual power plants in the industry, the interface must be able to be communicated using the existing standards for DER management. The IEC 61850-7-420 [15] standard handles communication of data from a specific range of DER types defined in the standard. As IEC 61850-7-420 has a similar conceptual model compared to the flexibility interface, this makes IEC 61850-7-420 a logical extension point for the flexibility interface. The implementation of the flexibility interface, requires a new set of logical nodes and classes to be added to IEC 61850-7-420, that can handle the flexibility blocks, but many of the common data classes can be reused. By implementing the flexibility interface as an extension to IEC 61850-7-420, it is possible to provide the flexibility information model as a part of the IEC 61850 [13] information model.

V. Communication in the Cloud

To better utilize the increase of decentralized energy networks, the communication path between SCADA [20] systems/VPPs and the DER units has been revisited. Moving from proprietary dedicated data lines, to modern internet technologies for the communication, arises network boundary and security issues. With impediments such as firewalls, routers and dynamic IP addresses, the current situation of communicating from a server located outside of the network and accessing a local unit e.g. at a farmers house, has to be changed. Instead of connecting from outside the network, the unit now connects from the inside to the server on the outside. This resolves a lot of the complications mentioned, but it brings along new criteria for the server to handle multiple connections and the amount of data that is being sent to the server.

With the cloud [21] becoming more commonly used on the internet [22] and with the research behind the security [23] [24], data handling [25] etc., the cloud would be an appropriate solution for handling the connections and storing the data.

Figure 8 shows a domain model, where the DER unit is connecting to the cloud. Using a WebSocket, it allows for real-time communication between the DER unit and the cloud. This maintains the criteria of the DPST node in the electrical connection points, as the real-time property between the DER unit and the server remain intact.



Owner
Manufacturer
VPP
CLOUD SERVICE
IEC 61850 Information Model
iPower Flexibility Interface Extension
Information Exchange Model
Distributed Multi-tenant Capability
Security
DER
DER Controller

Fig. 8. Domain model

The concept of dealing with DER-to-cloud communication embraces the idea, that the DER controller should be a basic computing unit, which transmits the raw data that it collects, through the lightweight and language neutral WebSockets, directly to the cloud service. The user that controls the DER submits a custom module, which is able to handle the raw data from the DER unit and transform it into a lightweight data structure, that the cloud service can use. The provided module is automatically and securely loaded, as the DER connects to the cloud service. When the data has been transformed, data monitoring, reporting and logging events and it is then persisted in the cloud service storage, where clients can query it. Finally, any additionally defined events triggered by the incoming data are handled and dispatched to clients if necessary.

From the view of an application client, the cloud solution is a multi-tenant cloud service, that has been implemented using Microsoft Azure[3]. The cloud service has support for communicating with the clients such as a VPP, SCADA system or client monitor applications, through an XML web service

[3]Windows Azure: Microsoft's Cloud Platform http://www.windowsazure.com/en-us/home/features/overview

and a web based RESTful interface. The cloud service exposes two interfaces for a VPP to communicate with using the IEC 61850 standard, mapped onto either the XML web service or the RESTful interface. Using the XML web service to query for information about a DER, returns a response using SOAP XML. The RESTful interface can return a response in either plain XML or JSON. The response type from the RESTful interface is determined by what the client has defined in the accept header in the HTTP request. Figure 9 illustrates the basic components involved with the communication between the VPP, DER and the cloud service.

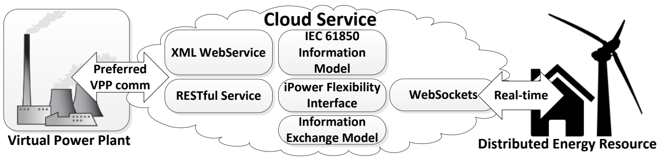

Cloud Service
XML WebService
IEC 61850 Information Model
Preferred VPP comm
RESTful Service
iPower Flexibility Interface
WebSockets
Real-time
Information Exchange Model
Virtual Power Plant
Distributed Energy Resource

Fig. 9. VPP and DER interaction with the cloud

To implement the RESTful interface, which is based on a concept of resource-based access, the object reference in the information model is mapped to be compliant with an URL. Each resource in the data model is accessed via URL nodes and updated the same way. As the naming convention of the IEC 61850 standard resembles a URL, changing the . (dot) to a / (slash) would satisfy the URL requirement [26]. A data object can then be referenced using the following path:

http://hostname/api/unit-id/device/node/class/attribute

RESTful communication uses the standard HTTP methods such as GET, PUT, POST and DELETE to manipulate the resources that are exposed through the interface.

VI. Case Studies

To utilize the Flexibily Interface, it has been implemented as logical nodes in the IEC 61850 standard. The implementation is deployed in the cloud and the following two case studies shows, how to access information on a specified unit and how to send a control command directly to a DER unit. As the implementation of the cloud solution is using industry standard for both the WebSocket and RESTful interface, it is not limited to certain operating system (OS). In the second case study an embedded Linux OS is used for the DER controller, as it does not have the same hardware requirements as e.g. Windows does.

A. Case Study: Communication with the cloud service

This case study shows, how a VPP and other applications can query data from a DER unit, using the RESTful interface provided by the cloud service. The DER unit queried in this case study is a Dantherm Micro CHP from Dantherm Power[4]. To query the RESTful interface, requesting a complete output of the unit, an HTTP GET request is created on the URL path:

http://hostname/api/<unit-id>

The response is in a format where the data stored in the cloud service, is mapped into an XML structure that represents the IEC 61850 object model standard. A simplified output of the XML response from the query on the Dantherm Micro CHP, is shown in listing 1.

```
<Server>
  <LD Name="DNTHRM1" Ref="DNTHRM1">
    <LN Name="LLN0" Type="LLN0" Ref="DNTHRM1/LLN0"/>
    <LN Name="Flexibility" Type="Flexibility" Ref="DNTHRM1/Flex">
      <DO Name="Type" Type="Type" Ref="DNTHRM1/Flex.Type">
        <DA FC="DC" Name="Name" Ref="DNTHRM1/Flex.Type.Name">Dantherm CHP</DA>
        <DA FC="DC" Name="Type" Ref="DNTHRM1/Flex.Type.Type">Micro CHP</DA>
      </DO>
      <DO Name="Electrical Connection Point" Type="ECP" Ref="DNTHRM1/Flex.Ecp">
        <DA FC="CF" Name="Geographical Connection Point" Ref="DNTHRM1/Flex.Ecp.
              ↪ GeoConPt">(8018)064514183456789012/452001452174</DA>
        <DA FC="MX" Name="Voltage Level" Ref="DNTHRM1/Flex.Ecp.VoltageLevel">
              ↪ 4000.0000</DA>
      </DO>
        .
        .
    </LN>
    <LN Name="Terminal" Type="Terminal" Ref="DNTHRM1/Terminal">
      <DO Name="GasInput" Type="GasInput" Ref="DNTHRM1/Terminal.GasInput">
        <DA FC="MX" Name="instMag.f" Ref="DNTHRM1/Terminal.GasInput.instMag.f">
              ↪ 70.0000</DA>
      </DO>
      <DO Name="ElectricalInput" Type="ElectricalInput" Ref="DNTHRM1/Terminal.
            ↪ ElectricalInput">
        <DA FC="MX" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalInput.instMag.f
              ↪ ">56.5000</DA>
      </DO>
        .
        .
    </LN>
  </LD>
</Server>
```

Listing 1. XML Response from the query to the RESTful interface

The response contains both the Flexibility Interface, mapped onto the IEC 61850 model and the standard IEC 61850 data objects for this DER unit.

B. Case Study: Communicating from the VPP to a Dachs DER

This case study explores how an application client or VPP can send control commands to a DER connected to the cloud service, see figure 10. A BeagleBone[5] microcontroller is used as the DER controller and is connected to a Dachs µCHP unit from Senertec through a RS232 serial connection. Using embedded Linux on the BeagleBone, a Python application communicates with the cloud service through a WebSocket.

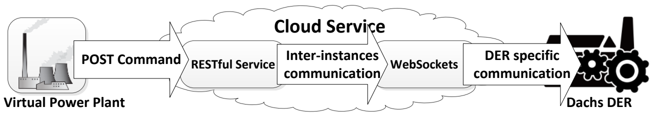


Fig. 10. Communication flow from a VPP to a DACHS DER

In the case where a VPP wants to send a control command via the RESTful interface, the VPP issues an HTTP POST request to the cloud service. As the cloud solution is a distributed application, that can scale to multiple server instances, the POST request has to be routed to the correct server instance through inter-instances communication, to the WebSocket server instance where the µCHP is connected. As the message arrives at the µCHP controller, it updates the values on the Dachs µCHP unit. After the data has been updated on the local µCHP unit, the data model in the cloud service will be updated when new data is sent from the µCHP controller to the cloud.

[4]Dantherm Power http://www.dantherm-power.dk

[5]BeagleBone from http://beagleboard.org/Products/BeagleBone

C. Case Study Conclusion

The cases presented in this section shows, how the Flexibility Interface can be utilized as an extension to the IEC 61850-7-420 standard and that the solution that is built using the cloud, can scale to easily handle large amounts of data, coming from the many DER units. The solution is based on a generic implementation of the IEC 61850 information model, as recommended by CENELEC, and is able to handle the current IEC 61850 data objects, the Flexibility Interface object model and future extensions to the IEC 61850 standard. The data can be accessed using either a RESTful interface or an XML Web service, which outputs the data in a standards compliant XML or JSON format.

VII. Conclusion

The goal of this paper has been to demonstrate how to ultilize a flexible interface for distributed energy resources, following CENELECs recommendation of using the IEC 61850 standard as information model. With the increase in decentralized energy systems, this paper has shown a cloud-based solution, where the requirements of real-time communication from the IEC 61850-7-420 standard has been satisfied using WebSockets. The prototype is able to handle RESTful communication from SCADA/VPP, where the information model is build using IEC 61850 and can be requested through an intuitive URL. Incorporating the flexible interface in the existing IEC 61850 information model is accomplished, by providing additional logical nodes and classes to the information model. This result combines the model, consumption, production and storage into one entity in the IEC 61850 standard. It has the advantages that non-IT knowledgeable users can create *demand-response* use cases to describe future usage requirements.

The advantages of facilitating the communication through the cloud, allows for a forward-looking, scalable and sustainable approach. The clients can access the data of their DER units securely through various means connection types and can communicate with the DERs in real-time, as each DER is connected with the cloud using WebSockets.

The usage of the cloud can be more than just a storage container for data from the DERs. Moving the services that a SCADA system and VPP normally would run locally, to a cloud service, would increases the value potential and add new possibilities for the services that are delivered to the clients of the solution.

References

[1] Danish Energy Agency, *Energy Policy in Denmark*. http://www.ens.dk/Documents/Netboghandel%20-%20publikationer/2013/Energy%20Policy%20in%20Denmark.pdf, 2012.

[2] A. B. Pedersen and E. B. Hauksson, "Enabling distributed energy resources in a virtual power plant using iec 61850," Master's thesis, Technical University of Denmark, DTU, 2010.

[3] H. Farhangi, "The path of the smart grid," *Power and Energy Magazine, IEEE*, vol. 8, no. 1, pp. 18–28, 2010.

[4] Cenelec, *European Committee for Electrotechnical Standardization*. http://www.cenelec.eu/, 2013.

[5] CEN-CENELEC-ETSI Smart Grid Coordination Group, *Smart Grid Reference Architecture*. http://ec.europa.eu/energy/gas_electricity/smartgrids/doc/xpert_group1_reference_architecture.pdf, 2012.

[6] J. Driesen and F. Katiraei, "Design for distributed energy resources," *Power and Energy Magazine, IEEE*, vol. 6, no. 3, pp. 30–40, 2008.

[7] S. You, "Developing virtual power plant for optimized distributed energy resources operation and integration," Ph.D. dissertation, PhD thesis, Technical University of Denmark, 2011.[2] SENERTEC," Technical Documentation CHP unit-DACHS HKA, SENERTEC", vol. Art. Nr. 00/4798, 2010.

[8] S. You, F. Marra, and C. Træholt, "Integration of fuel cell micro-chps on low voltage grid: A danish case study," in *Power and Energy Engineering Conference (APPEEC), 2012 Asia-Pacific*. IEEE, 2012, pp. 1–4.

[9] NIST, *NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 1.0*. http://www.nist.gov/public_affairs/releases/upload/smartgrid_interoperability_final.pdf, 2010.

[10] GridWise, *Smart Grid Interoperability Maturity Model Summary*. http://www.gridwiseac.org/about/imm.aspx, 2013.

[11] P. B. Andersen, B. Poulsen, C. Trholt, and J. Ostergaard, "Using service oriented architecture in a generic virtual power plant," in *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. IEEE, 2009, pp. 1621–1622.

[12] International Electrotechnical Commission (IEC), *IEC 62541 - OPC Unified Architecture*. http://www.iec.ch/, 2010.

[13] ——, *IEC 61850 - Power Utility Automation*. http://www.iec.ch/, 2003.

[14] R. Mackiewicz, "Overview of iec 61850 and benefits," in *Power Systems Conference and Exposition, 2006. PSCE'06. 2006 IEEE PES*. IEEE, 2006, pp. 623–630.

[15] International Electrotechnical Commission (IEC), *IEC 61850-7-420 - Communications systems for Distributed Energy Resources (DER) - Logical nodes*. http://www.iec.ch/, 2008.

[16] ——, *IEC 61850-7-2 - IEC 61850-7-2: Abstract communication service interface (ACSI) - Ed.2*. http://www.iec.ch/, 2003.

[17] Ministry of Science, Innovation and Higher Education, *SPIR - Strategic Platforms for Innovation and Research*. http://fivu.dk/en/research-and-innovation/councils-and-commissions/the-danish-council-for-strategic-research/spir, 2013.

[18] A. Troi., *About ipower*. http://www.ipower-net.dk/About_iPower.aspx, 2012, Mar.

[19] B. Biegel, *Flexibility Interface Information Modeling for Direct Control*. http://www.ipower-net.dk/~/media/iPower/Documents/fi.ashx, 2011.

[20] S. A. Boyer, *Scada: Supervisory Control And Data Acquisition*, 4th ed. International Society of Automation, 2009.

[21] G. Pallis, "Cloud computing: The new frontier of internet computing," *Internet Computing, IEEE*, vol. 14, no. 5, pp. 70–73, 2010.

[22] Cloud Provider USA, *Cloud Computing Statistics and Predictions for 2012*. http://www.cloudproviderusa.com/cloud-computing-statistics-and-predictions-for-2012/, 2012.

[23] B. R. Kandukuri, V. R. Paturi, and A. Rakshit, "Cloud security issues," in *Services Computing, 2009. SCC'09. IEEE International Conference on*. IEEE, 2009, pp. 517–520.

[24] D.-G. Feng, M. Zhang, Y. Zhang, and Z. Xu, "Study on cloud computing security," *Journal of Software*, vol. 22, no. 1, pp. 71–83, 2011.

[25] W. Wang, Z. Li, R. Owens, and B. Bhargava, "Secure and efficient access to outsourced data," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ser. CCSW '09. New York, NY, USA: ACM, 2009, pp. 55–66. [Online]. Available: http://doi.acm.org/10.1145/1655008.1655016

[26] A. B. Pedersen, E. B. Hauksson, P. B. Andersen, B. Poulsen, C. Træholt, and D. Gantenbein, "Facilitating a generic communication interface to distributed energy resources: Mapping iec 61850 to restful services," in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE, 2010, pp. 61–66.

Abbreviations

ACSI	Abstract Communication Service Interface
AWS	Amazon Web Services
CDC	Common Data Class
CHP	Combined Heat and Power
DER	Distributed Energy Resource
DA	Data Attribute
DC	Data Class
DO	Data Object
DT	Data Type
EC2	Elastic Compute Cloud
FC	Functional Constraints
FI	Flexibility Interface
FLEX	Flexible note
GAE	Google App Engine
GOOSE	Generic Object Oriented Substation Events
GSE	Generic Substation Events
GSSE	Generic Substation State Event
IAAS	Infrastructure as a Service
LD	Logical Device
LN	Logical Node
MAC	Message Authentication Code
MMS	Manufacturing Message Specification
MVC	Model View Controller
NIST	National Institute of Standards and Technology

PAAS Platform as a Service

RDS Relational Database Service

S3 Simple Storage Service

SAAS Software as a Service

SAN Storage Area Network

SCL Substation Configuration Language

SCSM Specific Communication Service Mapping

SGAM Smart Grids Architecture Model

SG-CG Smart Grid Coordination Group

SG-CG/RA Smart Grid Coordination Group Reference Architecture

SN Server Node

SNS Simple Notification Service

SOAP Simple Object Access Protocol

SQS Simple Queue Service

TDD Test-Driven Development

V2G Vehicle to Grid

VPC Virtual Private Cloud

VPP Virtual Power Plant

WCF Windows Communication Foundation

WP4 Work Package 4

Video Presentation

During the development of the prototype for this thesis the students took part of the Microsoft ImagineCup[1] competition with the prototype as a base for participating. As part of the ImagineCup competition a video presentation of the project was created. As the video presentation is a part of the competition the focus of the video is on the Microsoft Azure Cloud Platform architecture.

Distributed Energy Resource 2 Cloud
Lasse Orda & Jesper Bach
Technical University of Denmark
Applied Mathematics and Computer Science
00:09 / 11:06
ImagineCup 2013 - DER2Cloud Overview Video
YouTube DK

Figure C.1: Video Presentation on YouTube

The video presentation can viewed here[2]:

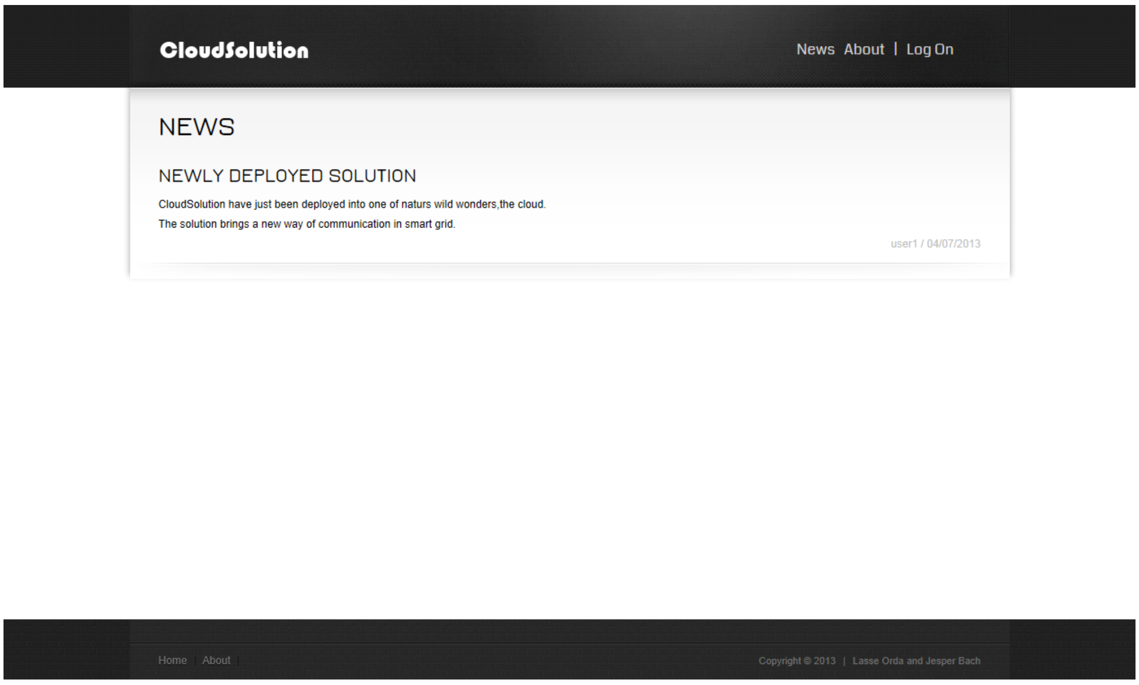
`http://www.youtube.com/watch?v=lRerrlKCDXI`

[1]Microsoft ImagineCup `http://www.imaginecup.com`
[2]Short URL to the video (Case sensitive): **`http://goo.gl/7O4Zf`** (7 OSCAR 4 ZULU foxtrot)

APPENDIX D

Screenshots

D.1 Website



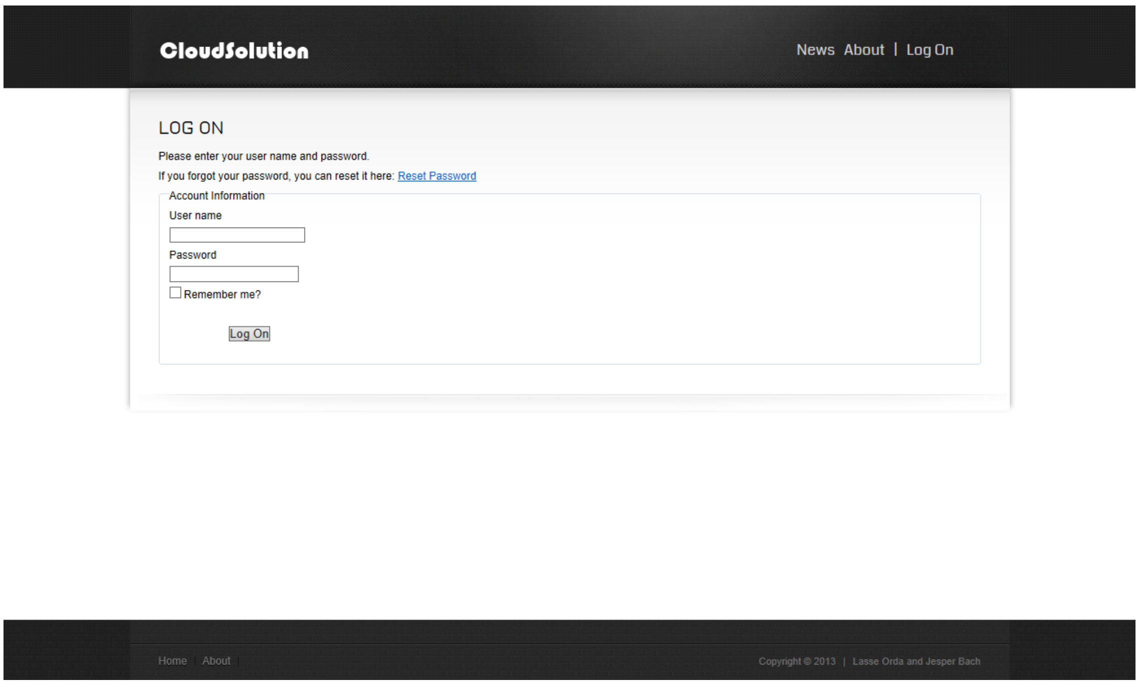
NEWS

NEWLY DEPLOYED SOLUTION

CloudSolution have just been deployed into one of naturs wild wonders,the cloud.
The solution brings a new way of communication in smart grid.

user1 / 04/07/2013

Figure D.1: CloudSolution - News 1



LOG ON

Please enter your user name and password.

If you forgot your password, you can reset it here: Reset Password

Account Information

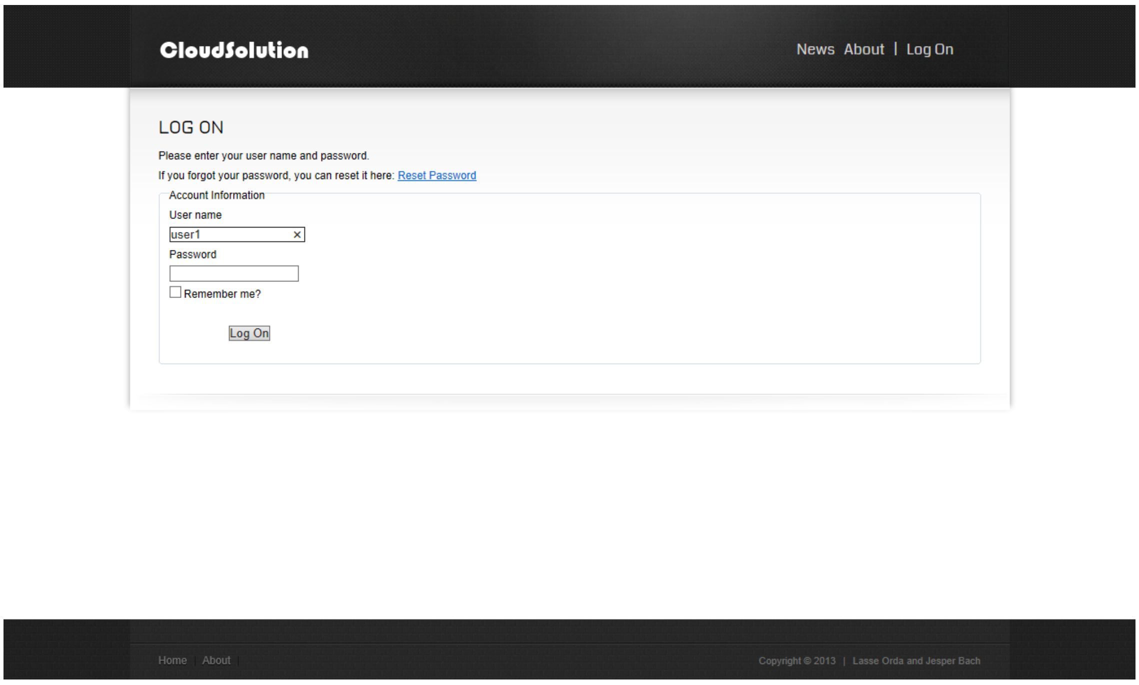
User name

Password

Remember me?

Log On

Figure D.2: CloudSolution - LogOn 1



LOG ON

Please enter your user name and password.

If you forgot your password, you can reset it here: Reset Password

Account Information

User name

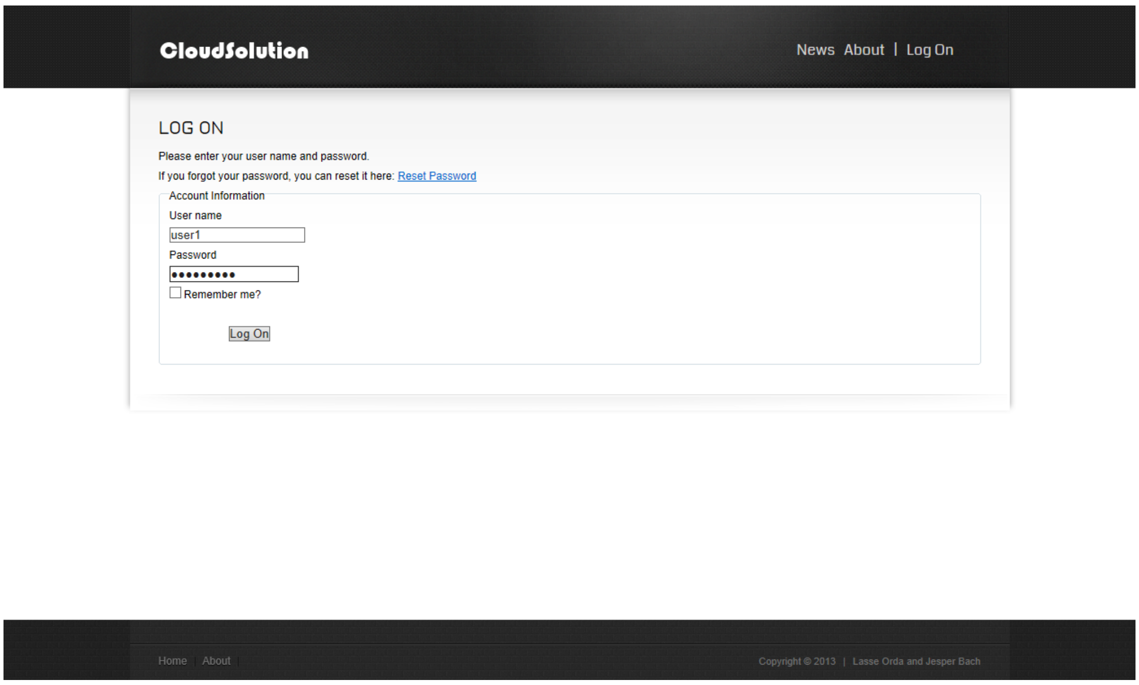
user1

Password

Remember me?

Log On

Figure D.3: CloudSolution - LogOn 2

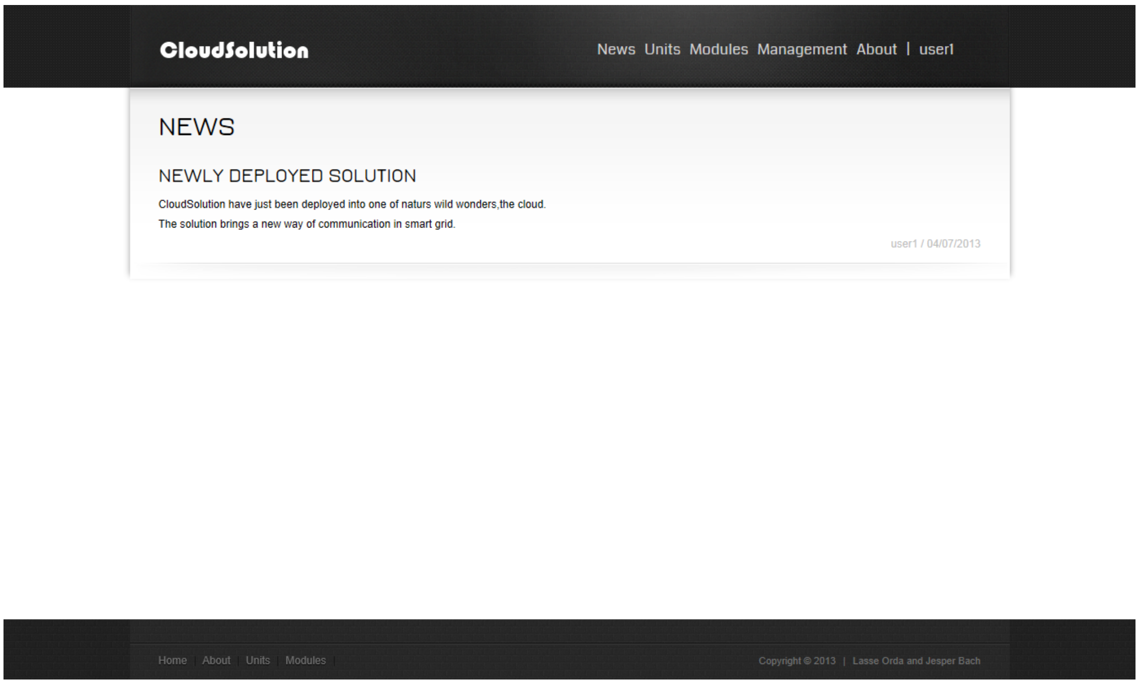


LOG ON

Please enter your user name and password.
If you forgot your password, you can reset it here: Reset Password

Account Information
User name
user1
Password
●●●●●●●●●
Remember me?

Figure D.4: CloudSolution - LogOn 3


CloudSolution
News Units Modules Management About | user1

NEWS

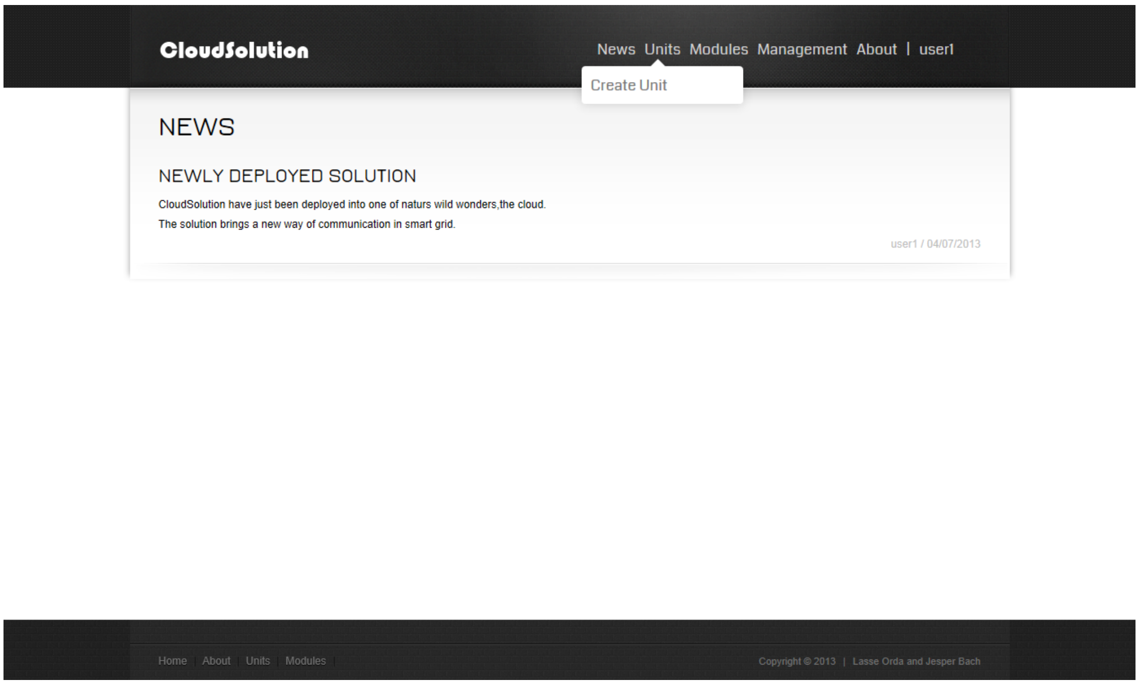
NEWLY DEPLOYED SOLUTION

CloudSolution have just been deployed into one of naturs wild wonders,the cloud.
The solution brings a new way of communication in smart grid.

user1 / 04/07/2013

Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.5: CloudSolution - News 2


CloudSolution
News Units Modules Management About | user1
Create Unit

NEWS

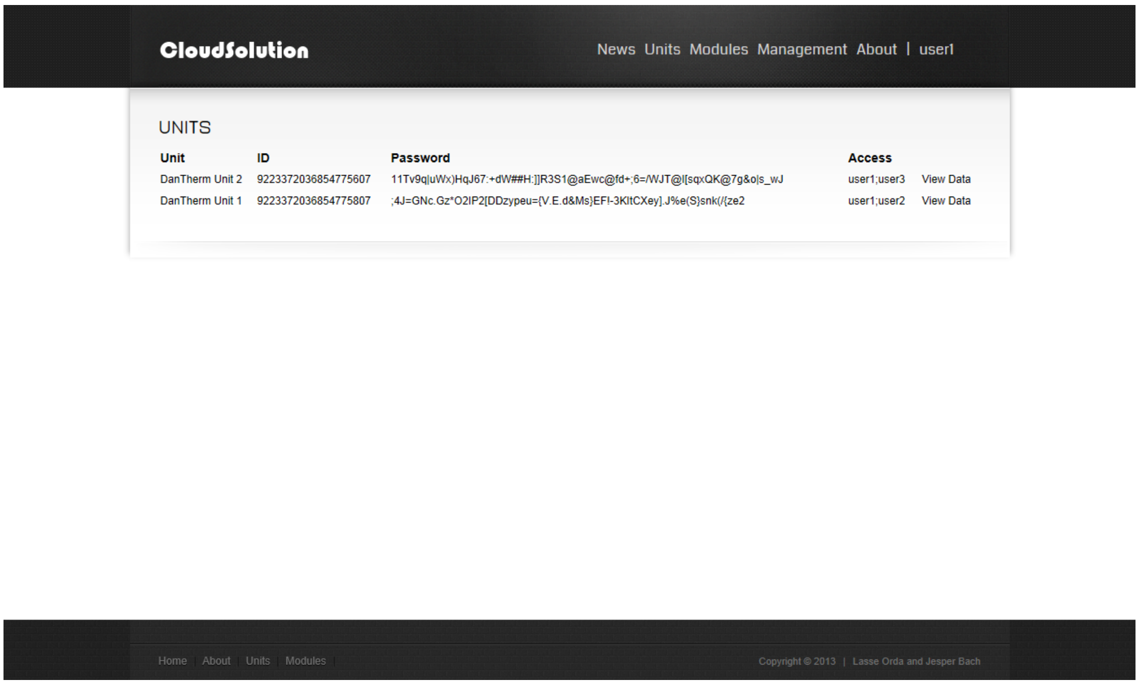
NEWLY DEPLOYED SOLUTION

CloudSolution have just been deployed into one of naturs wild wonders,the cloud.
The solution brings a new way of communication in smart grid.

user1 / 04/07/2013

Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.6: CloudSolution - News 3

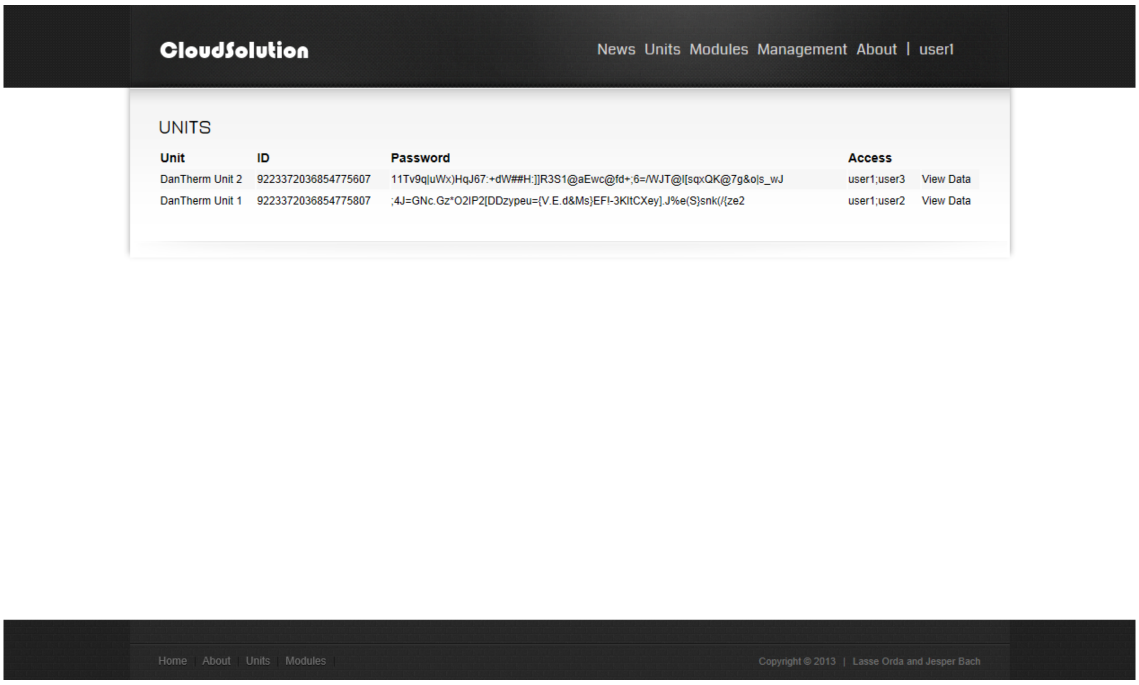

CloudSolution
News Units Modules Management About | user1

UNITS

Unit	ID	Password	Access	
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3	View Data
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EF!-3KltCXey].J%e(S}snk(/{ze2	user1;user2	View Data

Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

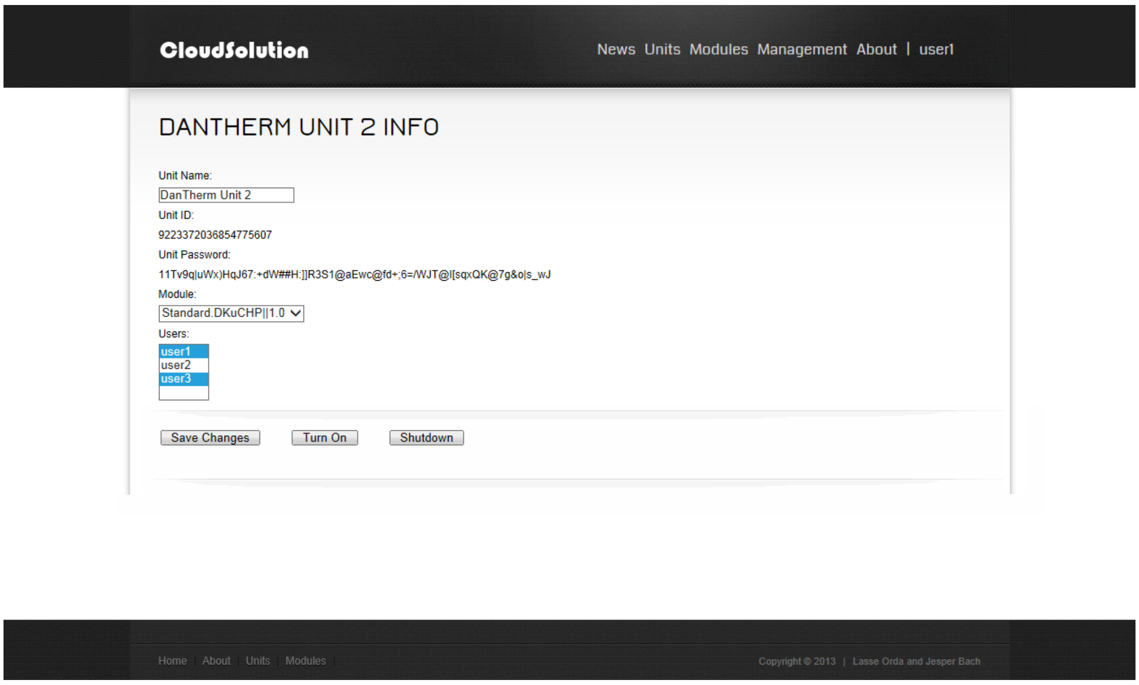
Figure D.7: CloudSolution - Units 1



UNITS

Unit	ID	Password	Access	
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3	View Data
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EF!-3KItCXey].J%e(S}snk(/{ze2	user1;user2	View Data

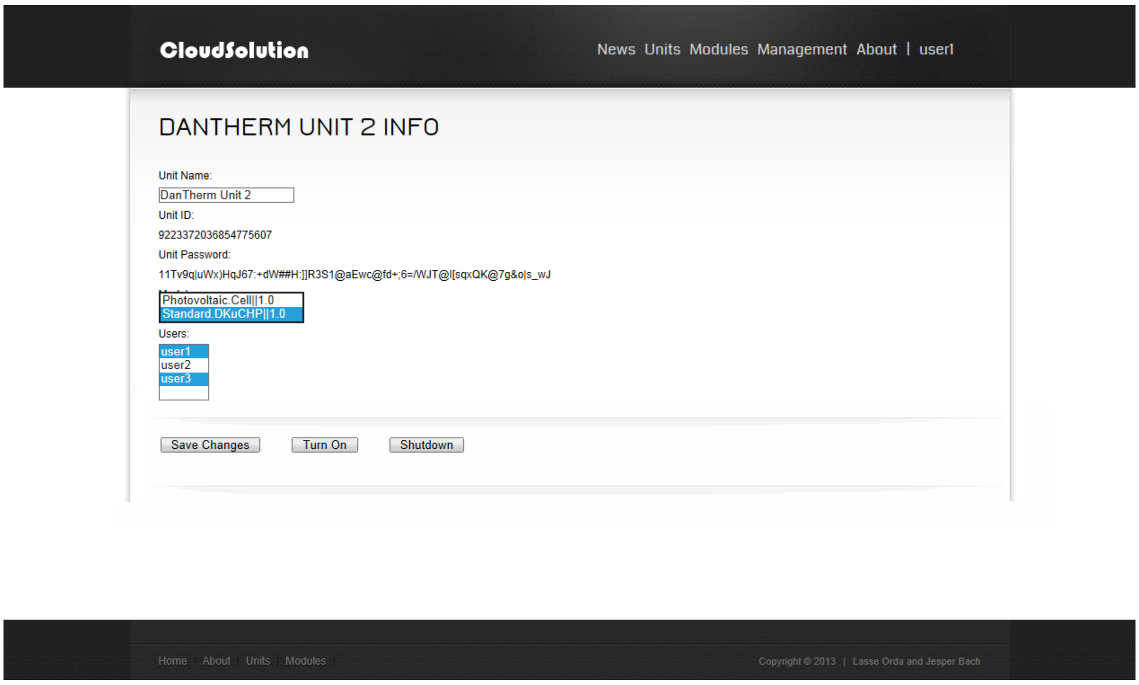
Figure D.8: CloudSolution - Units 2



DANTHERM UNIT 2 INFO

Unit Name:
DanTherm Unit 2
Unit ID:
9223372036854775607
Unit Password:
11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ
Module:
Standard.DKuCHP||1.0
Users:
[✓ user1]
user2
[✓ user3]

Figure D.9: CloudSolution - Info 1


CloudSolution
News Units Modules Management About | user1
Save Changes
Turn On
Shutdown
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

DANTHERM UNIT 2 INFO

Unit Name:

DanTherm Unit 2

Unit ID:

9223372036854775607

Unit Password:

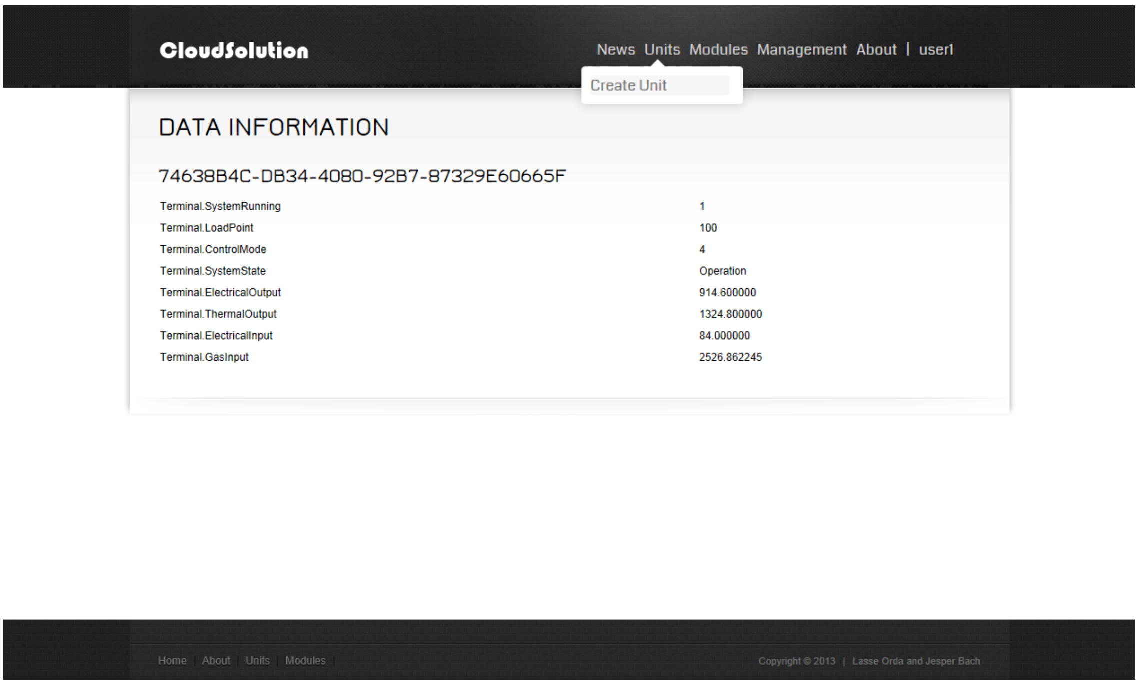
11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ

Photovoltaic.Cell||1.0
Standard.DKuCHP||1.0

Users:

user1
user2
user3

Figure D.10: CloudSolution - Info 2

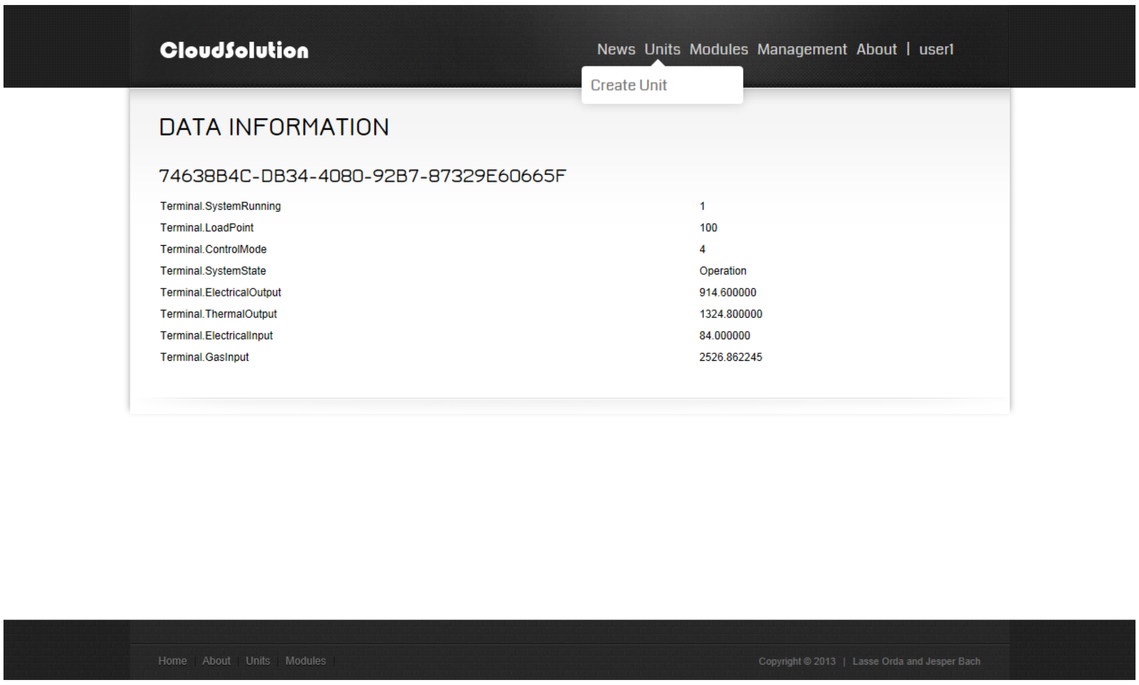

CloudSolution
News Units Modules Management About | user1
Create Unit
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

DATA INFORMATION

74638B4C-DB34-4080-92B7-87329E60665F

Terminal.SystemRunning	1
Terminal.LoadPoint	100
Terminal.ControlMode	4
Terminal.SystemState	Operation
Terminal.ElectricalOutput	914.600000
Terminal.ThermalOutput	1324.800000
Terminal.ElectricalInput	84.000000
Terminal.GasInput	2526.862245

Figure D.11: CloudSolution - Create 1



CloudSolution
News Units Modules Management About | user1
Create Unit

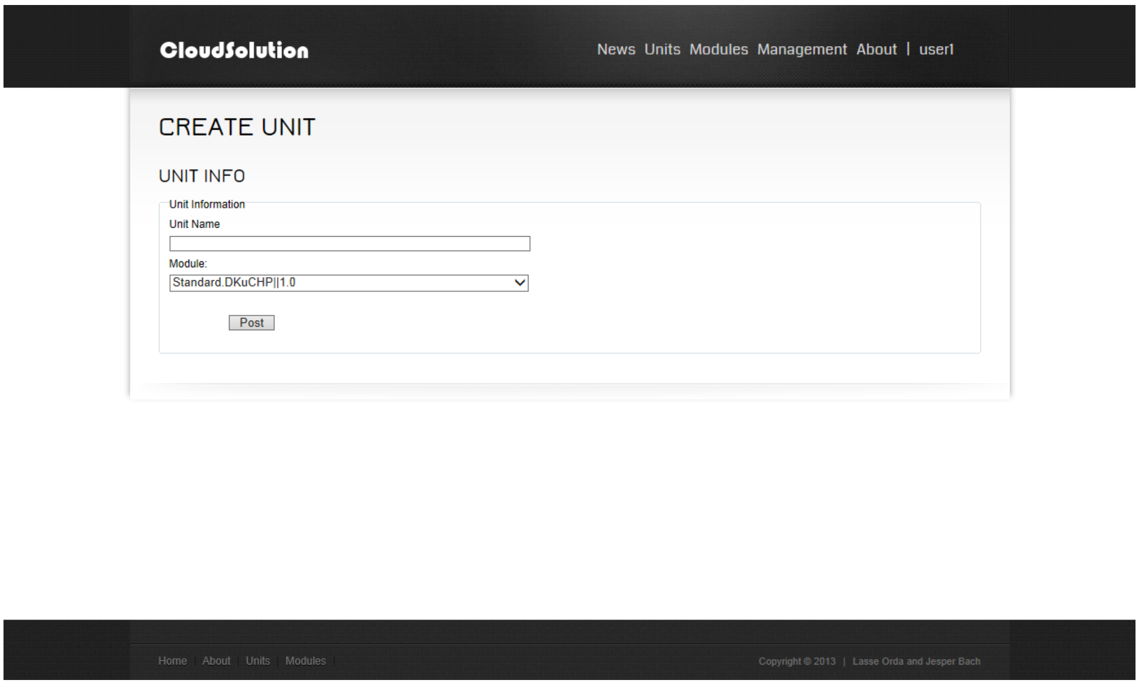
DATA INFORMATION

74638B4C-DB34-4080-92B7-87329E60665F

Terminal.SystemRunning	1
Terminal.LoadPoint	100
Terminal.ControlMode	4
Terminal.SystemState	Operation
Terminal.ElectricalOutput	914.600000
Terminal.ThermalOutput	1324.800000
Terminal.ElectricalInput	84.000000
Terminal.GasInput	2526.862245

Home | About | Units | Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.12: CloudSolution - Create 2



CloudSolution
News Units Modules Management About | user1

CREATE UNIT

UNIT INFO

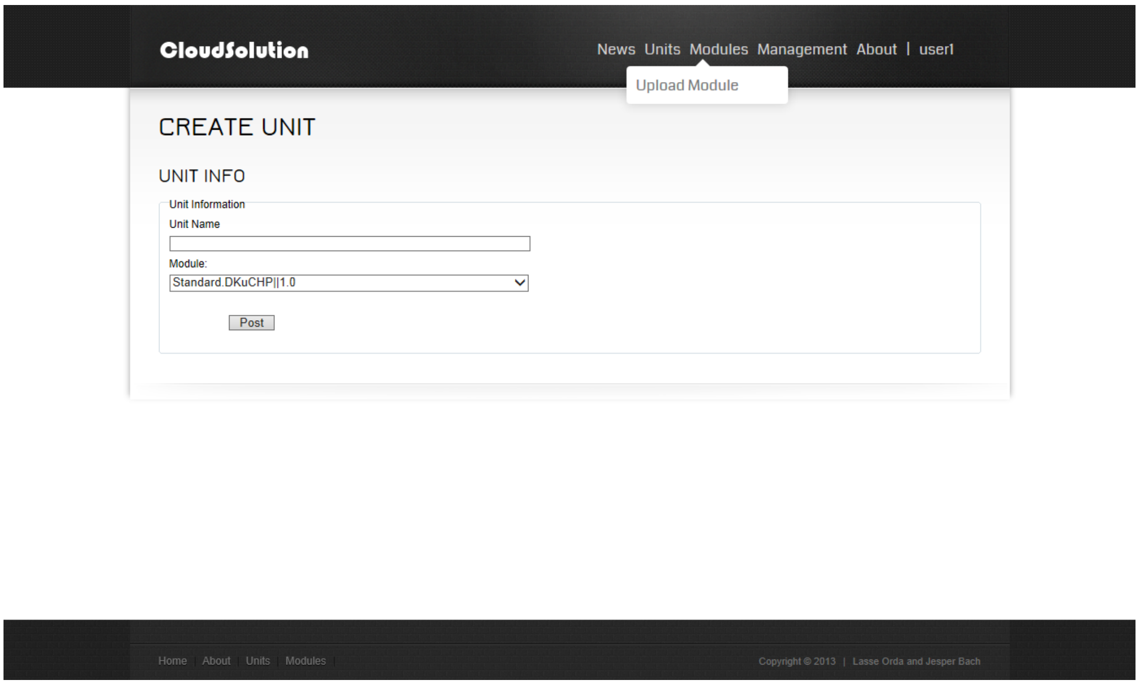
Unit Information

Unit Name:

Module: Standard.DKuCHP||1.0

Post
Home | About | Units | Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.13: CloudSolution - Create 3


CloudSolution
News Units Modules Management About | user1
Upload Module

CREATE UNIT

UNIT INFO

Unit Information

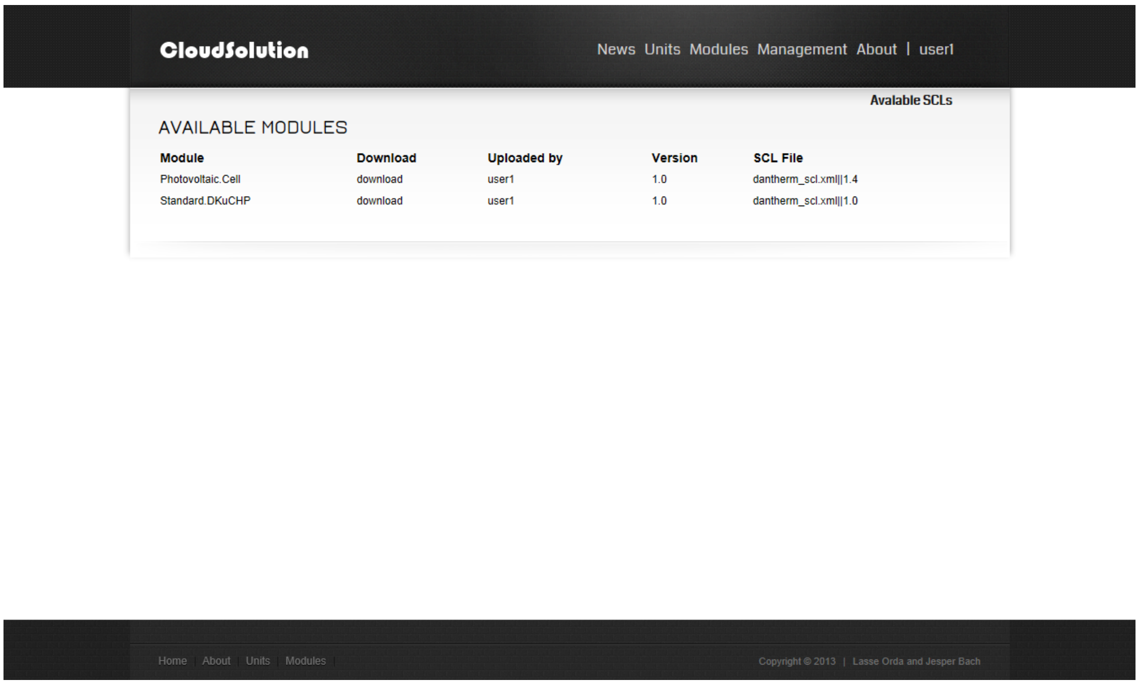
Unit Name

Module:

Standard.DKuCHP||1.0

Post
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.14: CloudSolution - Create 4

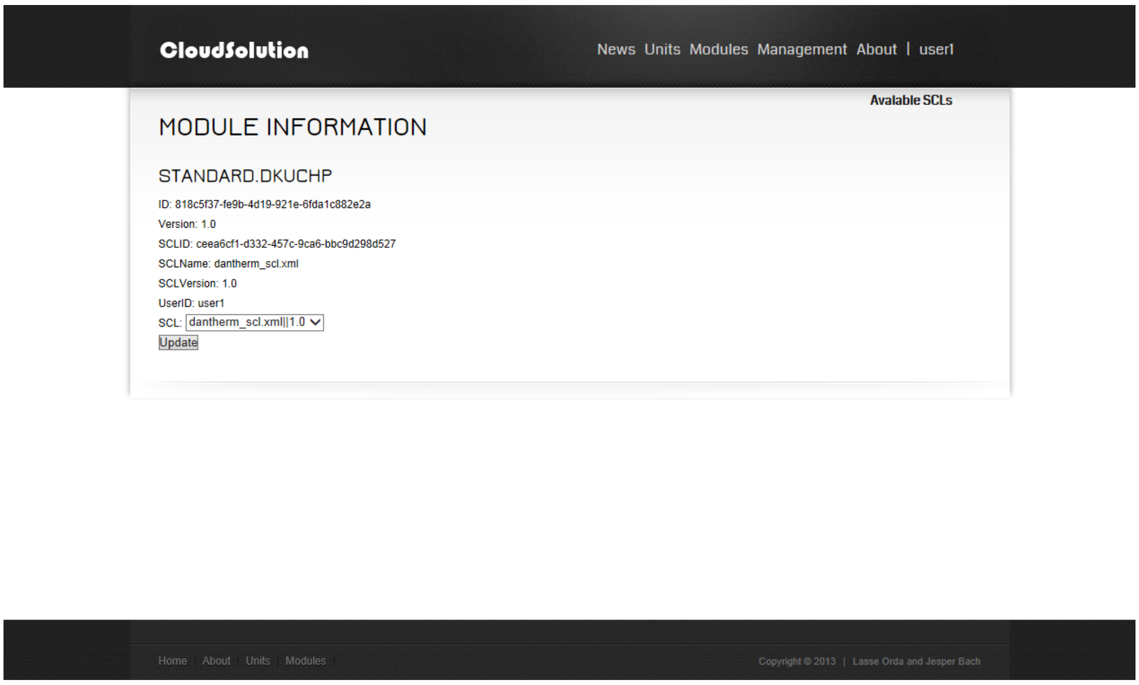

CloudSolution
News Units Modules Management About | user1
Avalable SCLs

AVAILABLE MODULES

Module	Download	Uploaded by	Version	SCL File
Photovoltaic.Cell	download	user1	1.0	dantherm_scl.xml||1.4
Standard.DKuCHP	download	user1	1.0	dantherm_scl.xml||1.0

Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.15: CloudSolution - Available Modules 1

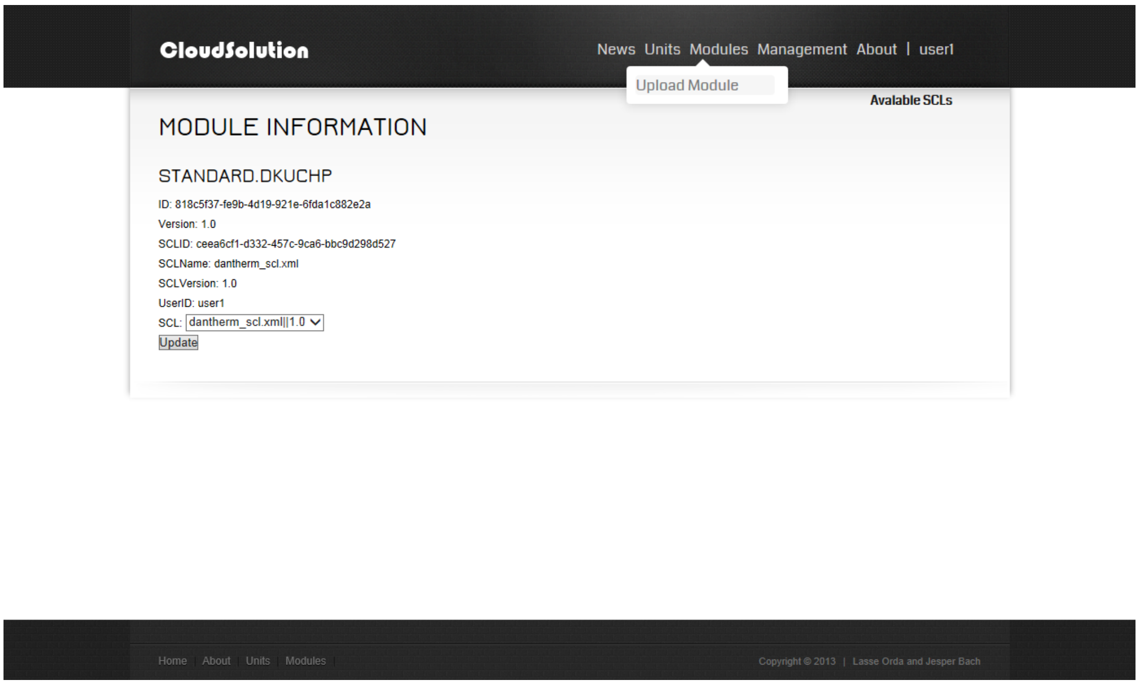

CloudSolution
News Units Modules Management About | user1
Avalable SCLs
Update
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

MODULE INFORMATION

STANDARD.DKUCHP

ID: 818c5f37-fe9b-4d19-921e-6fda1c882e2a
Version: 1.0
SCLID: ceea6cf1-d332-457c-9ca6-bbc9d298d527
SCLName: dantherm_scl.xml
SCLVersion: 1.0
UserID: user1
SCL: dantherm_scl.xml||1.0

Figure D.16: CloudSolution - Module Information 1

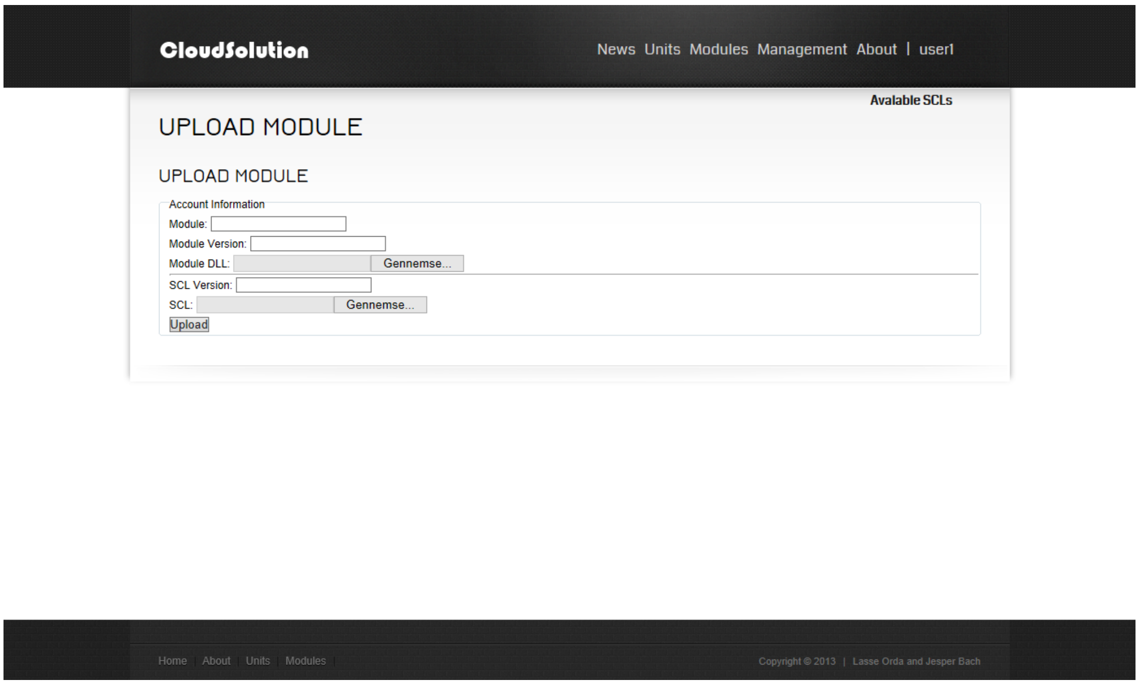

CloudSolution
News Units Modules Management About | user1
Upload Module
Avalable SCLs
Update
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

MODULE INFORMATION

STANDARD.DKUCHP

ID: 818c5f37-fe9b-4d19-921e-6fda1c882e2a
Version: 1.0
SCLID: ceea6cf1-d332-457c-9ca6-bbc9d298d527
SCLName: dantherm_scl.xml
SCLVersion: 1.0
UserID: user1
SCL: dantherm_scl.xml||1.0

Figure D.17: CloudSolution - Available Modules 2


CloudSolution
News Units Modules Management About | user1
Avalable SCLs

UPLOAD MODULE

UPLOAD MODULE

Account Information

Module:

Module Version:

Module DLL: Gennemse...

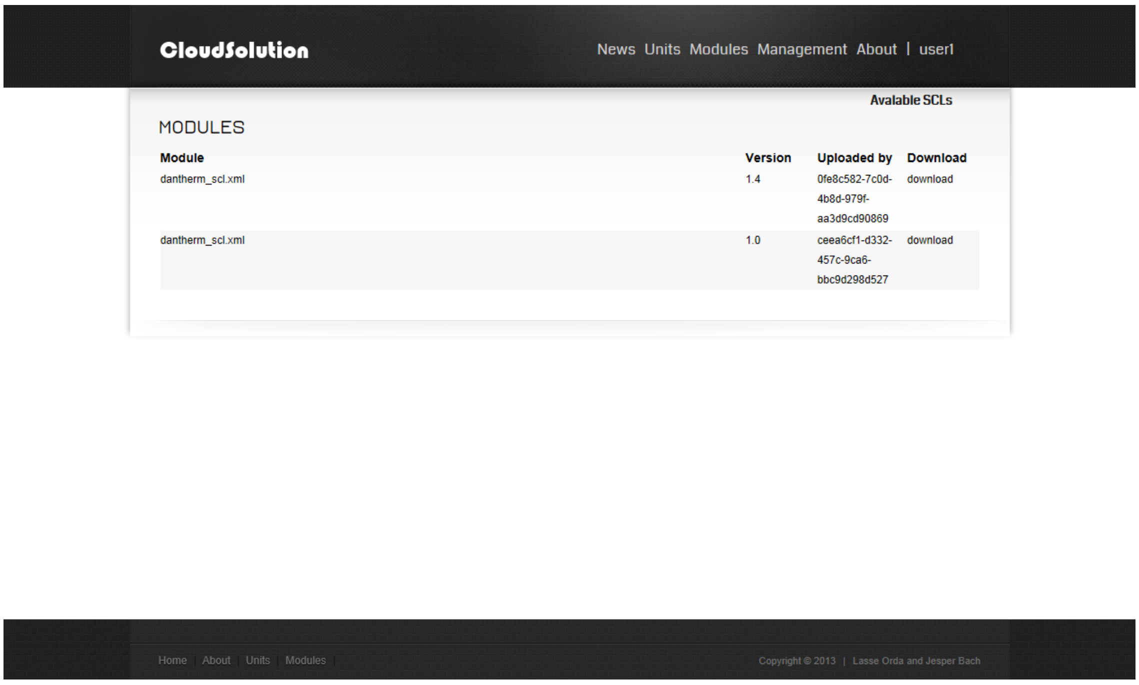
SCL Version:

SCL: Gennemse...

Upload

Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

Figure D.18: CloudSolution - Upload Module 1

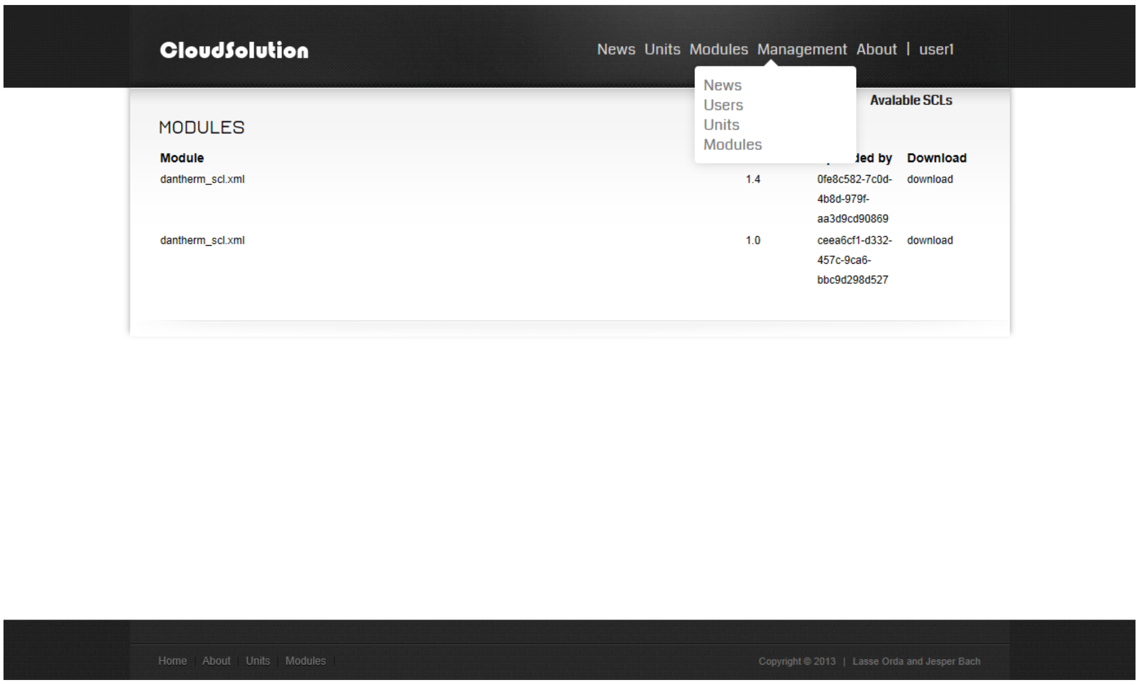

CloudSolution
News Units Modules Management About | user1
Avalable SCLs

MODULES

Module	Version	Uploaded by	Download
dantherm_scl.xml	1.4	0fe8c582-7c0d-4b8d-979f-aa3d9cd90869	download
dantherm_scl.xml	1.0	ceea6cf1-d332-457c-9ca6-bbc9d298d527	download

Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

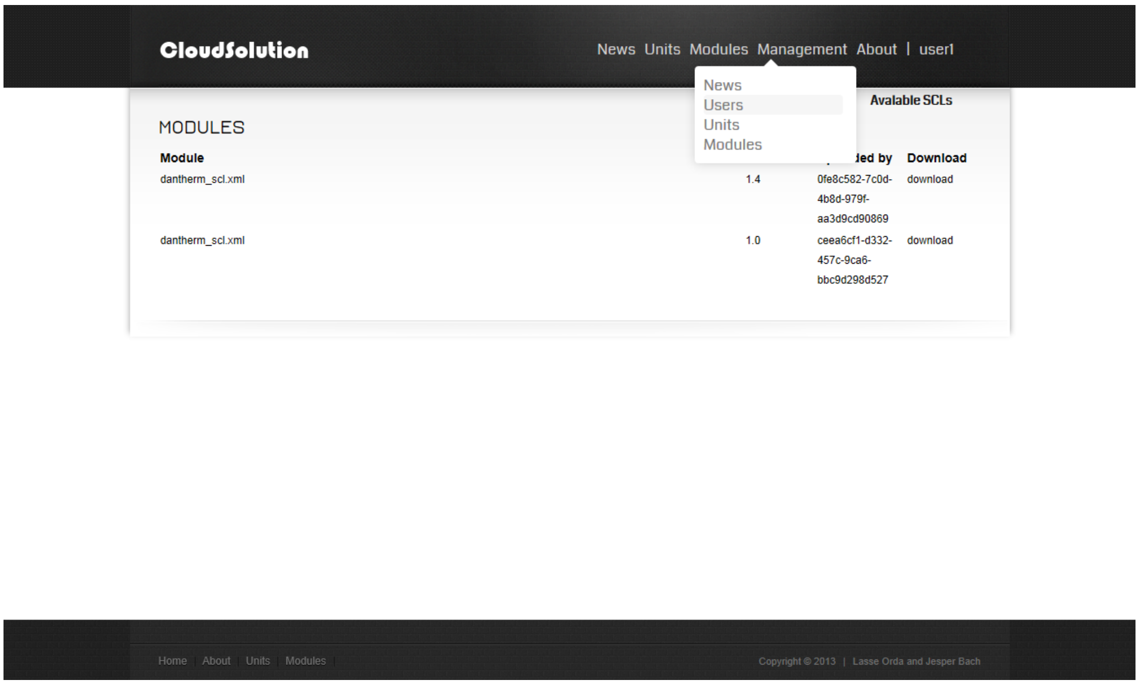
Figure D.19: CloudSolution - SCLs 1


CloudSolution
News Units Modules Management About | user1
News
Users
Units
Modules
Avalable SCLs
Home | About | Units | Modules |
Copyright © 2013 | Lasse Orda and Jesper Bach

MODULES

Module		ded by	Download
dantherm_scl.xml	1.4	0fe8c582-7c0d-4b8d-979f-aa3d9cd90869	download
dantherm_scl.xml	1.0	ceea6cf1-d332-457c-9ca6-bbc9d298d527	download

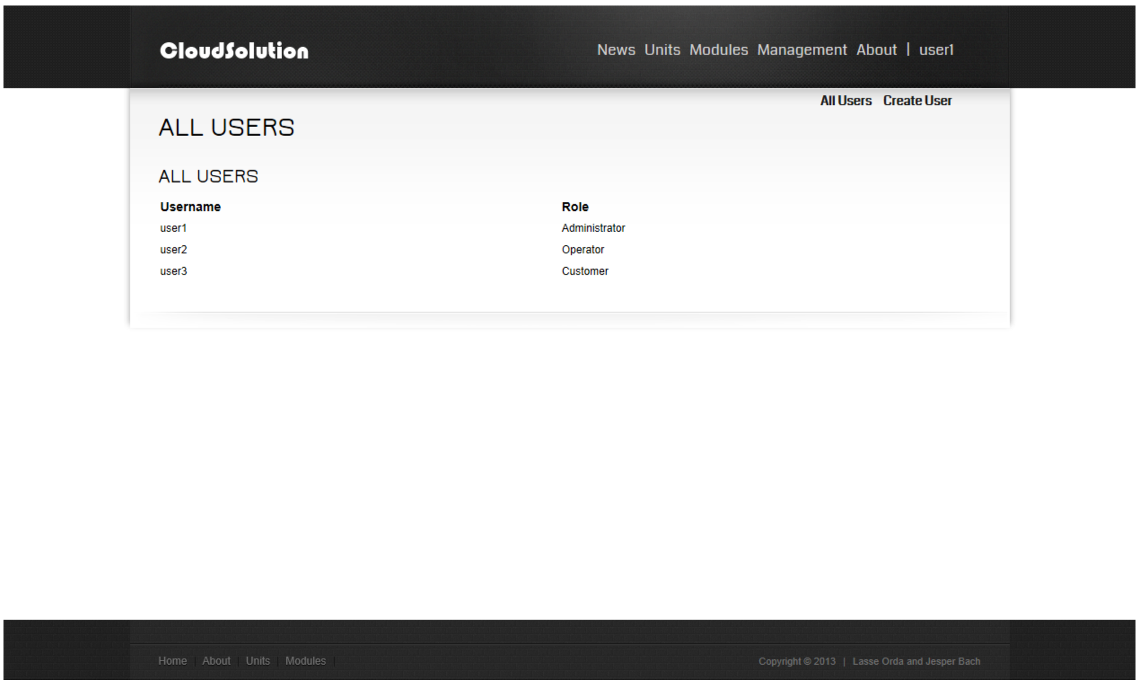
Figure D.20: CloudSolution - SCLs 2


CloudSolution
News Units Modules Management About | user1
News
Users
Units
Modules
Avalable SCLs
Home | About | Units | Modules |
Copyright © 2013 | Lasse Orda and Jesper Bach

MODULES

Module		ded by	Download
dantherm_scl.xml	1.4	0fe8c582-7c0d-4b8d-979f-aa3d9cd90869	download
dantherm_scl.xml	1.0	ceea6cf1-d332-457c-9ca6-bbc9d298d527	download

Figure D.21: CloudSolution - SCLs 3

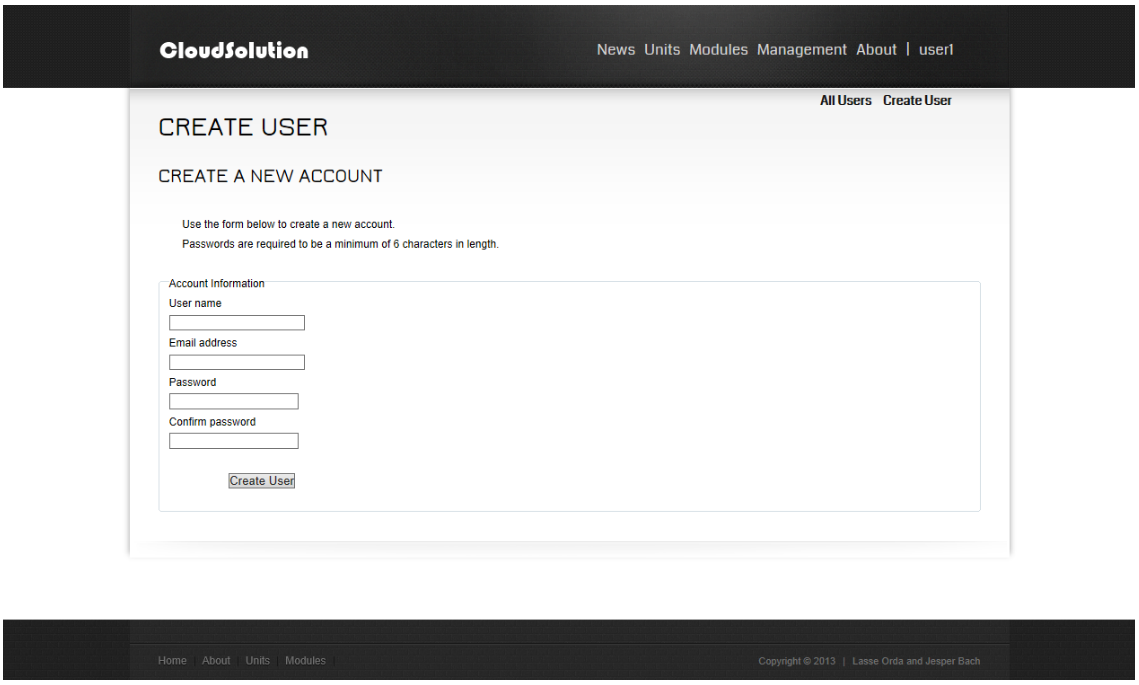

CloudSolution
News Units Modules Management About | user1
All Users Create User
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

ALL USERS

ALL USERS

Username	Role
user1	Administrator
user2	Operator
user3	Customer

Figure D.22: CloudSolution - All Users 1


CloudSolution
News Units Modules Management About | user1
All Users Create User
Create User
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

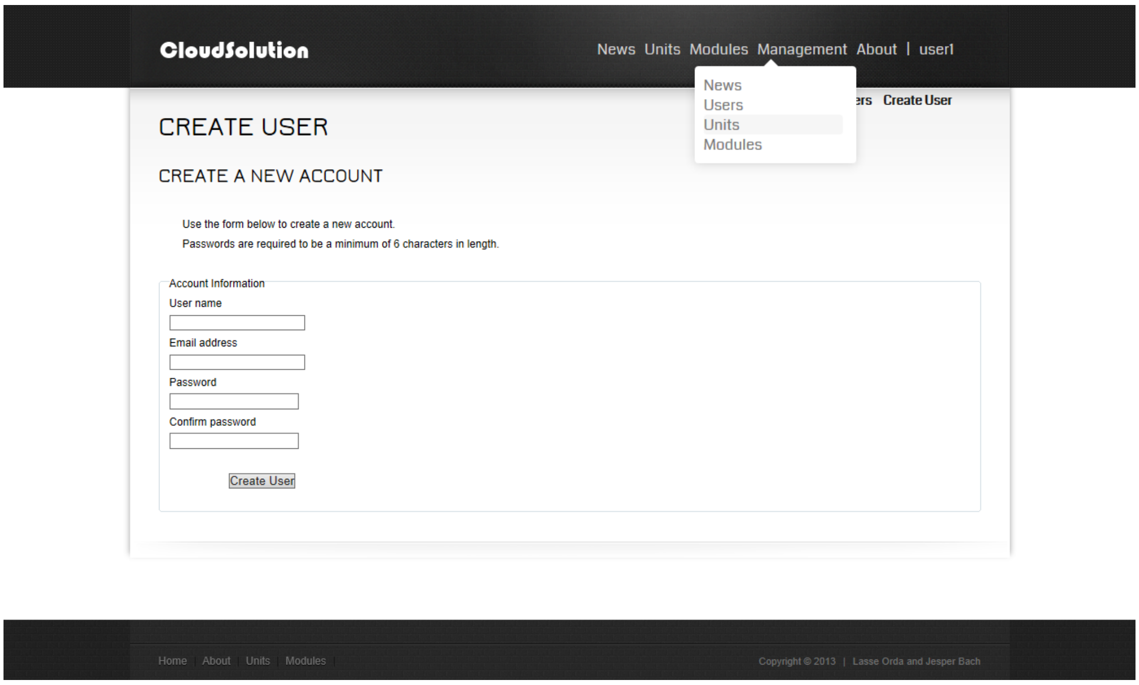
CREATE USER

CREATE A NEW ACCOUNT

Use the form below to create a new account.
Passwords are required to be a minimum of 6 characters in length.

Account Information
User name
Email address
Password
Confirm password

Figure D.23: CloudSolution - Create User 1


CloudSolution
News Units Modules Management About | user1
News
Users
Units
Modules
ers Create User
Create User
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

CREATE USER

CREATE A NEW ACCOUNT

Use the form below to create a new account.
Passwords are required to be a minimum of 6 characters in length.

Account Information

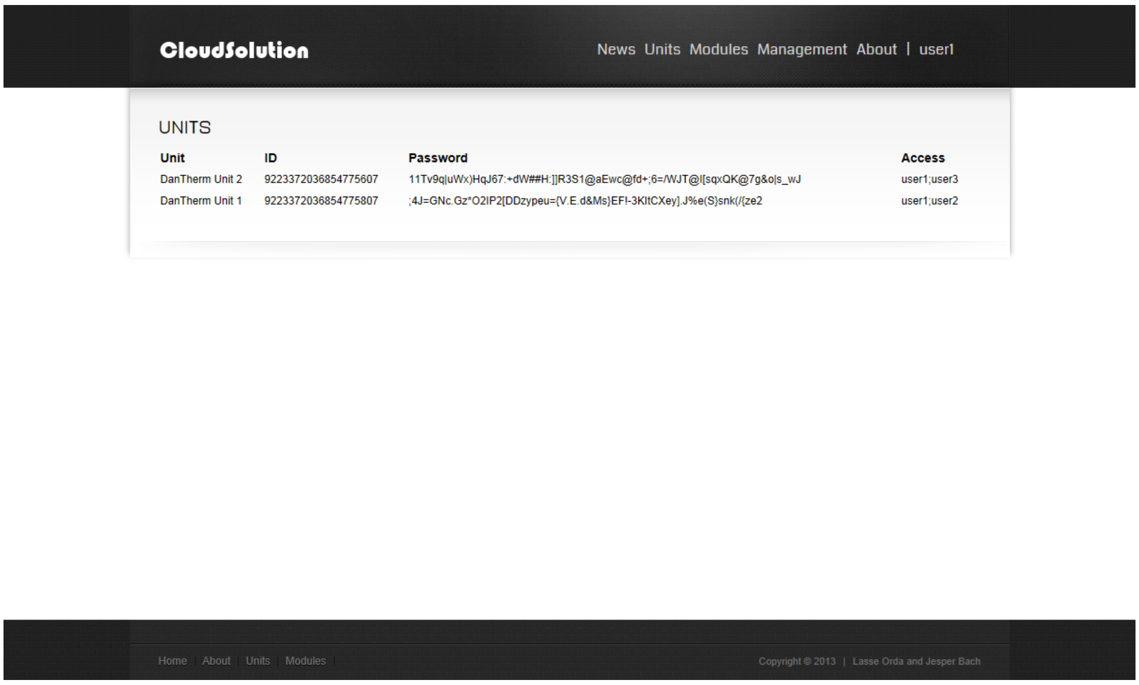
User name

Email address

Password

Confirm password

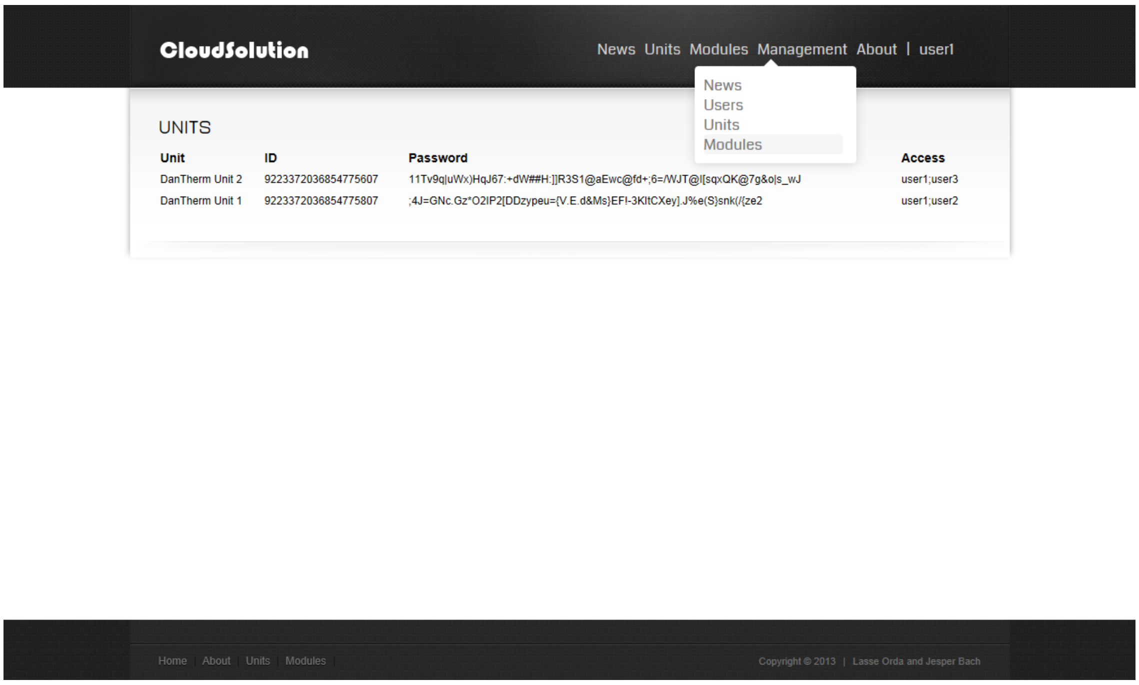
Figure D.24: CloudSolution - Create User 2


CloudSolution
News Units Modules Management About | user1
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

UNITS

Unit	ID	Password	Access
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EF!-3KltCXey].J%e(S}snk(/{ze2	user1;user2

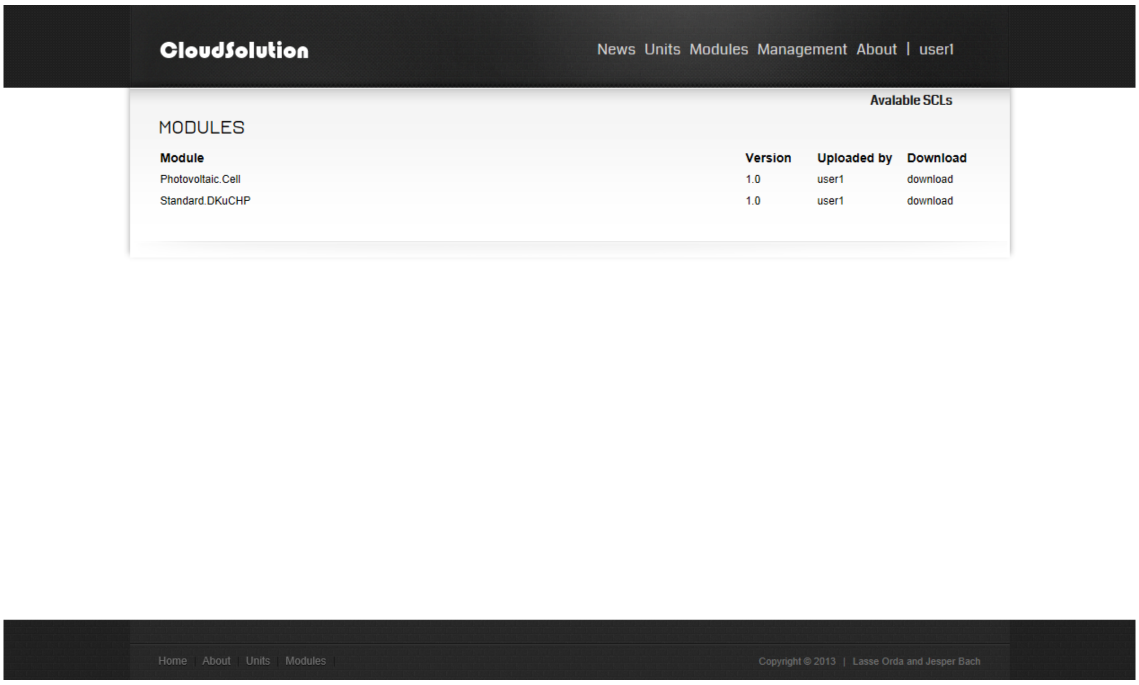
Figure D.25: CloudSolution - Units 1


CloudSolution
News Units Modules Management About | user1
News
Users
Units
Modules
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

UNITS

Unit	ID	Password	Access
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EFI-3KltCXey].J%e(S}snk(/{ze2	user1;user2

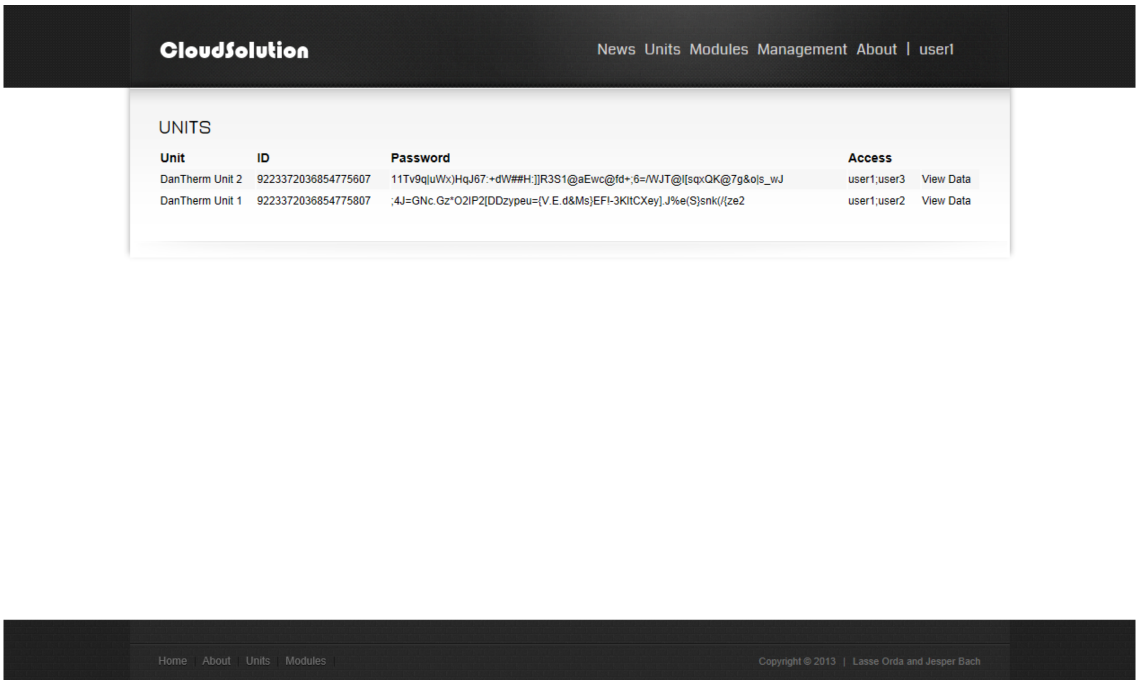
Figure D.26: CloudSolution - Units 2


CloudSolution
News Units Modules Management About | user1
Avaliable SCLs
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

MODULES

Module	Version	Uploaded by	Download
Photovoltaic.Cell	1.0	user1	download
Standard.DKuCHP	1.0	user1	download

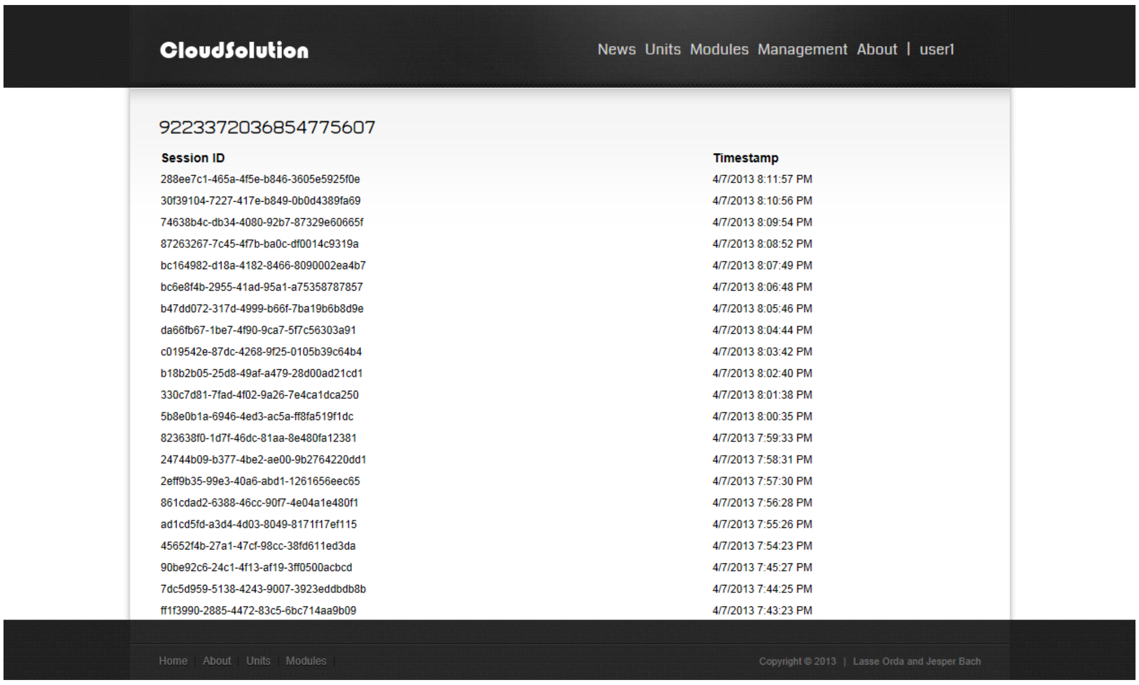
Figure D.27: CloudSolution - Modules 1


CloudSolution
News Units Modules Management About | user1
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

UNITS

Unit	ID	Password	Access	
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3	View Data
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EF!-3KltCXey].J%e(S}snk(/{ze2	user1;user2	View Data

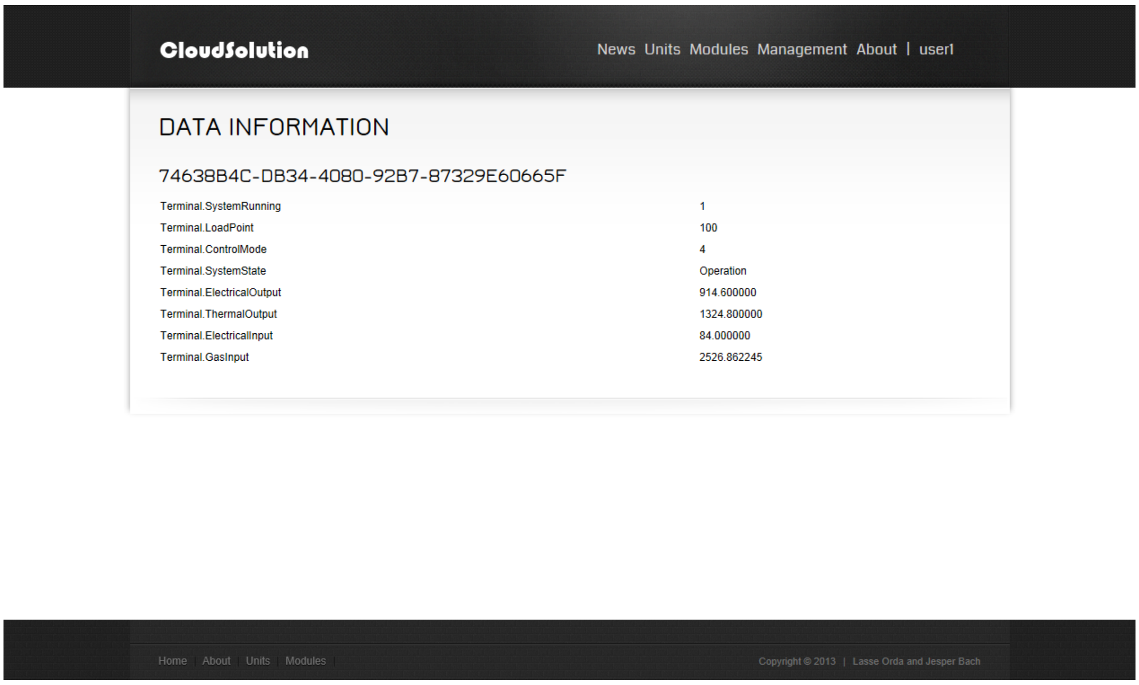
Figure D.28: CloudSolution - Units 3


CloudSolution
News Units Modules Management About | user1
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

9223372036854775607

Session ID	Timestamp
288ee7c1-465a-4f5e-b846-3605e5925f0e	4/7/2013 8:11:57 PM
30f39104-7227-417e-b849-0b0d4389fa69	4/7/2013 8:10:56 PM
74638b4c-db34-4080-92b7-87329e60665f	4/7/2013 8:09:54 PM
87263267-7c45-4f7b-ba0c-df0014c9319a	4/7/2013 8:08:52 PM
bc164982-d18a-4182-8466-8090002ea4b7	4/7/2013 8:07:49 PM
bc6e8f4b-2955-41ad-95a1-a75358787857	4/7/2013 8:06:48 PM
b47dd072-317d-4999-b66f-7ba19b6b8d9e	4/7/2013 8:05:46 PM
da66fb67-1be7-4f90-9ca7-5f7c56303a91	4/7/2013 8:04:44 PM
c019542e-87dc-4268-9f25-0105b39c64b4	4/7/2013 8:03:42 PM
b18b2b05-25d8-49af-a479-28d00ad21cd1	4/7/2013 8:02:40 PM
330c7d81-7fad-4f02-9a26-7e4ca1dca250	4/7/2013 8:01:38 PM
5b8e0b1a-6946-4ed3-ac5a-ff8fa519f1dc	4/7/2013 8:00:35 PM
823638f0-1d7f-46dc-81aa-8e480fa12381	4/7/2013 7:59:33 PM
24744b09-b377-4be2-ae00-9b2764220dd1	4/7/2013 7:58:31 PM
2eff9b35-99e3-40a6-abd1-1261656eec65	4/7/2013 7:57:30 PM
861cdad2-6388-46cc-90f7-4e04a1e480f1	4/7/2013 7:56:28 PM
ad1cd5fd-a3d4-4d03-8049-8171f17ef115	4/7/2013 7:55:26 PM
45652f4b-27a1-47cf-98cc-38fd611ed3da	4/7/2013 7:54:23 PM
90be92c6-24c1-4f13-af19-3ff0500acbcd	4/7/2013 7:45:27 PM
7dc5d959-5138-4243-9007-3923eddbdb8b	4/7/2013 7:44:25 PM
ff1f3990-2885-4472-83c5-6bc714aa9b09	4/7/2013 7:43:23 PM

Figure D.29: CloudSolution - Units 4


CloudSolution
News Units Modules Management About | user1
Home About Units Modules
Copyright © 2013 | Lasse Orda and Jesper Bach

DATA INFORMATION

74638B4C-DB34-4080-92B7-87329E60665F

Terminal.SystemRunning	1
Terminal.LoadPoint	100
Terminal.ControlMode	4
Terminal.SystemState	Operation
Terminal.ElectricalOutput	914.600000
Terminal.ThermalOutput	1324.800000
Terminal.ElectricalInput	84.000000
Terminal.GasInput	2526.862245

Figure D.30: CloudSolution - Units 5

D.2 Web API

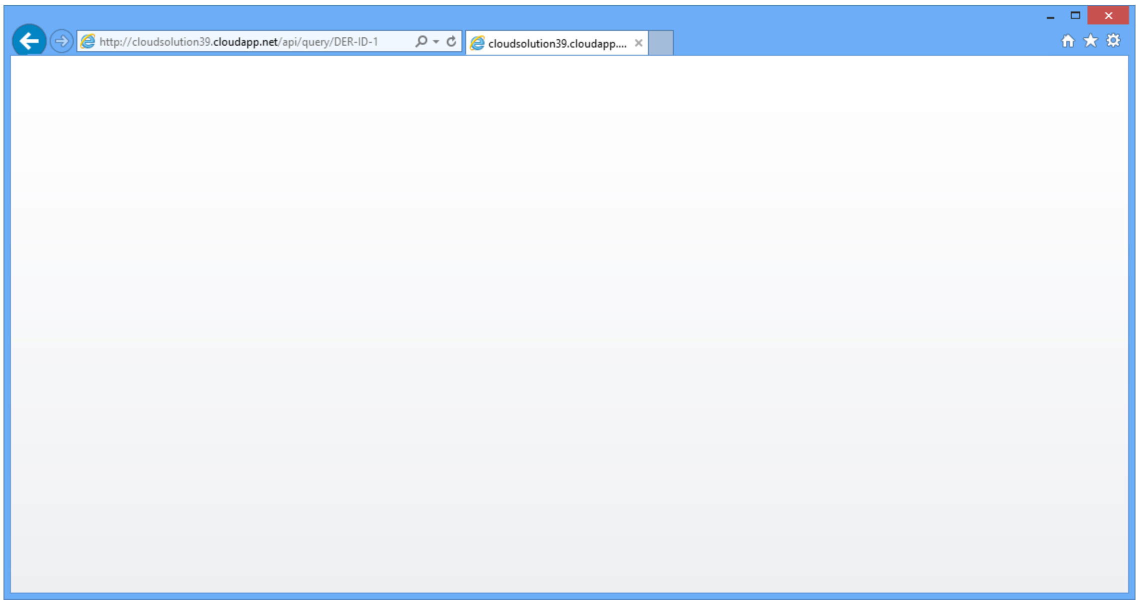

http://cloudsolution39.cloudapp.net/api/query/DER-ID-1
cloudsolution39.cloudapp....

Figure D.31: CloudSolution - Empty

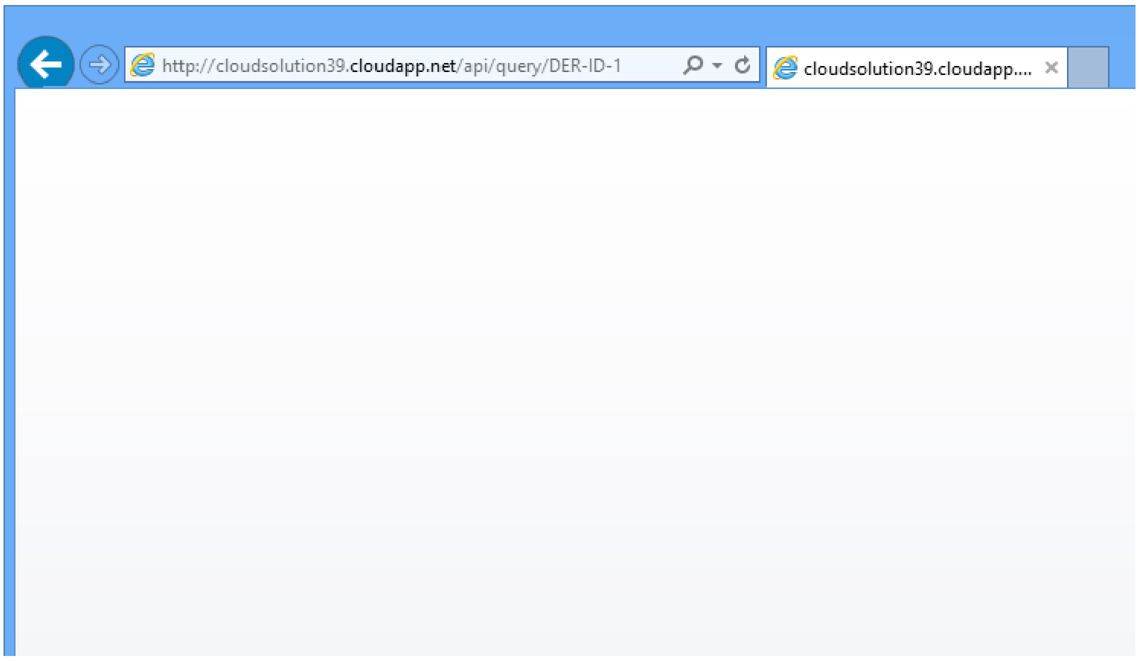
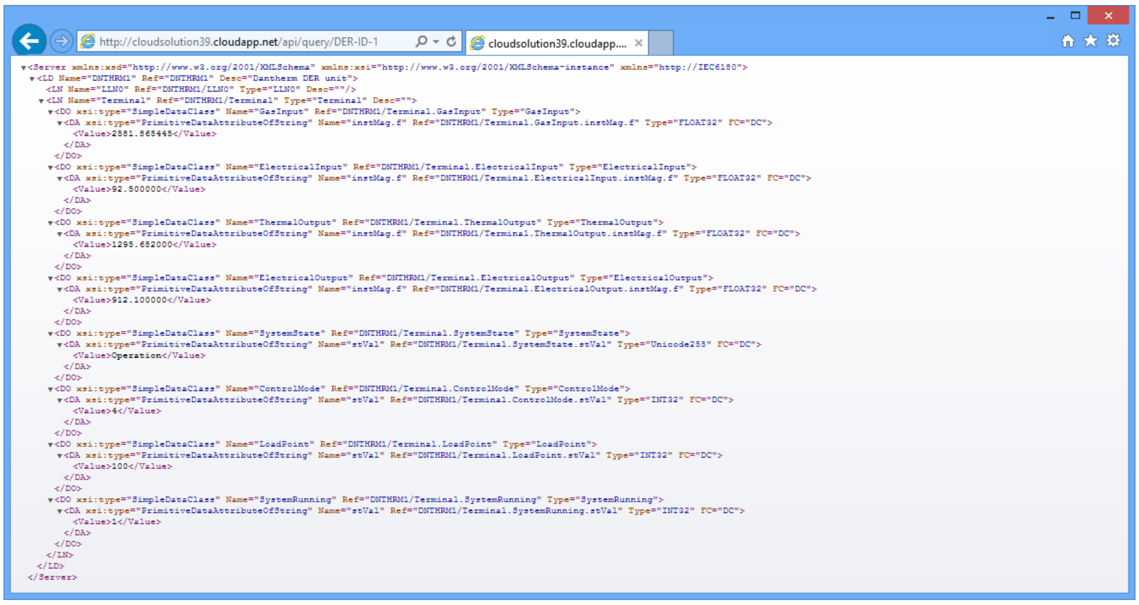

http://cloudsolution39.cloudapp.net/api/query/DER-ID-1
cloudsolution39.cloudapp....

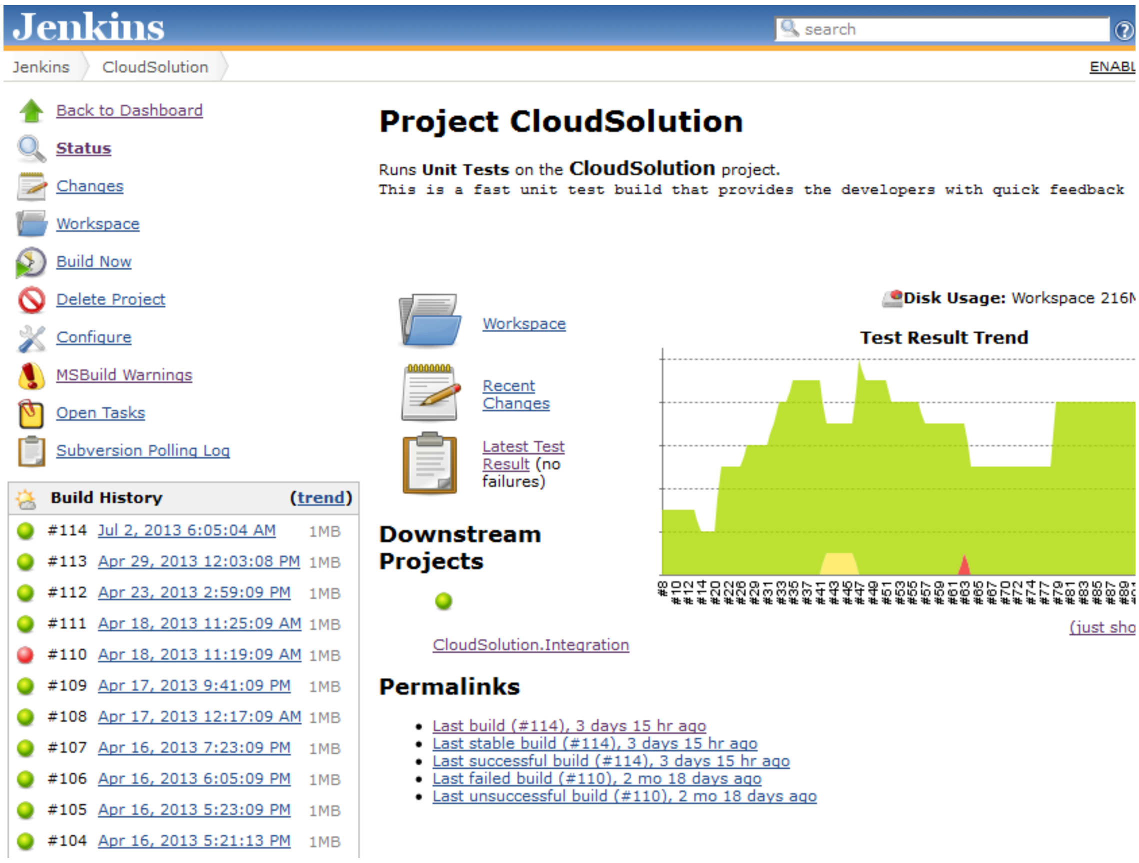
Figure D.32: CloudSolution - Small


http://cloudsolution39.cloudapp.net/api/query/DER-ID-1
cloudsolution39.cloudapp....

```
▼<Server xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://IEC6180">
 ▼<LD Name="DNTHRM1" Ref="DNTHRM1" Desc="Dantherm DER unit">
    <LN Name="LLN0" Ref="DNTHRM1/LLN0" Type="LLN0" Desc=""/>
  ▼<LN Name="Terminal" Ref="DNTHRM1/Terminal" Type="Terminal" Desc="">
   ▼<DO xsi:type="SimpleDataClass" Name="GasInput" Ref="DNTHRM1/Terminal.GasInput" Type="GasInput">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.GasInput.instMag.f" Type="FLOAT32" FC="DC">
       <Value>2581.865445</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="ElectricalInput" Ref="DNTHRM1/Terminal.ElectricalInput" Type="ElectricalInput">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalInput.instMag.f" Type="FLOAT32" FC="DC">
       <Value>92.500000</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="ThermalOutput" Ref="DNTHRM1/Terminal.ThermalOutput" Type="ThermalOutput">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ThermalOutput.instMag.f" Type="FLOAT32" FC="DC">
       <Value>1295.682000</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="ElectricalOutput" Ref="DNTHRM1/Terminal.ElectricalOutput" Type="ElectricalOutput">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalOutput.instMag.f" Type="FLOAT32" FC="DC">
       <Value>912.100000</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="SystemState" Ref="DNTHRM1/Terminal.SystemState" Type="SystemState">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.SystemState.stVal" Type="Unicode255" FC="DC">
       <Value>Operation</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="ControlMode" Ref="DNTHRM1/Terminal.ControlMode" Type="ControlMode">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.ControlMode.stVal" Type="INT32" FC="DC">
       <Value>4</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="LoadPoint" Ref="DNTHRM1/Terminal.LoadPoint" Type="LoadPoint">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.LoadPoint.stVal" Type="INT32" FC="DC">
       <Value>100</Value>
      </DA>
    </DO>
   ▼<DO xsi:type="SimpleDataClass" Name="SystemRunning" Ref="DNTHRM1/Terminal.SystemRunning" Type="SystemRunning">
    ▼<DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.SystemRunning.stVal" Type="INT32" FC="DC">
       <Value>1</Value>
      </DA>
    </DO>
   </LN>
  </LD>
</Server>
```

Figure D.33: CloudSolution - Full

D.3 Continuous Integration and Generated Reports



Project CloudSolution

Runs **Unit Tests** on the **CloudSolution** project.

```
This is a fast unit test build that provides the developers with quick feedback
```

Workspace

Recent Changes

Latest Test Result (no failures)

Disk Usage: Workspace 216M

Downstream Projects

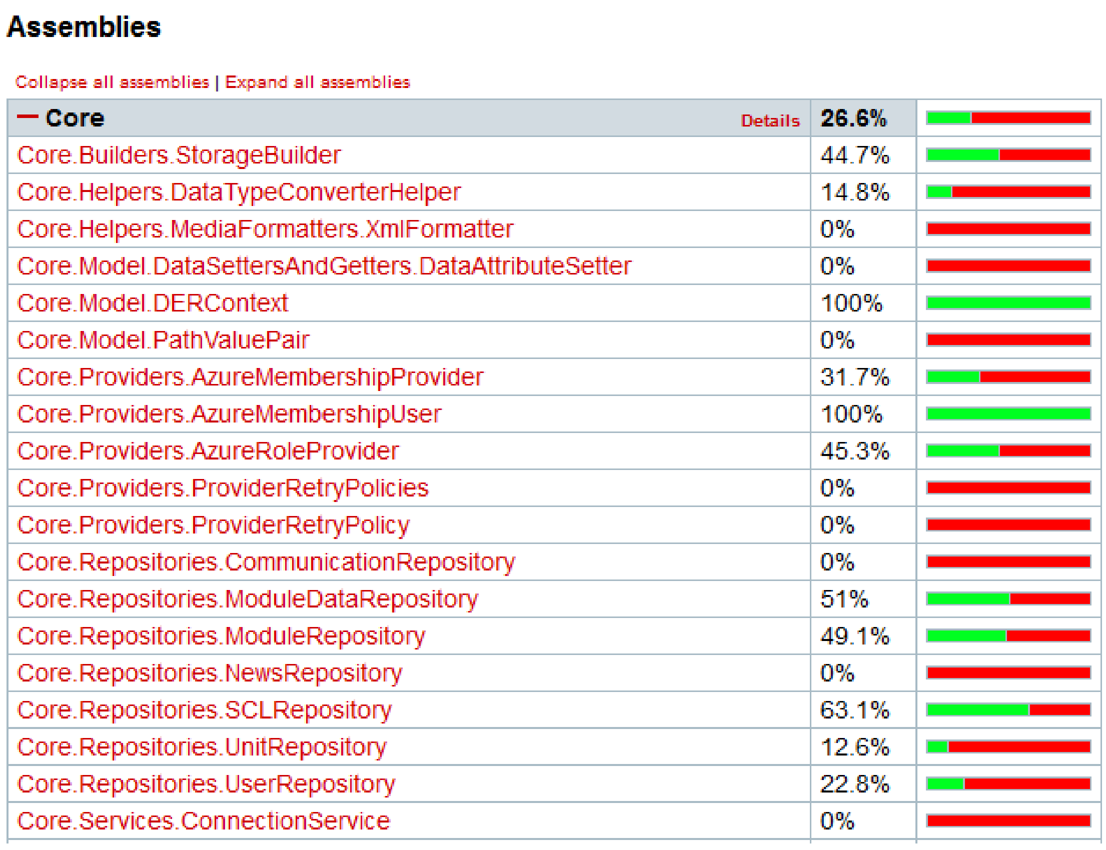
CloudSolution.Integration

Permalinks

- Last build (#114), 3 days 15 hr ago
- Last stable build (#114), 3 days 15 hr ago
- Last successful build (#114), 3 days 15 hr ago
- Last failed build (#110), 2 mo 18 days ago
- Last unsuccessful build (#110), 2 mo 18 days ago

Build History	(trend)	
#114	Jul 2, 2013 6:05:04 AM	1MB
#113	Apr 29, 2013 12:03:08 PM	1MB
#112	Apr 23, 2013 2:59:09 PM	1MB
#111	Apr 18, 2013 11:25:09 AM	1MB
#110	Apr 18, 2013 11:19:09 AM	1MB
#109	Apr 17, 2013 9:41:09 PM	1MB
#108	Apr 17, 2013 12:17:09 AM	1MB
#107	Apr 16, 2013 7:23:09 PM	1MB
#106	Apr 16, 2013 6:05:09 PM	1MB
#105	Apr 16, 2013 5:23:09 PM	1MB
#104	Apr 16, 2013 5:21:13 PM	1MB

Figure D.34: Continuous Integration Server: Jenkins

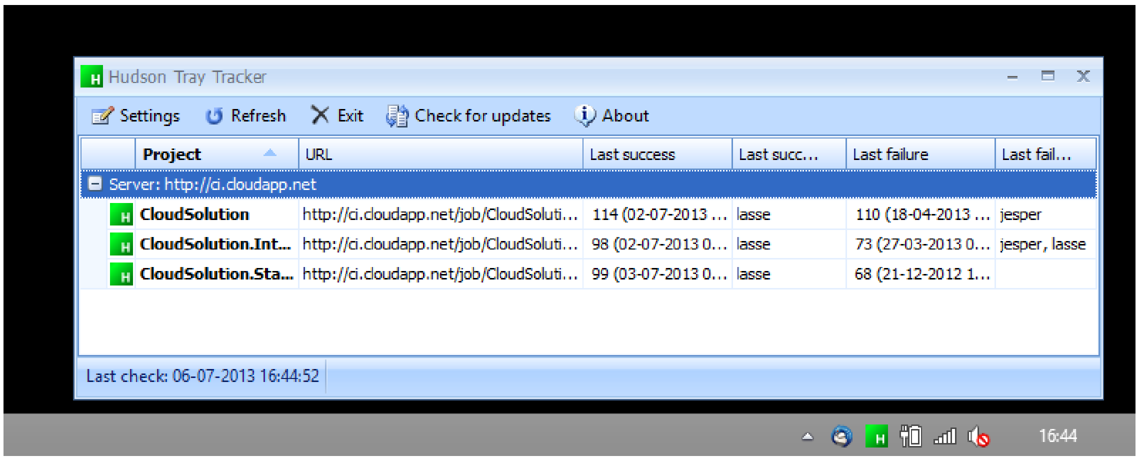


Assemblies

Collapse all assemblies | Expand all assemblies

— Core Details	26.6%
Core.Builders.StorageBuilder	44.7%
Core.Helpers.DataTypeConverterHelper	14.8%
Core.Helpers.MediaFormatters.XmlFormatter	0%
Core.Model.DataSettersAndGetters.DataAttributeSetter	0%
Core.Model.DERContext	100%
Core.Model.PathValuePair	0%
Core.Providers.AzureMembershipProvider	31.7%
Core.Providers.AzureMembershipUser	100%
Core.Providers.AzureRoleProvider	45.3%
Core.Providers.ProviderRetryPolicies	0%
Core.Providers.ProviderRetryPolicy	0%
Core.Repositories.CommunicationRepository	0%
Core.Repositories.ModuleDataRepository	51%
Core.Repositories.ModuleRepository	49.1%
Core.Repositories.NewsRepository	0%
Core.Repositories.SCLRepository	63.1%
Core.Repositories.UnitRepository	12.6%
Core.Repositories.UserRepository	22.8%
Core.Services.ConnectionService	0%

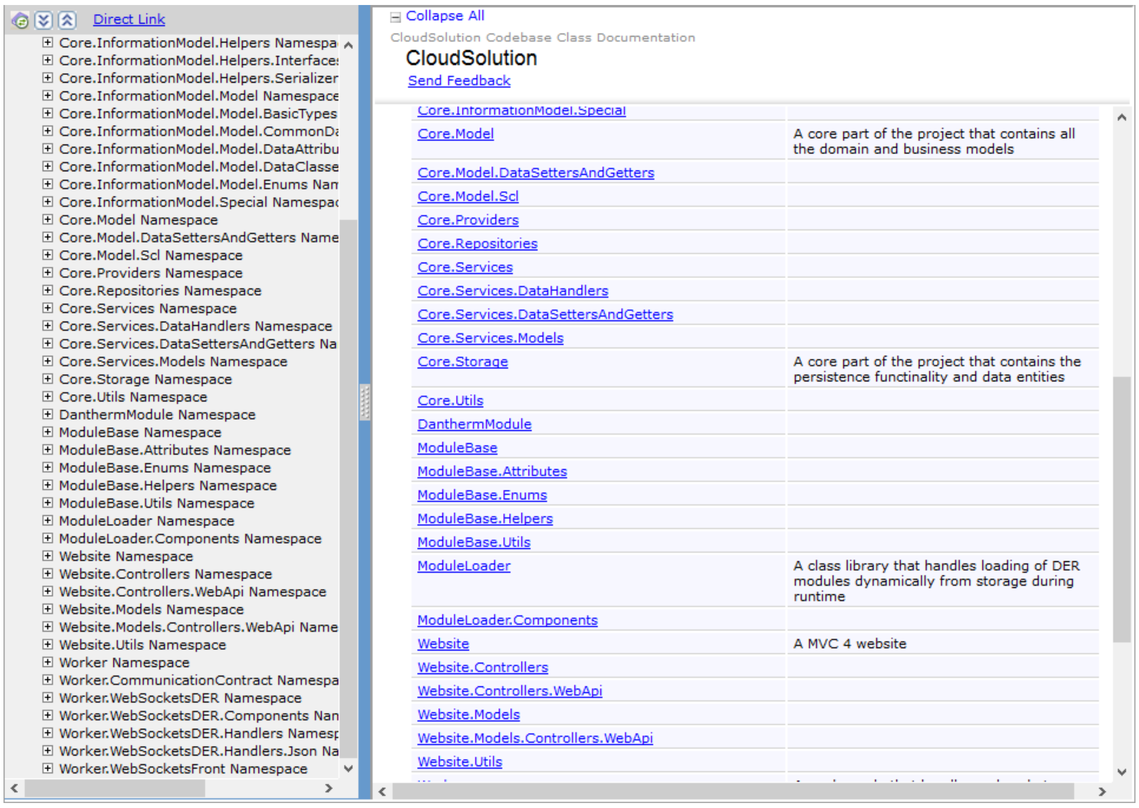
Figure D.35: Code Coverage Report



Project	URL	Last success	Last succ...	Last failure	Last fail...
Server: http://ci.cloudapp.net					
CloudSolution	http://ci.cloudapp.net/job/CloudSoluti...	114 (02-07-2013 ...	lasse	110 (18-04-2013 ...	jesper
CloudSolution.Int...	http://ci.cloudapp.net/job/CloudSoluti...	98 (02-07-2013 0...	lasse	73 (27-03-2013 0...	jesper, lasse
CloudSolution.Sta...	http://ci.cloudapp.net/job/CloudSoluti...	99 (03-07-2013 0...	lasse	68 (21-12-2012 1...	

Last check: 06-07-2013 16:44:52

Figure D.36: Tray Tracker for Jenkins CI Server


Direct Link

- Core.InformationModel.Helpers Namespa
- Core.InformationModel.Helpers.Interface:
- Core.InformationModel.Helpers.Serializer
- Core.InformationModel.Model Namespace
- Core.InformationModel.Model.BasicTypes
- Core.InformationModel.Model.CommonDa
- Core.InformationModel.Model.DataAttribu
- Core.InformationModel.Model.DataClasse
- Core.InformationModel.Model.Enums Nan
- Core.InformationModel.Special Namespac
- Core.Model Namespace
- Core.Model.DataSettersAndGetters Name
- Core.Model.Scl Namespace
- Core.Providers Namespace
- Core.Repositories Namespace
- Core.Services Namespace
- Core.Services.DataHandlers Namespace
- Core.Services.DataSettersAndGetters Na
- Core.Services.Models Namespace
- Core.Storage Namespace
- Core.Utils Namespace
- DanthermModule Namespace
- ModuleBase Namespace
- ModuleBase.Attributes Namespace
- ModuleBase.Enums Namespace
- ModuleBase.Helpers Namespace
- ModuleBase.Utils Namespace
- ModuleLoader Namespace
- ModuleLoader.Components Namespace
- Website Namespace
- Website.Controllers Namespace
- Website.Controllers.WebApi Namespace
- Website.Models Namespace
- Website.Models.Controllers.WebApi Name
- Website.Utils Namespace
- Worker Namespace
- Worker.CommunicationContract Namespa
- Worker.WebSocketsDER Namespace
- Worker.WebSocketsDER.Components Nan
- Worker.WebSocketsDER.Handlers Namesp
- Worker.WebSocketsDER.Handlers.Json Na
- Worker.WebSocketsFront Namespace

Collapse All

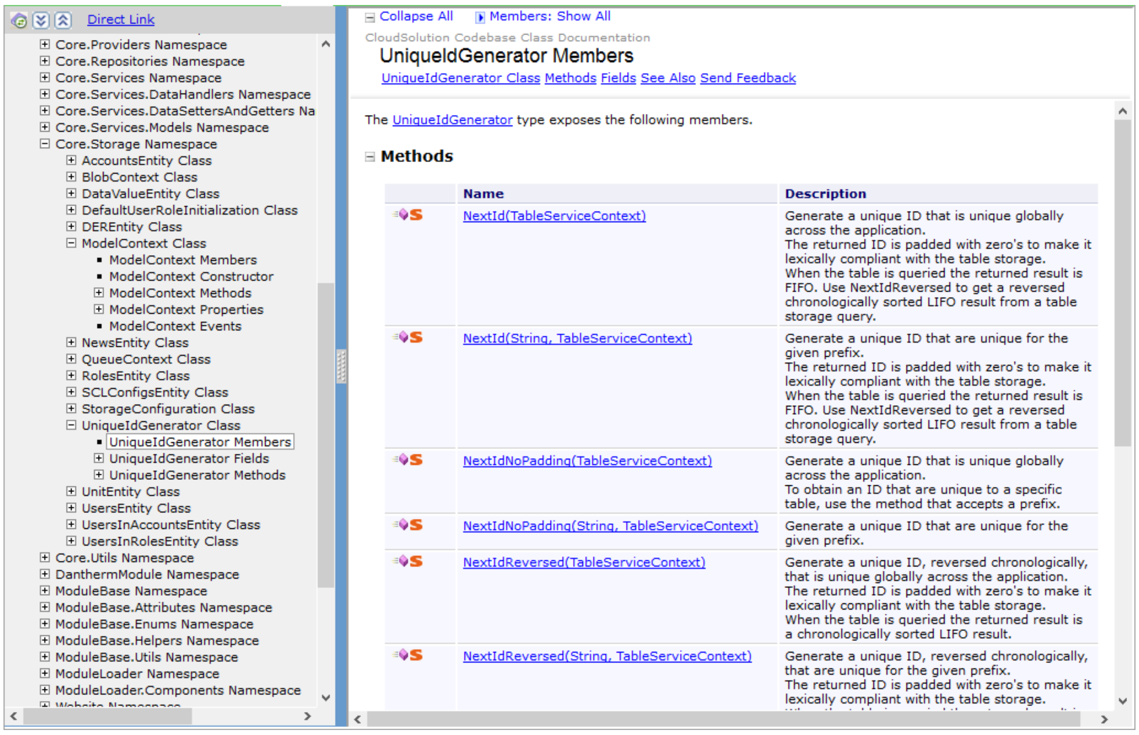
CloudSolution Codebase Class Documentation

CloudSolution

Send Feedback

Core.InformationModel.Special	
Core.Model	A core part of the project that contains all the domain and business models
Core.Model.DataSettersAndGetters	
Core.Model.Scl	
Core.Providers	
Core.Repositories	
Core.Services	
Core.Services.DataHandlers	
Core.Services.DataSettersAndGetters	
Core.Services.Models	
Core.Storage	A core part of the project that contains the persistence functinality and data entities
Core.Utils	
DanthermModule	
ModuleBase	
ModuleBase.Attributes	
ModuleBase.Enums	
ModuleBase.Helpers	
ModuleBase.Utils	
ModuleLoader	A class library that handles loading of DER modules dynamically from storage during runtime
ModuleLoader.Components	
Website	A MVC 4 website
Website.Controllers	
Website.Controllers.WebApi	
Website.Models	
Website.Models.Controllers.WebApi	
Website.Utils	

Figure D.37: Code API Docomentation


Direct Link

- Core.Providers Namespace
- Core.Repositories Namespace
- Core.Services Namespace
- Core.Services.DataHandlers Namespace
- Core.Services.DataSettersAndGetters Na
- Core.Services.Models Namespace
- Core.Storage Namespace
  - AccountsEntity Class
  - BlobContext Class
  - DataValueEntity Class
  - DefaultUserRoleInitialization Class
  - DEREntity Class
  - ModelContext Class
    - ModelContext Members
    - ModelContext Constructor
    - ModelContext Methods
    - ModelContext Properties
    - ModelContext Events
  - NewsEntity Class
  - QueueContext Class
  - RolesEntity Class
  - SCLConfigsEntity Class
  - StorageConfiguration Class
  - UniqueIdGenerator Class
    - UniqueIdGenerator Members
    - UniqueIdGenerator Fields
    - UniqueIdGenerator Methods
  - UnitEntity Class
  - UsersEntity Class
  - UsersInAccountsEntity Class
  - UsersInRolesEntity Class
- Core.Utils Namespace
- DanthermModule Namespace
- ModuleBase Namespace
- ModuleBase.Attributes Namespace
- ModuleBase.Enums Namespace
- ModuleBase.Helpers Namespace
- ModuleBase.Utils Namespace
- ModuleLoader Namespace
- ModuleLoader.Components Namespace

Collapse All   Members: Show All

CloudSolution Codebase Class Documentation

UniqueIdGenerator Members

UniqueIdGenerator Class Methods Fields See Also Send Feedback

The UniqueIdGenerator type exposes the following members.

Methods

	Name	Description
S	NextId(TableServiceContext)	Generate a unique ID that is unique globally across the application. The returned ID is padded with zero's to make it lexically compliant with the table storage. When the table is queried the returned result is FIFO. Use NextIdReversed to get a reversed chronologically sorted LIFO result from a table storage query.
S	NextId(String, TableServiceContext)	Generate a unique ID that are unique for the given prefix. The returned ID is padded with zero's to make it lexically compliant with the table storage. When the table is queried the returned result is FIFO. Use NextIdReversed to get a reversed chronologically sorted LIFO result from a table storage query.
S	NextIdNoPadding(TableServiceContext)	Generate a unique ID that is unique globally across the application. To obtain an ID that are unique to a specific table, use the method that accepts a prefix.
S	NextIdNoPadding(String, TableServiceContext)	Generate a unique ID that are unique for the given prefix.
S	NextIdReversed(TableServiceContext)	Generate a unique ID, reversed chronologically, that is unique globally across the application. The returned ID is padded with zero's to make it lexically compliant with the table storage. When the table is queried the returned result is a chronologically sorted LIFO result.
S	NextIdReversed(String, TableServiceContext)	Generate a unique ID, reversed chronologically, that are unique for the given prefix. The returned ID is padded with zero's to make it lexically compliant with the table storage.

Figure D.38: Code API Docomentation Opened

Application Instructions

The Application Instructions is a user guide and introduction document on how to use the Cloud Service Web Interface and RESTful WebApi Interface developed for this thesis. The Application Instructions was written as a part of the participation in the Microsoft ImagineCup competition, that the students of this thesis was a taking part in during the thesis period.

Using the WebPage

The primary use of the webpage is for users who have to either access data in a more visual manner or create new units.

The webpage can be found at the address:
http://cloudsolution39.cloudapp.net/

Three users have been created:

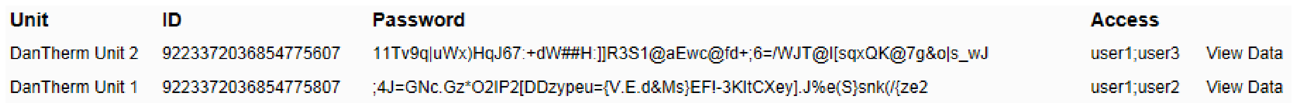
User	Password	Role
user1	password1	Administrator
user2	password2	Operator
user3	password3	User

And there is two distributed energy resources connected to the solution:

Unit Name	Unit Type	User Access
DanTherm Unit 1	Combined Head and Power	user1, user2
DanTherm Unit 2	Combined Head and Power	user1, user3

Units

Under 'Units' (http://cloudsolution39.cloudapp.net/Unit) can all distributed energy resources, that the current logged in user has access to, be viewed. As an overview of the units, the unit's name, is, password and access is shown.



Unit	ID	Password	Access	
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3	View Data
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EFI-3KItCXey].J%e(S}snk(/{ze2	user1;user2	View Data

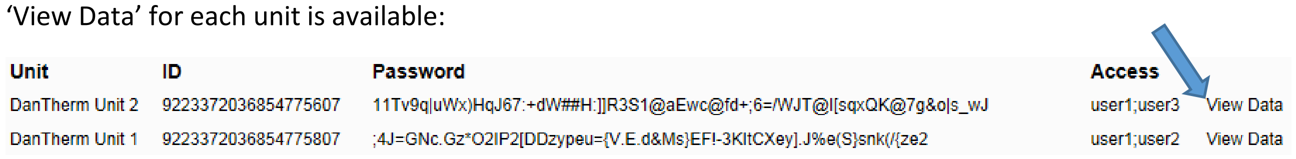
These data are needed for connecting a unit to the websocket available at:

Endpoint	Usage
cloudsolution39.cloudapp.net:10101	Configuration
cloudsolution39.cloudapp.net:10100	Sending data

Using the configuration endpoint, settings as the module or name can be set.
Using the data endport, the unit can sent raw data to the solution.

For a more detailed view and for editing data, each unit can be accessed by clicking on one of the rows. Here the unit name can be changed, as the unit name is not unit (only the ID is), the module can be changed (if a new version is to be used) or the users who can access the unit's information can be changed (the current user cannot be removed). Access to simple commands as 'Shutdown' or 'Turn on' is also visible. These commands uses the WebApi (see further in this text) to put setting into the unit. As a precaution, we have disabled this functionality because it is actually two running devices 'in the other end'.

The posted data from the unit can also be access through the 'Units' page. At the right side of the page, 'View Data' for each unit is available:



Unit	ID	Password	Access	
DanTherm Unit 2	9223372036854775607	11Tv9q|uWx)HqJ67:+dW##H:]]R3S1@aEwc@fd+;6=/WJT@l[sqxQK@7g&o|s_wJ	user1;user3	View Data
DanTherm Unit 1	9223372036854775807	;4J=GNc.Gz*O2IP2[DDzypeu={V.E.d&Ms}EF!-3KltCXey].J%e(S}snk(/{ze2	user1;user2	View Data

Each data set that arrives at the solution is saved in the table storage with the same session id. The 'View Data' page lists the latest 125 entities in a paged page. Accessing one of the session, the table storage is accessed and the data associated with the session is retrieved. As each unit can have different data points, the data are shown to the user, using the path in the IEC 61850 standard and the associated value.

Create Unit

Under the menu 'Units', a submenu for create new units can be found for users with 'Operator' and 'Administrator' roles.

Modules

The 'Modules' section provides an overview of each available module for the user. The user can *download* each module, access more information by clicking on the row or see all available SCL (Substation Configuration Language) under the 'Available SCLs' link.

When accessing a module by clicking on the row, each detail is shown and the user is able to change the SCL file associated with the module.

Under 'Available SCLs' all SCL's are shown and the user can download each configuration for further inspection.

Upload Module

Under the menu 'Modules', a submenu for uploading modules are visible if 'Operator' or 'Administrator'. This allows the user to upload a DLL file and a SCL.

Management

The management menu is only available for the 'Administrator' role. Here four submenus are shown:

- *News*
  Here the administrator can set the visible news or post a news.
- *Users*
  Access to see all users or create a new user.
- *Units*
  Show all the units in the system.
- *Modules*
  Show all the modules in the system.

Using the WebApi RESTful query interface

The application has a RESTful interface that provides access to the data from the DER units. When the data is queried, it is presented in a structure that is compliant with the IEC 61850 standard. The interface uses RSA public/private keys to encrypt the authentication credentials of the user logging in, however to enable you to play around with the query interface easily, we have enabled basic plain text authentication.

The query interface is found at the address location:
http://cloudsolution39.cloudapp.net/api/query

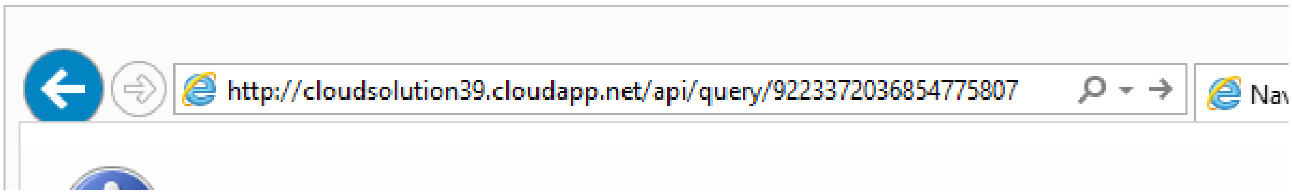
To access the query interface, use these credentials:

User:	user1
Password:	password1

You are now able to query a DER unit for its data. To query a DER unit you need to know its ID and what resources to request. Use the following ID and Logical Device:

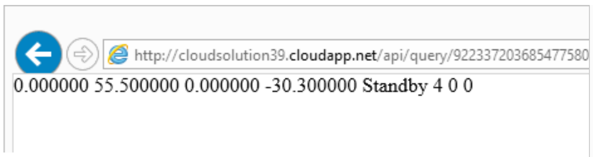
DER Unit ID: 9223372036854775807
Logical Device: DNTHRM1

The most basic query is to ask for everything on a given ID.
Try using this URL: http://cloudsolution39.cloudapp.net/api/query/9223372036854775807

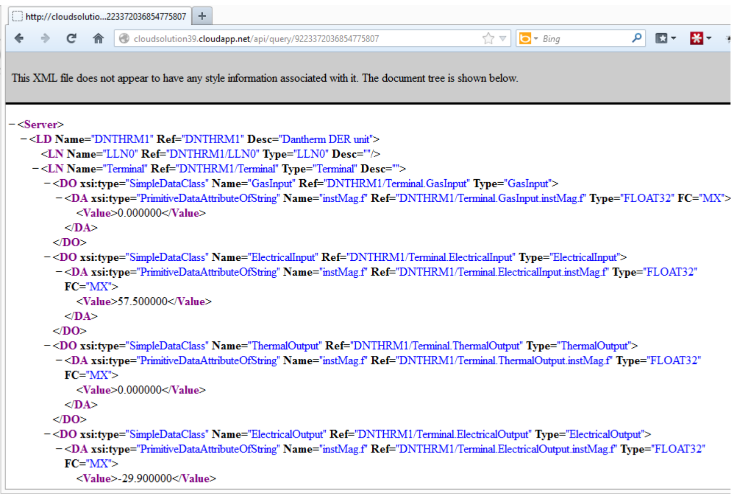


```
http://cloudsolution39.cloudapp.net/api/query/9223372036854775807
```

It will return all data for the DER unit. If you use Internet Explorer, it will most likely be formatted in nice readable way. However, since we want to view the entire XML DOM tree inclusive data attributes it is necessary to right click in Internet Explorer and chose "View Source" or use either Firefox or Chrome to play with the query interface.



```
http://cloudsolution39.cloudapp.net/api/query/922337203685477580
0.000000 55.500000 0.000000 -30.300000 Standby 4 0 0
```



```
cloudsolution39.cloudapp.net/api/query/9223372036854775807

This XML file does not appear to have any style information associated with it. The document tree is shown below.

− <Server>
  − <LD Name="DNTHRM1" Ref="DNTHRM1" Desc="Dantherm DER unit">
      <LN Name="LLN0" Ref="DNTHRM1/LLN0" Type="LLN0" Desc=""/>
    − <LN Name="Terminal" Ref="DNTHRM1/Terminal" Type="Terminal" Desc="">
      − <DO xsi:type="SimpleDataClass" Name="GasInput" Ref="DNTHRM1/Terminal.GasInput" Type="GasInput">
        − <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.GasInput.instMag.f" Type="FLOAT32" FC="MX">
            <Value>0.000000</Value>
          </DA>
        </DO>
      − <DO xsi:type="SimpleDataClass" Name="ElectricalInput" Ref="DNTHRM1/Terminal.ElectricalInput" Type="ElectricalInput">
        − <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalInput.instMag.f" Type="FLOAT32"
          FC="MX">
            <Value>57.500000</Value>
          </DA>
        </DO>
      − <DO xsi:type="SimpleDataClass" Name="ThermalOutput" Ref="DNTHRM1/Terminal.ThermalOutput" Type="ThermalOutput">
        − <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ThermalOutput.instMag.f" Type="FLOAT32"
          FC="MX">
            <Value>0.000000</Value>
          </DA>
        </DO>
      − <DO xsi:type="SimpleDataClass" Name="ElectricalOutput" Ref="DNTHRM1/Terminal.ElectricalOutput" Type="ElectricalOutput">
        − <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalOutput.instMag.f" Type="FLOAT32"
          FC="MX">
            <Value>-29.900000</Value>
```

The query can be filtered using a concept from the IEC 61850 standard called Functional Constraints, which is a category assigned to each data value. When applying a Functional Constraint, only vaules in that category is returned. There are quite a lot of defined categories in the IEC 61850 standard. The DER unit in this example makes use of the following Functional Constraints: **ST** – Status information and **MX** – Measured value.

It is also possible to specify a level depth for the size of the returned data tree. Ie. If a level depth of 2 is specified, the contents of two child levels down from the parent is returned.

Functional Constraints and level depth is applied to the request using query parameters.

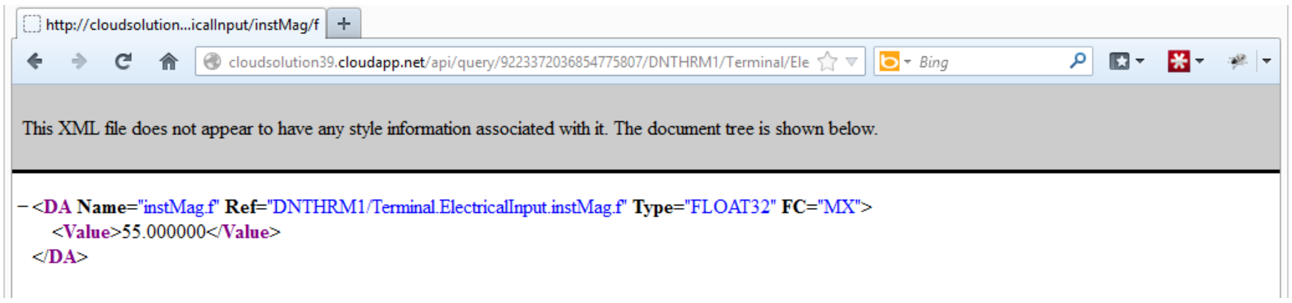
When performing a query, a resource path can be provided in the URL to query into more specific resources in the available data. In order to define a resource path, a basic understading of the data structure is needed. The data tree consists of a *server node* (A DER unit) with a number of *logical devices* (Computers or Terminals within a DER unit). Each *logical device* have a number of *logical nodes* (Data categories) which can contain *data objects* (Data value descriptions). A *data object* can contain *data objects* or *data attributes* (Container for data value) and *data attributes* can contain *data attributes* or *data values* (The actual value). This means that *data objects* and *data attributes* can be recursively defined.

Server Node	- DER unit
Logical Device	- A Terminal within a DER unit
Logical Device	- Another Terminal
Logical Node	- A data category (there a many different categories)
Data Object	- A data description (can contain other data objects recursively)
Data Attribute	- A data value container (can also be defined recursively)
Data Value	- A data value
Data Attribute	- A data value container that contains another data attribute
Data Attribute	- Another data value container
Data Value	- A data value

The reasoning for this data structure is to provide a generic information model that can handle all the current defined data types in the IEC 61850 standard and future extensions.

The RESTful interface allows for quering into this data structure by providing a path of logical devices, data objects and data attributes in the URL.

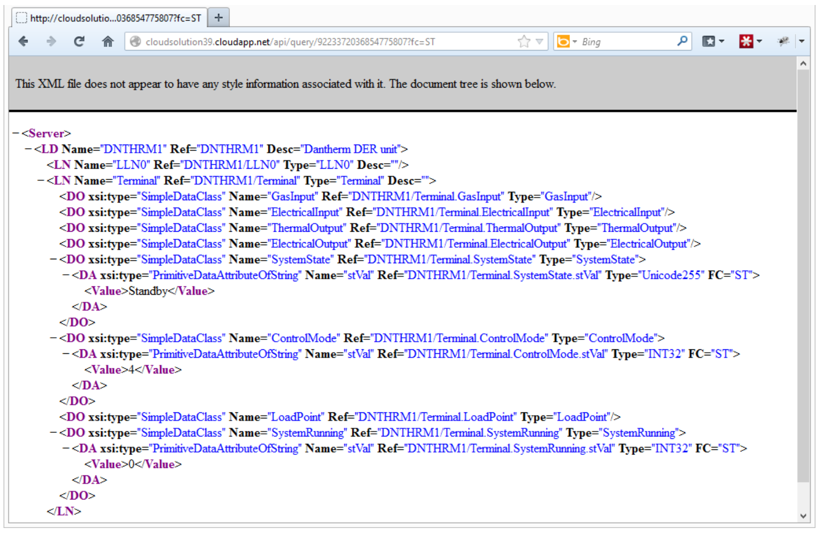
Here we query into the data by using a query path that points to one specific data attribute.
Using this URL:
http://cloudsolution39.cloudapp.net/api/query/9223372036854775807/DNTHRM1/Terminal/ElectricalInput/instMag/f


http://cloudsolution...icalInput/instMag/f
cloudsolution39.cloudapp.net/api/query/9223372036854775807/DNTHRM1/Terminal/Ele
Bing

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <DA Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalInput.instMag.f" Type="FLOAT32" FC="MX">
    <Value>55.000000</Value>
  </DA>
```

Now let's try to use a Functional Constraint to filter on the ST category, so only status information is returned.
Using the URL: http://cloudsolution39.cloudapp.net/api/query/9223372036854775807?fc=ST

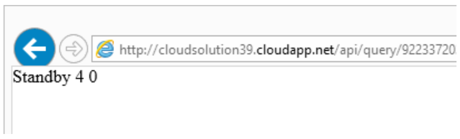


http://cloudsolutio...036854775807?fc=ST

cloudsolution39.cloudapp.net/api/query/9223372036854775807?fc=ST

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <Server>
  - <LD Name="DNTHRM1" Ref="DNTHRM1" Desc="Dantherm DER unit">
      <LN Name="LLN0" Ref="DNTHRM1/LLN0" Type="LLN0" Desc=""/>
    - <LN Name="Terminal" Ref="DNTHRM1/Terminal" Type="Terminal" Desc="">
        <DO xsi:type="SimpleDataClass" Name="GasInput" Ref="DNTHRM1/Terminal.GasInput" Type="GasInput"/>
        <DO xsi:type="SimpleDataClass" Name="ElectricalInput" Ref="DNTHRM1/Terminal.ElectricalInput" Type="ElectricalInput"/>
        <DO xsi:type="SimpleDataClass" Name="ThermalOutput" Ref="DNTHRM1/Terminal.ThermalOutput" Type="ThermalOutput"/>
        <DO xsi:type="SimpleDataClass" Name="ElectricalOutput" Ref="DNTHRM1/Terminal.ElectricalOutput" Type="ElectricalOutput"/>
      - <DO xsi:type="SimpleDataClass" Name="SystemState" Ref="DNTHRM1/Terminal.SystemState" Type="SystemState">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.SystemState.stVal" Type="Unicode255" FC="ST">
            <Value>Standby</Value>
          </DA>
        </DO>
      - <DO xsi:type="SimpleDataClass" Name="ControlMode" Ref="DNTHRM1/Terminal.ControlMode" Type="ControlMode">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.ControlMode.stVal" Type="INT32" FC="ST">
            <Value>4</Value>
          </DA>
        </DO>
        <DO xsi:type="SimpleDataClass" Name="LoadPoint" Ref="DNTHRM1/Terminal.LoadPoint" Type="LoadPoint"/>
      - <DO xsi:type="SimpleDataClass" Name="SystemRunning" Ref="DNTHRM1/Terminal.SystemRunning" Type="SystemRunning">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="stVal" Ref="DNTHRM1/Terminal.SystemRunning.stVal" Type="INT32" FC="ST">
            <Value>0</Value>
          </DA>
        </DO>
      </LN>
```

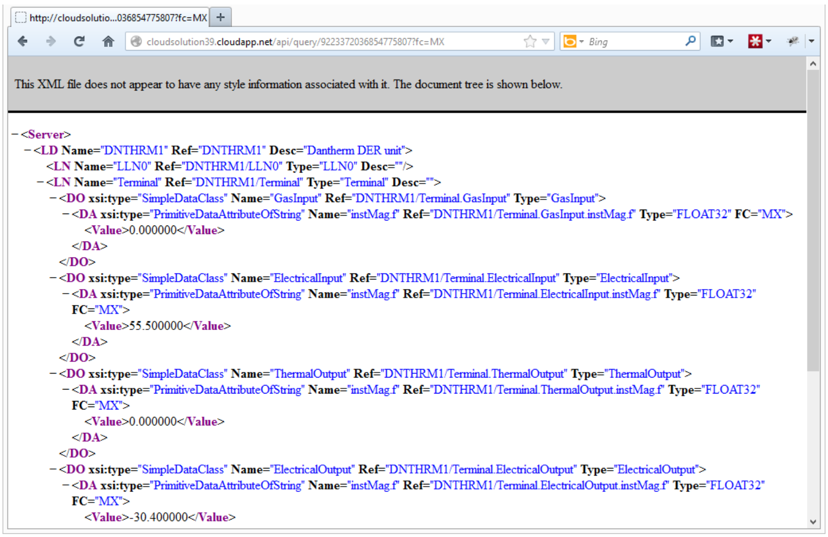


http://cloudsolution39.cloudapp.net/api/query/92233720

```
Standby 4 0
```

Now only the status information values are returned and none of the measured values.

And to return only the MX – measured values, we can filter on the MX category.
Using the URL: http://cloudsolution39.cloudapp.net/api/query/9223372036854775807?fc=MX

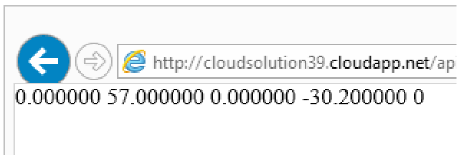


http://cloudsolutio...036854775807?fc=MX

cloudsolution39.cloudapp.net/api/query/9223372036854775807?fc=MX

This XML file does not appear to have any style information associated with it. The document tree is shown below.

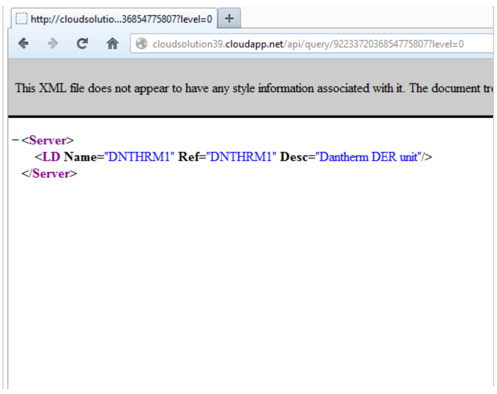
```
- <Server>
  - <LD Name="DNTHRM1" Ref="DNTHRM1" Desc="Dantherm DER unit">
      <LN Name="LLN0" Ref="DNTHRM1/LLN0" Type="LLN0" Desc=""/>
    - <LN Name="Terminal" Ref="DNTHRM1/Terminal" Type="Terminal" Desc="">
      - <DO xsi:type="SimpleDataClass" Name="GasInput" Ref="DNTHRM1/Terminal.GasInput" Type="GasInput">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.GasInput.instMag.f" Type="FLOAT32" FC="MX">
            <Value>0.000000</Value>
          </DA>
        </DO>
      - <DO xsi:type="SimpleDataClass" Name="ElectricalInput" Ref="DNTHRM1/Terminal.ElectricalInput" Type="ElectricalInput">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalInput.instMag.f" Type="FLOAT32"
          FC="MX">
            <Value>55.500000</Value>
          </DA>
        </DO>
      - <DO xsi:type="SimpleDataClass" Name="ThermalOutput" Ref="DNTHRM1/Terminal.ThermalOutput" Type="ThermalOutput">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ThermalOutput.instMag.f" Type="FLOAT32"
          FC="MX">
            <Value>0.000000</Value>
          </DA>
        </DO>
      - <DO xsi:type="SimpleDataClass" Name="ElectricalOutput" Ref="DNTHRM1/Terminal.ElectricalOutput" Type="ElectricalOutput">
        - <DA xsi:type="PrimitiveDataAttributeOfString" Name="instMag.f" Ref="DNTHRM1/Terminal.ElectricalOutput.instMag.f" Type="FLOAT32"
          FC="MX">
            <Value>-30.400000</Value>
```



http://cloudsolution39.cloudapp.net/ap

```
0.000000 57.000000 0.000000 -30.200000 0
```

Finally we can limit the depth of the returned query by using the depth level parameter.
Using the URL: http://cloudsolution39.cloudapp.net/api/query/9223372036854775807?level=0
Returns only the content of the parent.
And using: http://cloudsolution39.cloudapp.net/api/query/9223372036854775807?level=2
Returns a little more details.

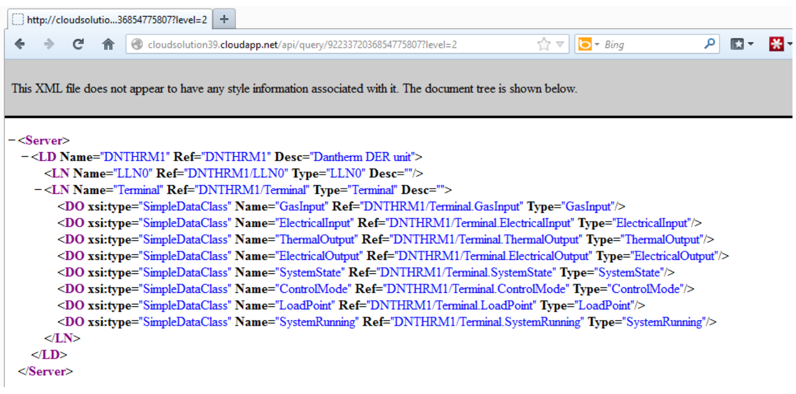


http://cloudsolutio...36854775807?level=0

cloudsolution39.cloudapp.net/api/query/9223372036854775807?level=0

This XML file does not appear to have any style information associated with it. The document tr

```
- <Server>
    <LD Name="DNTHRM1" Ref="DNTHRM1" Desc="Dantherm DER unit"/>
  </Server>
```



http://cloudsolutio...36854775807?level=2

cloudsolution39.cloudapp.net/api/query/9223372036854775807?level=2

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <Server>
  - <LD Name="DNTHRM1" Ref="DNTHRM1" Desc="Dantherm DER unit">
      <LN Name="LLN0" Ref="DNTHRM1/LLN0" Type="LLN0" Desc=""/>
    - <LN Name="Terminal" Ref="DNTHRM1/Terminal" Type="Terminal" Desc="">
        <DO xsi:type="SimpleDataClass" Name="GasInput" Ref="DNTHRM1/Terminal.GasInput" Type="GasInput"/>
        <DO xsi:type="SimpleDataClass" Name="ElectricalInput" Ref="DNTHRM1/Terminal.ElectricalInput" Type="ElectricalInput"/>
        <DO xsi:type="SimpleDataClass" Name="ThermalOutput" Ref="DNTHRM1/Terminal.ThermalOutput" Type="ThermalOutput"/>
        <DO xsi:type="SimpleDataClass" Name="ElectricalOutput" Ref="DNTHRM1/Terminal.ElectricalOutput" Type="ElectricalOutput"/>
        <DO xsi:type="SimpleDataClass" Name="SystemState" Ref="DNTHRM1/Terminal.SystemState" Type="SystemState"/>
        <DO xsi:type="SimpleDataClass" Name="ControlMode" Ref="DNTHRM1/Terminal.ControlMode" Type="ControlMode"/>
        <DO xsi:type="SimpleDataClass" Name="LoadPoint" Ref="DNTHRM1/Terminal.LoadPoint" Type="LoadPoint"/>
        <DO xsi:type="SimpleDataClass" Name="SystemRunning" Ref="DNTHRM1/Terminal.SystemRunning" Type="SystemRunning"/>
      </LN>
    </LD>
  </Server>
```

The DER units we currently have connected to our cloud application are Read-Only, as they are real DER units: Combined Heater and Power units (CHP) provided by Dantherm A/S. Sending random commands to these running machines would be unfortunate and as such, we cannot provide access to updating resources through the RESTful interface. However, this is what a POST request to update a resource on a DER would look like. This update request was created using JavaScript to build a JSON based POST request to the URL: http://cloudsolution39.cloudapp.net/api/update/<der unit>/<resource path> and the body of the request contains the new data value.

POST Request	**Response**
POST /api/update/<der id>/<path> Host: 127.0.0.1 User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0 Accept: */* Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate DNT: 1 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 X-Requested-With: XMLHttpRequest Content-Length: 25 Connection: keep-alive Pragma: no-cache Cache-Control: no-cache { "value" : "12345.678" }	HTTP/1.1 200 Value accepted Cache-Control: no-cache Pragma: no-cache Transfer-Encoding: chunked Content-Type: text/plain; charset=utf-8 Content-Encoding: gzip Expires: -1 Vary: Accept-Encoding

The important thing to look at here, is that the POST request sends a data value, using a specific path, to the RESTful interface and the interface returns with a HTTP response value of 200 OK, which means the request was successful.

Depending on the type of request and the interval the DER unit uses to communicates with our cloud service (different commands may prompt different communications intervals), it may take some amount of time before the result of the command sent to the DER is reflected in a new data query to the interface. Ex. If a shutdown command is issued to a water boiler type of DER unit, it can take more than 8 hours before the DER unit is in a turned off state. It would be normal, that the DER unit would have a sequence of states such as: Running, Shutting Down, Turned Off and would communicate the Shutting Down status immediately to our cloud service upon receiving a shutdown command.

The following is a sample of links to queries on the RESTful interface that displays different results:

1. http://cloudsolution39.cloudapp.net/api/query/9223372036854775807
   - Basic query that displays all data

2. http://cloudsolution39.cloudapp.net/api/query/9223372036854775807/DNTHRM1/Terminal/
   - Query the Terminal logical node on the DNTHRM1 logical device

3. http://cloudsolution39.cloudapp.net/api/query/9223372036854775807/DNTHRM1/Terminal?fc=MX
   - Query the Terminal logical node and only return values from the category MX

4. http://cloudsolution39.cloudapp.net/api/query/9223372036854775807/DNTHRM1/Terminal/SystemState
   - Query for the system state status value on the DNTHRM1 logical device

Class diagrams and Models

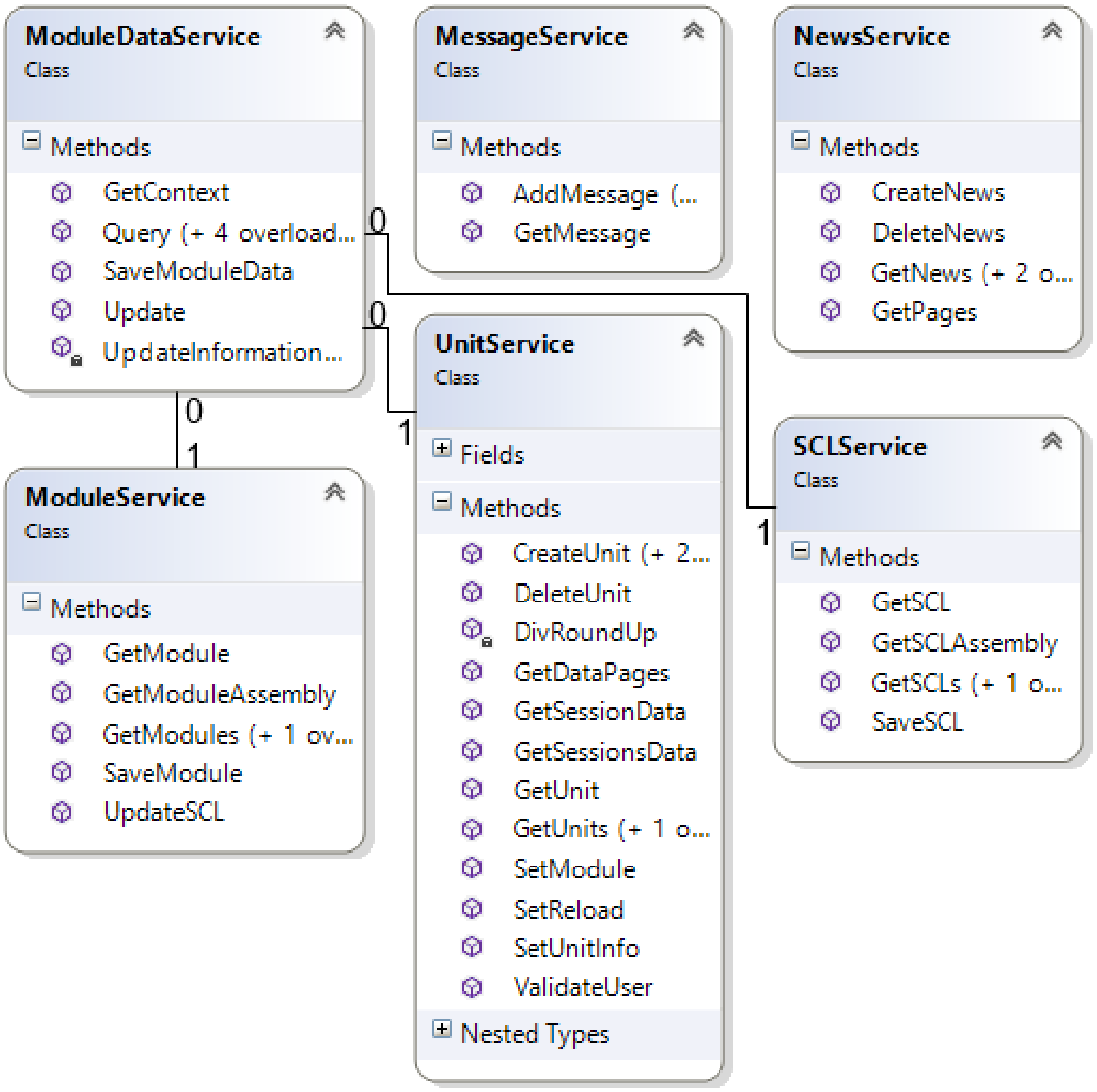


Figure F.1: Class diagram: Service Layer

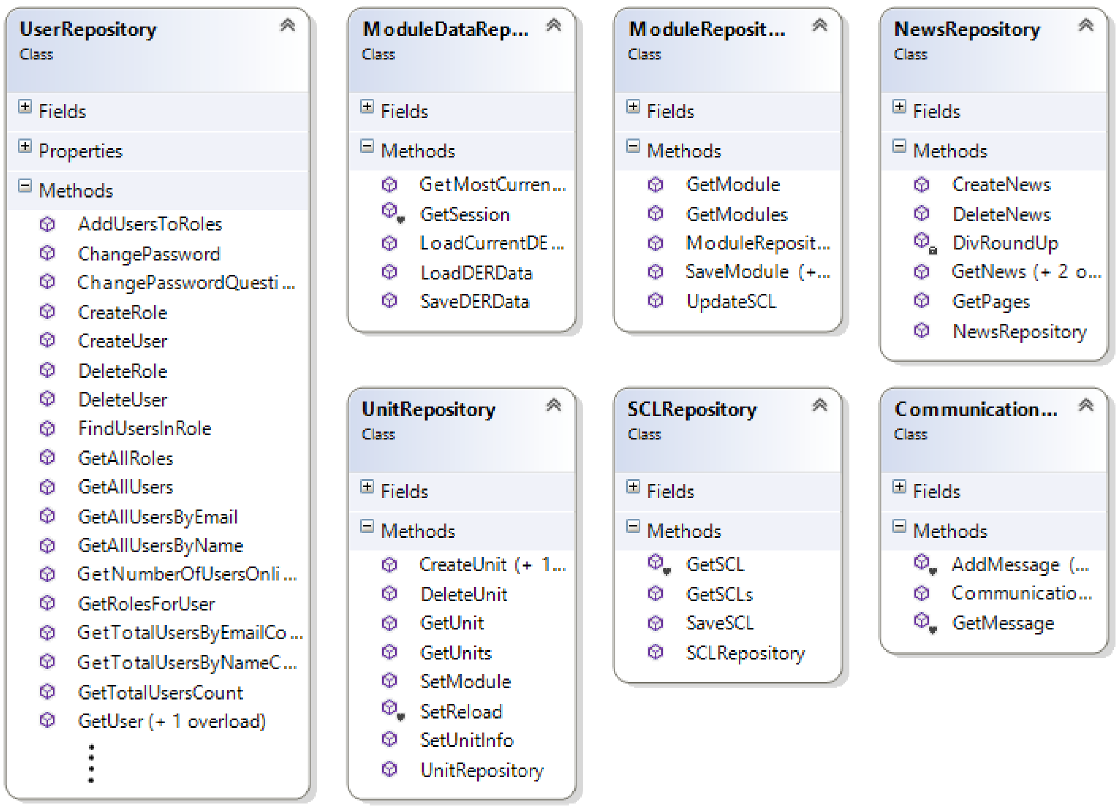


Figure F.2: Class diagram: Repository Layer

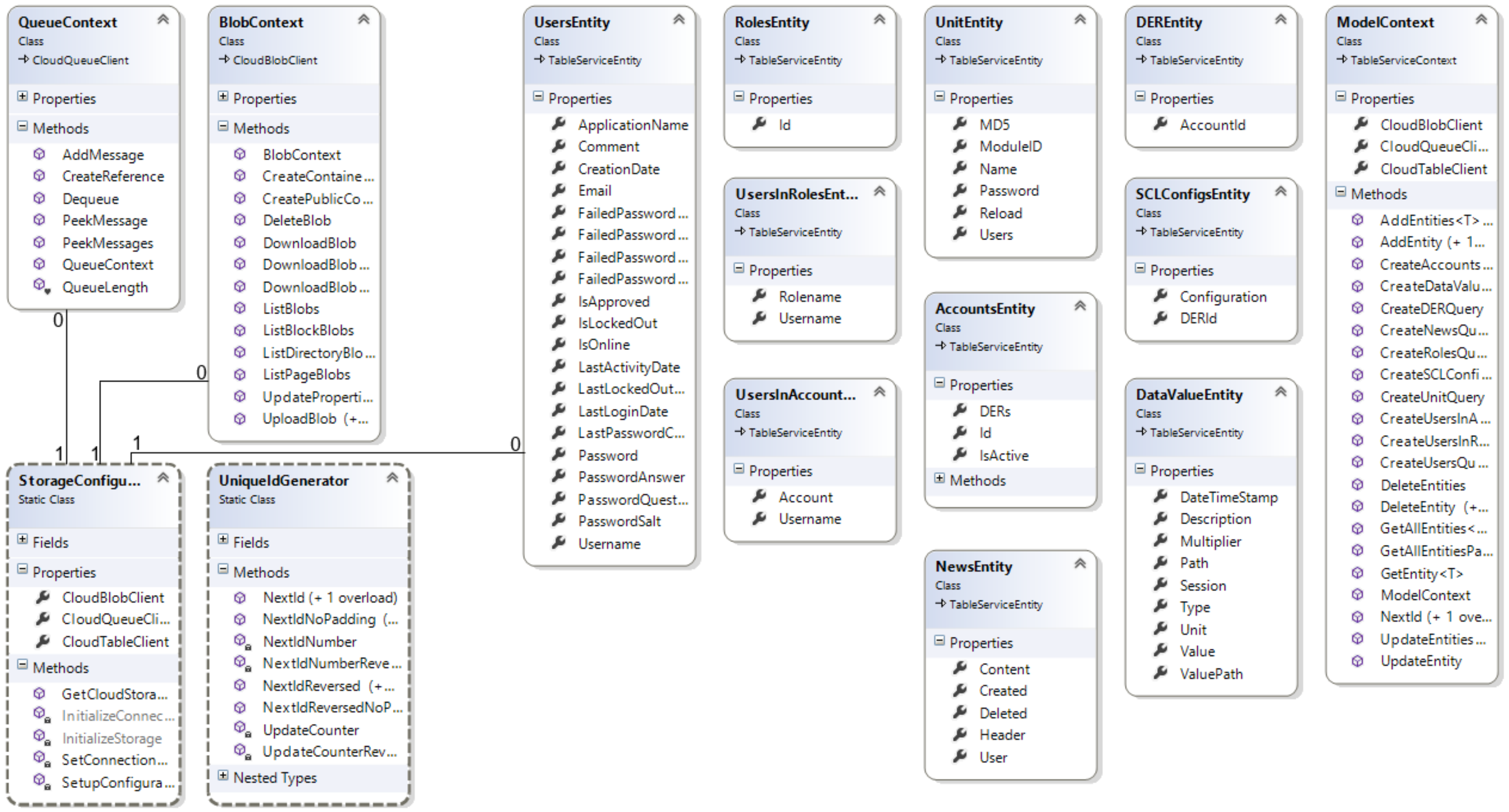


Figure F.3: Class diagram: Storage

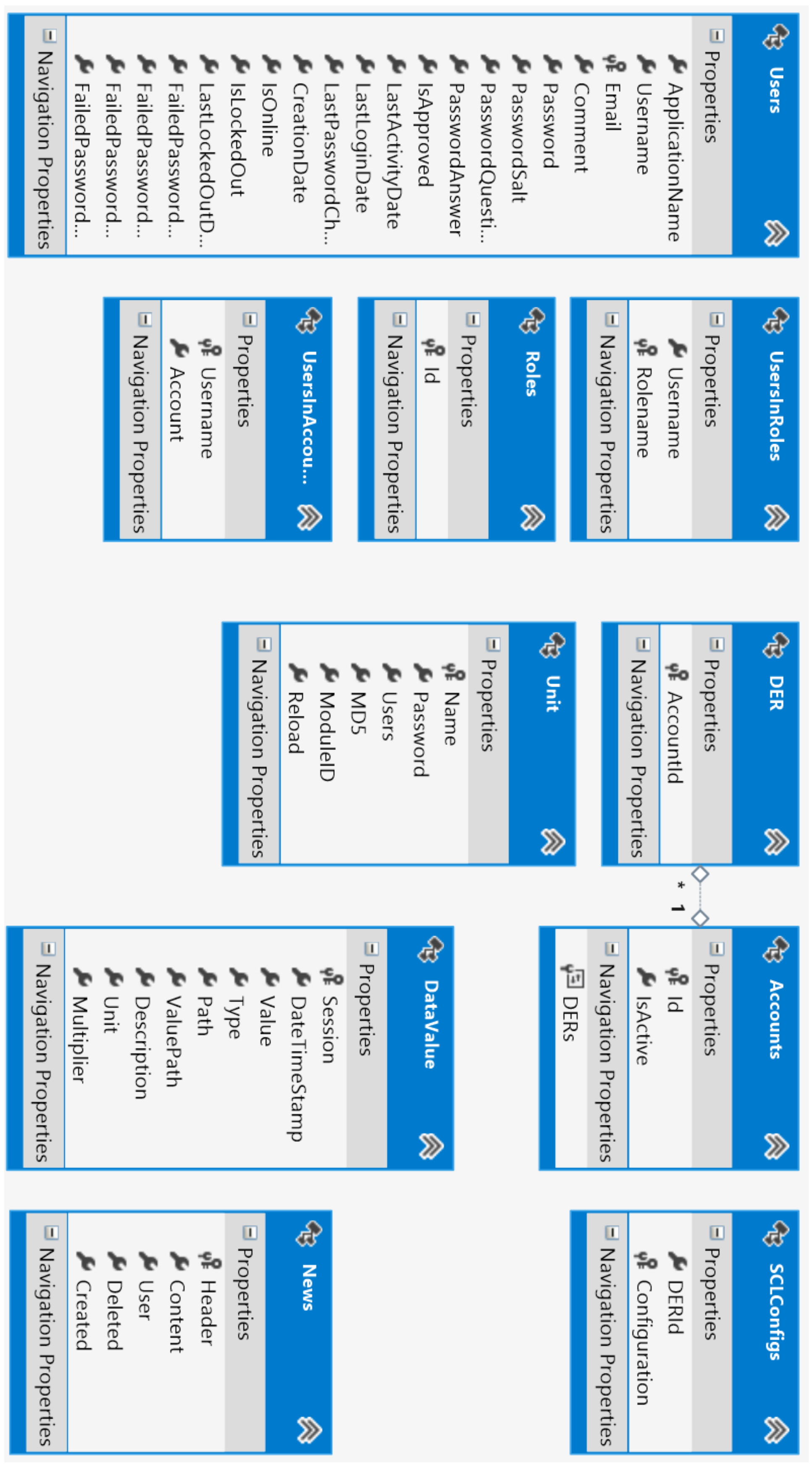


Figure F.4: Model Generated Entities for the Storage

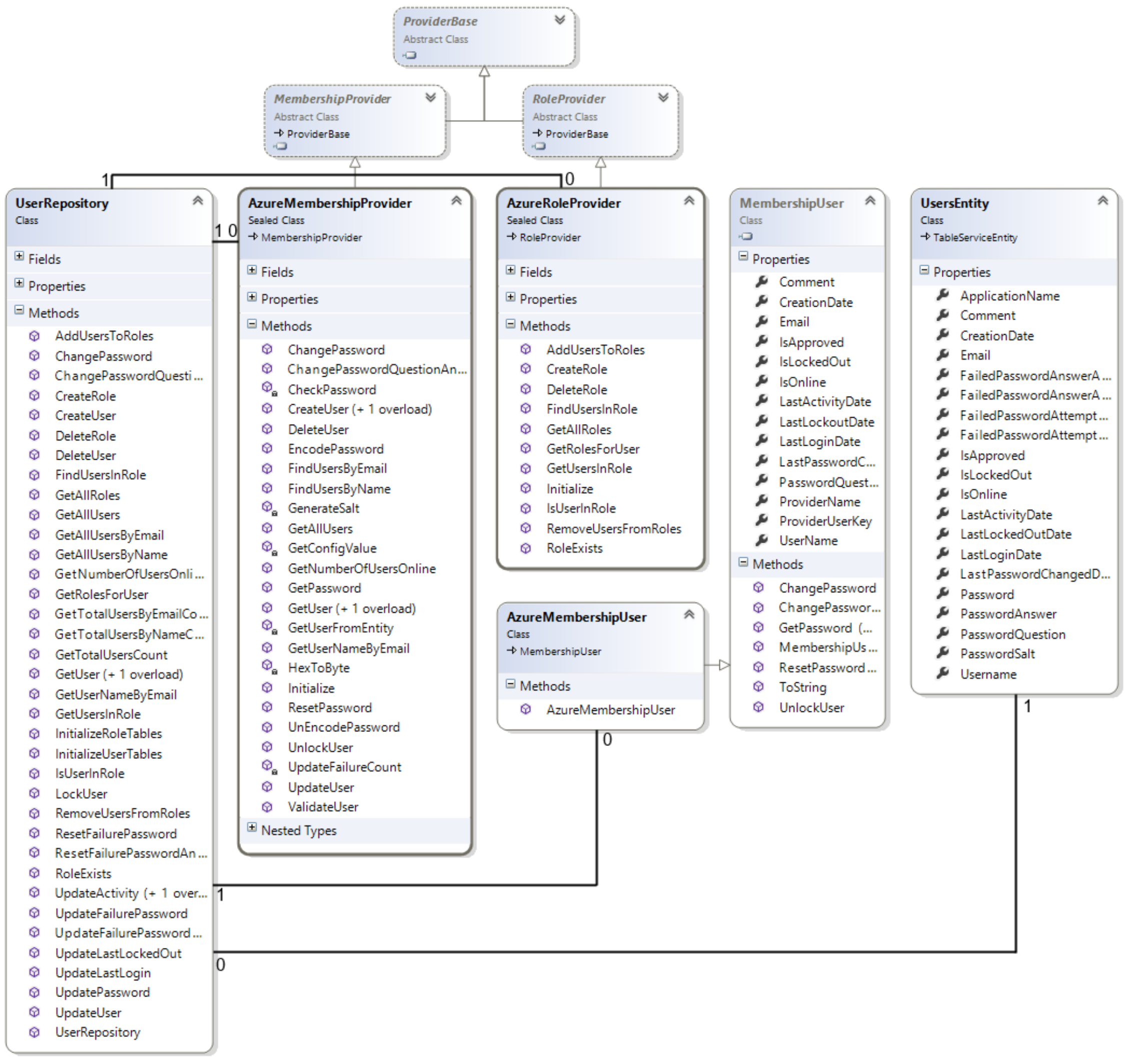


Figure F.5: Class diagram: Users and Membership Providers

Use Cases

G.1 Use Case : Create Unit

Use Case 1	**Create Unit**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Farmer
Postcondition:	A new unit created with a unique ID and a password

Main Success Scenario:

1. Actor logs into the website (see use case “Login On Website”)
   1. Redirected to main site
2. Navigate to the ’Create Unit’ page
3. Enter a unit name in the text box
4. Select a module to use
5. Send the post request to create the unit

Extensions:

1.a Unable to log in

5.a Invalid log in data

1. System shows failure message
2. User returns to step 3

G.2 Use Case : Login on Website

Use Case 2	**Login on Website**

Scope:	System-wide
Level:	Subfunction
Primary Actor:	Customer & Farmer
Preconditions:	Actors has a user ID and a Password
Postcondition:	User has successfully logged into the website

Main Success Scenario:

1. Actor enters the Website
2. Navigates to the 'Log On' page
3. Enter the actors user name in the first text box
4. Enter the related password in the second text box
5. Clicks on the 'Log On' button

Extensions:
5.a Invalid login data

1. System shows failure message
2. User returns to step 3

G.3 Use Case : Upload Module & SCL

Use Case 3	**Upload Module & SCL**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Customer
Postcondition:	Module & SCL uploaded

Main Success Scenario:

1. Actor logs into the website (see use case “Login On Website”)
2. Navigate to the ’Upload Module’ page
3. Enter the module name
4. Enter the module version
5. Select the module DLL
6. Enter the SCL version
7. Select the SCL file
8. Upload the files

Extensions:

1.a Unable to log in

  1. Redirected to main site

5.a Invalid module file

  1. System shows failure message
  2. User returns to step 3

7.a Invalid SCL file

  1. System shows a failure message
  2. User returns to step 3

8.a Module with identical version already exists

  1. System shows failure message
  2. User returns to step 3

8.b No module file selected

  1. Only the SCL is being uploaded

8.b No SCL file selected

  1. Only the module is being uploaded

G.4 Use Case : Accessing Unit Data

Use Case 4	**Accessing Unit Data**

Scope:	System-wide
Level:	User-goal
Primary Actor:	Farmer

Precondition:

1. Customer has access to the unit
2. The unit has posted data into the cloud

Postcondition:	A new module has been added to the system with an associated SCL

Main Success Scenario:

1. Actor logs into the website (see use case “Login On Website”)
2. Navigate to the ’Units’ page
3. Find the unit in a list of units
4. Accessing the unit
5. Each unit data is shown in a list

Extensions:
1.a Unable to log in

   1. Redirected to main site

G.5 Use Case : Connect to CloudSolution

Use Case 5	**Connect to CloudSolution**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Unit
Precondition:	Unit is created in the CloudSolution
Postcondition:	Unit is sending data to the cloud

Main Success Scenario:

1. Enter ID and Password in the Unit
2. Start authentication
3. Retrieve data from DER
4. Send data

Extensions:

2.a Authentication failed

   1. Error message is shown

G.6 Use Case : Authenticate Unit

Use Case 6	**Authenticate Unit**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Unit
Precondition:	Unit has knows the ID and Password
Postcondition:	Unit has been authorized to transmit data

Main Success Scenario:

1. Opens a WebSocket with the cloud service
2. Transmit a Login command with ID and Password
3. Retrieve a confirmation from the cloud service

Extensions:

1.a The Socket was unable to open

   1. Error message is shown

2.a Authentication failed

   1. Error message is shown

G.7 Use Case : Send Raw Data

Use Case 7	**Send Raw Data**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Unit
Precondition:	Unit has been authorized by the cloud service
Postcondition:	Successfully transmitted raw data to the cloud

Main Success Scenario:

1. Opens a WebSocket with the cloud service
2. Transmit raw data and an associated ID
3. Retrieve a confirmation from the cloud service

Extensions:

1.a The Socket was unable to open

  1. Error message is shown

2.a An exception was found with the data

  1. Error message is shown
  2. Transmission is starting over

G.8 Use Case : Query Data

Use Case 8	**Query Data**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Customer
Precondition:	

1. The customer must have authenticated with the interface.
2. A unit must have been created.
3. The customer must have a unit id.

Postcondition:	A response in XML or JSON is generated and sent to the customer.

Main Success Scenario:

1. Enter unit id and IEC reference in request
2. Send request

Extensions:

1.a Add functional constraints to request

  1. The response is filtered according to the functional constraints

G.9 Use Case : Post Data

Use Case 9	**Post Data**
Scope:	System-wide
Level:	User-goal
Primary Actor:	Customer

Precondition:

1. The customer must have authenticated with the interface.
2. A unit must have been created.
3. The customer must have a unit id.

Postcondition:	Data is posted to the DER unit

Main Success Scenario:

1. Enter unit id and IEC reference in request
2. Add values to be updated on the reference to the body of the request
3. Send request

Extensions:

1.a Wrong unit id used in query

1. Error message is shown which states that wrong arguments was supplied with the request

---

References

[1] Rolf Adam and Walter Wintersteller. What's So Smart about the Smart Grid? http://www.strategy-business.com/media/file/leading_ideas-20080902.pdf, 2008.

[2] Amazon. Building GrepTheWeb in the Cloud, Part 1: Cloud Architectures. http://aws.amazon.com/articles/1632, 2008.

[3] Amazon. Architecting for the Cloud: Best Practices. http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf, 2011.

[4] Amazon. Overview of Amazon Web Services. http://media.amazonwebservices.com/AWS_Overview.pdf, 2013.

[5] Peter Bach Andersen, Bjarne Poulsen, C Trholt, and Jacob Ostergaard. Using service oriented architecture in a generic virtual power plant. In Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on, pages 1621–1622. IEEE, 2009.

[6] Kent Beck. Test-Driven Development. Addison-Wesley, 2007.

[7] Benjamin Biegel. Flexibility Interface – Information Modeling for Direct Control. http://www.ipower-net.dk/~/media/iPower/Documents/fi.ashx, 2011.

[8] BizCloud. BizCloud Overview of Top 10 Security Threats of Cloud Computing. http://bizcloudnetwork.com/bizcloud-overview-of-top-10-security-threats-of-cloud-computing, 2011.

[9] IBM Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall. Cloud Computing. www.ibm.com/developerworks/websphere/zones/hipods/, 2007.

[10] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. Cloud Computing Principles and Paradigms. Wiley Publishing, 2011. ISBN 9780470887998.

[11] CEN-CENELEC-ETSI Smart Grid Coordination Group. Smart Grid Reference Architecture. http://ec.europa.eu/energy/gas_electricity/smartgrids/doc/xpert_group1_reference_architecture.pdf, 2012.

[12] Cenelec. European Committee for Electrotechnical Standardization. http://www.cenelec.eu/, 2013.

[13] European Commission. Protection of personal data. http://ec.europa.eu/justice/data-protection/, 2013.

[14] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. Introduction to Algorithms. McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

[15] D. Crockford. JavaScript Object Notation. `http://www.json.org/` and `http://www.ietf.org/rfc/rfc4627.txt?number=4627`, 2006.

[16] Danish Energy Agency. Energy Policy in Denmark. `http://www.ens.dk/Documents/Netboghandel%20-%20publikationer/2013/Energy%20Policy%20in%20Denmark.pdf`, 2012.

[17] Energihjem.dk. Køb solceller inden det er for sent – 57.000 kr. i tilskud. `http://www.energihjem.dk/kob-solceller-inden-det-er-for-sent-57-000-kr-i-tilskud/`, 2012.

[18] Energinet.dk. Ancillary services to be delivered in Denmark - Tender conditions. `http://energinet.dk/SiteCollectionDocuments/Engelske%20dokumenter/El/8871-11%20v3%20Ancillary%20services%20to%20be%20delivered%20in%20Denmark%20-%20Tender%20conditions.%20Valid%20from%203%20October%202012.pdf`, 2012.

[19] Energinet.dk and The Danish Energy Association. Smart Grid i Danmark. `http://energinet.dk/SiteCollectionDocuments/Engelske%20dokumenter/Forskning/Smart%20Grid%20in%20Denmark.pdf`, 2010.

[20] Energinet.dk and Dansk Energi. Smart Grid i Danmark 2.0. `http://energinet.dk/SiteCollectionDocuments/Danske%20dokumenter/Forskning/Smart_Grid_i_DK.pdf`, 2011.

[21] Energinet.dk and Dansk Energi. Kortlægning af den danske elbranches Smart Grid FUD-indsats. `http://www.danskenergi.dk/~/media/Smart_Grid/Kortlaegning%20af%20den%20danske%20elbranches%20Smart%20Grid%20FUD%20indsats.pdf.ashx`, 2011.

[22] H. Farhangi. The path of the smart grid. Power and Energy Magazine, IEEE, 8(1):18–28, 2010. ISSN 1540-7977. doi: 10.1109/MPE.2009.934876.

[23] Global Smart Grid Federation. Smart Grid Definition. `http://www.globalsmartgridfederation.org/smartgriddef.html`, 2012.

[24] I. Fette and A. Melnikov. RFC: The WebSocket Protocol. `http://tools.ietf.org/html/rfc6455`, December 2011.

[25] Martin Fowler. Inversion of Control Containers and the Dependency Injection pattern. `http://www.martinfowler.com/articles/injection.html`, 23 January 2004.

[26] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2.

[27] Gartner. Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 2Q12 Update. `http://www.gartner.com/resId=2126916`, 2012.

[28] Go'Energi. Elproduktion og -forbrug. http://energinet.dk/DA/KLIMA-OG-MILJOE/Elsektorens-miljoepaavirkninger/Elproduktion-i-Danmark/Sider/Elproduktion-og-forbrug.aspx, 2012.

[29] Google. Google App Engine. https://developers.google.com/appengine/, 2013.

[30] Google. Google Cloud Platform Overview. https://developers.google.com/cloud/samples/photofeed/overview, 2013.

[31] Google. Google Cloud Platform. https://developers.google.com/cloud/, 2013.

[32] European Technology Platform Smart Grid. Strategic Deployment Document. http://www.smartgrids.eu/documents/SmartGrids_SDD_FINAL_APRIL2010.pdf, 2010.

[33] GridWise. Smart Grid Interoperability Maturity Model Summary. http://www.gridwiseac.org/about/imm.aspx, 2013.

[34] Erik Hansen and Landbrugnet. Brug for flere vindmøller i landbruget. http://landbrugsyd.landbrugnet.dk/Artikler/9887/brug-for-flere-vindmoeller-i-landbruget, 06-02-2012.

[35] Van Hoboken, Joris V. J., Axel Arnbak, and Van Eijk. Cloud Computing in Higher Education and Research Institutions and the USA Patriot Act. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2181534, 2012.

[36] Home.dk. Solceller på taget er 'hot'. http://home.dk/bolignyt/flere-artikler/juni-2012/solceller-paa-taget-er-%E2%80%99hot%E2%80%99, 2012.

[37] IBM. Cloud computing service models. http://www.ibm.com/developerworks/cloud/library/cl-cloudservicemodels/?cmp=dw&cpb=dwcld&ct=dwnew&cr=dwnen&ccy=zz&csr=021011, 2011.

[38] IBM DeveloperWorks and A. Rodriguez. RESTful Web services: The basics. https://www.ibm.com/developerworks/webservices/library/ws-restful, 2008.

[39] International Electrotechnical Commission (IEC). Basic communication structure for substation and feeder equipment – Principles and models. http://www.iec.ch/, 2003.

[40] International Electrotechnical Commission (IEC). Communication networks and systems in substations Part 7-2: Basic communication structure for substation and feeder equipment - Abstract communication service interface (ACSI). http://www.iec.ch/, 2003.

[41] International Electrotechnical Commission (IEC). Communication networks and systems in substations Part 7-3: Basic communication structure for substation and feeder equipment - Common Data Classes. http://www.iec.ch/, 2003.

[42] International Electrotechnical Commission (IEC). Communication networks and systems in power utility automation - Requirements for web-based and structured access to the IEC 61850 information models. `http://www.iec.ch/`, 2010.

[43] International Electrotechnical Commission (IEC). Communication networks and systems for power utility automation – Part 80-3: Mapping to Web Services – Requirement Analysis and Technology Assessment. `http://www.iec.ch/`, 2012.

[44] International Electrotechnical Commission (IEC). International Electrotechnical Commission 61850 Standard. `http://www.iec.ch/`, 2003.

[45] International Electrotechnical Commission (IEC). Compatible logical node classes and data classes. `http://www.iec.ch/`, 2003.

[46] International Electrotechnical Commission (IEC). Communication Networks and Systems for Power Utility Automation for Distributed Energy Resources (DER). `http://www.iec.ch/`, 2008.

[47] International Electrotechnical Commission (IEC). IEC 62541 - OPC Unified Architecture. `http://www.iec.ch/`, 2010.

[48] International Energy Agency. International Energy Outlook 2011. `http://www.eia.gov/forecasts/ieo/pdf/0484(2011).pdf`, 2011.

[49] Blizcloud jason. Understanding the Business Benefits of Platform as a Service. `http://bizcloudnetwork.com/understanding-the-business-benefits-of-platform-as-a-service`, 2011.

[50] Alex C. Lakatos. The USA Patriot Act and the Privacy of Data Stored in the Cloud. `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2181534`, 2012.

[51] R.E. Mackiewicz. Overview of iec 61850 and benefits. In Power Systems Conference and Exposition, 2006. PSCE'06. 2006 IEEE PES, pages 623–630. IEEE, 2006.

[52] J.D. Meier, Carlos Farre, Jason Taylor, Prashant Bansode, Steve Gregersen, Madhu Sundararajan, and Rob Boucher. Improving Web Services Security. Microsoft, 2008.

[53] Microsoft. Windows Azure Cloud Platform. `http://www.windowsazure.com/en-us/`, 2013.

[54] Microsoft. Windows Azure Architecture. `http://msdn.microsoft.com/en-us/library/windowsazure/dd163896.aspx`, 2013.

[55] Microsoft MSDN Library. Introduction to Membership. `http://msdn.microsoft.com/en-us/library/yh26yfzy%28v=vs.100%29.aspx`, 2012.

[56] Microsoft MSDN Library. ASP.NET MVC 4. `http://msdn.microsoft.com/en-us/library/dd381412%28v=vs.108%29.aspx`, 2012.

[57] Microsoft MSDN Library. Optimistic Concurrency. `http://msdn.microsoft.com/en-us/library/aa0416cz%28v=vs.71%29.aspx`, 2012.

[58] Microsoft MSDN Library. The Repository Pattern. http://msdn.microsoft.com/en-us/library/ff649690.aspx, 2012.

[59] NIST. NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 1.0. http://www.nist.gov/public_affairs/releases/upload/smartgrid_interoperability_final.pdf, 2010.

[60] Department of Justice. The USA PATRIOT Act: Preserving Life and Liberty. http://www.justice.gov/archive/ll/what_is_the_patriot_act.pdf, 2001.

[61] NIST National Institute of Standards and Technology. The NIST Definition of Cloud Computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

[62] Anders Bro Pedersen and Einar Bragi Hauksson. Enabling distributed energy resources in a virtual power plant using iec 61850. Master's thesis, Technical University of Denmark, DTU, 2010.

[63] Anders Bro Pedersen, Einar Bragi Hauksson, Peter Bach Andersen, Bjarne Poulsen, Chresten Træholt, and Dieter Gantenbein. Facilitating a generic communication interface to distributed energy resources: Mapping iec 61850 to restful services. In Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on, pages 61–66. IEEE, 2010.

[64] Steve Resnick, Richard Crane, and Chris Bowen. Essential Windows Communication Foundation. Addison-Wesley, 2008.

[65] Margaret Rouse. Definition : cloud computing. http://searchcloudcomputing.techtarget.com/definition/cloud-computing, 2010.

[66] Ahmed Shafiu and Melanie Watts. D4.3: A Working Document on Communication and Control Requirements. http://www.smart-a.org/D4.3_Comm&Control_requirements_finalversion.pdf, 2007.

[67] Claus Sonne and Jyllands-Posten. Elbiler på vej mod gennembrud. http://jyllands-posten.dk/motor/article4730930.ece, 2012.

[68] Projektleder Steen Luk and DLBR Energi Invest. Vindenergi i landbruget. http://www.dlbr.dk/Produkter/DLBRSpecialraadgivning/DLBREnergiInvest/2011_11_Vindenergi_i_landbruget.htm, 2011.

[69] La'Quata Sumter. Cloud computing: security risk. In Proceedings of the 48th Annual Southeast Regional Conference, ACM SE '10, pages 112:1–112:4, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0064-3. doi: 10.1145/1900008.1900152. URL http://doi.acm.org/10.1145/1900008.1900152.

[70] TorchLMS. The Cloud: The Benefits of SaaS. http://torchlms.com/blog/learning-technology/benefits-of-software-as-a-service, 2012.

[71] A. Troi. About ipower. http://www.ipower-net.dk/About_iPower.aspx, 2012, Mar.

[72] Twenties. Providing flexibility with a virtual power plant. http://www.twenties-project.eu/system/files/Deliv_10_2.pdf, 2012.

[73] Kenneth W Umbach. What is" push Technology"? California State Library, California Research Bureau, 1997.

[74] Anthony T. Velte, Toby J. Velte, and Robert Elsenpeter. Cloud Computing: A Practical Approach. McGraw Hill, 2010. ISBN 978-0-07-162695-8.

[75] Zack Whittaker. Microsoft admits Patriot Act can access EU-based cloud data. http://www.zdnet.com/blog/igeneration/microsoft-admits-patriot-act-can-access-eu-based-cloud-data/11225, 2011.

[76] Zack Whittaker. PRISM: Here's how the NSA wiretapped the Internet. http://www.zdnet.com/prism-heres-how-the-nsa-wiretapped-the-internet-7000016565/, 2013.

[77] World Wide Web Consortium (W3C). Single Object Access Protocol Specification. http://www.w3.org/TR/soap, 2007.

[78] Shi You, Francesco Marra, and Chresten Træholt. Integration of fuel cell micro-chps on low voltage grid: A danish case study. In Power and Energy Engineering Conference (APPEEC), 2012 Asia-Pacific, pages 1–4. IEEE, 2012.

[79] Alex Yu Zhang. Smart Grid Environmental Benefits. http://www.smartgridnews.com/artman/publish/article_289.html, 2007.