Analysis, Design and Development of a Generic Framework for Power Trading

Rasmus Skovmark, s001509
Johan Holkmann Jacobsen, s001768

Supervisors:
Preben Nyeng
Bjarne Poulsen
Jacob Østergaard

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

Abstract

This project presents analysis, design and development of a generic framework for automatic real-time trading of power among distributed energy resources in the power grid.

Trading power requires the producer to assign a balance responsible entity, which has a considerable cost and often leaves power producers with small producing units unable to profit from trading. This is unfortunate since small producing units usually requires a smaller warm-up/start-up time, which makes them ideal for regulating power balance in the power grid.

This project documents the design and development of a generic framework based upon a service oriented architecture. This framework can be used by power producers of any size to automatically trade power. There is currently no such software existing although the possibilities for this kind of framework have previously been identified. This project emphasizes the modeling of the framework and the implementation of several prototypes solutions.

The prototype of the framework is implemented as a client/server structure in a service oriented architecture based upon .Net Framework 3, WCF and WPF. The server, functions as a broker agent where each producer will be able to join the framework as a client and do online trading real-time. The server makes a deal on a market outside the framework of delivering a certain amount of energy. The server now has the responsibility of making the clients produce this energy by using a power control method. Two models are designed and implemented.

The first design is based on a price signal, where the server continuously informs the clients of the current price. When more power is needed it increases the price to encourage more clients to produce and if too much power is produced the price signal is decreased to make clients stop production.

The second design is based on bids from each client. The bids determines how much power each client wants to produce and at what price. When the server needs more power all valid bids are evaluated and the cheapest are accepted until the power reaches the wanted level.

A case study of the framework prototype is performed, which demonstrates that the principles of the framework can be exploited in a useful manner. The case study is a proof of concept, and shows that the price signal model is the best method for production units that have some unpredictability in their producing plans (e.g. wind power plant). The bid model is the best model for clients with units having a consistent and reliable power production. Both models are feasible solutions and function well. The server is able to stabilize the power level with the right marginal values applied. This project does not conclude on which model is capable of creating the largest profit. The framework in this project can be used as the core for further development and deployment.

Keywords: power producers, trading power, regulating market, service oriented architecture.

Preface

This report documents the MSc project carried out by R. Skovmark and J.H. Jacobsen at Institute of Informatics and Mathematical Modeling and Ørsted•DTU, Technical University of Denmark, January 1st – August 13th 2007.

The project was supervised by PhD student Preben Nyeng, Lector Bjarne Poulsen, Professor Jacob Østergaard. We would like to thank them for valuable cooperation and feedback throughout the project.

Jacob Østergaard and Bjarne Paulsen have encouraged us to submit this project to an IEEE conference. The project has been submitted to the conference "2008 IEEE PES Transmission and Distribution Conference and Exposition" in Chicago, Il.

We would also like to thank Thomas Eriksen, Energi Danmark, Anders Houmøller, Mikael Togeby, EA Energy Analyses and Søren Stig Abildgaard, EC Power who have helped us find valuable information in our research of the power market.

August 17th 2007, DTU

Johan Holkmann Jacobsen

Rasmus Skovmark

1 Introduction

The purpose of this chapter is to introduce the reader to the thesis and the report in general. The chapter gives a short description of the background for the project, which are the possibilities of power trading and especially in a virtual power plant.
This project contains analysis, design and development of a generic framework prototype that enables automatic real-time trading among power producers. This generic framework for automated power trading will be denoted GFPT[1] throughout the report.
This chapter includes background information, project description, problem definition, solution outline, vision and mission. Finally this chapter gives an overview of each chapter in the report.

1.1 Background

In 1879 Thomas Edison invented the electric lamp and soon after he patented an electric distribution system. A generating station in Manhattan supplied 59 customers with electricity and this was be the beginning of the electric distribution system as we know it today.
The world is covered in networks of power lines. These networks make it possible to transfer energy from the producers to the end users. An example of this is electricity generated by a wind power plant, transferred through power lines to transformers and substations before finally being used for making a lamp glow in the house of a consumer.

1.2 Overview of power market

Power plants produce power (electricity). In the Nordic countries power plants operate in a joint market and trade power on the Nord Pool exchange. Power is here sold once a day in 24 one-hour slots for the next day. This market is called the spot market. Energinet.dk is the operator of the transmission system and has the responsibility for keeping the system in balance. This is done by trading power on the regulating market.
Trading power requires a balance responsible entity to handle production plans and legal issues. These tasks traditionally require human resources and trading power is therefore an option, which can only be performed profitable by producers of a certain size. This leaves many small producers (wind power plants, combined heat and power units, fuel cells, etc.) with no reasonable way to trade power for profit.
A combined heat and power plant may need to produce a lot of heat during winter time. The enhanced production of heat means that more electrical power is produced than can be consumed. This excessive power can be traded to consumers in need of power.
During summer it might be cheaper to buy power than to produce it since there is no need for heat anyway.
Smaller producers have the advantage of much faster start and stop times (warm-up period). This makes them more suitable for regulating the balance in the system. Even though the producers are small, a great number of cooperating producers would be able to make a difference. This would be possible if these producers joined a network where they appeared and acted as one virtual power plant.
Nord Pool has secured production slots for one-hour one day prior to the actual power production. These slots are based on a forecast and create an inflexible production without consideration of the actual power consumption. Ideally a power plant would start up and shut down (assuming this would be a cost-free task) as the markets fluctuate and hereby optimize its own production. One hour slots might be ideal for large production plants but in a framework of many smaller producers

[1] Generic Framework for automated Power Trading
[2] http://www.dongenergy.com/da/aktiviteter/el+og+varmeproduktion/elproduktion/asnaesvaerket.htm

trading should be done real-time between the best producers, making the system as effective and flexible as possible.
To accommodate the above requirements for flexibility the solution needs to be developed using a generic virtual interface. This enables consumers and producers of different interests and capabilities to use the same interface for trading the required information and afterwards act on this information to decide whether or not the production should occur.
Trading power could be done automatically real-time (online) using e.g. web services in service-oriented architecture. A server could function as a broker agent that gets the market price from a market outside the framework like e.g. Nord Pool. Each power plant has all necessary information about its capacity and the price it is willing to produce at. This information can be used by the server to calculate which producers that most effectively could produce power at a given time and inform these to begin production. As more power plants want to join the system, the generic architecture simply allows us to add each of them to the network, instantly being part of the server's calculations. In fact this whole system functions as one virtual power plant.

1.3 Problem definition

The purpose of this project is to analyze, design and develop a generic framework in which power producers can trade power real-time automatically (GFPT). The GFPT will be the first of its kind, and the project will emphasize the analysis of the power market and modeling of a framework prototype with some power control solutions. The GFPT will be tested in a case study to examine if such a framework will be able to optimize trade opportunities for power producers and hereby improve efficiency and economy.

1.3.1 Vision

The vision of this project is to model a state-of-the-art GFPT, which improves trade opportunities for especially small power producing units. Also vendors of producing units will be able to benefit from improved market conditions as well as overall efficiency and economy for the society will improve. The project will document the analysis of the power market and identify its problems in order to model the GFPT. This project's aim is to design and implement a prototype of the framework. A case study will demonstrate if requirements are met and if the framework can be a feasible solution.

1.4 Solution Outline

The GFPT will be based on a service oriented architecture where a server controls the automatic trading. Figure 1.1 shows the basic principle of the GFPT. The framework consists of a server and a number of clients who are power producers. The server handles all trading with external markets (Nord Pool exchange, regulating market, power suppliers etc.) by using a broker agent, who can be either a human resource or software.
Inside the framework clients join the network by using a standard interface through web services. The figure shows 2 clients with different production units, but a client could also be a consumer as well as a producer in principle. The clients use this interface to communicate with the server and evaluate how much power to produce, if any at all. The internal control of the production differentiates from client to client and cannot be created as a generic solution because the controlling of a wind power plant differentiates from the controlling of a fuel cell. Each client therefore makes an integration of their own specific controlling software, e.g. SCADA or PDA into the generic interface.

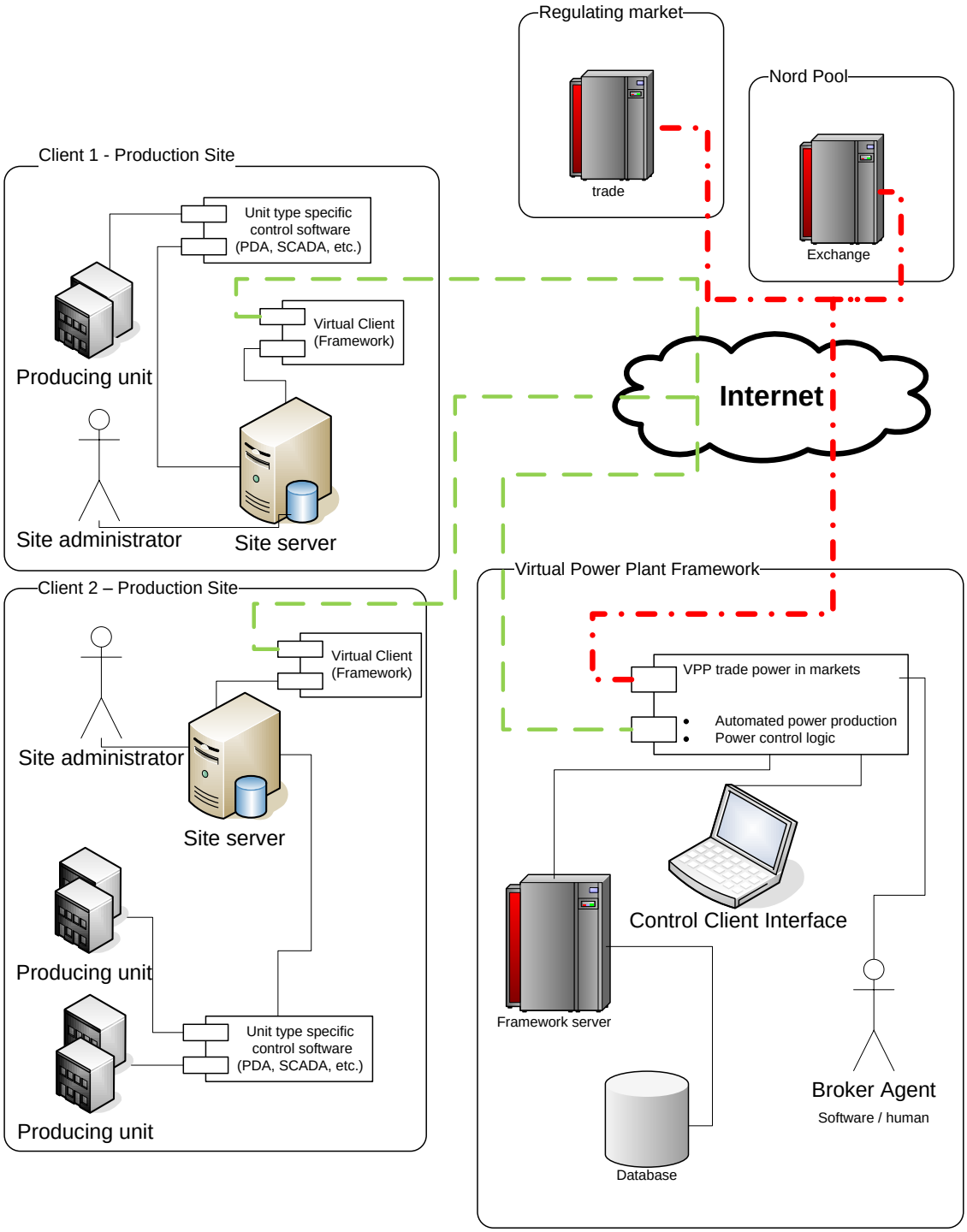


Figure 1.1 - Solutions overview

The server in the virtual power plant framework has control logic which has the task of encouraging clients to produce the right amount of energy by using a power control model. This can be done in several ways. Solutions could include using a price signal set by the server or a bidding process, where clients send bids.

1.4.1 Scope and project limitations

This project will cover analysis of the power market and possibilities for a GFPT. The design and development of the GFPT will be a prototype only and the functionality will be as simple as possible since this project emphasizes on the construction of a realistic framework rather than going into details. Further development of the GFPT will be possible in future works.
The design of the prototype will include the communication between clients and server, server side logic to control power production and user interface. The design will not include forecasting, breakdown or external market trading although these topics are described in the analysis since they all are important parts of a GFPT.

1.4.2 Technologies

This project will make use of the latest technologies in software development as for example .Net Framework 3.0 and its Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF) and Windows Workflow Foundation (WWF).
.Net Framework 3.0 is integrated in Windows Vista and it is the future development platform for the Windows platform.
WCF unifies several communication methods as e.g. Web Services into a single service-oriented model (using SOAP message as communication) for distributed programming, and it is intended for rapid application development.
WPF is a graphical feature which provides a model for building user interfaces in Windows applications. WPF is based on DirectX and basically enables better control, design and development of graphical areas of applications.
WWF integrates workflow control with Visual Studio and allows the execution as well as debugging of a workflow. Declaring the structure of a workflow is based on XML but can be expressed in any .NET programming language.
The development platform and the technologies chosen are described in detail in chapter 4.

Content of the thesis

Chapter 1: Introduction. This chapter includes an introduction to the project, problem description, vision and mission and a short solution outline.

Chapter 2: Analysis. The first part contains a thoroughly analysis of the electric distribution system and the power market, which is the domain area of this project.
The second part describes the main problems and challenges being the problems of the broker logic, forecasting of power production and handling breakdowns. It also will outline possible solutions to the problems and overall requirements for the project. The first solution is based on a price signal while the seconds uses bids from clients.

Chapter 3: Conceptual Design. This chapter include use cases, design of system architecture, flowcharts, events and classes.

Chapter 4: Technologies. The first part of this chapter gives a throughout description of service oriented architecture. Pros and cons of using service oriented architecture to implement the desired functionality will be discussed. The second part of the chapter briefly describes the chosen technologies and development techniques. That includes .Net 3 framework, WCF and WPF.

Chapter 5: The purpose of this chapter is to describe the physical design and implementation of the prototype based on the analysis in chapter 2 and the conceptual design in chapter 3. The

implementation will use technologies described in chapter 4. The framework will support multiple module based production controlling methods where the price signal model and the bid control model will be implemented in this prototype. The chapter describes the events and classes, database design and shows flowcharts and state diagrams.

Chapter 6: Case Studies. This chapter describes the case studies in which the prototype is tested by simulating a number of clients. Each power control model is in particular tested for performance and constancy when stabilizing the power production.

Chapter 7: Conclusion. This chapter describes which parts of the project that succeeded and what was learned.

Appendix: The appendixes include source code, installation notes and a list of sources.

2 Analysis

The purpose of this chapter is to describe analysis of the domain area, power market and problems of the GFPT, solution outlines and requirement specification.
This chapter is divided in 2 parts. The first part contains a thoroughly analysis of the electric distribution system and the power market, which is the domain area of this project.
The second part describes the problems and challenges of the GFPT server logic, forecasting of power production and breakdown handling. Also an outline of 2 solutions to the power control problem will be described. The analysis results in a requirement specification.

2.1 Analysis of Domain Area

The development of the GFPT requires knowledge of the domain in which the framework should function. The purpose of this part is to thoroughly analyze the domain area and describe the electrical distribution network and the power market. This will supply the background information necessary to fully understand the interests of this project. The description is based on conditions in the Nordic power market but the generic framework should be a universal solution that could be applied all over the world or inside embedded sub systems.

2.1.1 Definition of power and energy

In this project electrical power production will use some terms that needs to be clarified. In particular:

- Power
- Energy

Power is measured by watt (W), while energy (amount of energy) traditionally in electrical power business is measured by watt-hours (1000 Wh = 1 kWh). The International system of units however measure energy in joule, which is equivalent to 1 watt-second.
Common terminology adds to the confusion since the word power has several meanings and is also used in many ways (incorrect, one might argue) to denote some part of electrical energy. For example power market, power production, etc.

2.1.2 Power market players

Many different players are connected in some way to the electrical distribution system and power market. These players are a part of the model and it is therefore necessary to define which roles each player carries out. A player has often more than one role (e.g. the owner of a combined heat and power unit may be a producer as well as an end user). In Figure 2.1 and Figure 2.1 relationships between players are illustrated in rich pictures.
Figure 2.1 shows how power in a physical form is distributed among the players. The power producers produce power which is transported to the power grid. The power grid has a high voltage transmission net which is maintained by the transmission system operator, while local distribution nets are maintained by a distribution network company. The end user spends power while physical unbalance in the grid is regulated by the transmission system operator.

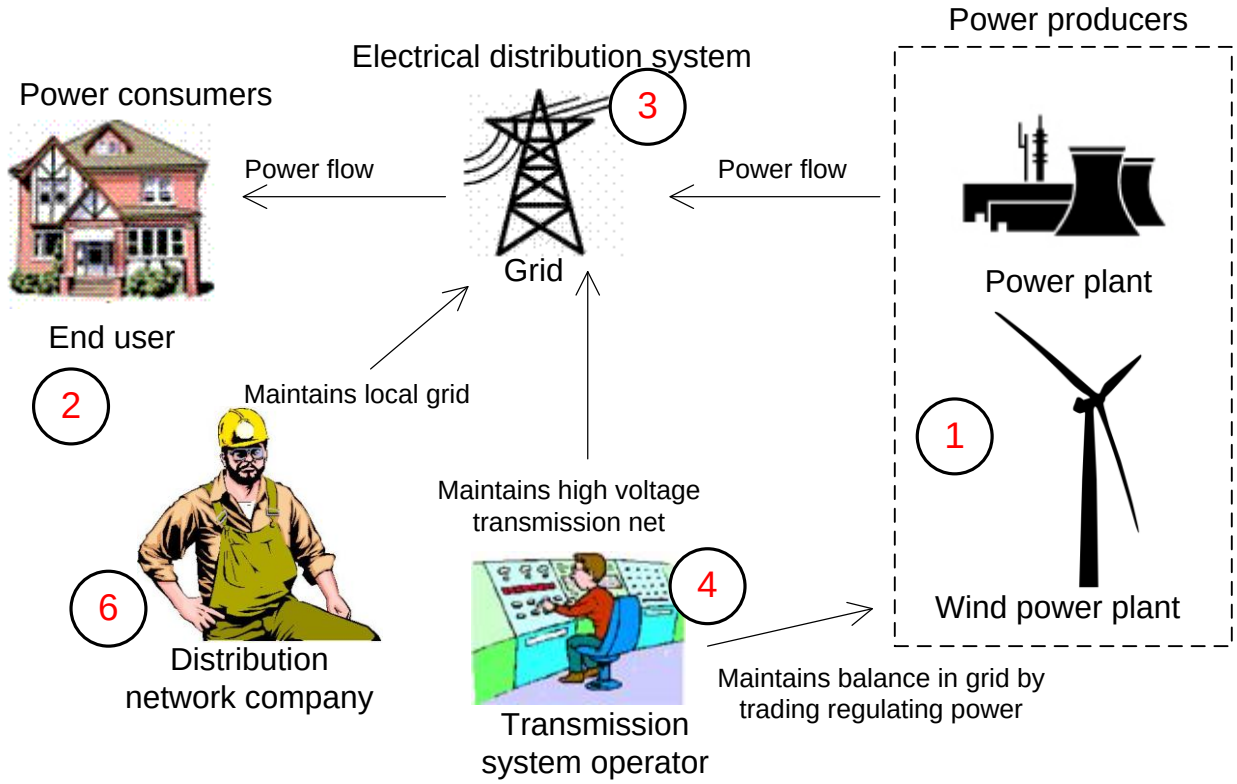


Figure 2.1 Shows physical flow of power and how players interact in the domain area.

In Figure 2.2 the financial flow of power is depicted. Power producers sell their power to markets or in a bilateral agreement with a power supplier. The power supplier sells the power to the end user.
All producers and consumers must have a balance responsible entity who delivers plans for power production and consuming to the transmission system operator. If the plan fails the balance responsible is financial responsible.
Each player in the power markets will be described in the following sections. Numbers in the headlines correspond to the numbers in Figure 2.1 and Figure 2.2. Based on the problem definition in chapter one a GFPT is sought to be developed. In Figure 2.2 the red circle collects the players that could be involved in the GFPT.

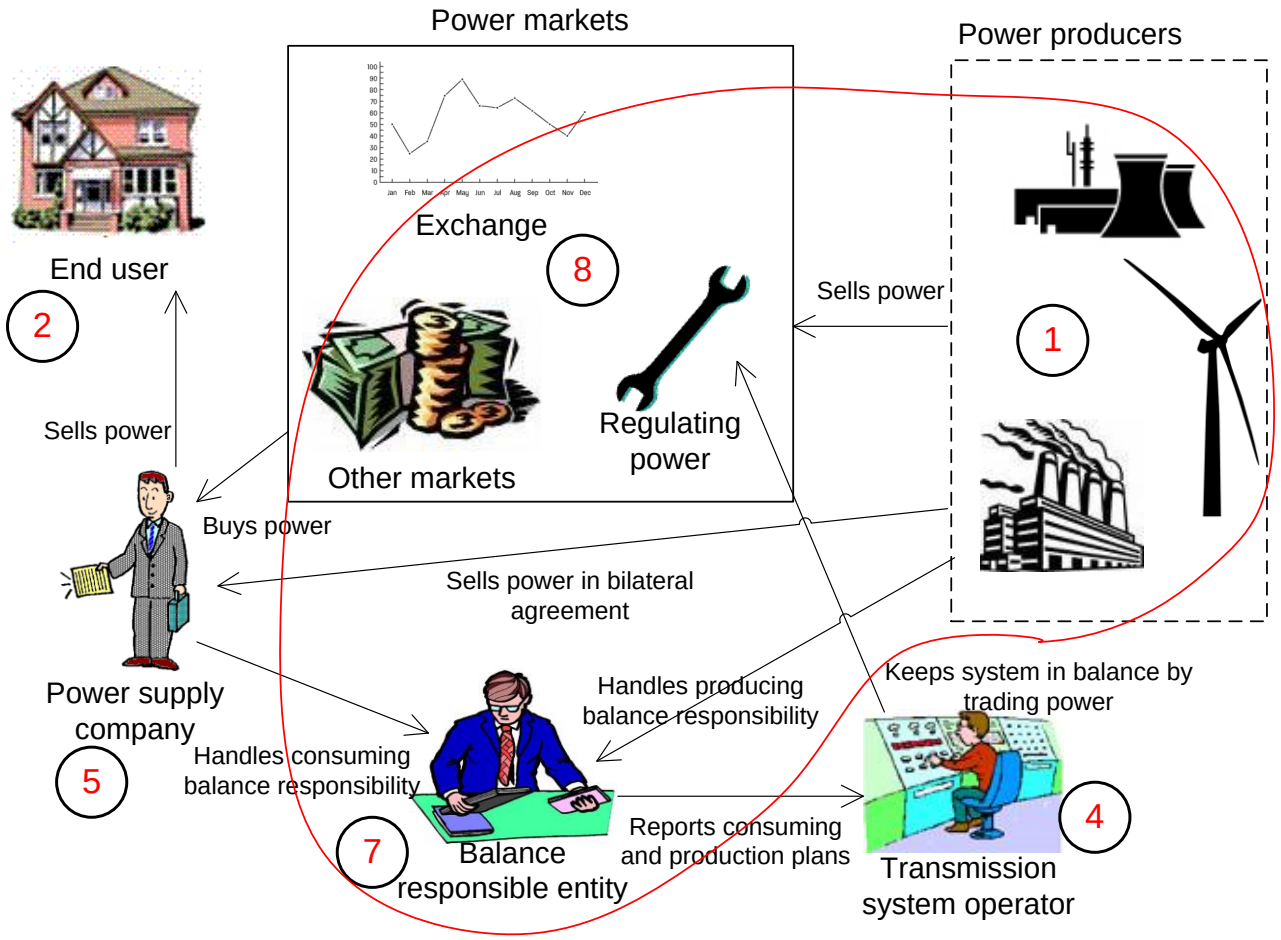


Figure 2.2 Shows economical flow when players interact in the domain area. The red circle gathers the players that are involved in the GFPT.

2.1.3 Power Producers (1)

The producer is an entity that produces power, and then either uses the power himself or sells it. Examples of producers are: A wind power plant, a housing co-operative with a heat- and power unit or a fossil fuel power plant.

Most important to the electrical distribution network system are the large power plants, which produce 60% of power in the network. In Denmark many of these power plants use coal or gas as fuel and are combined heat and power plants. In Denmark the company Dong Energy and Vattenfall own the largest power plants. The largest power plant in Denmark is Asnæs Power Station capable of producing 1057 MW[2].

Some of the power producers which could benefit from using a GFPT are:

- Combined heat and power units
- Units based on renewable energy resource with a unreliable production

Combined heat and power units

Combined heat and power units are common as supply source for the electrical distribution network and smaller units for e.g. industries, some local residence areas, which uses district heating and as

[2] http://www.dongenergy.com/da/aktiviteter/el+og+varmeproduktion/elproduktion/asnaesvaerket.htm

an emergency energy source at hospitals. Their advantage is that at certain times power and heat can be produced cheaper than the cost of buying it from a supplier.
There are many types of units. Large power plants can adjust their production to specific needs. In this project focus is on smaller local units.
For smaller units the relationship between heat and electrical power production of the plant is dimensioned to fit the specific needs for the users of this particular unit and can in general not be changed afterwards. Producing heat will always produce electrical power as well and vice versa. Combined heat and power units can profit from producing power. Especially during wintertime when it is more likely to be cold and an increased heat production may be necessary. This might mean that there is produced more electrical power than the producer can use himself. Selling power to the distribution network does however require the use of a balance responsible player, which often results in too many expenses for a producer to be able to profit from such a sale.
The smaller units also have the advantage that they can start and stop production within a short time with few expenses. That makes them very useful for regulating imbalance in the network.
Figure 2.3 shows a picture of a 13 kW combined heat and power unit from the manufacturer EC Power and a 75 kW wind power plant from Vestas.

Figure 2.3 - ***A 13 kW EC Power combined heat and power unit and a 75 kW Vestas wind power plant.***

Renewable energy

Renewable energy is power derived from sources, which are able to either regenerate or are sustainable. Wind power plants, hydroelectric dams and solar power plants are all units, which use renewable energy to produce power. The downside of this production is that these power plants often dependent on weather or other outside factors. Wind power plants can only produce power when the wind is blowing, solar power plants depend on the sun and tidal power units produce power in line with the tide.
Forecasting plays an important role for planning power production in these plants. Unlike combined heat and power units, which to some extend can plan their production to optimize their profit a wind power plant only has the opportunity to produce when there is wind. Political forces encourage an increase in renewable energy production to protect the environment.
Another factor regarding the plants based on renewable energy is the fact that the outside factors in general follow all plants at the same time, at least for the same geographical area. When a production is feasible because of strong wind or sun, the production of the neighbor's plants will also be feasible, increasing the general power supply and lowering the price of power, according to the general rules of supply and demand.

2.1.4 End users (2)

The end user is the player that consumes electricity. The electricity can have been produced by the end user himself or bought from a power supplier. A typical end user could be a household, a business or anything that consume electrical power. This means that most power producers also are consumers. A power plant for example uses power to lighting, control devices etc.

2.1.5 Electrical Distribution System (3)

The electrical distribution system consists of producers, end users and the actual network connecting producers and end users physically to each other. In Denmark the network have 4 different grids:

- High voltage transmission network (400 kV)
- Transmission network (150 kV in West Denmark and 132 kV in East Denmark)
- Distribution network (50 kV)
- Local distribution network (230-400 V)

The transmission system operator in Denmark, Energinet.dk, owns the high voltage network. This grid connects the largest power plants and the different parts of the country, see Figure 2.4. Also high voltage cables connect Denmark with other countries. In transformer stations power is distributed to smaller electrical networks, which interconnects the end users.

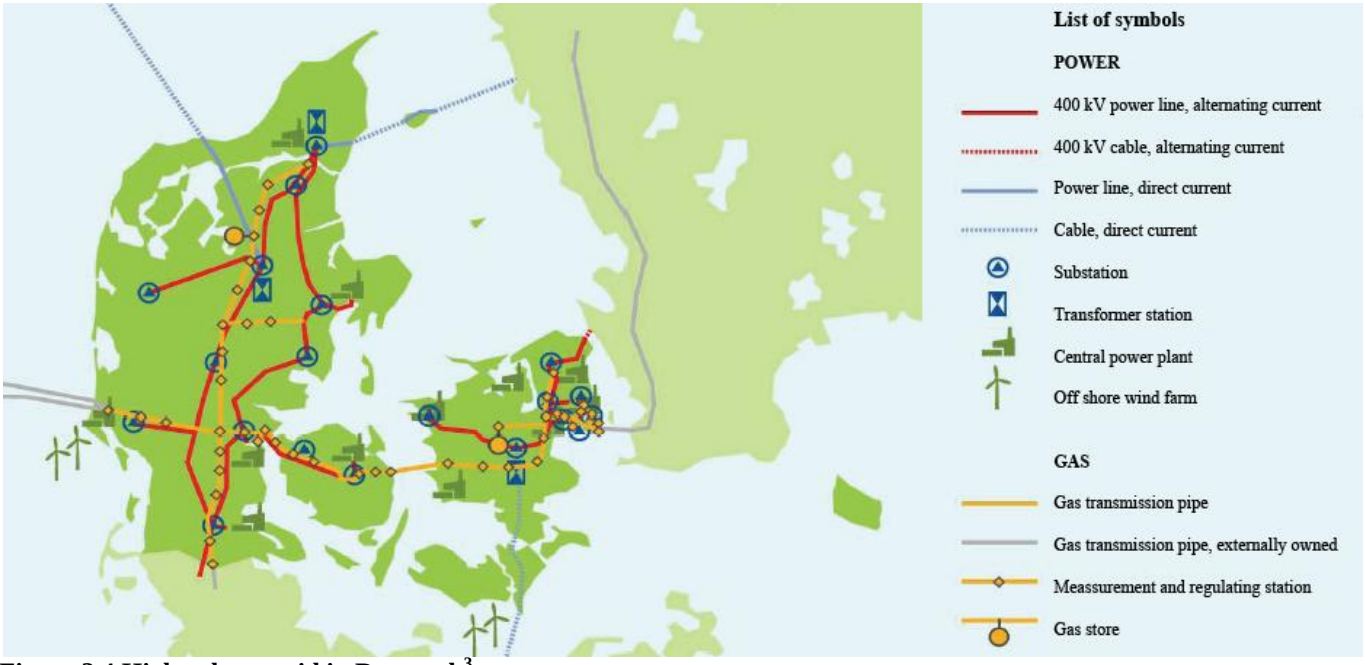


Figure 2.4 High voltage grid in Denmark[3]

2.1.6 Transmission System Operator (4)

Transmission System Operator (TSO) is the company with the overall responsibility for making sure the system is functioning and in balance at any time. In Denmark this company is Energinet.dk. If there is an unbalance in the system the TSO must regulate this by buying or selling power on a special market for this called the regulating market. The TSO makes deals with power plants to be on standby for producing or end current production in case it is needed. The producer is obliged to send in a price bid for a certain amount with regularly intervals and must be able to start a

[3] www.energinet.dk

production no later than 15 minutes after the TSO accepts a bid. The TSO can however also accept bids from producers who are not on standby.
The TSO supervises the electrical distribution system but also receives daily production and consuming plans from the balance responsible players, which it then approves (or do not accept). Detailed schedules and statistics helps keeping up the balance.

2.1.7 Power supplier (5)

The power supplier is typically the company, which sells electricity to the end user, but may also be a company, which alone trade power in wholesale in the exchange or in bilateral agreements with the producer or another power supplier. The power supplier (if selling to end users) may forecast how much power its customers supposedly will use.

2.1.8 Distribution Network Company (6)

A distribution net company maintains part of the distribution network. The company also makes all necessary measurements. All measurements are sent to the TSO to help keeping the system in balance. Also measurements are used for settling accounts.

2.1.9 Balance Responsible Entity (7)

One of the most important principles in the electrical distribution network is that the system is always functioning and the supply is available to all part of the network. Within recent time several large breakdowns has occurred in both New York and eastern Denmark – leaving big cities without power.
The system must always be in balance. This means that production and import of power must equalize the demand and export of power. This is relevant both on a national level as well as with each individual player in the electrical distribution network. If a producer, customer, power supplier or other player does not want the balance responsibility himself it must be given to another entity.
In real life smaller end users only spend power and have no power production themselves. Today balance responsibility often lies with different power suppliers, to whom the end user pays a fee for each spent kWh.
On a larger scale the TSO has the responsibility of maintaining a stable electric power system. If unbalance occur in frequency or voltage, or a breakdown in a power plant occurs the TSO reacts by buying or selling power in the regulating market.
The balance responsible player is an entity that has the financial responsibility for keeping a part of the system in balance. There is always a balance responsible player whether its production, consuming or trading.
The balance responsible player each day sends production or consuming plans for the following day to the TSO. This plan should be balanced. If the plan for some reason does not work out (e.g. lack of wind for a wind power plan) there is an unbalance in the system. The TSO handles the physical unbalance, but since the balance responsible player has the financial responsibility accounts are settled by measurements from an independent trusted third part.
The BRP (Balance Responsible of Production) is an entity that handles plans for production. In Denmark, Energi Danmark is certified BRPs and is a major player in the Danish market. Energi Danmark has lots of clients for whom they function as BRP. However the smallest producer is 150kW. This is the minimum size of a producing unit[4], where it is possible to make a reasonable profit when expenses for the BRP are taken into consideration.

[4] Conversation with Thomas Eriksen, Energi Danmark

2.1.10 Power market (8)

In many ways the power market acts like any other market, power is traded like any commercial product. The obvious benefits of the market are also the same. Producers can sell their supply (of power) while consumers can purchase power and thus satisfy their demand.
In the Nordic power market all balance responsible companies must each afternoon deliver plans for production, consuming and trading for the next day to the TSO. When the plans are approved the TSO takes over the responsibility of keeping the system in balance the next day (operating day).
Figure 2.5 shows a graph of the price of power (DKK/MWh) on Nord Pool Spot market throughout 48 hours. The figure shows that the price for producing power between 17.00 and 19.00 is very high compared to the rest of the day in East Denmark and Germany (Kontek), while very low in Sweden. That is because there is a large demand for power at this time because people are using electrical devices for cooking, watching TV, etc, and though power production is large enough to cover demand in Sweden , the power cable connection to East Denmark cannot transfer enough power to satisfy the needs of this area. This problem is called a bottleneck and that is why prices are very high some places and low in other places.
If unbalance occurs the TSO can take action by buying or selling power on the regulating market.
After the operating day measurements from production and consuming are compared with the original plans. Sometimes the plan does not match the actual production and consuming. There can be many different reasons for this, breakdowns, inaccurate forecasts, etc. The difference in the plan is settled financially with the TSO.

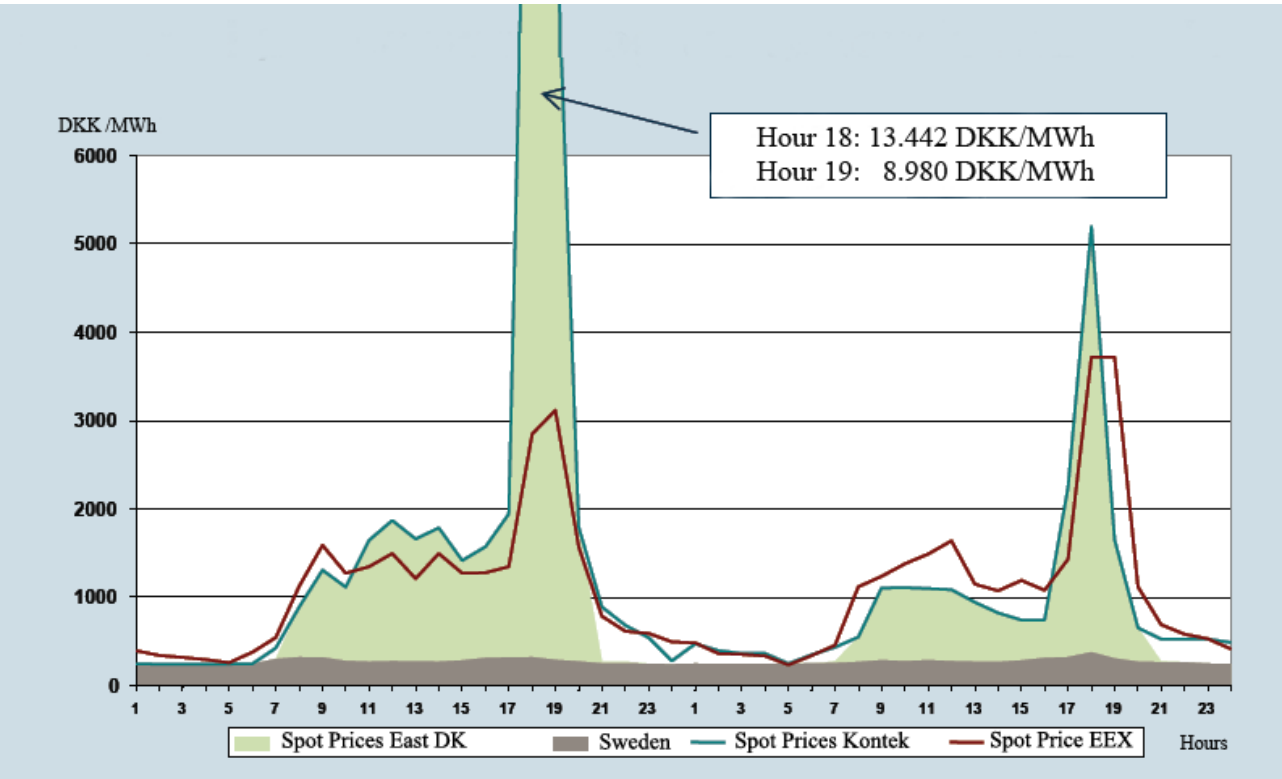


Figure 2.5 Price of power (DKK/MWh) on Nord Pool Spot market throughout 48 hours. This figure shows that due to power line bottlenecks between land areas the price for producing power between 17.00 and 19.00 is very high in east Denmark and Germany (Kontek), while low in Sweden[5].

[5] http://www.energinet.dk

There are many different reasons for trading power. Most of all producers want to profit from selling power. An excessive production of power can be sold.
Power can be traded either on an exchange like Nord Pool or in the regulating market.
When a producer trades power a balance responsible player must be assigned. The fee for this player is of such a figure that many smaller power plants cannot afford to trade power. A 2 mill kr. deposit is required by the transmission system operator from the balance responsible player. The price a producer should pay for having Energi Danmark as the balance responsible player is an entrance fee of 25,000 kr. and then an annual fee of 100,000 kr. [6] This amount means that a producer must have a relatively large production, which outmatches these fees to be able to profit from using the exchange. The producer also has the opportunity to trade power in bilateral agreements with consumers or power suppliers or wholesale merchants, but assigning a balance responsible player is still mandatory.

Nord Pool

Nord Pool is the Nordic exchange for trading power owned by the Nordic TSOs and about 700 Danish producers trade at Nord Pool through their balance responsible player. The exchange has 2 markets Elspot and Elbas.
In the Elspot market power is traded for the following day (operating day) in slots of 1 hour. All players must give a bid before 12:00. All bids from producers and consumers are calculated by Nord Pool to ensure the balance. The result is the system price. If there are bottlenecks on the high voltage grid between the different regions of Nord Pool (Finland, Sweden, Norway, Denmark East, Denmark West and Northern Germany) a price will be calculated for each region influenced by bottlenecks.
Nord Pool operates with a double bidding concept which means that all entities that want to either sell or buy power gives a bid for an amount of energy and a price. All bids are listed and matched and the highest accepted bid is then defined as the market price, which is the price that all bids then are traded for.
At 13:00 Nord Pool announces all trades by amount and price for the operating day.
Elbas is a short-term market on which trades can be made until one hour before the operating hour. The purpose of this market is to allow players to trade in case of breakdowns or poor forecasts to obtain balance.

Regulating Market

The TSO must always keep the system in balance. If unbalance occurs the TSO must buy or sell regulating power. This is done in the regulating market, which functions by bids offered by each producer. There is a min and max price and amount (between 10 and 50 MW) for bids. The producer must be able to start production no later than 15 minutes after a bid has been accepted.
The regulating market in the Nordic countries is an IT-platform called Nordic Operational Information System. There are two ways of trading here.

- The producer (or consumer) makes a deal with the TSO for the following month to be standing by for regulating power and is paid a fee for doing so. A bid must be offered each hour during the period.
- The producer has made no deal but simply offers a bid. If the bid is not accepted the producer receives no payment.

[6] http://www.energinet.dk

Smaller power plants are very suitable for regulating power because their ability to start and stop within a short amount of time. However a small unit of 15 kW does not regulate much power. The minimum is 10 MW to function in the regulating market. This means a lot of small units (maybe 10s of thousands) must join a virtual power plant if they must be able to make a difference. Another option could be to divide a region into smaller local areas each balanced regulating power from local virtual power plants.

2.1.11 Other Projects

A research project in the southern part of Denmark is interesting for the development of power supply. The project supplies all households in a large region with differentiated energy bills, which means that the price of power changes for each hour and each household has the choice of spending energy on the cheapest time of day. Each night a special energy-broadcast will appear on TV (like the weather forecast) and the customers can see when power prices are expected to be low the next day. Turning on the dishwasher or doing laundry at nighttime will suddenly affect the household economy.
The national power supply Energinet.dk is the founder of the project and it will run for 2 years from spring 2007. Energinet.dk is interested in finding out if a price-flexible power market will help the customers spend energy when there are lots of it and save energy when there are not. If this is the case it will be an advantage for both the economy and the environment.

2.1.12 Players with interests in GFPT

Many players have an interest in the framework.

- Society will benefit from a more efficient power production socio-economically
- Power producers get opportunity to trade real-time instead of one hour slots.
- Smaller producers will be able to trade power.
- Balance responsible will be able to use smaller fast-starting producers to even out an unbalance faster than normally.
- The vendors of smaller power producing units will be able to sell more units because their possibilities of joining this framework and thus making a profit.
- The owner of the framework will most likely want to profit from a production.

The framework's interests change depending on who owns it.
If an independent company owns the framework, it will be run commercially trying to maximize a profit.
If the framework is owned by a group of cooperating producers they most likely want to split the profit in some fair way, but each producer also wants to profit the most from his own production. This leaves some conflicting interests since the producer wants to get paid the highest possible price for producing.
If the system is owned by the TSO with a no-profit-policy then the interest will probably be keeping the system in balance rather than a profit.

2.1.13 Concluding remarks

Balance of the electrical distribution system is very important, but the market could be both more effective and faster responding if smaller production units were used. These units however have difficulties trading their power because of the expenses of using a balance responsible player. Virtual power plants containing many smaller plants could be an effective way of sharing expenses and common tasks. The distribution system could obtain an even more effective and faster

regulating ability if trading could be done automatically in real-time. The type of producers in the network might be wind power plants, combined heat and power units or any other type of producer.

In the analysis of the domain area the main challenges of the power market has been identified. The problems and challenges will be investigated according to a possible GFPT solution based on the values in the projects vision.

The main issues that should be solved with a GFPT are the following:

- Enabling small producers to trade power in a virtual power plant
- Automatic trade of power to minimize human resources.
- Real-time trade of power optimizing balance regulating.
- Controlling power production.

2.2 Problems and solutions

This part describes how a GFPT can be a solution to the issues that was found by the analysis of the domain area. In the light of these problems this section breaks each problem down into several parts and these are each investigated and analyzed as also all aspects of a GFPT are. The result should be a description of possible solutions with pros and cons. Finally 2 different solutions to controlling power level in a GFPT is described in details in order for it to function as a starting point of requirement specification and conceptual design in the next chapter.

2.2.1 Real-time and automatic trading

One of the problems in the power market is that trading cannot be done in real time. As described earlier in section 1.2, the Nord Pool exchange has secured slots for one-hour one day ahead. The one hour slots in a future solution should ideally disappear in a manner that would make the exchange between the best producers as flexible as possible.
So far trading has been done manually by a judicial entity with this responsibility. Software can be used to implement automatic trading. This might eliminate human resources from the actual trading functionality and decrease human errors, but also optimize production. An example is a wind power plant where changing weather makes production difficult to plan. The wind might be blowing for 5 minutes followed by 5 minutes of calmer winds. These production peaks could be sold in an automatic trading system.

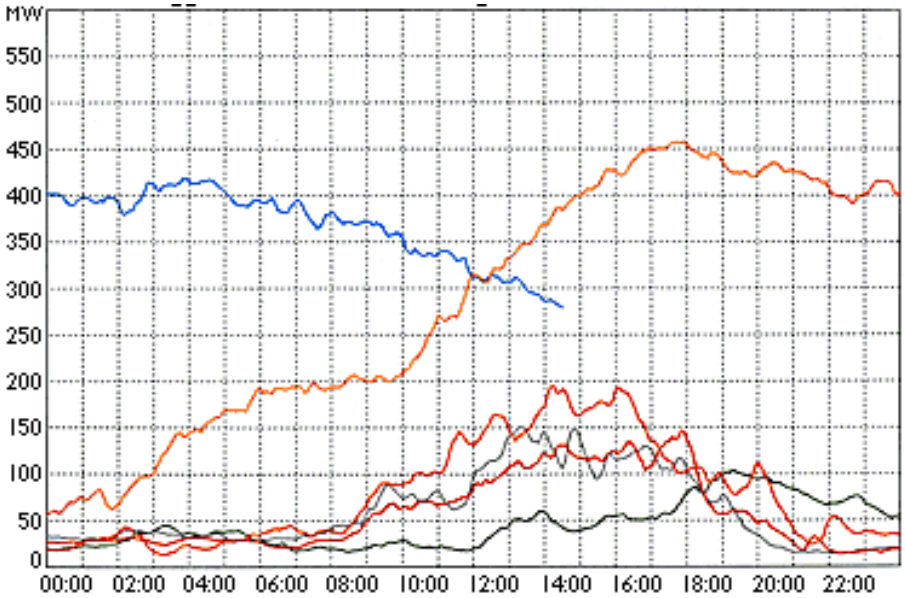


Figure 2.6 The graph shows power output from all wind power plants in West Denmark in June 1997. Each curve represents one day of production. It appears that 2 of the days (blue and orange) there were a significant wind while the 4 days were relatively calm[7].

Figure 2.6 illustrates a typical week in Denmark and shows that variations in power production by wind power plants are common. The figure might suggest that production from wind power plants in general matches consumption of power. During the night the wind is typically calm compared to daytime, where consumption of power is larger due to the fact that people are awake and using

[7] www.windpower.org

power and also the wheels of the industries are turning. This is not necessarily true. Weather forecast reference states that Wind often increases at night[8].

2.2.2 Politics & Legislations

Danish legislation has been changed rapidly through the last decade to ensure that all power producers can trade power in the free market. The Danish Energy Authority stated that this new legislation would also have a positive impact on wind power production. Legislation also favors power plants using renewable energy, providing extra benefits and reduced fees[9].
All though there seems to be greatly dived opinions on the energy plan published by the Danish government 2007 [9,10], mostly from the opposing political parties, it is easily concluded that wind power and other renewable energy is thought to play a more important role in the power supply the next 10-20 years. The Danish Social Democrats energy plan publish in 2006 states that renewable energy must be 80% of the power supply by 2025 [8].
This all contributes to the fact that there will be a greater need of supplying power when it is needed instead of making large reserves and that makes it more lucrative for a project like this, which seeks to make real-time automatically trading possible.

2.2.3 Combined heat and power production in the future

In 2006 DGC made an analysis of the capability of combined heat and power production in Denmark in the future. Denmark is already world leading when it comes to district heating, where 80% of the heat is from a combined heat and power production. The analysis also investigates the possibilities of placing a unit (e.g. 1 kW) in each home with the capability of producing both heat and power. The analysis concludes that there is a significant potential of 2000 MW electric power. Placing a unit in each home would help reducing power-fallouts and could be used as regulating balance. However problems exist at this time around the circumstances for settling of accounts and the mandatory use of a balance responsible of production[10].
It gives this project a new perspective when taken the potential for power producing units into consideration. How could a generic framework handle 1,000, 10,000 or maybe 100,000 producers in a virtual power plant?

2.2.4 Service oriented architecture

Trading power could be done e.g. by using web services in a service-oriented architecture.
The framework has a server and a number of clients. The server functions as a controlling instance and a broker agent and is the interface to the outside world.
Each power plant is a client and has all necessary information on its own power production and the price it is willing to produce at.
Web services are used as communication between server and clients. As more power plants want to join the system, the generic architecture simply allows us to add each of them to the system. This whole system functions as a virtual power plant itself for Nord Pool.
The framework in this project will function with few clients but as more clients join the possibilities of making greater profit improves. Expenses of deploying and maintaining the framework are split between more clients. Keeping the system in balance is important and with many producers it is more likely that someone is willing to produce at a certain price. Forecasting of production planning based on statistics will also be more accurate in a framework with many clients.

[8] http://www.windpower.org/en/tour/grid/index.htm
[9] http://www.dkvind.dk/nyheder/foer_2005/ny_elforsyningslov_kort.htm
[10] Kraftvarme NYT, August 2006, p. 21-24

2.2.5 Client

A client in this project is the owner of one production site. Each client can have one or more producing units or be a virtual power plant itself. The client has a communication interface to the server and a communication interface to the production units to ensure that once an automatic trade has been done the production actually starts.
The client has framework software which contains generic variables that ensure that trading can be done automatically. This can be simple if only one type of units is used in the GFPT or very comprehensive if many different units (wind power plants, solar cells, fuel cells, combined heat and power plants) are used. The software in this project must be able to handle all kind of units. The most basic variables that should be used for trading power must therefore be set by a custom-made program in each client that handles communication between the framework-part and the unit-part. These variables could be e.g. producing power level, wanted minimum price for production and startup delay of production.

2.2.6 Control and decision making

It is possible to allow the server to know everything about each client. A logic module in the server could check all production details from each client and find out which clients are cheapest in production and start these. This would be fine if all clients had the same owner. The most effective production would always be at hand and control of all producing units could be done from the server. This solution though has some disadvantages.
The server must have information about communication methods for all possible production units and how they work. Information about both the different types of production units and about each vendor's machines must also be available. This is a severe challenge to both implementing and updating the software.
Also many situations could occur where the producing unit should not do as the server logic expects. E.g. if water in a heat and power plant is so warm that production is impossible.
Most importantly the framework in this project is most likely to work in a network where each producing unit has different owners. For independent producers information about the producing units and business should be confidential to avoid other interests in exploiting any possible weaknesses. The control of each unit should also be handled by the client alone. In the end all acceptances of a trade should be made by the client itself and never the server.
In this project the clients in the framework will be considered independent and therefore all information in the client is confidential and unknown to the server.

2.2.7 Server

The server is the vital part of this framework. It is the component that communicates with each client and if necessary the outside world. The framework could basically live on its own if both producers and consumers exist among the clients and if their production and consumption of power equalize each other, or if production/consuming could be regulated to fit the opposite. However the framework is meant to communicate with a source outside the framework like e.g. Nord Pool, but it could also be any other exchange, the regulating market, consumers or making bilateral agreements with power suppliers. The server could make deals or trade power and then at the same time or delayed, profit on this trade by using the clients production.
Figure 2.7 shows how the GFPT is thought to be divided into several modules each with their tasks to perform. The server has some logic to control production, a user control interface to administrate the virtual power plant as the framework is and some communication layer. A GFPT would also need some forecasting logic to base its production and external trading on.

The PowerService module handles communication within the framework. The server logic is the logic that controls that a certain production level is maintained within the framework by trading power with clients. The control interface is the tool an administrator uses to set up and control the framework. In principle this module could be automatic but in the prototype that will be designed in this project the interface will help the user to quickly change variables, which is useful when testing different production situations.
The forecast logic makes forecasts of what level of power production the entire framework will produce in the future. The external market service is the software or human resource that trade with markets outside the framework. If the clients also include consumers as well as producers this module might not be necessary since the server logic in that case could maintain a self sufficient framework. These modules will not be implemented in the prototype but will be described later in this chapter as low-priority problems.

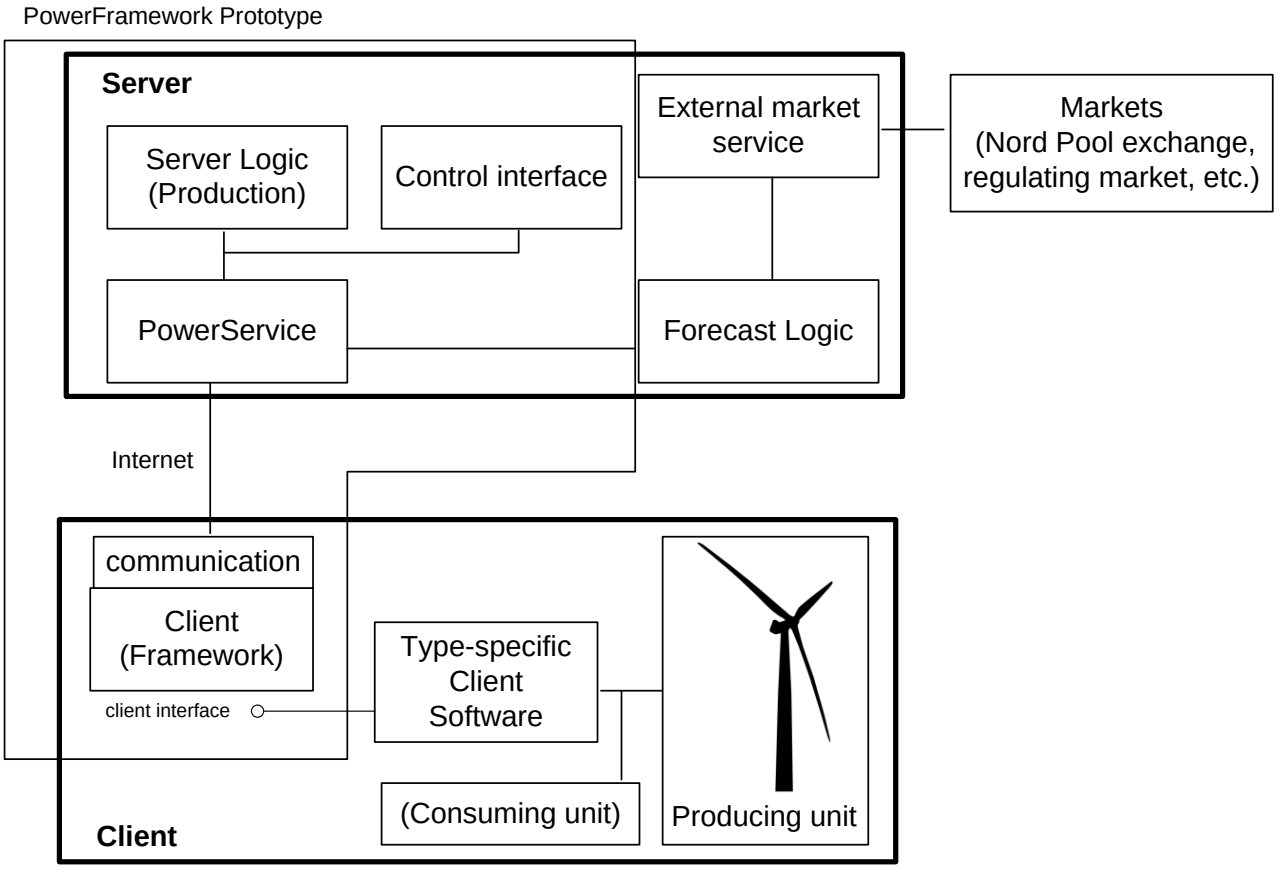


Figure 2.7 - Listing of the GFPT components.

The Client has parts of the framework software installed. The client module is the same for all clients but can be set up differently to match the client type. The clients need a type-specific piece of software to connect the producing units to the generic framework. The client could just as well have consuming units, but this is not implemented in the prototype in this project.

Communication

The server has communication layers for clients and for the outside world. Many different communication methods could be used. In this project web services will be used as method

communication between server and clients. Communications between outside world and the server is out of scope and will only exist as a model. Web service could be appropriate to use here as well.

Server logic

The server has several parts, which performs certain logic functions in the framework.
The forecast part describes how much profitable production the clients are going to deliver at any time. The forecast might include weather forecasts (wind power plans), statistics, data from the clients or other methods. The power that a server trades with the outside world should ideally be a result of the forecast.
The broker agent makes sure that all the deals made with an external source are kept. This could be a deal on Nord Pool Elspot for 1 hour of 50MW. The broker agent would then have to tell the servers control logic to trade from clients within that hour until 50MWh is reached.

2.2.8 Economy & Real Time (Online)

Trading real-time is practically impossible. First of all there is a certain time of communication between server and client. This time is not defined. Because it is unknown how long time a communication package is to travel through the communication link (Internet). More importantly a producer must be paid a price for the amount, which is produced. The amount is measured in e.g. kWh, in other words a certain power value (W) over a period of time (h). Because the function of the price might not be continuingly it cannot be integrated over time to find the amount the producer should be paid. Figure 2.8 shows the difference between trading in one-minute slots or in 10 seconds slots. It illustrates the fact that to be able to receive payment for a production in a given time there cannot be such thing as real-time trading. The price is not a continuing curve and cannot be integrated without loss of information. Thus the GFPT must use timeslots.

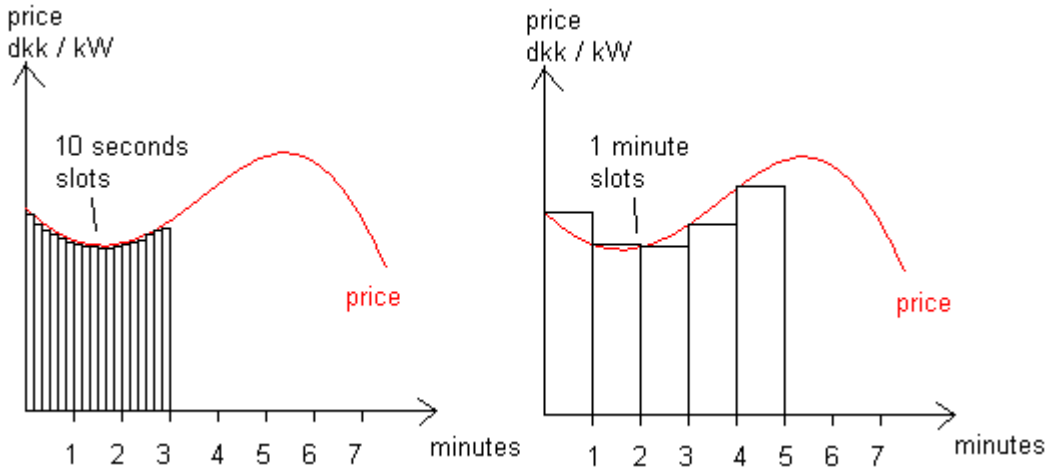


Figure 2.8 illustrates the fact that to be able to receive payment for a production in a given time there cannot be such thing as real-time trading. The price is not a continuing curve and cannot be integrated without loss of information.

2.2.9 Measured production

When a producer has made a deal with the server on how much should be produced the producer should by default be obligated to follow this plan. There are many different reasons for the producer not being able to keep to the plan. The producer should if possible give the server as much information as possible regarding how much has been produced at anytime. The distribution network company will be used as a trusted source of measurements and will at a later time inform

both producer and server of the exact amount of power actually produced. If one part broke the agreement of producing too little or too much a compensation (or reward) of some sort should probably be given. The producer will typically be hooked up directly to the distribution net and the server will have no real-time data of the client's production other than what the client tells the server itself.

2.2.10 Profit / loss splitting

The broker agent in the server trades power outside the framework and afterwards the agent must make sure that the servers' control logic and clients inside the framework fulfill these agreements. If the agent makes a contract to deliver 10MW within the next hour, the clients have to produce 10MWh. The framework should be as effective as possible, which often means that the clients, who can produce power at the lowest price, are in fact the ones that should produce. This will result in a financial result, giving the best profit. Occasionally it might be impossible to make a profit and thus the framework suffer a loss instead.
Profit and loss is by default the responsibility of the framework owner. If the clients share the framework ownership, the profit should be split. If the framework is owned by an independent company some parts of the profit could be split between clients and the rest could be given to the owner.
A loss could also be split between the clients or covered by a fund for this purpose. Loss throughout a longer period should not be able to occur. That would make it a bad business to run the framework at all. The broker agent trades with markets outside the framework and has the responsibility for making intelligent trade decisions based on forecasting, which will ensure a profit. In other words if it is not possible for any clients to produce power cheaper than the price the framework is selling the power to an external customer, then no trading should be done.

2.2.11 Profit split

There are several ways that the profit can be split between the clients:

1) Equal to the size of the clients ownership share of the company (like a stock)
2) Equal to the client's share of total power production throughout e.g. a year
3) Equal to the client's share of profit made in the specific periods when the client have produced.

It might also be a combination of one or more of the above. This is a political decision but influence how clients decide to produce.
If a client knows that his profit will be equal to the difference between the price he is paid and the price the external customer has paid the framework, then the client would be encouraged to produce in peak hours (evening) when the price often is high.
This is a problem which is outside the scope of this project. Economical research and analyses can be made to optimize how this is best performed in a framework. It might depend on many different data as types of producers, number of clients, geographical locations, legislation, etc.

2.2.12 Payment of clients

When a client decides to produce he gets an immediate price for the power. If more clients produce they might have given different prices if price bids are used. Here is a new problem. Should a client be paid the amount he has requested in his bid or the price which is given by the client with the highest accepted price (market price)? This is further looked into in the description of the price bidding model.

2.2.13 Power control and broker solutions

The broker agent is the software or human resource in the framework, which has the task of optimizing the production and profit. The framework has these important tasks:

1) Forecast production in order to make a profitable trade with markets outside the framework.
2) Control the production inside the framework to make the largest profit possible.

The power control can be constructed in many different ways. One way is to construct a price signal, which goes up and down making it favorable or not for producers to produce power at certain time. Another solution is to use bids from clients to decide who can produce. A third option is for the broker agent to single-handed decide who can produce.
All these models can also be used in one solution. For some clients a price signal may be the best and for other it may be the bid solution.

2.2.14 Solution models for power control

The control of power level in the GFPT can be designed in many ways. In the following sections 2 power control models will be described in details. The 2 solutions are based on the analysis in the previous sections.

1) Price signal model
2) Bid model

The purpose of the 2 solutions is to analyze how 2 different power control models would affect the GFPT functionality.
The goal is to find out in which cases each model is best to use. The models will also be a starting point for the design and development of the prototype. A case study will be undertaken to test if the 2 models in the prototype function and behave as expected in this analysis.

2.2.15 Solution model 1: Price signal

The price signal model is based on the following design:

1. The server sets the price for producing and clients then decide themselves if they want to produce (if it profitable for them).
2. The server regulates the price signal
    a. If too few clients produce, the server increases the price to encourage producing.
    b. If too many clients produce, the server decreases the price to try and make clients stop their production.

Signal Behavior

The following example shows how the server regulates the price signal.

Need for higher price signal		
Current price signal = 1 Current production = 10 kW Wanted production = 15 kW		
Client	Min. Price	Power (kW)
A	1	10
B	2	10
C	3	10

Table 1 example of need for higher signal.

Need for lower price signal		
Current price signal = 2 Current production = 20 kW Wanted production = 15 kW		
Client	Min. Price	Power (kW)
A	1	10
B	2	10
C	3	10

Table 2 example of need for lower signal

The first table shows that only one client is producing and the server needs more power. The price signal must therefore be increased. In the second table client B has started to produce because the

price signal has reached the client's minimum price of 2. Now the total production is higher than the wanted production and the price signal must be lowered again.
The above example also illustrates a problem. The signal will go up and down with no stability because the wanted production of 15 kW never can be satisfied. It will either be 10 or 20 kW. This is not a problem in itself, but because an electrical system should in general have balanced power.
This problem can be solved by creating a marginal threshold of adequate size, or by producing too much for a period of time followed by a period with too little production. This could be done by mathematically integrating the history of production. The marginal threshold may be a fixed value or a percentage. If the production is within this threshold the price signal will ignore that the current production is not exactly the wanted production. In the example a threshold limit of 5 kW or 33% would stabilize the price signal. The framework however is meant to have a large number of clients and a threshold of 1-2% might be enough and also more realistic.
The price of the price signal is for a predefined amount of energy (e.g. 1 kWh).

Client behavior

The price signal is published by the server by setting a global variable available to all clients. Each client decides if he wants to produce power at that price. If the client is producing already and the signal has changed to an unacceptable price, then the client ends the production.
Because a given price only can be given for an amount of energy – the client will produce for a certain period. The period of production at this price can be:

1. A specified period of time specified by the server (e.g. 5 minutes).
2. A specified period of time specified by the client.
3. A period of time until the client updates the next time and the price signal has changed.

If the price signal has to work properly by encouraging or discouraging clients to produce by regulating the price signal it should be real-time or as close as possible to this. In this project the time the client is producing at the price signal price will be from the time the client states that production has started until either the production is stopped or the price signal changes.
One problem is if the client does not update. If the server suddenly needs a lot of clients to produce, one client starts production while the price signal is very high. If the client updates one hour later and the price signal have fallen to a very low level after 30 minutes, where the client cannot afford to produce.
What is the client's income?

1. The very high price for the whole hour of production (framework loss).
2. The very high price for the first 30 minutes and the very low price in the last 30 minutes (client's loss).

The solution to this problem is to demand a regular update-interval for each client. This could be e.g. once every 10 seconds or every minute. The price signal might have changed in this period but since the update-interval is relatively small the client's payment will be from the registered start-time till the registered stop-time. If the price signal changes the payment will be from the start-time till the time the client updates the signal.
The start, stop and updating times are set whenever the client updates and the information about starting and stopping is solely based on clients' information. This information will be compared with data from a trusted source like the local distribution network company before the actual payment takes place.
The problem still exists if there is a breakdown in connection between client and server. Should the client continue or stop its production? This will be handled in the breakdown section.
In this project a client's update-limit is created to ensure proper working under normal circumstances.

Start up time and price

Some power producers use units which have a start-up time before they are able to produce. E.g. some combined heat and power units. This warm-up period may also have a price, which means that starting a production can be more expensive than continuing a production. In this project the client will have an option of declaring both a minimum price for starting a production and a stop price, which denotes the price when the client do not make a profit from producing anymore.
The missing production in a start-up time can be handled by the client keeps producing for the same period of time after registering a production stop with this server. This way the total energy produced is correct but the actual period of production has been pushed slightly.
Start-up and stop price can also be the cause of a problem. In a framework with few clients it is possible to find a scenario where the price signal never stabilizes and clients are forced to start and stop production. The following example illustrates this problem.

Need for higher price signal		
Current price signal = 1 Current production = 1 kW Wanted production = 2 kW		
Client	Start/stop price	Power (kW)
A	1 / 0,8	1
B	2 / 0,2	3

Table 3

Need for lower price signal		
Current price signal = 0.4 Current production = 3 kW Wanted production = 2 kW		
Client	Min. Price	Power (kW)
A	1 / 0.8	1
B	2 / 0.2	3

Table 4

The price signal will go up and down forcing A and B to alternate starting and stopping. In the first case the price signal rises until B starts. Then the production is too large and the price signal will fall. A will have to stop at 0.8 but the price signal continue to fall until B stops producing at 0.2. Then the signal goes up again until 1 is reached and A starts.

Constraints

The framework must have a maximum and minimum price for the signal to ensure that the system does not get out of hand if e.g. no clients are available (price signal will keep rising).

Calculating price signal

The server in the framework has in advance of a time period made a deal with a source outside the framework to deliver an amount of energy. E.g. an amount of 10MWh must be delivered through Nord Pool within the next hour. This means an average of 10MW power.
The server should have 2 tasks:

1. Calculating the wanted average power until the end of this period based on the actual production in framework which has already taken place.
2. Adjusting the price signal to accommodate the wanted average power level.

The first task is (nearly) not necessary. If the system works in real-time and enough clients exist, then the average power level should be reached within a short period of time and thus remaining at this level. If a client stops production the power in the framework drops, but the price signal rises and a new client will (in the best-case scenario) immediately start producing at the new high price.
In reality the clients will spend some time updating and registering production which means that there is a lack of production in the beginning. This means that the server calculates a higher average power level for the rest of the period. Another option is to make up for the power lack at once by producing more than needed and then return to the average power level.

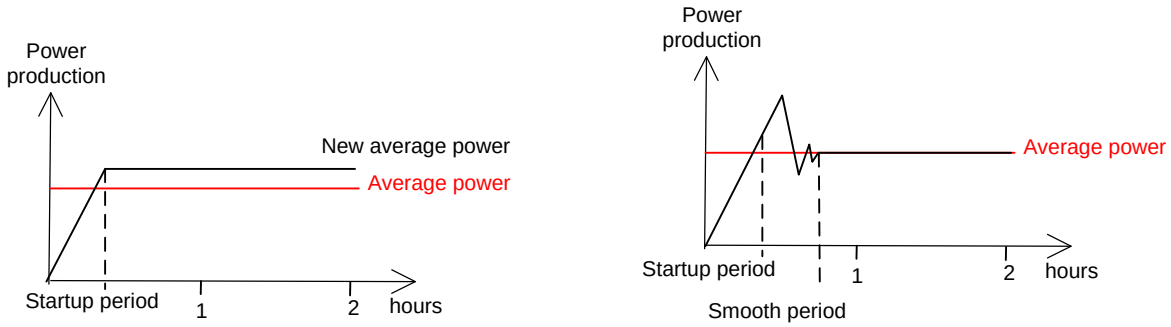


Figure 2.9 - The figures show price signal and wanted power level behavior.

Figure 2.9 shows how one model sets a new average power level for the rest of the a period if any variations has occurred, while the other model show how the startup period is followed by a smooth period where the price signal returns to the original average power level by producing extra power to equal the lack of power in the beginning.

Controlling a production could be split into to 2 parts.

1. Controlling that a specified power level is withheld in the framework.
2. Controlling that the wanted energy has been in fact produced and if not the power level in [1] should be regulated.

Pros and cons

The price signal has advantages and disadvantages.
Pros:

- Flexible, signal is changed all the time to fit power production
- Unpredictable production can easily be changed

Cons:

- Flexible, clients might have to start and stop all the time if the price signal goes up and down.
- Clients are not guaranteed a period of production.
- Clients make a profit when price signal is larger than their marginal costs.

2.2.16 Solution model 2: Price bidding

The second solution model is price bidding from clients. The TSO trades regulating power by accepting bids from producers. The solution model is based on the same methods as the regulating market use.
This solution works like this:

1) Clients states bids on how much, how long and at what price their willing to produce power.
2) The power control decides who can produce based on these bids.

Real-time or bidding rounds

One problem in this solution is figuring out when the server should accept bids. In general bids should be ordered ascending by price. The producer, who is willing to produce at the lowest price, should be the first to be accepted. Bids can be accepted real-time or in bidding rounds with specified deadlines like in a traditional auction.
Real-time acceptance of bids means that whenever the server needs more clients to produce it looks to the list of bids ordered by price and accepts a client until no more power is needed. Real-time acceptance makes the system flexible, because regulating power production can be done immediately, which is one of the abilities wanted in this project. One flaw of real-time acceptance is that clients do not know when their bid might be accepted (or not) and planning production can therefore be difficult. Another problem with real-time trading is illustrated if no bids are available at that specific time the server must chose the first bid whether it is good or not. This is acceptable when dealing with a large number of clients because the chance of no bids or mere expensive bids is low.
The server could also define a deadline for when bids are accepted like a traditional auction. After a deadline (bidding round), which could be every minute, every 10 minutes or maybe every hour, the cheapest bids would be accepted. This is a solution which is easy to administrate but lacks a bit of flexibility. It is a very feasible solution from the view of the server.
Another solution could be to start accepting bids when a certain number of clients have given their bids. This could be a good solution for a system with few clients to ensure that e.g. minimum of 3 bids has been made before bid acceptance starts. The problem is that a number of clients might have to wait unnecessary long time for the deadline to occur.

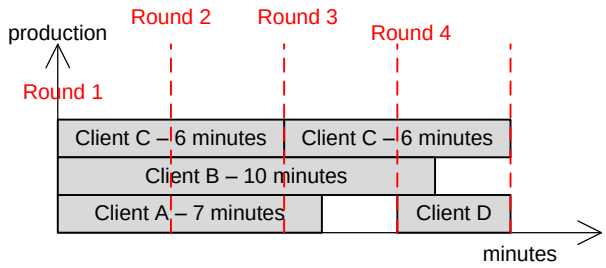


Figure 2.10 shows a model of traditional bidding. Red lines are representing bidding rounds every 3 minutes.

In the following example the wanted power value (W) is 3 (kW). For simplicity all clients send bids representing 1 kW so only price pr. kWh and length of production time vary.

Round 1 (Current production = 0)		
Client	Price pr. kWh	Length of production time
A	2,00	7
B	2,50	10
C	4,00	6
D	5,25	3
E	6,10	5

Table 6

Round 2 (Current production = 3)		
Client	Price pr. kWh	Length of production time
D	5,25	3
E	6,10	5

Table 5

Round 3 (Current production = 2)		
Client	Price pr. kWh	Length of production time
C	4,00	6
D	5,25	3
E	6,10	5

Table 7

Round 4 (Current production = 2)		
Client	Price pr. kWh	Length of production time
D	5,25	3
A	5,75	7
E	6,10	5

Table 8

The tables show which bids get accepted (highlighted) in each bidding round to keep the production at 3 kW when using traditional bidding rounds. The next example illustrates real-time bid acceptance.

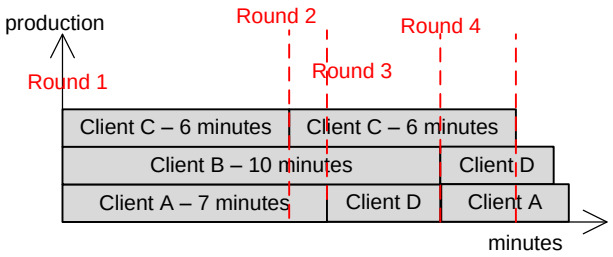


Figure 2.11 – Real time bid acceptance

Figure 2.11 shows a model of real-time bid acceptance. Red lines are representing each bidding round, which occurs whenever the power production falls to a lower value than the wanted production.

Round 1 (Current production = 0)		
Client	Price pr. kWh	Length of production time
A	2,00	7
B	2,50	10
C	4,00	6
D	5,25	3
E	6,10	5

Table 10

Round 2 (Current production = 2)		
Client	Price pr. kWh	Length of production time
C	4,00	6
D	5,25	3
E	6,10	5

Table 9

Round 3 (Current production = 2)		
Client	Price pr.	Length of production

Round 4 (Current production = 1)		
Client	Price pr.	Length of production time

	kWh	time
D	5,25	3
A	5,75	4
E	6,10	5

Table 11

	kWh	
D	5,25	3
A	5,75	4
E	6,10	5

Table 12

Again, the tables show which bids are accepted (highlighted) in each bidding round to keep the production at 3 kW when using real-time bid acceptance.
In this project the solution model using bid will be designed to use real-time bid acceptance.

Power, energy and duration of producing

When the server accepts a bid from a client, the client is obligated to produce the power that is specified in the bid. The production in the bid can be denoted as one of the following:

1. Power (W)
2. Energy = Power x time (kWh)

If the momentarily power production is used there must also be noted a period of production.
Power level in the producing time can be

1. Constant power level throughout production.
2. Variation in power level, but with an average power level that matches a constant power level.

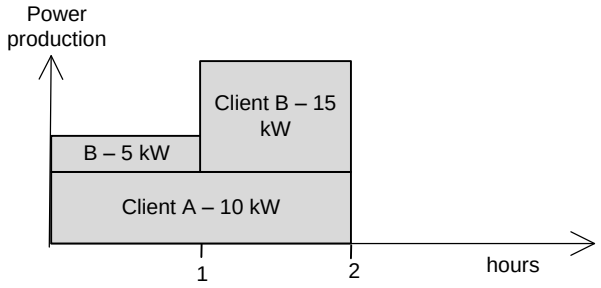


Figure 2.12 shows Client A and B, who both produce an amount of 20 kWh, but with different levels of power during their 2 hour production time.

If a constant power level is mandatory it is easier to administrate for the control logic. If variation in power level is allowed as seen in Figure 2.12 the framework will be more flexible for power producers. It also means that it is not possible to use logic that looks into the history of power production to set the wanted power production, because clients might make up for any lacks later in their production period. In this project the focus is on smaller units, which in general do not have the ability to change their power level. A constant power level is hence expected.
The energy stated in a bid must be produced within a certain time. There are several solutions to this problem.

1) After a bid has been accepted the client has a defined amount of time to produce. E.g. 1 hour.
2) The client defines his own timeslot and states exactly when the energy will be produced. E.g. 13:33 – 13:37.
3) The server defines a number of timeslots throughout the day and the client make bids on producing in one or more of these slots. E.g. 5-minute slots 13:05-13:10, 13:10-13:15, etc. The length of time for each slot should be decided based on the type of producers in the framework. This could also be 1 hour, 1 minute or even 1 second slots.

Client forecasting logic

The model with bids from clients creates a new problem: Forecasting. When a client make a bid on producing power he must have some sort of knowledge about how much he can or wants to produce. This might be easy for a large power plant with a regular production plan, which can afford resources for planning based on statistics or other models. This is not as easy for a small producer because the expenses of using advanced software or human resources to plan and predict the production are much larger relative to the profit the producer is capable of making.
This project focus on automating trade possibilities for producers and in particular smaller producers and the framework should be as flexible as possible so it is easy to join for a producer with no time and resources for production planning.

Broker agent forecasting

When a client sends a bid it is also a piece of information that the server can use in forecasting production for the framework. This helps the agent calculate how much power, to trade with the external markets.
If all clients in the framework were encouraged to deliver bids for all timeslots in the next 24 hours this information could be used in long term planning of trade possibilities for the agent. The clients should then be able to alter their bids up or down until the defined deadline for each timeslots (e.g. 1 minute prior to acceptance of bids). This is challenging the traditional bidding process like in auction where a bid cannot be lowered once it has been given.
It is important however to note that the bids might not give an accurate picture of the capability of the entire framework. Producers might give the same bid for each timeslot during the next 24 hours, but not be able to produce in all timeslots. The producer could have a limited production unit like a combined heat and power unit. One attribute of this type of units is a heat-reserve (water-container) which only can be heated to a certain extend and after this need a period of cooling with no production.
A wind power plant might expect the wind to blow but as weather changes the bids would be changed according to these conditions.
To help the broker agent forecast production in the entire framework each client could give trusted information about how much power they intend or wish to produce.

Bidding

A clients bid must have some information.

1) Price per kWh.
2) Amount of energy (kWh) or power level(W).
3) Time data, either a timeslot or time limit indicating when the production is going to take place.

The price is the price pr. kWh the producer wants to be paid for producing the amount specified in the bid.
The amount of power could be both an amount delivered no later than the specified time limit or a steady balanced production throughout the timeslot.
A 4^{th} data, deadline for bid, could indicate when the bid is no longer valid. This is data, which possibly can be omitted depending on how the bidding process is organized.

1) Bids given by the client is valid until the server-specified deadline for the timeslot is reached.
2) Bids given by the client is valid until the server-specified deadline for the timeslot but can be altered either one-way or both up and down.

3) If there are no specified timeslots and bids are accepted by the server at arbitrary times a deadline for when the bid is no longer valid is a good idea. A client would be able to give a new bid with a regular interval e.g. minute, each bid is then valid for this interval. This way bids would be suited to fit the production conditions at the site.

All data could have a minimum and maximum. In the regulating market a minimum bid is 10MW and maximum price 5,000 euro/MWh. In this framework clients are expected to be as small as 1kW and these limits, if any, should be set to fit the majority of the clients in the framework.
Another option is for the bid to include the power the client intends to produce instead of amount. The difference is that producing at 10MW for one hour results in the amount 10MWh, but the amount of 10MWh could also be produced by 60MW in 10 minutes (1/6 hour). The first scenario would have a balanced production while the last would be 50 minutes of no power and 10 minutes with high power.

Bid acceptance process

The server has the responsibility of deciding, which bids are accepted. In the simplest form the decision could be made automatically by ordering all bids and then accepting the lowest bids. This is also the principle of the regulating market in Denmark.

List of bids (all 1 kWh)	
Client	Bid (price pr. kWh)
A	10
B	15
C	8
D	12
E	18

Table 13

Ordered list of bids	
Client	Bid
C	8
A	10
D	12
B	15
E	18

Table 14

The tables show a simple example. All clients (A-E) have sent bids offering to produce 1 kWh. The agent however only needs a production of 3 kWh. The 3 low bids C, A and D are accepted.
The price is not the only data to take into consideration. Bids will typically be different amounts and bids for different time periods. This makes it an optimization problem for the agent to decide who should produce.
The optimization problem is illustrated in this example: The broker agent has agreed upon delivering 10 MWh between 8:00 and 9:00 to the market outside the framework. 3 clients have given bids for this period as shown in the following tables.

Client	Price pr. kWh	Amount (MWh)
A	10	5
B	20	10
C	30	5

Table 15

Accepted bids	Amount (MWh)	Total price
A + B	15	250
B	10	200
A + C	10	200
A	5	50

Table 16

The tables show an example of one optimization problem. The broker agent has agreed upon producing 10 MWh.

If the server chooses the cheapest bids it is A and B. But this also results in an additional 5 MWh is produced. In Denmark there is a balance market, whose task is to charge consumers that spend more power than they planned and "reward" producers if they produce more than planned. This basically means that producers usually get paid for power produced even if they do not have planned it. This

is typical for wind power plants. The price is somewhat lower than if the power was sold in an exchange or regulating market, because other producers have to decrease their production to keep the balance in the system. It might even be 0 or negative price. It might turn out to be cheaper for the server only to accept bid A and the 5 MWh and instead pay the balance market or regulating market for not complying with the production plans.
If the server accept bids to fulfill the production plan (10MWh) it must choose either bid A and C or B alone. In this case the total price is the same.
It is important for the logic in the broker agent to have some guidelines to follow in this situation like a set of rules for which priority price, amount and timeslots have.
The framework is intended to have many clients. If there are 3000 bids (relative smaller) instead of 3 it will most likely be easier to hit the right amount and timeslots.

Client income

When the server has accepted a number of bids, the clients are notified and start producing. It is assumed that each client produce exactly the amount specified in the bid. If not, accounts will be settled when measured production data from the 3rd party distribution network company is received.
Client can meanwhile send its own production data measurements to the broker agent making it possible to adjust unbalance inside the framework faster, if too little or too much is produced.
There are several options to how much each producing client is paid:

1) Clients are paid the amount, which they have specified in the bid.
2) Clients with accepted bids are all paid the same amount, equal to the most expensive bid the server accepts (market price) after each bidding round.

These options have advantages and disadvantages, which both influence and are influenced by other options in the framework. Most significant is how the framework's profit is split among the clients.
If all clients are paid the amount specified in their bid and no more, a smart client will start bidding at a high price, slowly lowering the price by each new bid until either a bid has been accepted or the client's marginal-price (when a profit no longer can be made) is reached. This solution might involve some kind of gambling or guessing for the client. If all clients are in a process of constantly altering their bids it results in an unstable system with fluctuations. On the other hand the client will have bigger chance of having their bid accepted if he bids his marginal price.
If all clients with accepted bids are paid the same price as the most expensive bid, which the server accepts it means that clients with low bids will be rewarded for offering a lower bid. This solution is an incitement for clients to bid the marginal-price.

Pros and cons

The bid model has advantages and disadvantages.
Pros:

- Clients can decide the duration of production
- Clients are guaranteed the production once a bid is accepted
- Server is able to forecast production based on given bids
- Clients are paid the bid price optimizing framework profit.

Cons:

- If power production is unreliable (e.g. wind power plant) clients are stuck with a bid that cannot be changed.
- Clients might speculate in giving higher bid prices to increase their own profit.
- Clients have to send in bids

2.2.17 Concluding solution remarks

In the previous sections 2 solutions to the power control problem has been made. The price signal is flexible and is able to change real-time to accommodate the wanted power level. The bid model guarantees clients a specified production period. Each power control model will be developed as part of the GFPT prototype and will be tested in the case study.

2.2.18 Low priority problems: Forecasting and breakdowns

The following problems and challenges of the framework are not going to be a part of the prototype. These issues have been included in the analysis because of their importance in the GFPT, but they will be low prioritized in the actual design and development of the prototype. This is in order to keep the prototype as simple as possible and focus on the actual power trading. This section will give a description of these problems and suggestions to possible solutions. The design and implementation of these features will be possible in future works of developing the prototype.

2.2.19 Forecasting

The framework has a lot of data on how much clients produce. Furthermore data exist in bids of how much clients are able to produce. Based on this information the system should be able to forecast how much power production will be in the future. This is important because deals with markets outside the framework are not made real-time.

Other data can also be used to make an accurate forecast. Weather information can be used to forecast production for wind power plants (wind level), combined heat and power units (temperature), solar cells (hours of expected sunshine) but also the time of day can be used to calculate the need for e.g. heat during night, heated water when people shower in the morning. Each client could also submit information about what it intends to produce for the following day.

2.2.20 External Trading

The server needs to trade power with a market outside the framework unless there consumers and producers in the framework produce and spend the same energy. In this project only producing clients exist. An extension to the framework could be to have consumers as well.

In previous sections power markets have been described. The forecasting module of the framework should give a realistic picture of how much and when power is produced in the framework. Exchanges like Nord Pool could be used but also the regulating market, since this already functions automatically by software. The framework system could be integrated to trade on the regulating market, which would mean no human labor should be used to control trading.

2.2.21 Balance responsibility

When trading power the balance responsibility must be handled by the entity called balance responsible of production. If the framework acts like a virtual power plant it only needs one entity to handle this responsibility, while the expenses will be shared among the clients. This is a fact by principle but in reality rules set by the transmission system operator on power trading and responsible balance entities might need to be revised to make it legal.

2.2.22 Adjusting unbalance in the framework

The server keeps the system in balance in general by using one of the price models described or a combination of the two. If a client's bid is accepted and it then finds itself unable to produce the amount specified in the bid there will be a problem. The client will not produce as much power and the framework will neither because it expects the client to deliver. This unbalance could be regulated with a functionality which enables the client to tell the server that it cannot produce the agreed amount. The server then can try to get this power from another client in the framework. Part of the solution is already designed since clients leave a log in the database every time it updates. From this data the server will be able to monitor if the client is producing the expected amount. If the client does not it might not mean that it is a bad thing. The client might have some knowledge

about its own production that the server does not. E.g. a wind power plant might expect it production to increase considerable in the last part of its production period due to weather changes.

2.2.23 Security

Many security issues should be dealt with in a framework like the one which is designed in this project. Any activities that involve financial transactions require a design that prevents criminal actions. Integrity must also be kept to avoid loss of important data and is therefore essential to the project. Confidentiality is important in this project although it is not a catastrophe if data is seen by someone without the required rights. It is not company secrets that are exchanged but simply producing information and price signals. Availability has high priority because the framework is based on web services and frequent updating throughout the system.
Security technology will be described in the technology chapter.

Breakdowns

In case of Internet (or otherwise connected) breakdowns the system is not able to work properly. Several breakdown scenarios exist.

1. Client Internet connection
2. Server breakdown
3. Client Power cable breakdown

If one client's Internet connection is lost it presents a problem in the bid model if a bid is active or accepted. It presents a problem in the price signal model if the client is producing.
An overall solution should be that these circumstances are agreed upon how to be handled when the client joins the framework. Some types of clients may prefer that if breakdowns occur they stop producing, others may prefer that they keep producing (if a new start-up is considerable expensive).
Several solutions exist:

1. If a client has not updated within the time limit defined then he is considered to have stopped production. This might mean unnecessarily many unforced production stop in reality, because Internet traffic might be congested heavily at certain times. The client will know that the Internet connection is down and can stop production, but it might not be possible (if the client has as minimum production period that is not reached yet).
2. If the client has not updated within the time limit it is assumed that production continue either at the same price, as the last update or just following the current price signal. This is risky because maybe the client cannot produce anymore, or produce anymore with a profit, and prefer to stop. If the connection is lost for a long time it could mean a considerable loss for the client if he has to pay for the period of no production.
3. Declining price could be used. If the client has not updated the price of the signal will decline with a certain amount e.g. every minute until it reaches 0.

If the server breaks down the system of the framework will work, but clients will still be able to produce. Emergency procedures could be set in effect to step in (like the regulating market) if this happens. Some producers may be on standby to accommodate this situation.
If the clients power cable breakdown the power cannot be transferred to the framework unless it can be stored in some power storage device.

2.3 Requirement Specifications

The following section list the system requirements for this project based on the analysis of the domain and problem area. The specification use high-level requirements to sketch a rough draft of the entire GFPT. The actual prototype that will be developed at a later stage in this project does not need to fulfill all requirements. The requirements are based on the previous sections of this chapter including specifically the solutions to problems and the two power control solutions, price signal and price bid model, which will be implemented in the prototype and tested in the case study.

The requirement specification is normally used as a basis for the agreement between the customer and the supplier of the software product.
In this project there is no specified customer. As stated in the previous sections many groups of people can have an interest in the framework and it is assumed that a customer would be a private company, possibly owned by a cooperation of power producers.
The specification is divided into functional requirements and non-functional requirements. Functional requirements define the specific functions, tasks and behavior of the system. Non-functional requirements define constraints on design, implementation, security and quality.

2.3.1 Functional requirements

The functional requirements are based on the analysis of the power market and power producers.

1. The software must be able to trade power automatically.
2. The server must be able to adjust the frameworks power production level (sum of all clients' production). In full-scale GFPT this should be done automatically, but in the prototype this should be done by the control interface.
3. All production and price information should be stored in a database.

User interface

4. The administrator must through the interface be able to change updating intervals and possibly other variables.

Price model 1: Signal

5. The server must be able to set the price signal automatically.
6. The clients should be able to check the price.
7. Clients must be able to decide themselves (automatically) when to start and stop production based upon the value of the price signal.
8. Each client must notify the server at any change in production.

Price model 2: Bid

9. The system must make it possible for each client to give bids to the server if he wishes to produce and then check if bids are accepted in order to start production.
10. The server must accept bids available when more power or continuous production is needed.

2.3.2 Non-Functional Requirements for the GFPT prototype

The non-functional requirements in this section are the requirements that define constraints on the prototype. The requirements will be tested in the case study later in this thesis.

Scalability

The system architecture must use service oriented architecture to be able to function with anything from 10 to 10,000 clients, but the prototype will only be tested using a simulation program of 10 clients.

Usability

The framework should be as automatic as possible so that clients can make their units produce with no or minimum human interaction.
The server should have an administration interface for set up and control abilities.

Extensibility

The system should be a model with simple functions and be easy to extend with new functionality and abilities in a future solution.

2.3.3 Non-Functional Requirements for a GFPT

The non-functional requirements in this section are the requirements that define additional constraints on the GFPT in general. The non-functional requirements are defined merely to give an idea of what to expect of a GFPT based on the analysis of the power market and power producers. These non-functional requirements will not be included in the development of the GFPT prototype and therefore not tested in the case study either.

Scalability

The system architecture must be a generic framework and use service oriented architecture to be able to function with anything from 10 to 10,000 clients, based on the fact that a minimum of 10 MW is needed to trade on the regulating market. If the average client has a 10kW unit a minimum of clients would be well over 1,000 since not all can be expected to produce at all times.

Access

The system must be easy to join for power producers (clients) and a simple interface must exist which only requires a minimum of integration for the smallest producers, but also with opportunity for large producers to include functionality.

Reliability

The system should be available and functioning at all times with a minimum downtime.

Security

Security must be able to be applied in an extended version of the framework, but should not be implemented in this solution.

2.4 Concluding remarks

This chapter has analyzed the power markets and found that some of the problems can be improved by developing a GFPT as suggested in the problem definition.
The analysis showed that the power market could be both more effective and faster responding when regulating system balance if smaller production units were used due to their fast start-ups. These units however have difficulties trading their power because of expenses. Virtual power plants containing many smaller plants could be an effective way of sharing expenses and common tasks. The distribution system could obtain an even more effective and faster regulating ability if trading could be done automatically in real-time. The problems and challenges in power market:

- Enabling small producers to trade power in a virtual power plant
- Automatic trade of power to minimize human resources.
- Real-time trade of power optimizing balance regulating.
- Controlling power production level.

The development of a GFPT could solve these problems. Clients join the framework which handles trading automatically. The framework server has a module that regulates power level in the framework by some production control logic. This logic can be designed in many different ways. In this project 2 models have been analyzed:

- Price signal
- Bid model

The price signal regulates a price for an amount of energy. Producing clients update the price signal with a regular interval. If a client can benefit from producing at this price it lets the server know that it is now producing. The server regulates the signal either up and down if more or less clients are needed to produce power. The signal is thought to be a good solution for clients with a varying or unstable production with no or very low start-up time (e.g. wind power plants).
The bid model makes use of bids from clients. The framework will have a function that enables clients to automatically send bids by a custom-set interval as well as price, production period etc. The bid model is thought to be good for clients who prefer to have a scheduled production. This could be e.g. a combined heat and power unit.
Requirements for the prototype of the GFPT were specified for both functionality and non-functionality. The prototype with its 2 power control models will be designed and developed in the next chapters and functionality and requirements are tested in the case study to see if they match the projects vision of building a state-of-the-art framework.

3 Conceptual Design

The purpose of this chapter is to describe the conceptual design of a GFPT. This will be done using the problem analysis and solution descriptions of chapter 2, while fulfilling the requirement specification.
A model of conceptual design will be achieved using use cases to identify events and classes. The system architecture will be described and flowcharts constructed. The conceptual design of the GFPT will be used in chapter **Error! Reference source not found.** to design and implement the prototype.

3.1 Use cases

The conceptual design will begin with identifying use cases for each actor in the GFPT.
The following use cases are based on descriptions of the problem area in the previous chapter and are directly linked with the functional requirements from chapter 2. Each actor has some specific tasks both in general but also specific for the power control model.

Actor	**Model**	**Functional Req.No.**	**Use Case**	**Description**
Client Server		4	Set data	Sets variables in database, e.g. minimum price, power, etc.
Client		3, 8	Send production status / update	Sends information about the amount of power being produced and whether the client has started, stopped or is continuing production.
Client		7, 9	Start production	Decides to start production
Client		7, 9	Stop production	Decides to stops production
Client		1	Trade power	Function that uses client data and power control model to make a trade.
Client	Price signal	6	Get price (signal)	Gets value of price signal at the server and store timestamp in database for calculating payment.
Client	Bid	9	Send bid	Gives a bid on energy in a time period and at a certain price.
Client	Bid	9	Check if bid has been accepted	Check if bid has been accepted or not
Server		1	Trade Power	Trades power by using one of the power control models.
Server		3, 8	Log Production	Stores information about each clients production
Server		2	Set wanted production	Set/calculates the energy the framework needs to produce to fulfill agreements.
Server	Price signal	5	Set price signal	Sets value of price signal based upon calculations on wanted production.
Server	Bid	10	Accept bid	Accepts bid

Figure 3.1 shows the basic functions of a power producer. This includes the start and stop of production and trading power. The GFPT in this project must function automatically and the tasks split up in smaller subtasks.

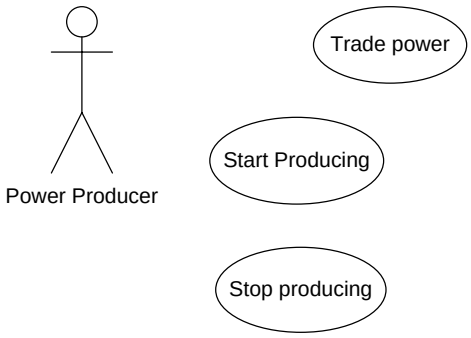


Figure 3.1 shows the basic tasks the power producer must perform.

In Figure 3.2Figure 3.2 and Figure 3.3 the task of trading power is divided into the price signal model and the bid model. Both models use shared functionality and only the evaluation regarding price model differentiates between them.

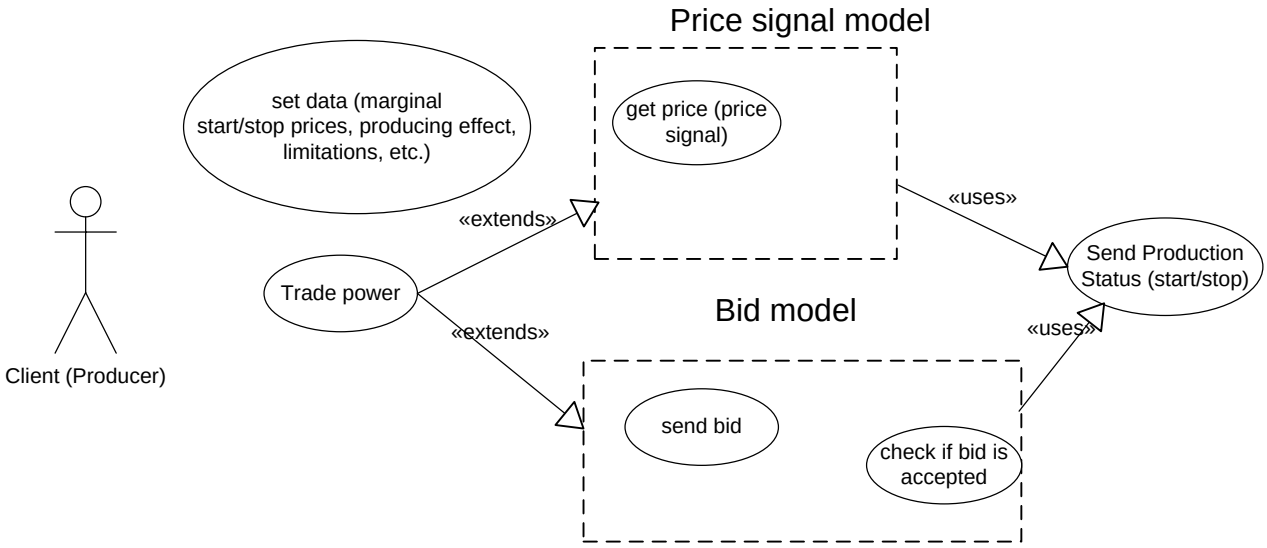


Figure 3.2 shows use case diagram for the client in the framework.

The same figure can be set up for the server. Instead of evaluating whether to produce the server evaluates if the current combined production is acceptable. If changes needs to be made the server handles this, regarding to the control model in use. E.g. accept new bids or change the price signal.

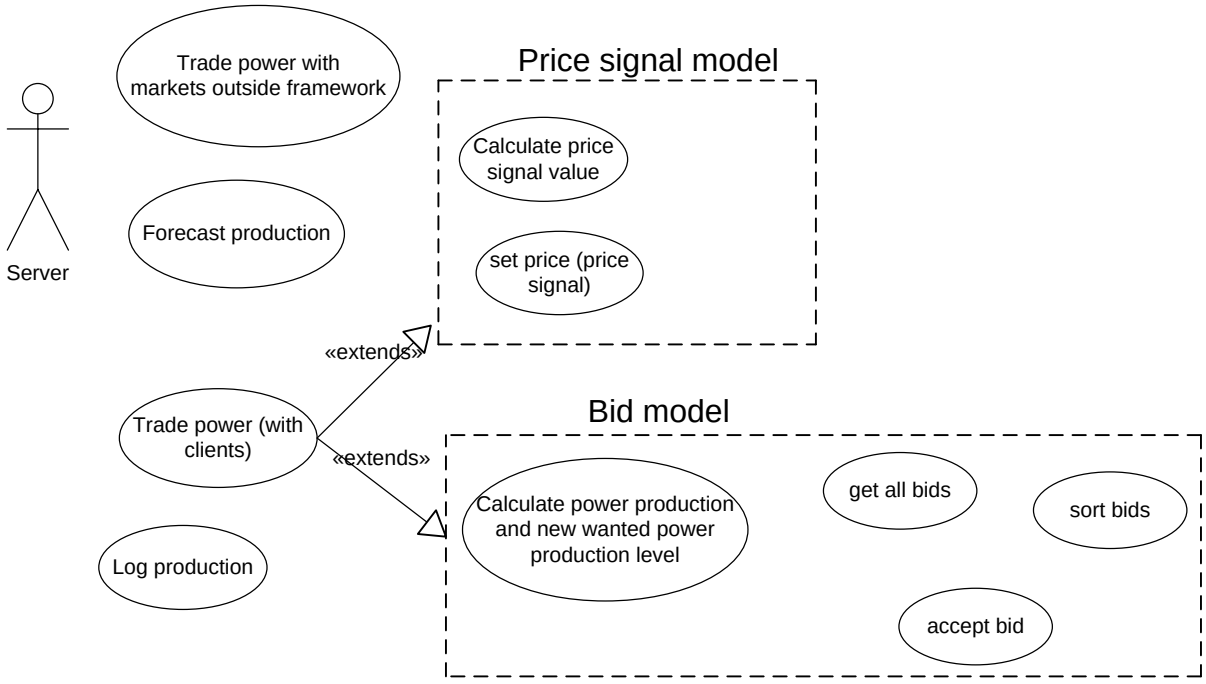


Figure 3.3 shows use case diagram for the server.

3.2 Defining the system

To specify the needs and expectations for future developers and users of the framework a system definition is made. As a template for defining the important elements, the CATOFP[11] model is used. This model can iteratively be compared to the system definition until an acceptable definition is reached. The same can be done when some of the elements in the model changes. The six elements are described below.

Conditions: The conditions that further use and development of the system are subdued.

Area of usage: Administration, monitoring and control of a problem range by one or more organizational units.

Technology: The technology used to develop the system, and the technology that the system is developed for.

Object system: The future users understanding of a problem range.

Functionality: The main functions, offered by the system to support tasks within the range of usage.

Philosophy: The philosophy behind the IT-system.

[11]Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, Jan Stage, Objekt Orienteret Analyse og Design (BATOFF Model)

The content of the CATOFP model is based on the basic communication structure of the framework combined with the two client control models. The result of the CATOFP model is as follows.

Conditions:	The framework must supply the basic functionality for implementing the production control scenario between the producing client and the server controlling the needed production. The framework must be module based enabling the developers to add new functionality and expanding the general functionality of the system core. The modules integrates the capabilities of the framework into a simple object structure where production control software can use a simplified control interface to decide upon the current production level.
Area of usage:	The main control center of the virtual power plant uses the framework to communicate and control the clients. The clients use the framework to integrate communication into the control software for their production.
Technology:	The framework is developed on Microsoft .NET Framework 3, using web services based on WCF. The control interface is combining the framework with WP.
Object system:	Client, Server, and Control Signal.
Functionality:	Simplify the control interface to power producing clients. Enable the combined framework to supply production statistics and current performance.
Philosophy:	To simplify and standardize the implementation of the client control scenario.

The criteria of the CRTOFF model has been used to compile the following system definition

3.2.1 System Definition

The framework should be a collection of modules to simplify the development process of new controlling software for the implementation of smaller power plants. The framework should implement the communication structures of the control modules. The WCS based communication should however enable other frameworks or client implementations to communicate with the open platform.
The proof of concept simulation should implement a control application for the combined virtual power plant as well as a simple client simulation application. This simulation should implement the price bidding and price signal control protocols.

3.2.2 Analyzing the problem area

The purpose of this section is to define and describe an object system. The problem area is the part of the environment administrated, monitored or controlled by the system. This section will try to describe the real life scenario as future users need to think of it. Focus will be kept on the general view instead of on specific details.

The product of this section is a coherent model of the object system described in the system definition.
The most important part of this analysis is to understand the object system and not the technical specifics of the IT system. The technical specifics are described in detail later on. To obtain this understanding, the following activities are made:

- Flow charts
- Event table
- Class diagram
- State diagram

Working through these activities is an iterative process, where further awareness is achieved along the way. The activities are therefore not processed once in the described order, but instead repeatedly carried out moving back and forth between them, until the end of the analysis.

3.2.3 Flow charts

In this section flowcharts are created to show how the conceptual design of the solution functions. The flowcharts are later used in the physical design and implementation.

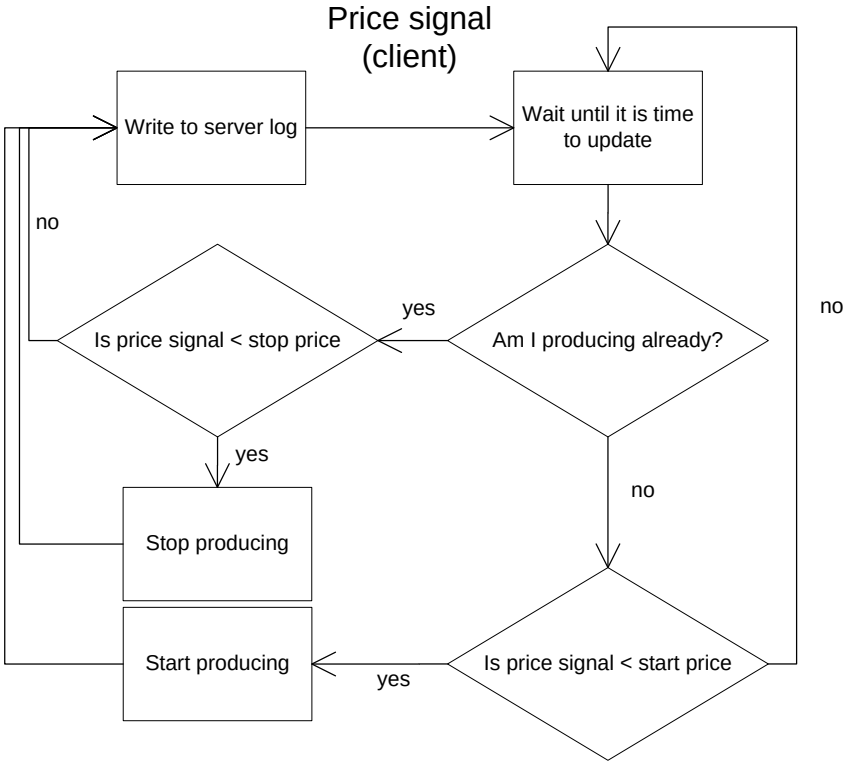


Figure 3.4 shows a simplified flowchart for the price signal process for the Client class.

In Figure 3.4 the flowchart for the client using the price signal is shown. The client must update within a certain time limit. When updating, the client must take into account whether or not a production is currently running. If not the client should start the production when the price signal is greater than its minimum start price. If the client is already producing the client should stop producing when the price signal has fallen below the client's stop price and continue producing if it is still favorable.

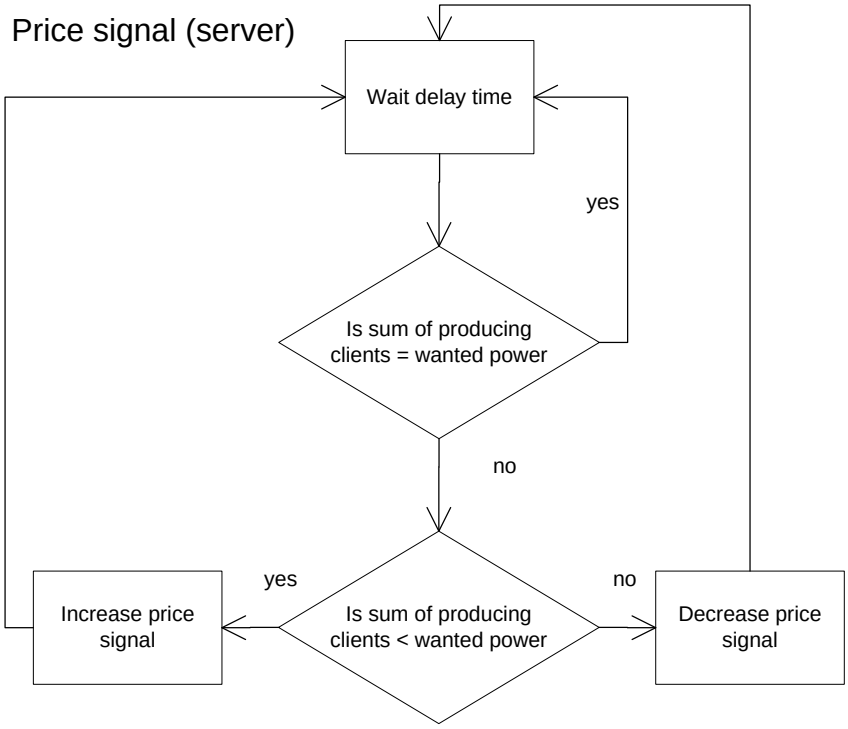


Figure 3.5 shows a simplified flowchart for the price signal process for the Server class.

In Figure 3.5 the flowchart for the server is shown. The server must like the clients update with regular intervals. When the server updates it compares the combined production of the framework with the amount of power wanted. If the amount is inadequate or substantial the price signal is either increased or decreased to encourage the client to modify their production.

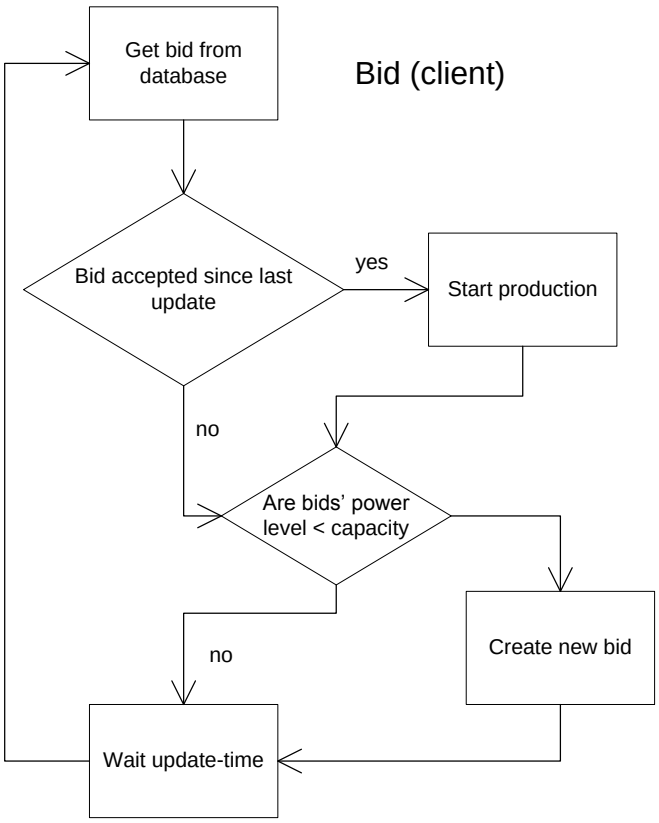


Figure 3.6 shows a simplified flowchart for the bid process for the Client class.

In Figure 3.6 the flowchart for a client using the bid model is illustrated. The client gets data on its own active bids from the server. If any of them have been accepted the client must start the production. Then the client must check if its current production capacity is higher than the total power level specified in the bids. In other words the client must check if it able to produce more than it has already bid. If it is a new bid should be created.

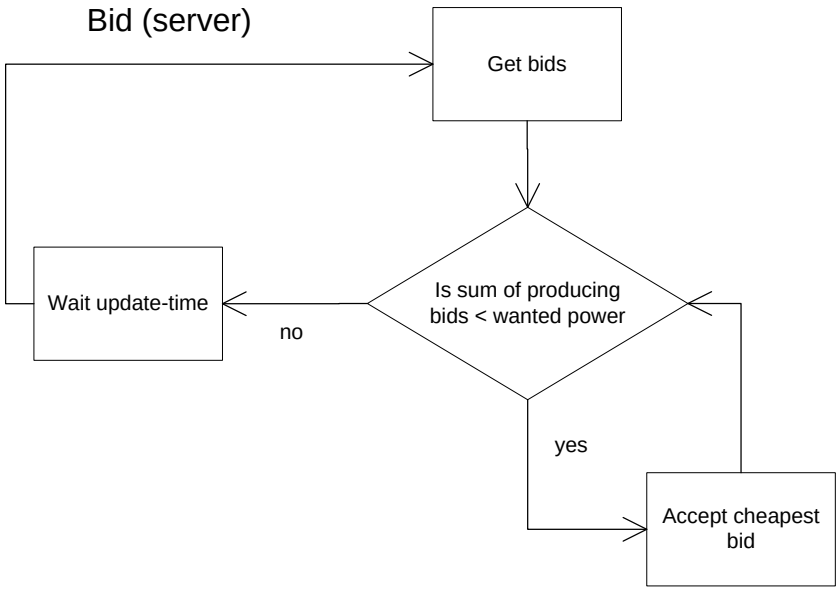


Figure 3.7 shows a simplified flowchart for the bid process for the server class.

In Figure 3.7 the flowchart of a server handling the bid model is illustrated. The server gets all bids from the database. If the current production is lower than the wanted level then the cheapest bid is accepted. This process is repeated until the wanted power level is reached.
These are the simplified flowcharts. In the physical design some extra parts must be added. The client must check if any of the accepted bids are about to expire and if it should make a new bid on continuing this production. If this is not handled without gaps the client would have to stop and start the production between each bid.

3.2.4 Event table

This activity forms a table that illustrates the connection between the essential classes and events in the system. An *event* is defined as "an immediate occurrence involving one or more objects". Classes and events are identified from the problem area and the first iteration is making an uncritical selection in order to get a larger number of objects.
The following iterations perform a critical selection where classes or events are removed from the table if the IT system does not need to store the information kept in the element. This ends up with a table containing the left classes and events. The connections between these are marked.

Locating the Classes

The uncritical selection creates Table 17 where the following selection of each class is marked with either "+" (chosen) or "-" (discarded). The first uncritical selection is the result of a brainstorm focusing on nouns where the final selection of classes is made from an evaluation of each class against the problem area. As an example of this, elements as "ProductionUnit" and "PriceLog" are discarded because they are not part of the problem area since even though they are part of the IT system, no information needs to be registered about them.

Class	Critical selection
Client	+
Control Type	+
Price Bid	+
Price Bid Settings	+
Price Signal	+
Price Signal Settings	+
Production Log	+
Server	+
Client Type	+
Power Service	+
Client Interface	-
Server Interface	-
Price Log	-
Statistics	+
Production Unit	-
Price Control	-

Table 17 – Choosing the important classes

Locating the Events

Selecting events are done in the same manner as the selection of classes except from focusing on verbs instead of nouns in the first uncritical selection.

Events	Critical selection
Load Client Information from database	+
Save Client Information to database	+
Load a list of all clients on a server	+
Load Control Type from database	+
Save Control Type to database	+
Load Client Type from database	+
Save Client Type from database	+
Client evaluate production	-
Client evaluate price	-
Evaluate price control	+
Get a price bid from the database	+
Get all valid bids for a server	+
Get all valid bids for a client	+
Load a bid from the database	+
Save a bid to the database	+
Load price bid settings for a server	+
Save price bid settings to a server	+
Get the current price signal from the database	+
Get the current energy produced in the system	+
Update the price signal in the database	+
Load price signal settings from the database	+
Save price signal settings to the database	+
Clear production log information	+
Add production to production log	+
Load server information from the database	+
Save server information from the database	+
Load statistics information from the database	+

Table 18 - Choosing the important events

The Event table

The chosen classes and events are inserted in a table (Table 19). If a connection exists between an event and a class this is marked.

Event	Class										
	Client	Control Type	Price Bid	Price Bid Settings	Price Signal	Price Signal Settings	Production Log	Server	Client Type	Power Service	Statistics
Load Client Information from database	●	●							●	●	
Save Client Information to database	●	●							●	●	
Load a list of all clients on a server	●							●		●	●
Load Control Type from database	●	●								●	
Save Control Type to database		●								●	
Load Client Type from database	●								●	●	
Save Client Type from database									●	●	
Evaluate price control	●		●		●		●	●		●	
Get a price bid from the database	●		●							●	
Get all valid bids for a server			●					●		●	
Get all valid bids for a client	●		●							●	
Load a bid from the database	●		●					●		●	
Save a bid to the database	●		●					●		●	
Load price bid settings for a server		●	●	●				●		●	
Save price bid settings to a server				●				●		●	
Get the current price signal from the database	●	●			●			●		●	
Get the current energy produced in the system			●		●		●			●	
Update the price signal in the database					●	●		●		●	
Load price signal settings from the database					●	●	●	●		●	
Save price signal settings to the database						●				●	
Clear production log information							●	●		●	
Add production to production log	●		●		●		●			●	
Load server information from the database								●		●	
Save server information from the database								●		●	
Load statistics information from the database										●	●

Table 19 – Connection between classes and events

The event table illustrates the connection between the chosen classes and events as they are interpreted in the problem area. The information from the table is used further during the rest of the activities in the analysis of the problem area.

3.2.5 Conceptual Class Diagram

The purpose of creating a class diagram is to look into the structural connections between classes and objects in the object system. There are two main types of structures connecting classes.

- Abstract, static connections between classes.
- Concrete, dynamical connections between some of the objects in the classes.

An example of an abstract, static connection between two classes can be seen in Figure 3.9Figure 3.9.

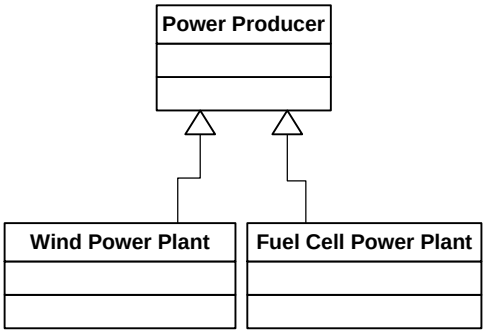


Figure 3.8 – An example of an abstract static connection between classes. This is a generalization structure.

Connections between classes can be viewed upon as the classes "Wind Power Plant" and "Fuel Cell Power Plant" both is a subset of the class "Power Producer". I.e. both wind power plants and fuel cell power plants are power producers. The other way around however is not necessarily true. A power producer is not necessarily a wind power plant.
This form of connection between classes is known as a *generalization structure*. This is because the class "Power Producer" is a generalization of the classes "Wind Power Plant" and "Fuel Cell Power Plant". It can also be viewed upon as "Wind Power Plant" and "Fuel Cell Power Plant" is specializations of "Power Plant".

Two forms of concrete, dynamical connections between classes are used. One example of these is viewed in Figure 3.9 which shows the connection between the class "V90-3MW Wind Power Plant"[12] and the class "Wing". This figure shows that a V90-3MW wind power plant is fitted ("contains") with 3 wings. This means that "V90-3MW Wind Power Plant" is a unified whole that among other objects includes a wing. The numbers means that a "V90-3MW Wind Power Plant" has three wings and one wing is only part of exactly one "V90-3MW Wind Power Plant". This structure is known as the *aggregation structure*.

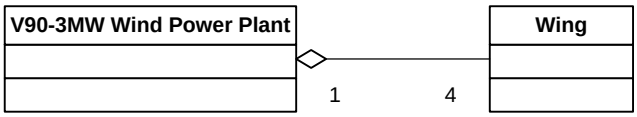


Figure 3.9 – An example of a concrete dynamic connection between classes. This is an aggregation structure.

The second form of concrete, dynamical connections between classes (Figure 3.10) are known as the *Association Structure* and informs about the connection between the class "Wind Power Plant" and the class "Owner". The two classes are coordinately meaning that one of the classes is not part of the other class as the "Wing" class in the example above. The illustration only shows a connection between the classes, in this example the ownership of the wind power plant. The numbers mean that one owner has one or more wind power plants as well as a wind power plant has zero to more owners.

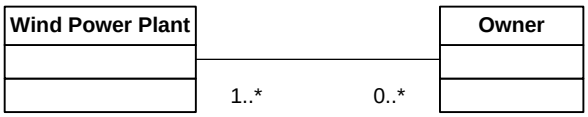


Figure 3.10 – An example of a concrete dynamic connection between classes. This is an association structure.

[12] A specific wind power plant type from Vestas, having 3 wings.

The design of a class diagram is done by locating a number of classes and structuring these relating each other using the two types of structures described above.
The classes from the critical selection when creating the event table is reused when creating the class diagram. The purpose is to structure the elements and their connections in between in a well-arranged manner. The result is displayed in Figure 3.11.

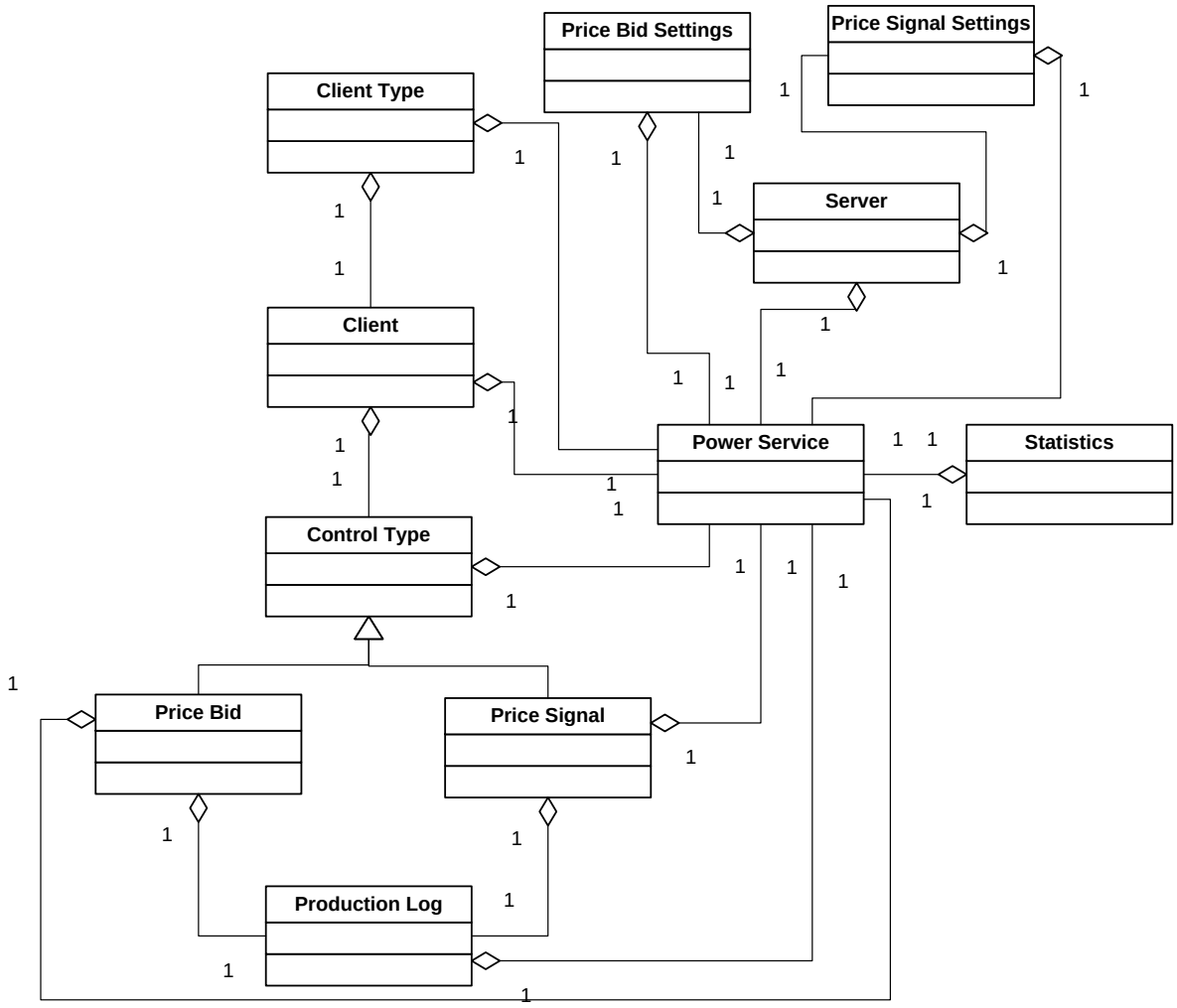


Figure 3.11 – Power Framework Conceptual Class Diagram

3.3 System architecture

The system architecture for the GFPT is illustrated in Figure 3.12. The architecture is based on a data layer (database) accessed through WCF service as a communication layer.

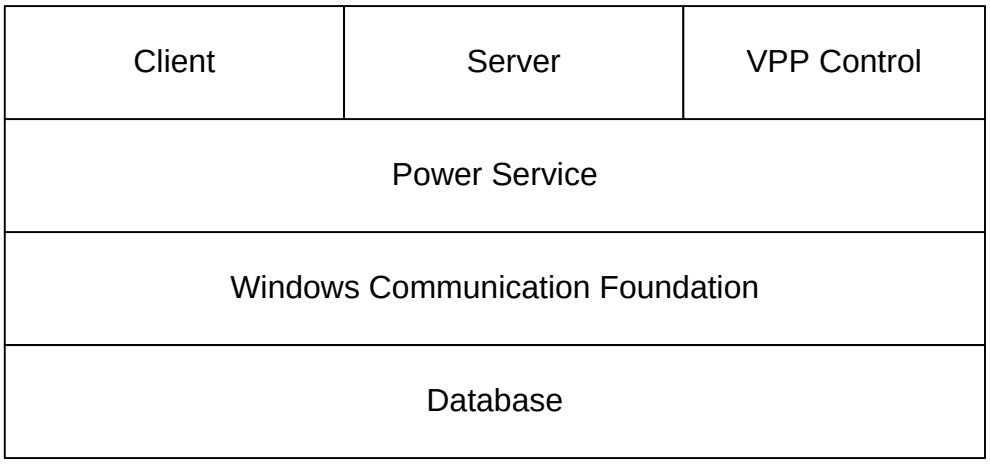


Figure 3.12 System Architecture

The role of each basic component will be described in the following.

3.3.1 Client

The client is the basic type used to model the implementation of a power producing unit of any type. The purpose of the client is to generalize the architecture of the power producers and provide the general interface for the control variables used inside the framework.
A power producer can then use this interface to identify when begin or end production as well of controlling the amount of power to produce.
The actual, and client specific, control software e.g. SCADA can then be used to physically control the production equipment.

3.3.2 Server

The server identifies the controlling segment of the framework. In general, the server implementation is as generic as the client implementation. The server is the interface for controlling the amount of power produced by the framework, as well as the acceptable prices to pay for this power.
The implemented control models are specifically engineered to minimize the cost of the power produced but other control criteria can just as easy be implemented.
The server module could be controlled by integrating the price structure and power production into a stock exchange like control application automatically bidding the capacity of the framework to an existing market.
It could also be implemented as a simple control platform for a virtual power plant, where a user could specify how much power the framework should produce, disregarding the power prices.

3.3.3 VPP Control

The VPP control application is an extremely simple implementation of a server control platform. The application acts as a simple interface for configuring the control types of the server.

3.3.4 Power Service

The power service is the framework API used for all communication between the objects of the framework. All communication in general is sent through this interface and all settings and information produced are stored in the database, which is only accessible through this API.

3.3.5 Windows Communication Foundation

The WCF layer is the technology used for all components of the GFPT to communicate in between. This layer is a part of the general design structure of the framework and is part of the power service from where the platform is extended into basically every class of the framework communicating with either the database or other classes.

3.3.6 Database

The database is used for storing data shared between the applications of the framework as well as a reference for all information produced by the framework.
The database is used for saving the general information such as client identification, production logs and general settings.
Temporary information used inside local objects, such as the current capacity of a client is not transferred to the database. This information is updated by the client and only used for calculations inside this object.
The database is also used for doing advanced calculations and manipulation of the above information, such as accepting the best feasible bids based on cross checking current production log entries and current price bids.

3.4 Concluding remarks

The purpose of this chapter was to create a coherent model of an object system defining the fundamental design for the GFPT. The functional requirements of the system were used to form a number of use cases specifying the general usage scope of implementations based on the framework structure.
Based on these use cases the objects important for the design were identified and their interaction in between characterized in terms of their events. This resulted in a conceptual class diagram. The general system architecture was designed with a database as the data layer, WCF service as communication layer, the GFPT as a platform and with a server part, client part and a user interface for virtual power plant control.
The next chapter examines the technologies which will be used for the implementation. The actual implementation of the object system in the framework prototype is described in chapter 5.

4 Technology

This chapter examines some of the technologies that can be used in this project. The purpose of the chapter is to find and describe the newest and strongest technology platform for creating the strongest solution to the current problem. A decision was made upon the Microsoft platform already in the preliminary phase, and the general framework will be developed as an add-on to the Microsoft .NET Framework.
This add-on can be used to implement the communication platform used between the components of the system, using the standard functionality of the framework.
The implementation will be done in the C# programming language, focusing on the newest technologies available for this platform.
The first part of this chapter gives a throughout description of service oriented architecture.
Pros and cons of using service oriented architecture to implement the desired functionality will be discussed.
The second part of the chapter briefly describes the chosen technologies and development techniques.

4.1 Service Oriented Architecture, SOA

A Service Oriented Architecture is an architecture based on the interoperability of a number of web services. A web service is a general term of a web based application that uses open standards to publish an interface to a remote client.
The remote client can be another web service or an application that uses the information published by the service.
OASIS (the Organization for the Advancement of Structured Information Standards) has the following definition for SOA[13]:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

SOA is an architectural style more than a technology. There is nothing new to the design and ways of thinking being done in web services. The first release of CORBA (Common Object Request Broker Architecture) was introduced in October 1991, but even though the technology has evolved the general concepts are the same.
Services are communicating through a collection of related endpoints. The actual code behind the interfaces can be updated and changed as long as the endpoint of the service remains the same. The applications and services depending on the output of the service will function regardless.
This is why web services are called loose coupled. They can communicate between programming languages and platforms because the interface is defined as a standardized endpoint. Each endpoint is described using the WSDL (Web Service Description Language).

4.1.1 Advantages of SOA

When an endpoint is made available it can be accessed by anyone from any platform that supports web services. This can of course be restricted to specific users and application if needed. This

[13] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, OASIS Reference Model for Service Oriented Architecture 1.0

interoperability with other platforms makes it very simple to integrate functionality into different applications and systems.
The specific operations being done behind the end point is hidden from the users. It is being done on the server side and is encapsulated so that all operations are out of range from any malicious users. This also makes it possible to update and modify the actual operations behind the end point without changing or notifying the depending systems. As long as the interface remains unchanged this does not affect any sub systems since all logic is hidden from the outside world.
If a functionality is published through a given web service it can be reused in all subsystems afterwards. The operations never need to be implemented more than once. This does not only save the work of implementing the same functionality in different sub systems and languages but also minimizes the work needed to update and maintain code.
The interface of web services are designed to share all public information in a manner that publishes the functionality of the web service in a standardized way so that other applications and web services automatically can detect the services offered[14].

4.1.2 Disadvantages of SOA

SOA results in an extra XML layer which means more parsing and composition. All variable types as converted to text and back, even though the application development is done using the same programming language. This causes applications using web services to run slower and require more processing power.
Standards and products are still evolving which potentially can make interfaces and contract specifications change making it very difficult to update a service being used by cross platforms.
The interface of a public service is locked. Once the service contract is published and implemented in other applications the depending applications will fail if the service end point is changed. This makes it difficult make changes to the logic that requires a redefinition of the end point.

4.1.3 SOA in the virtual power plant framework

The general purpose of the virtual framework is to create an implementation platform, where any type of producer can be a part of the virtual power plant. Each client is connected through a virtual interface exposing the variables needed to calculate production capacity and communicate production decisions.

The combined forces of all virtual clients create one large virtual power plant as shown in Figure 4.1.

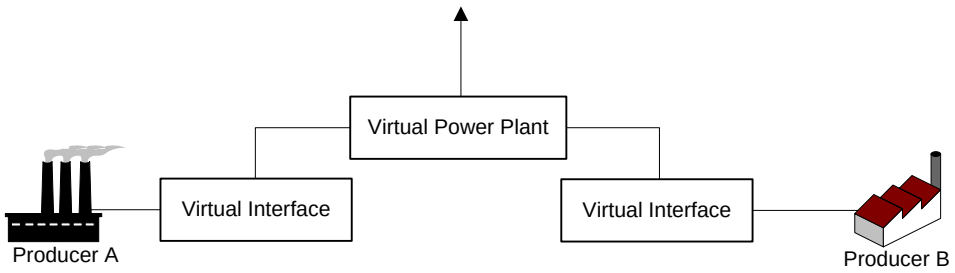


Figure 4.1 - Virtual Power Plant Components

[14] WS-Discovery – Web Services Dynamic Discovery defines a multicast discovery protocol to locate services on a network.

The production code to control a specific production environment is developed at the clients. Future works could include a specification module to define PLC integration or standards for a production unit e.g. a specific wind power plant type.
The virtual interfaces communicate through the internet with the main service. This interface can ideally be developed using the techniques available from the web services. This enables the loose coupling of the services to allow other systems to integrate communications with the service end points as long as the defined standards are maintained.
This creates a more flexible framework where each specific module can be used or reused when integrating existing or new components.
A local server at a production site can create an instance of the virtual interface and equip this interface with the local parameters needed to decide upon the best production. The local system also has access to the variables available in the virtual interface that decides whether the production should be increased, decreased or stopped.
The production code running on a proprietary platform inside a PLC or SCADA system can use the information available from the interface to regulate the production equipment and redefine the new production parameters in the interface.
Using SOA as a platform for creating this will enable the system to benefit from the advantages already described. The disadvantages are acceptable as the evolving standards can be accommodated by updating the virtual components allowing backwards compatibility with older systems, where possible.
The logic developed at a client side for controlling the production is hardly linked to the interface. If the module for calculating prices changes the interface, the entire production site needs to reevaluate the decision making procedures. These updates cannot be done frequently anyway and the disadvantage of a locked interface is not increased further by choosing the SOA.
The amount of information traded between the components is minimal. The extra processing power for passing data to XML upon communication is therefore minimal. Even thought the amount of clients can be increased, this number is depending on the success of the system. If a large number of clients are involved, the system can be scaled to fit the amounts of request required by the clients without straining each client remarkable.

4.2 Technologies

This section briefly describes the technological foundation that the solution is based upon. Each technology is introduced and the reason for choosing the technology is mentioned.

4.2.1 The Microsoft .NET Framework 3.0

The Microsoft .NET Framework is the newest development platform available from Microsoft. The framework makes a programming library available for creating new applications. The force of the framework is the Common Language Infrastructure (CLI). The development process produces managed code and the CLI allows it to be executed on a defined hypothetical machine, optimizing for architecture and producing a closed execution environment. Ideally this creates a platform independent environment comparable to Java Virtual Machine, where the UNIX platform is covered by the Mono[15] project.

[15]The Mono project provides the .NET platform for a number of other OS than Windows. http://www.mono-project.com

The frameworks consist of different versions of components. Each framework update is a combination of adding new technology and replacing existing with stronger components. The current version is 3.0 but when working with the framework, components from each version is used. The .NET Framework 3.0 is a collection of classes added to the .NET Framework 2.0. This collection originally referred to as WinFX consists of the newest managed code programming model for Windows. It supports communication across technology boundaries enabling the newest technology able to interface with existing technology and other programming languages.

The framework fundamental technologies consist of the following:

4.2.2 Windows Presentation Foundation, WPF

The WPF provides the newest technology for presenting information to the user. This combined with the user interoperability provides the perfect fundament for creating the user interfaces to manipulate data and present system controls and status. This is available for creating real time data driven and user manipulated interfaces.
One of the success criteria for the final solution is the ability to create a simple and intuitive control panel for the virtual power plant. This can be achieved through the WPF.

4.2.3 Windows Communication Foundation, WCF

The WCF codenamed Indigo allows different software components to communicate. It consists of a new set of .NET libraries for use with the second version of the .NET Common Language Runtime environment. [16]
The Communication Foundation is used for building web services in managed code and incorporates functionality to talk to other distributed systems. This makes it our preferred platform for creating the services used in our solution.

4.2.4 Windows Workflow Foundation, WF

The WF provides the programming model for building workflow enabled applications. This technology could be used for modelling the solution but another approach was chosen because the WF primarily focuses on the modelling and support of business processes.

4.2.5 Windows CardSpace

This technology has its primary focus on the digital identity of users. It could be used for creating a profile for each production site or the local users configuring the system from here. This is however considered a part of the security package and the functionality has not been considered further.

4.3 Development Techniques

This section briefly describes the development techniques used in developing the prototype framework and the proof of concept implementation of it.
The techniques are combined for a number of reasons. One of them being, to benefit from the strengths and avoiding the weaknesses of each. The other is the time frame of the project. The general recommendation of the Framework design is to develop different test scenarios that need to use the framework, in a manner of specifying the interface to the framework from the angle of the end users. The time frame does however not allow the implementation of numerous solutions to optimize the framework structure.

[16] McMurtry, Mercuri, Watling, Winkler, Windows Communication Foundation Unleashed, page 1

4.3.1 Test-driven Development, TDD

The test driven development scenario is the newest Microsoft recommendation for producing stable software when doing extreme programming. The basic concept is to design a test case as the first step and then afterwards implementing the code to pass this test.

4.3.2 Framework Design

The basic purpose of a framework is to standardize the integration of a solution using the designed protocol. The components available in the framework are creating a uniform, secure and reliable interface designed to accommodate the necessary security and performance requirement for a successful implementation.

The framework should be

- Designed to evolve
- Consistent Design
- General-Purpose part of framework

The framework design principle recommended by the designers of the Microsoft .NET Framework is to base the framework design on specific user scenarios and use this experience to design a supporting object model.[17] The framework design should be intuitive, self-documenting and simple scenarios able to be implemented without the need for reference documentation.[18]

4.4 Security

The security aspect of the solution designed is very important giving a number of aspects. The most conspicuous reason is the money involved. This creates a potential possibility for offences against the properties of the parties involved.
Production units in between can also speculate in production margin and economy of their competitors in an effort to maximize their profit or harm their competitors.
A denial of service (DoS) attack can also be used to disable communication to the controlling service of the virtual power plant, rendering all the production units in the dark. This can be used to hurt the parties involved or for the purpose of blackmail.
The basic security concepts are covered as standard features in the programming environment available. The basic security concepts are defined as confidentiality, integrity and availability. The concepts are to be applied at all levels of the communication, from e-mails to the system protocols. The focus here will however only be at the protocols at hand, and the general security aspects should be defined in a surrounding IT security strategy.

4.4.1 Confidentiality

The information shared between entities must remain confidential. This can easily be complied by the use of the embedded encryption solutions available in the WS-Security specification.

4.4.2 Integrity

The integrity concept is a bit more difficult to define than the information confidentiality, as the basic option is to guarantee that the information shared reaches the recipient without being altered.

[17] Cwalina, Abrams, Framework Design Guidelines, page 13
[18] Cwalina, Abrams, Framework Design Guidelines, page 24

If a price signal is altered before the destination is reached, clients make decisions on incorrect information. This is of course a fatal consequence for the system.
The integrity is however a standard component of the WS-Security specification.

4.4.3 Availability

Insurance that the service is always available is a very important criterion for the producers. This can however not be guaranteed in a solution as the above. Redundancy in the system networks combined with power backup and a DoS defense can help reduce the down time and time frames where the system is unavailable, but the system needs a backup plan for when to act if the main system becomes unavailable.
The information shared between clients is limited or nonexistent. This means that if a producer looses contact to the main system, the protocol itself does not know if all clients are without contact or if the problem is local for the current producer.
This needs to be taken into consideration when decision is made upon whether the client should continue or suspend production.

4.4.4 Security in the virtual power plant framework

As specified above the aspects of confidentiality and integrity is a general part of the WS-Security specification. This means that the development environments support enabling and disabling encryption and signatures as a simple property to enable or disable.
The virtual power plant framework in general is a proof of concept example of a simple framework based on the two theories of price control. The general problem of confidentiality and integrity is therefore ignored at this point, but the solution developed could easily be wrapped in the standards of the WS-Security specification to comply with the security needs of an actual implementation of the system.
The problem faced in availability is however not a simple standard component to add on to the project. A lot of things can be done to minimize the occurrence of the down time, but no matter how much focus is put into this optimization the protocol for handling an actual offline system needs to be defined. This will be thought of as a part of the implementation.

5 Implementation

The purpose of this chapter is to design and implement parts of a prototype framework to be used as a proof of concept analysis model for the proposed solution. The prototype is based on the conceptual design which solves problems identified in the analysis in chapter 2. The framework itself is a fast track for developing new controlling applications when implementing power production systems.

The framework will support multiple module based production controlling methods where the price signal model and the bid control model will be implemented in this prototype.

A program to simulate a number of clients will be created. To easily control the clients a server interface will be implemented to control the virtual power plant and view statistics of prior and current client productivity.

5.1.1 Class Descriptions

This section will go through each class describing the class usage and the properties and methods defined in the class. The class diagram available from the implementation in Visual Studio is shown below. This diagram does not show any UML relations except from direct heritage which is only used in the ControlType classes.

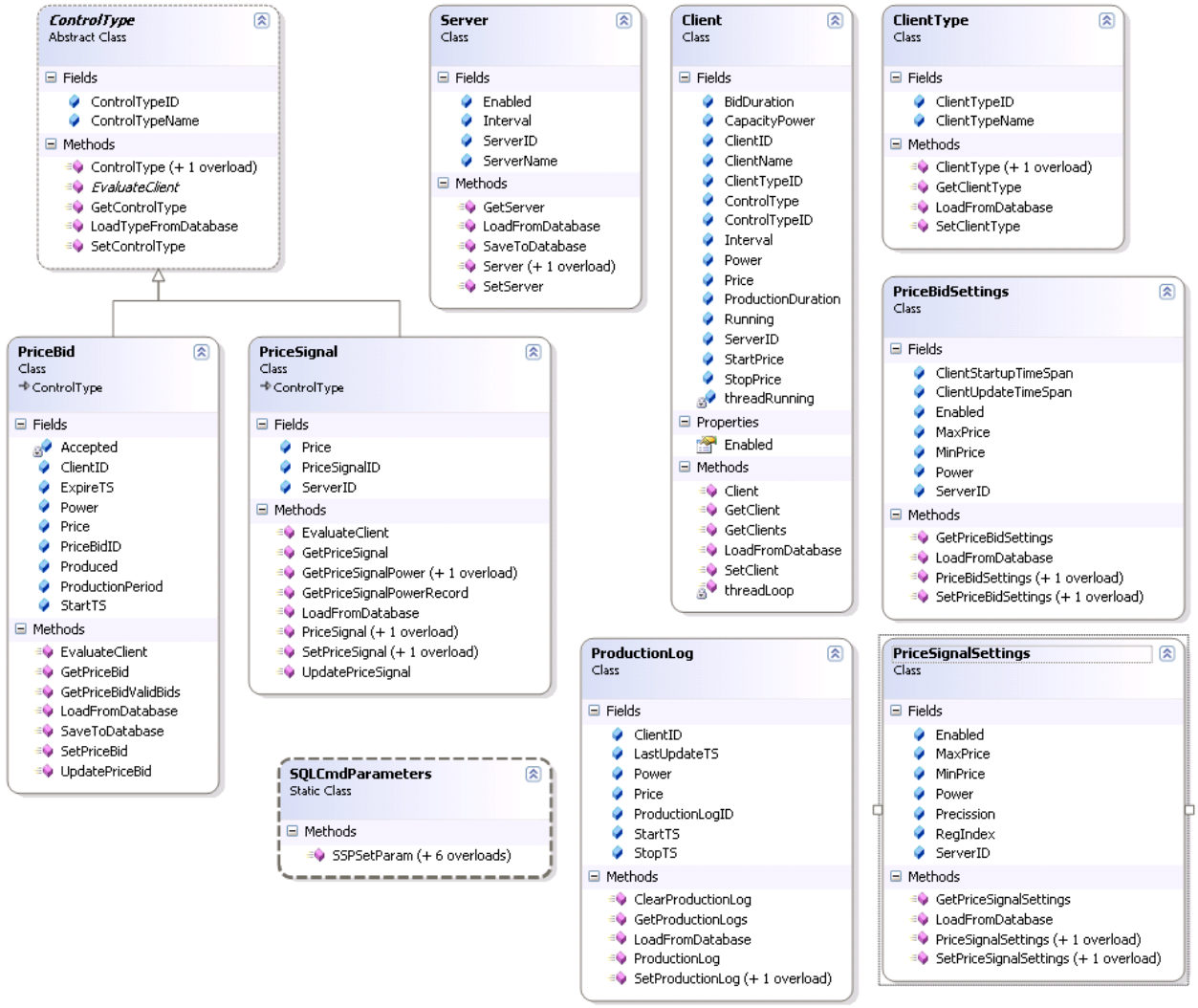


Figure 5.1 – GFPT Class Diagram

5.1.2 Client

The client class is used for storing information related to the client implementation. The client keeps a lot of general information used as an interface between the production equipment and the control type of the server. These values are used in a continuous loop where each iteration evaluates them against the configured control type to decide whether the production should begin, continue or end.

Fields	Description
BidDuration	Indicates the preferred bid duration
CapacityPower	The production capacity of the client
ClientID	The unique id to identify the client.
ClientName	The name of the client.
ClientTypeID	The unique id to identify the client type.
ControlType	The control type object.
ControlTypeID	The unique id to identify the client control type.
Interval	The client update interval.
Power	The current production of the client.
Price	The current production price for the client.
ProductionDuration	Indicates the preferred production duration.
Running	Indicates if the client is running / producing.
ServerID	The unique id to identify the server.
StartPrice	The price needed for the client to begin producing if stopped.
StopPrice	The minimum feasible price for a client to continue a running production.
ThreadRunning	Private indication of whether the clients internal thread is running.

Properties	Description
Enabled	Public indication of whether the clients internal thread is running.

Methods	Description
Client	Client constructor.
GetClient	Returns a client from the database.
GetClients	Returns a list of clients from the database.
LoadFromDatabase	Loads the client object from the database.
SetClient	Saves a client to the database.
ThreadLoop	The thread loop continuously updates the client when the client is enabled.

5.1.3 Control Type

The control type class is a template class as defined earlier in the design pattern section. The class templates any current or future control type and any of these should inherit from this template class.

Fields	Description
ControlTypeID	The unique ID to identify the control type.
ControlTypeName	The name of the control type.

Methods	Description
ControlType	Class constructor.

EvaluateClient	Evaluate Client against the control type criteria.
GetControlType	Gets a control type from the database.
LoadTypeFromDatabase	Loads a control type object from the database.
SetControlType	Saves a control type to the database.

Price Bid

The price bid class holds the implementation of the price bid control type. This class defines what a price bid is and includes all methods for handling this control type for both the server as well as the client.

Fields	**Description**
Accepted	The accepted value indicates if the server has accepted the bid.
ClientID	The unique id to identify the client that produced the bid.
ExpireTS	The time stamp for when the bid expires
Power	The Power proposed by the bid.
Price	The price that the client wants for the production.
PriceBidID	The unique ID to identify the price bid.
Produced	The produced value indicates if the production has ended.
ProductionPeriod	The period in seconds that the production will last.
StartTS	The time stamp for when client started the production.

Methods	**Description**
EvaluateClient	The price bid control types evaluation of the client.
GetPriceBid	Gets a price bid from the database.
GetPriceBidValidBids	Gets a list of valid price bids for a client or a server.
LoadFromDatabase	Loads a price bid object from the database.
SaveToDatabase	Saves a price bid object to the database.
SetPriceBid	Saves a price bid to the database.
UpdatePriceBid	Used by the server to accept new price bids when needed.
AcceptFeasibleBids	Accepts feasible bids for a server.

Price Bid Settings

The price bid settings class holds the settings used by a server running the price bid control type. The server uses this configuration to evaluate which bids to accept and how much power the virtual power plant should produce.

Fields	**Description**
ClientStartupTimeSpan	The maximum time span used by an acceptable client to begin production.
ClientUpdateTimeSpan	The minimum time span in which the client should synchronize with the server
Enabled	Property to enable or enable the price bid.
MaxPrice	The maximum acceptable price.
MinPrice	The minimum acceptable price.
Power	The combined power of all bids.
ServerID	The unique id to identify the server for which the settings apply.

Methods	Description
GetPriceBidSettings	Gets a price bid settings record from the database.
LoadFromDatabase	Loads a price bid settings object from the database.
PriceBidSettings	Class constructor.
SetPriceBidSettings	Saves a price bid settings record to the database.

Price Signal

The price signal class holds the implementation of the price signal control type. This class defines what a price signal is and includes all methods for handling this control type for both the server as well as the client.

Fields	Description
Price	The last price fetched by the price signal object.
PriceSignalID	The unique id to identify the price signal.
ServerID	The unique ID to identify the server.

Methods	Description
EvaluateClient	The price signal control types evaluation of the client.
GetPriceSignal	Gets a price bid settings record from the database.
GetPriceSignalPower	Gets the current power of all clients running the price signal control type.
GetPriceSignalPowerRecord	Gets the current Power of all clients running the price signal control type, returned as a dataset.
LoadFromDatabase	Loads a price signal object from the database.
PriceSignal	Class constructor
SetPriceSignal	Saves a price bid signal record to the database.
UpdatePriceSignal	Sets the current Price Signal Price.

Price Signal Settings

The price signal settings class holds the settings used by a server running the price signal control type. The server uses this configuration to calculate a new control signal and evaluate how much power the virtual power plant should produce.

Fields	Description
Enabled	Identifies whether the price signal is enabled.
MaxPrice	The maximum price to send out through the signal.
MinPrice	The minimum price to send out through the signal.
Power	The ideal energy to be reached by the price signal producers.
Precission	The required precision of the combined result.
RegIndex	The regulating index to increase / decrease the price signal.
ServerID	The unique id to identify the server.

Methods	Description
GetPriceSignalSettings	Loads a price signal settings record from the database.
LoadFromDatabase	Loads a price signal settings object from the database.
PriceSignalSettings	Class constructor.

SetPriceSignalSettings	Saves a price signal settings record to the database.

Production Log

The production log class is used by the client to store information about how much power is produced at a given time as well as the price agreed with the server. The production log class is used to store this information and generate a log in the database whenever the variables change.

Fields	**Description**
ClientID	The unique id to identify the client.
LastUpdateTS	The last update of the production record.
Power	The production power.
Price	The production price.
ProductionLogID	The unique id to identify the production log.
StartTS	The production start time stamp.
StopTS	The production stop time stamp.

Methods	**Description**
ClearProductionLog	Removes all log entries for a client. If no client is specified, the method removes all production logs.
GetProductionLogs	Returns all production log records from the database.
LoadFromDatabase	Loads a production log object from the database.
ProductionLog	Class constructor.
SetProductionLog	Saves a production log record to the database.

Server

The server class is used to store information about a given server. This gives the possibility of using multiple servers and different client control models in the same solution based on the same backend database.

Fields	**Description**
Enabled	Indicates whether the server is enabled.
Interval	The server update interval.
ServerID	The unique id to identify the server.
ServerName	The name of the server.

Methods	**Description**
GetServer	Gets a server record from the database.
LoadFromDatabase	Loads a server record from the database.
SaveToDatabase	Saves a server record to the database.
Server	Class constructor.
SetServer	Saves a server record to the database.

Client Type

The client type class holds information about client types. The different client types can be used to differentiate between clients and divide them into categories.

Fields	Description
ClientTypeID	The unique id to identify the client type.
ClientTypeName	The name of the client type.

Methods	Description
ClientType	Class constructor.
GetClientType	Loads a client type record from the database.
LoadFromDatabase	Loads a client type object from the database.
SetClientType	Saves a client type record to the database.

5.2 Database design

The GFPT is based on trading information between the clients and the server. A lot of the decisions within the GFPT are based upon this data. The database is the central point of the framework information flow.
The database contains a number of tables in which data is stored. Each table has a fixed number of columns, representing each attribute and a number of rows, representing entries in the table. The database is normalized to the extent of the third normal form.

5.2.1 Normalization

Normalization of a database is to change the design of the database to avoid duplicates and avoid possible inconsistencies. This improves conditions when maintaining, expanding and updating tables. Normalization can be done to various degrees described by a number of normal forms. A database in 3rd degree normal form will also comply with 2nd as well as 1st normal form.

First normal form

Normalization in first normal form removes duplicate records. Each table must have a primary key which is unique identifier in one column or in a combination of several columns. The database complies with the first normal form.

Second normal form

Normalization in second normal form is when data in one column which is functional dependant on data in another column is moved to a new table. The database complies with the second normal form.

Third normal form

Normalization in third normal form has been achieved when no data is transitively dependent on data in another column which is not a primary key. The database complies with the third normal form.

Entity Relationship Diagram

The ER diagram displays an overview of the database entities and their relationship to each other.

Figure 5.2 – Database ER Diagram

5.2.2 Tables

Each table will be described in detail in the following section.

PriceSignal

Column	Description
PriceSignalID	Unique id to identify the record.
ServerID	Link reference to the server using the price signal.
Price	Price value of the price signal.
AddTS	Timestamp for when the record was added.

PriceBidSettings

Column	Description
ServerID	There can only be one record for each server. The ServerID is therefore used as the primary key as well.
Power	The energy the price bid control should produce.
MinPrice	The minimum price for an acceptable bid.
MaxPrice	The maximum price for an acceptable bid.
ClientUpdateTimeSpan	The minimum time span in which the client should synchronize with

	the server.
ClientStartupTimeSpan	The maximum time span used by an acceptable client to begin production
Enabled	Indicates whether the price bid is control type is enabled.

PriceBid

Column	Description
PriceBidID	Unique id to identify the record.
ClientID	Link reference to the client that made the bid.
Power	The energy included in the bid.
ProductionPeriod	The period for which the production should occur.
Price	The price of the bid.
ExpireTS	The timestamp for when the bid expires.
Accepted	Indicates whether the bid is accepted.
StartTS	Timestamp for when the production of the bid started.
Produced	Indicates whether the bid has been produced.

Server

Column	Description
ServerID	Unique id to identify the record.
ServerName	The name of the server.
Interval	The update frequency of the server.
Enabled	Identifies if the server is enabled.

PriceSignalSettings

Column	Description
ServerID	There can only be one record for each server. The ServerID is therefore used as the primary key as well.
Precission	The required precision of the acceptable produced amount.
RegIndex	Regulating index for regulating the price signal.
Power	The energy that should be reached by the price signal.
MinPrice	The minimum acceptable price for the price signal.
MaxPrice	The maximum acceptable price for the price signal.
Enabled	Identifies if the price signal is enabled.

Client

Column	Description
ClientID	Unique id to identify the record.
ServerID	Link reference to the server where the client is running.
ClientTypeID	Link reference to the client type.
ControlTypeID	Link reference to the control type.
ClientName	The name of the client.

ClientType

Column	Description
ClientTypeID	Unique id to identify the record.

ClientTypeName	Name of the client type

ProductionLog

Column	Description
ProductionLogID	Unique id to identify the record.
ClientID	Link reference to the client that produced.
Power	The energy produced.
Price	The agreed price for the power.
StartTS	Timestamp for when the production started.
StopTS	Timestamp for when the production ended.
LastUpdateTS	Timestamp for when the record was last updated.

ControlType

Column	Description
ControlTypeID	Unique id to identify the record.
ControlTypeName	Name of the control type

5.2.3 Stored Procedures

All communication between the framework and the database is done using stored procedures. The stored procedures are a set of SQL statements that are stored in the database in compiled form. This use of stored procedures has some advantages over doing this implementation in the belonging programs.

Stored procedures can be shared in between applications. This guarantee the same result in between programs as implementation is done using the same code. Stored procedures are compiled queries and therefore they perform better than new queries unknown to the server.

Data manipulation can be done locally inside the SQL server without transferring working calculations and result sets to the client. The only information sent between the client and the server is the request and the result.

As an example of this the *AcceptFeasibleBids* procedure are explained in detail after the stored procedure overview below.

Stored Procedure	Description
GetPriceSignalSettings	Returns a price signal settings record.
SetPriceSignalSettings	Stores a price signal settings record.
GetControlType	Returns a control type record.
SetControlType	Stores a control type record.
GetPriceBid	Returns a price bid record.
SetPriceBid	Stores a price bid record.
GetPriceSignal	Returns the current price signal
GetClientTypes	Get a list of available client types.
GetClient	Get a client record.
GetClients	Get a list of clients.
SetClient	Stores a client record.
SetProductionLog	Stores a production log record.
GetProductionLogs	Returns a production log error.
GetCurrentProduction	Returns the current production of the combined clients.

GetPriceBidSettings	Returns a price bid settings record.
SetPriceBidSettings	Stores a price bid settings record.
GetPriceSignalPower	Returns the current power produced by all clients running the price signal control type
SetPriceSignal	Sets a new price signal price.
GetPriceBidValidBids	Returns a list of all valid bids for the server.
ClearProductionLog	Clears the production log for a server or a client.
AcceptFeasibleBids	Accepts all feasible bids
GetAccumulatedProduction	Returns the accumulated production for a given period.
GetServer	Returns a server record.
SetServer	Stores a server record.

The implementation of the AcceptFeasibleBids is shown below. The statements perform some internal requests and it uses the result to perform a fetched cursor loop that updates the desired records.
If this is done at the program level, all statements would have to be transferred between the database and the server, evaluated at the server and the update statements resulted by the evaluation would have to be sent back to the database.
All this is now done inside the SQL database and only the request and the result is transferred back and forth. This gives the optimal performance.

```
-- =============================================================
-- Author:                  Johan Holkmann Jacobsen
--                          Rasmus Skovmark
-- Create date: 20070522
-- Description:             Accept feasible bids
-- =============================================================
CREATE PROCEDURE [dbo].[AcceptFeasibleBids]
  @ServerID AS UniqueIdentifier
AS
BEGIN
  DECLARE @RunTS AS DateTime
  SELECT @RunTS = GetDate()

  DECLARE @AverageStartupTime AS int
  SELECT
    @AverageStartupTime = (ClientUpdateTimeSpan+ClientStartupTimeSpan)/2
  FROM PriceBidSettings
  WHERE
    ServerID = @ServerID

  DECLARE @NeededCapacity AS Decimal
  SELECT @NeededCapacity=Power FROM PriceBidSettings WHERE ServerID = @ServerID

  DECLARE @CurrentCapacity AS Decimal
  SELECT
    @CurrentCapacity=IsNull(SUM(Power), 0)
  FROM PriceBid A
    INNER JOIN Client B ON A.ClientID = B.ClientID
  WHERE
    B.ServerID = @ServerID AND
    A.Accepted = 1 AND
    (A.StartTS+(A.ProductionPeriod/(86400.0))+ @AverageStartupTime) > @RunTS

  DECLARE @MissingProduction AS Decimal
  SELECT @MissingProduction = @NeededCapacity - @CurrentCapacity


  DECLARE tmp_Cursor CURSOR FOR
```

```
  SELECT
    A.Power, A.PriceBidID
  FROM PriceBid A
    INNER JOIN Client B ON A.ClientID = B.ClientID
    INNER JOIN PriceBidSettings C ON A.Price BETWEEN C.MinPrice AND C.MaxPrice
  WHERE
    B.ServerID = @ServerID AND
    A.Accepted = 0 AND
    A.ExpireTS > @RunTS
  ORDER BY Price ASC

  DECLARE @Power Decimal
  DECLARE @PriceBidID UNIQUEIDENTIFIER

  OPEN tmp_Cursor
  FETCH NEXT FROM tmp_Cursor INTO @Power, @PriceBidID
  WHILE (@@FETCH_STATUS <> -1)
  BEGIN
    IF(@@FETCH_STATUS <> 2)
    BEGIN
      IF(@NeededCapacity > 0)
      BEGIN
        SELECT @MissingProduction = @MissingProduction - @Power
        UPDATE PriceBid SET Accepted = 1 WHERE PriceBidID = @PriceBidID
      END
    END
    FETCH NEXT FROM tmp_Cursor INTO @Power, @PriceBidID
  END
  CLOSE tmp_Cursor
  DEALLOCATE tmp_Cursor
END
```

5.3 Program Implementation

The implementation is not limited to the framework, but also includes a sample implementation of a control application, a server and an application simulating a number of clients. These external applications are developed based on the exposed functionality of the framework. This is done exactly as intended in a real life scenario.
The client simulator is implemented so it can both simulate clients running by the price signal control type and clients running by the price bid control type. Both models could easily be implemented running parallel but the test scenario is done using either and therefore the client simulator is not tested using parallel control types.
The implementation of the surrounding applications of the GFTP is straight forward and can be found in the code section on the CD. This section will focus on the client implementation of the control types.
The client has a reference to any control type through a control type object, unaware of the type of this. Each client then has its own thread for updating. This thread is automatically started when the client is enabled.
The control type is declared as a control type object and since all control types inherit from this basic type they can be assigned as an instance of this object.

```
/// <summary>
/// The Control type object.
/// </summary>
public ControlType ControlType = null;
```

When the internal thread in the client makes its iteration it executes the EvaluateClient function with an instance of itself.

```
/// <summary>
/// The thread loop continuously updates the client when the client is enabled.
/// </summary>
void ThreadLoop()
{
  while (Enabled)
  {
    try
    {
      ControlType.EvaluateClient(this);
    }
    catch (Exception ex)
    {
      throw ex;
    }
    Thread.Sleep(Interval);
  }
}
```

The evaluation which is specific for each control type is then executed within the control type object. This enables the possibility for adding new control types without changing the client implementation.
The control type still has access to the client object since this is passed on to it as an object reference. The implementation of the EvaluateClient functions is shown below starting with the price signal.
The price signal is the simplest evaluation of a client. Basically this control type gets the newest price signal from the database and uses this to evaluate the start and stop prices of the client against this new price.
If the price is within the accepted price range for feasible production, the client is either started or production is continued. If the price is outside the range, the client is either stopped or not started depending on the current state.

```
/// <summary>
/// The price signal control types evaluation of the client.
/// </summary>
public override void EvaluateClient(Client CL)
{
    Decimal oldPrice = CL.Price;
    PriceSignal PS = new PriceSignal(CL.ServerID);
    CL.Price = PS.Price;
    // Stop Production if not feasable
    if (!CL.Running && (CL.Price >= CL.StartPrice)) { CL.Running = true; }
    // Stop Production if not feasable
    if (CL.Running && (CL.Price <= CL.StopPrice)) { CL.Running = false; }

    if (CL.Running)
    {
        CL.Power = CL.CapacityPower;
        ProductionLog.SetProductionLog(CL.ClientID, CL.Power, CL.Price);
    }
    else
    {
        ProductionLog.SetProductionLog(CL.ClientID, 0, CL.Price);
    }
}
```

The implementation of the price bid is in general the same evaluation, but the function is a bit more complex since more than one bid is used to make up the production plan.

The structure of the price bid evaluation is the following:

1. Get all active bids for the client
2. End production of bids where production is finished
3. Begin production of bids accepted but not currently started
4. Make a new bid on the remaining capacity

```
/// <summary>
/// The price bid control types evaluation of the client.
/// </summary>
public override void EvaluateClient(Client CL)
{
    CL.Power = 0;
    CL.Price = 0;
    Decimal tmpPower = 0;
    Decimal tmpPrice = 0;
    Decimal remainingCapacity = CL.CapacityPower;
    Boolean tmpRunning = false;
    DateTime LastAcceptedBidStopTS = DateTime.MinValue;
    PowerServiceClient proxy = new PowerServiceClient("BasicHttpBinding_IPowerService");
    DataSet PriceBids = proxy.GetPriceBidValidBids(Guid.Empty.ToString(), CL.ClientID.ToString());
    PriceBid Bid = new PriceBid();
    PriceBidSettings PBSettings = new PriceBidSettings(CL.ServerID);
    DateTime FetchTS = DateTime.MinValue;
    for (int i = 0; i < PriceBids.Tables[0].Rows.Count; i++)
    {
        Bid.PriceBidID = (Guid)PriceBids.Tables[0].Rows[i]["PriceBidID"];
        Bid.ClientID = (Guid)PriceBids.Tables[0].Rows[i]["ClientID"];
        Bid.Power = (Decimal)PriceBids.Tables[0].Rows[i]["Power"];
        Bid.ProductionPeriod = (int)PriceBids.Tables[0].Rows[i]["ProductionPeriod"];
        Bid.Price = (Decimal)PriceBids.Tables[0].Rows[i]["Price"];
        Bid.ExpireTS = (DateTime)PriceBids.Tables[0].Rows[i]["ExpireTS"];
        Bid.Accepted = (Boolean)PriceBids.Tables[0].Rows[i]["Accepted"];
        if (PriceBids.Tables[0].Rows[i]["StartTS"].ToString() != "")
        {
            Bid.StartTS = (DateTime)PriceBids.Tables[0].Rows[i]["StartTS"];
        }
        else
        {
            Bid.StartTS = DateTime.MinValue;
        }
        Bid.Produced = (Boolean)PriceBids.Tables[0].Rows[i]["Produced"];
        FetchTS = (DateTime)PriceBids.Tables[0].Rows[i]["FetchTS"];
        // Accepted but not started
        if (Bid.Accepted)
        {
            tmpRunning = true;
            // Save the time stamp for the longest running bid.
            if ((Bid.StartTS.AddSeconds(Bid.ProductionPeriod)) > LastAcceptedBidStopTS)
            {
                LastAcceptedBidStopTS = Bid.StartTS.AddSeconds(Bid.ProductionPeriod);
            }
            // Start bid if not started
            if (Bid.StartTS == DateTime.MinValue)
            {
                Bid.StartTS = FetchTS;
                Bid.SaveToDatabase();
            }
            // End bid if time is up
            if ((Bid.StartTS != DateTime.MinValue) &&
                (Bid.StartTS.AddSeconds(Bid.ProductionPeriod) < FetchTS))
            {
                Bid.Produced = true;
                Bid.SaveToDatabase();
            }
            // Remove bid from remainin capacity because
            // the bid is running out an needs to be taken
            // over by a new bid.
            if (FetchTS > Bid.StartTS.AddSeconds(PBSettings.ClientStartupTimeSpan +
                PBSettings.ClientUpdateTimeSpan))
            {
```

```
                // Calculate new price amount
                //
                // (OldPower*OldPrice)+(NewPower*NewPrice)
                // ---------------------------------------
                // (OldPower + NewPower)

                tmpPrice = ((tmpPower * tmpPrice) + (Bid.Power * Bid.Price)) /
                            (tmpPower + Bid.Power);
                // Calculate new power amount
                tmpPower = tmpPower + Bid.Power;
                // Calculate remaining capacity
                remainingCapacity = remainingCapacity - Bid.Power;
            }
        }
        else
        {
            // Calculate remaining capacity
            remainingCapacity = remainingCapacity - Bid.Power;
        }
        if (Bid.Accepted && !Bid.Produced)
        {
            if ((CL.Power + Bid.Power) > 0)
            {
                Decimal ClientUnitPrice = 0;
                if (CL.Power > 0)
                {
                    ClientUnitPrice = CL.Price / CL.Power;
                }
                CL.Price = (((ClientUnitPrice) + (Bid.Price / Bid.Power)) * 2) *
                           (CL.Power + Bid.Power);
            }
            else
            {
                CL.Price = 0;
            }
            CL.Power += Bid.Power;
        }
    }
    // Make a bid for the remaining capacity
    if (remainingCapacity > 0)
    {
        PriceBid newBid = new PriceBid();
        newBid.PriceBidID = Guid.NewGuid();
        newBid.ClientID = CL.ClientID;
        newBid.Power = remainingCapacity;
        newBid.ProductionPeriod = CL.ProductionDuration;
        if (FetchTS == DateTime.MinValue)
        {
            FetchTS = DateTime.Now;
        }
        if(CL.Running)
        {
            newBid.Price = CL.StopPrice;
            newBid.ExpireTS = LastAcceptedBidStopTS;
        }
        else
        {
            newBid.Price = CL.StartPrice;
            newBid.ExpireTS = FetchTS.AddSeconds(CL.BidDuration);
        }
        if (newBid.ExpireTS < DateTime.Now)
        {
            newBid.ExpireTS = DateTime.Now;
        }
        newBid.Accepted = false;
        newBid.StartTS = DateTime.MinValue;
        newBid.Produced = false;
        newBid.SaveToDatabase();
    }
    CL.Running = tmpRunning && (CL.Power > 0);
    if (CL.Running)
    {
        ProductionLog.SetProductionLog(CL.ClientID, CL.Power, CL.Price);
```

```
    }
    else
    {
        ProductionLog.SetProductionLog(CL.ClientID, 0, CL.Price);
    }
}
```

The above implementations of the different EvaluateClient functions in the two control types shows the differences in complexity as well as the ability of creating new and more advanced control types in a future solution.

5.4 Test

The entire implementation is done as a proof of concept implementation. The focus has been on testing the behavior and reliability of the control types instead of exposing vulnerabilities in all parts of the code.
Testing was done at function level and the exceptions raised when errors occur in this level are always raised to the applications above for explicit handling.
These errors are exposed to the user by standard message boxes in the client simulator and the control application. The server however does not have a user interface and a log file is used instead. This log file includes start up times and a timestamp of errors and the modules in which they occur.

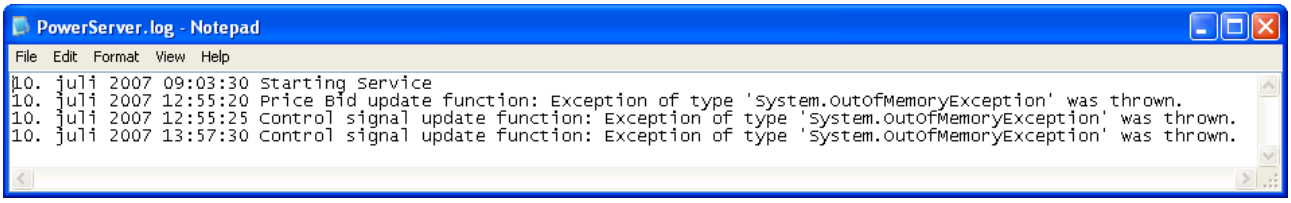


Figure 5.3 - Example of errors in the log file

Testing of the combined solution is limited to the above since a real life implementation needs an extension of the framework as well as a number of applications build on the framework. These applications are responsible for doing the actual error handling, as long as the framework either returns a trustworthy result or fails.

5.5 Conclusion

The framework is a simple and very easy to use component when implementing solutions based on the GFTP.
The implementation where each client runs in its own thread is a simple integration where a new solution can just add an instance of a client to the existing control software and the entire integration into the GFTP is implemented. This design is however not very efficient when simulating a number of clients. The test performance is very poor because a realistic scenario would be of 10,000 clients running on 10,000 computers each client using one thread per computer. The test scenario however is 10,000 clients on the same computer giving an application of 10,000 threads, which is more than our test environment could handle.

6 Case Study

6.1 Introduction

The purpose of the case study chapter is to use the prototype and a simulation program to test how the GFPT works in a (simulated) reality. The program simulates a number of clients communicating with the server across the internet and tests all functional requirements. This simulation will then be used to analyze the actual performance and stability of the two power control models and the framework as a whole.
The purpose of this chapter is to document the case studies, which investigates the simulation of the framework in working, whether it fulfills the project's requirements and problem definition or not, and if it is a good solution to the problems of power trading discussed in chapter 2.

6.2 System overview

The system used in the case studies has three basic modules:

- Client simulator
- Server
- Control

The server is the core of the framework which handles communication and price regulating logic.
The control interface is a part of the framework and makes it possible for the administrator to set up and change the energy production, marginal values etc.
Figure 6.1 shows how the Client simulation is a program, which uses the framework to create a number of virtual clients. Each client exists in its own thread. A graphical user interface is created to control client simulation.

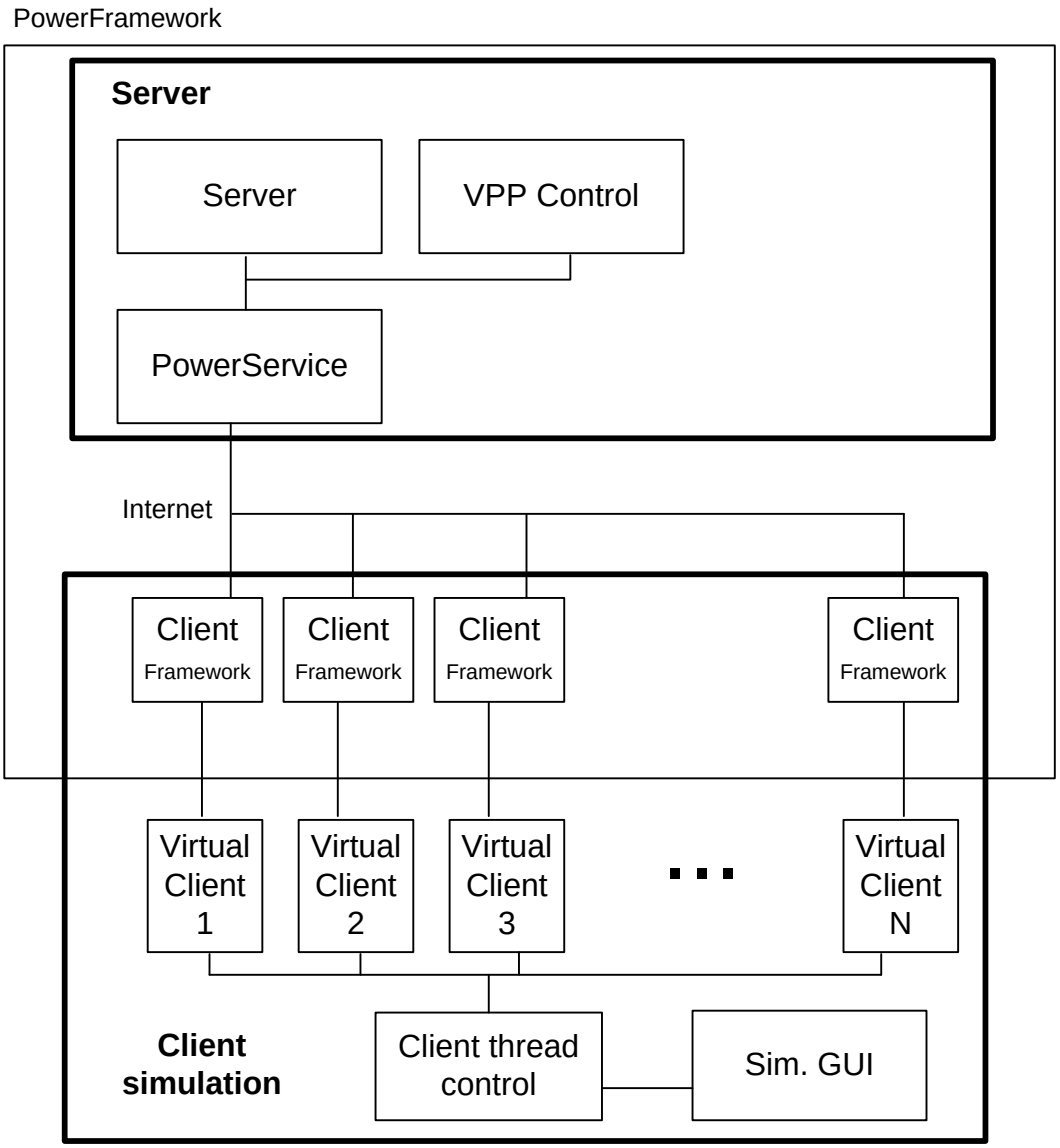


Figure 6.1 shows an overview of the system used in the case study. A program simulates a number of clients by creating a thread for each virtual client.

6.2.1 Client simulation

Client simulation is a program that simulates a number of clients. The program starts each client in its own thread thus making each thread function as a virtual client. Each client uses the framework to trade power.

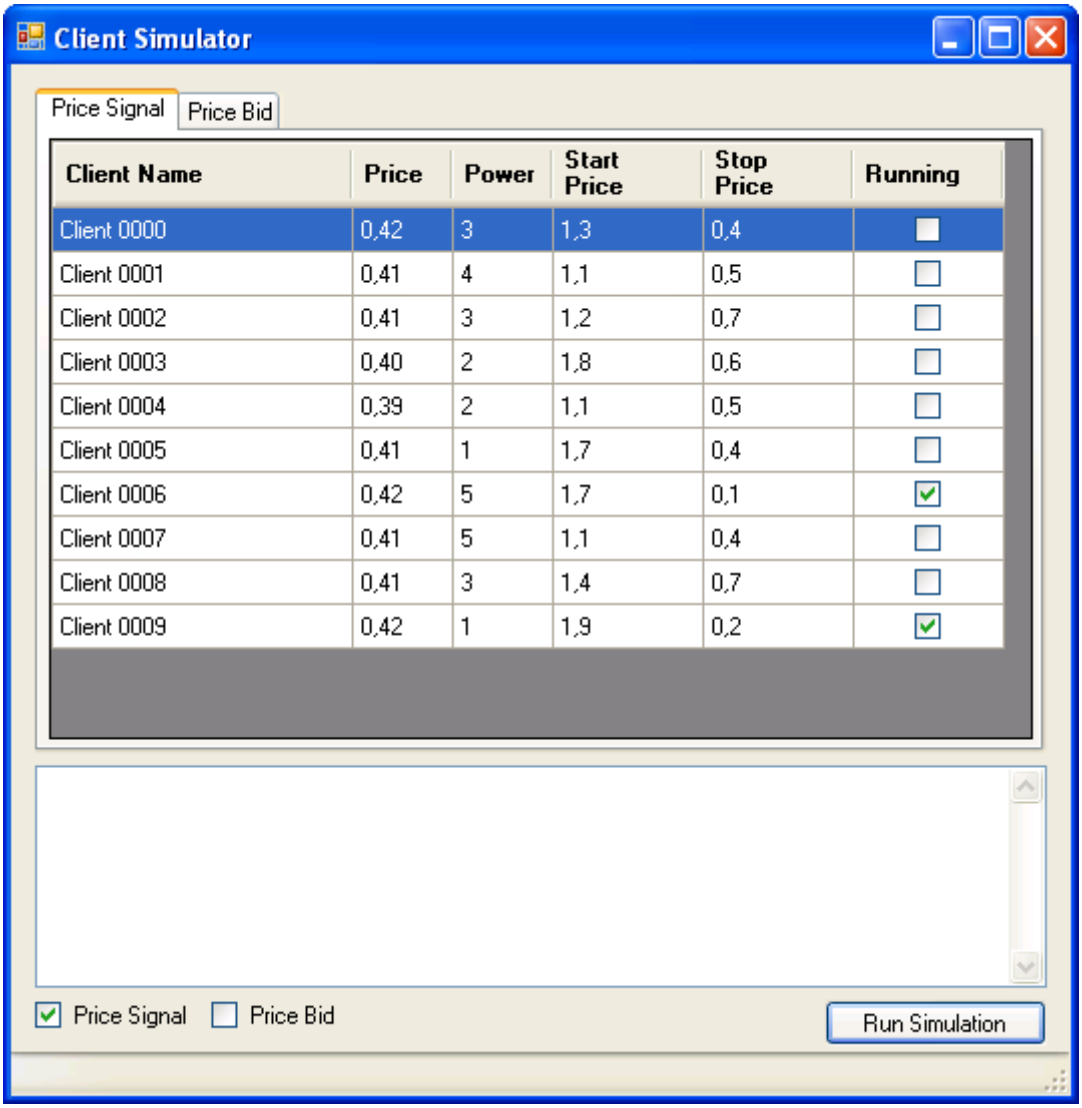


Figure 6.2 shows the Client simulator program with 10 active clients using the price signal model.

The user interface of the client simulator allows the administrator to use either the price signal or the price bid model or both. The program lists the number of active clients and their respective data. A column named running is indicating whether or not the client is currently producing. The program is a standalone application. There is no information on the value of the price signal or bid process. The control interface of the framework handles this.
The client simulator is started by clicking Run Simulation. In Figure 6.2 a simulation of 10 clients is shown. Client 6 and 9 is producing.
The simulator for the price signal shows Client name, the value of the price signal the last time the client has updated, the power level the client wants to produce, minimum start price, minimum stop price.
In the following table variables for client simulation is shown. To make the simulation more realistic all clients have randomized variables.

Variables	**Values**
Number of clients	10
Clients update interval	10-20 seconds

Power level	1-5 kW
Start price	1-2
Stop price	0-1
Bid production duration	60-360 seconds
Bid valid duration	30-60 seconds

Figure 6.3 Client simulation variables

A screenshot from the simulation of the price bid model is shown in Figure 6.3. The window show client name, price (when producing), power level (when producing), start price, stop price and whether the client is producing or not (running). In Figure 6.4 clients 502, 503, 505, 506, 507 and 508 have had their bids accepted and are now producing.

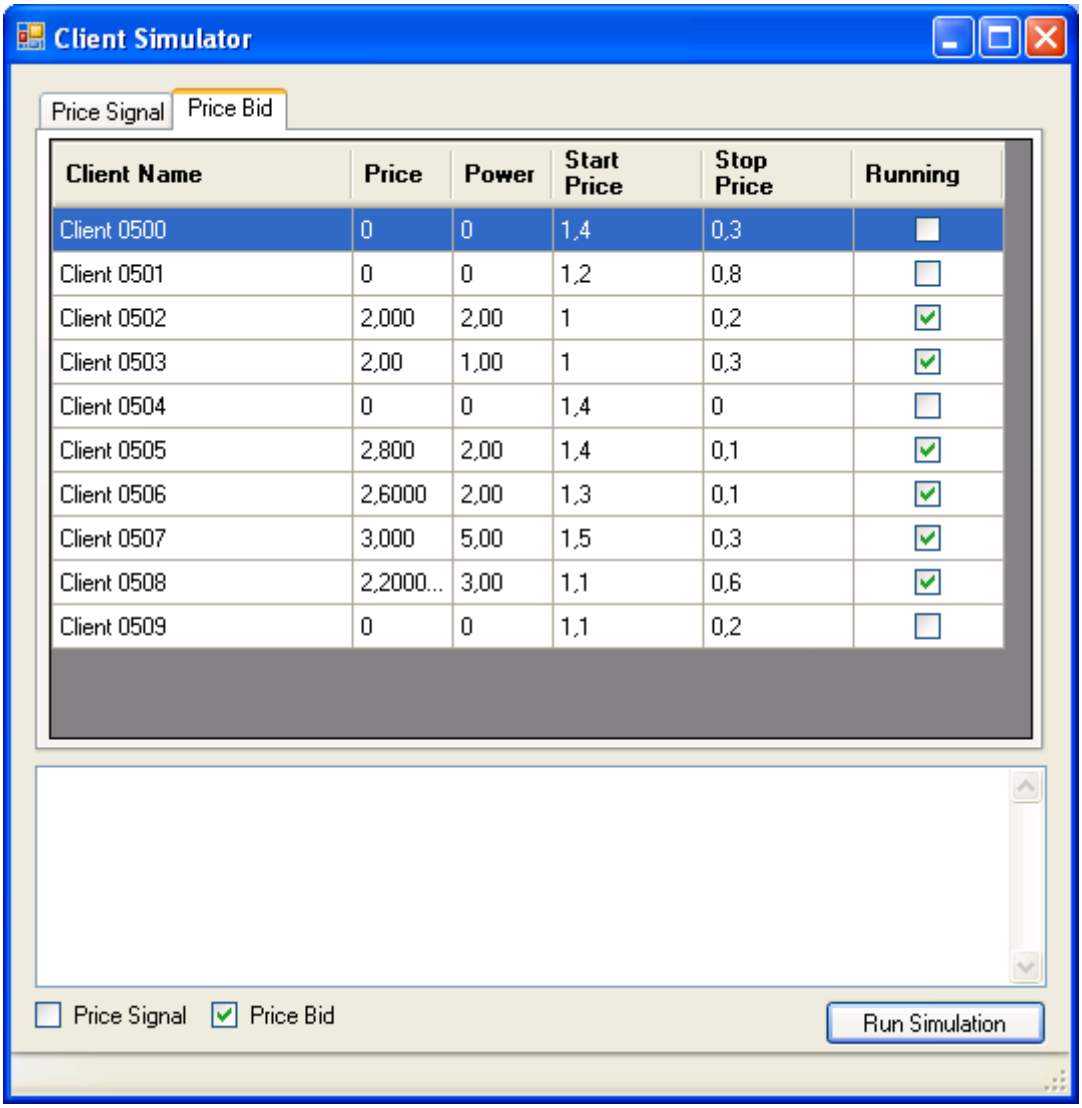


Figure 6.4 Client simulation of price bid model

6.2.2 Server

The server has the communication and logic of the power control. There is no user interface for the server, but when the service is running it can be seen in the windows services as in Figure 6.5 where it is named Power Server.

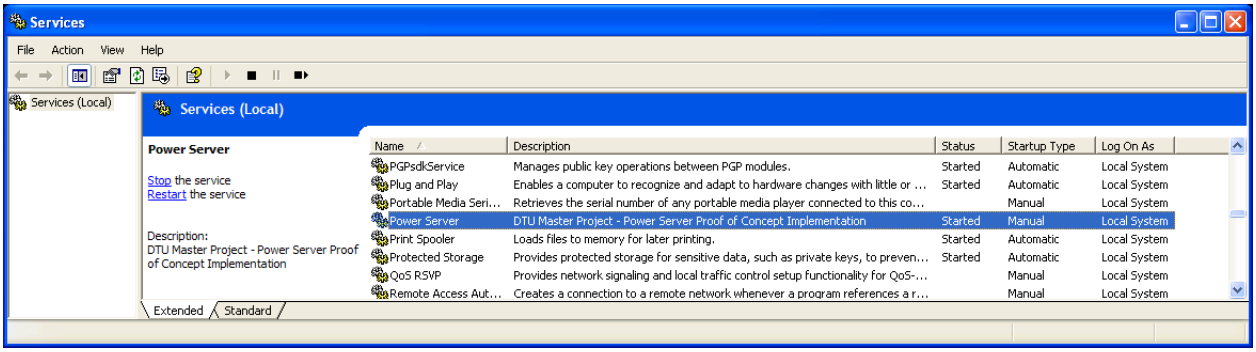


Figure 6.5 Power Service

6.2.3 Control Interface

The control interface is the administrator's tool to set up variables and change them. The control interface has all the necessary information the administrator need. For each power control model it shows the variables which can be altered. It also shows the current power level.
In this project the software does not calculate if the power production throughout a period of time has been too low or too high. In the control interface it is only possible to set the wanted production level, while statistics on already produced power can be calculated and showed by using the production log in the database. This task could easily be implemented to be done automatically.

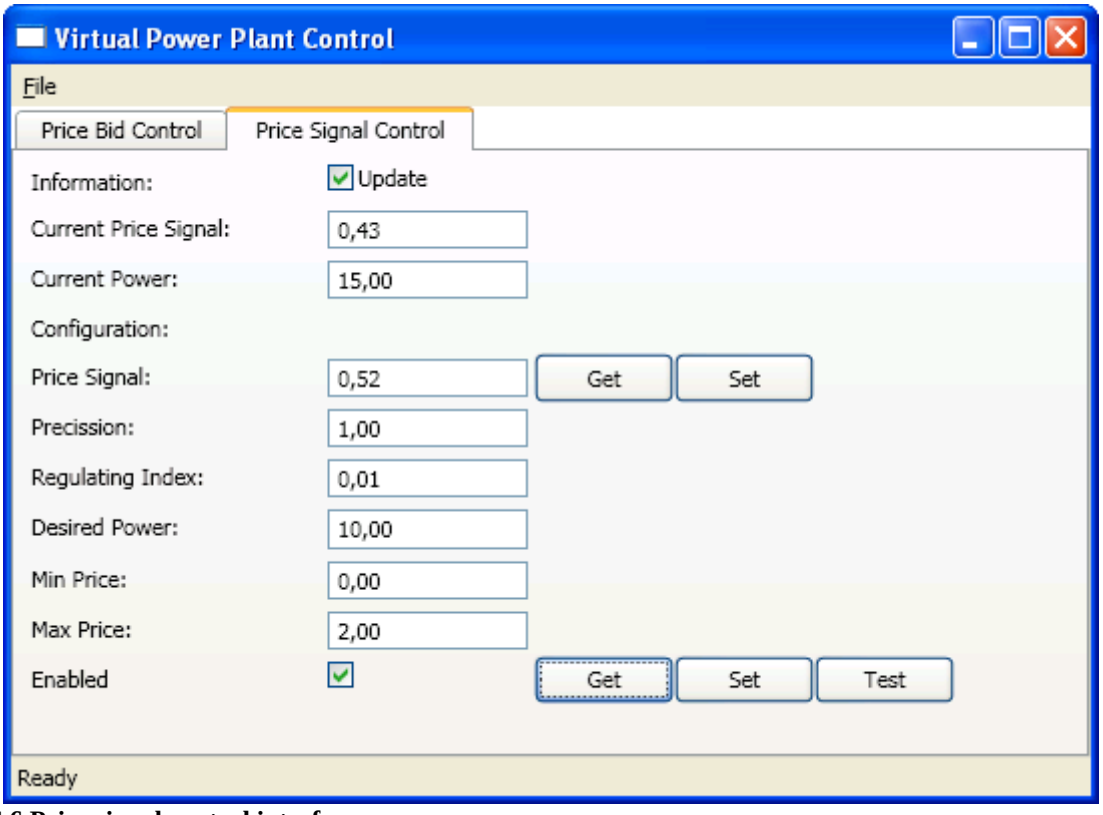


Figure 6.6 Price signal control interface

In Figure 6.6 the control interface is illustrated by a screenshot taken while the price signal model was in use. The variables in the control user interface are described in the following table.

Variables	Description
Update	If checked, the price signal is in progress and updating with a regular interval.
Current Price Signal	The price signal as it is right this moment.
Current Power	Denotes the power level the clients are producing at present time.
Price Signal	It is possible to get the price signal from the database or set it with these two buttons.
Precision	If current power level is within this amount from the desired level the price signal stops adjusting. This is the margin.
Regulating Index	The amount the price signal will adjust in each update if not the desired power level matches the current power level.
Desired Power	The power level the framework wants to produce.
Min Price	The lowest price allowed for the price signal.
Max Price	The highest price allowed for the price signal.
Enabled	If checked the price signal is enabled and clients can trade.
Get	Gets values of the above variables from the database.
Set	Sets values in the database.

Table 20 Price Signal Control Interface Variables

In the Virtual Power Plant Control there is also the option of running the bid model. Figure 6.7 shows the user interface of the bid model.

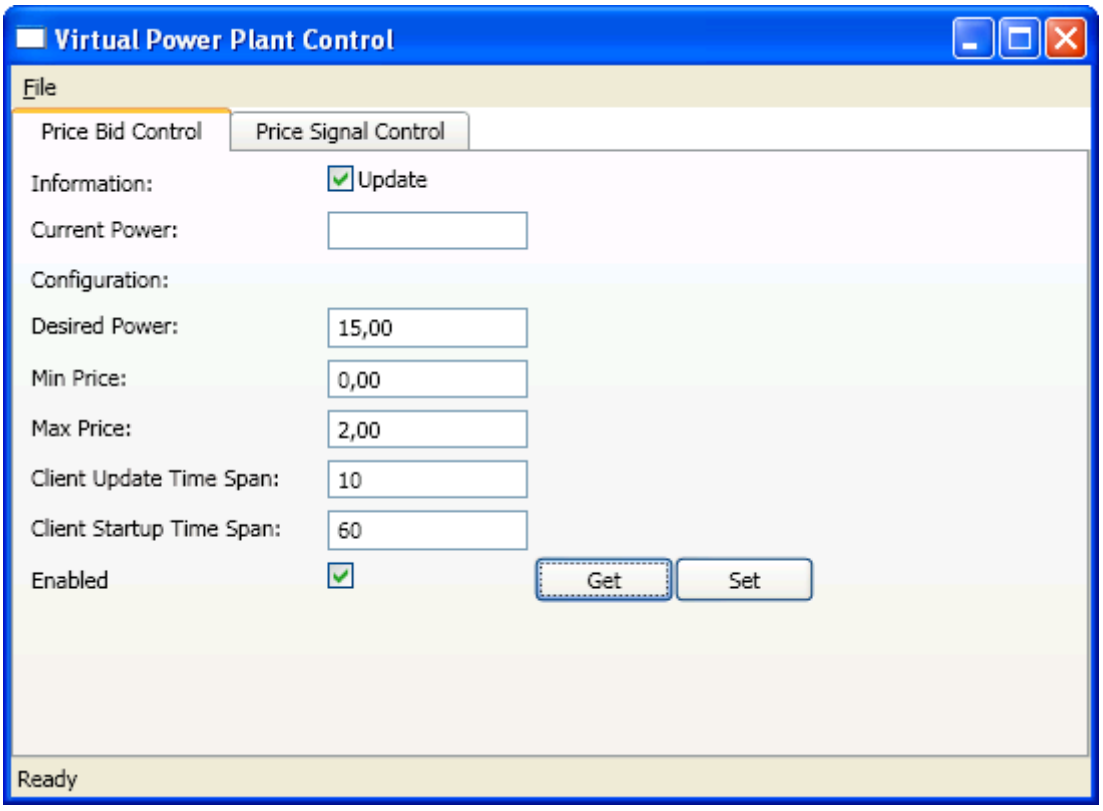


Figure 6.7 Bid model control interface

The variable in the bid model control user interface are described in the following table.

Variables	Description
Update	If checked, the bid model is in progress and updating with a regular interval.
Current Power	Denotes the power level the clients are producing at present time.
Desired Power	The power level the framework wants to produce.
Min Price	The lowest price allowed for bids.
Max Price	The highest price allowed for bids.
Client Update Time Span	How often a client must update.
Client Startup Time Span	The time a client on average use on starting up its production.
Enabled	If checked the price signal is enabled.
Get	Gets values of the above variables from the database.
Set	Sets values in the database.

Table 21 Bid model control interface variables

6.3 Testing

In the case studies the framework prototype is tested by the simulation of a realistic scenario. Each case study has a number of simulated clients, who wants to produce power. The program could also have been tested with real clients (producing units), but experience shows that this often increases the amount of testing significantly due to trouble shooting.
The case study investigates if a realistic simulation of 10 clients will stabilize and if it does how long it will take.
The simulations have been run on 2 different computers:

- Laptop computer, 1,6 GHz P4, 1 GB RAM
- Stationary computer, Core 2 duo, 2,4 GHz, 2 GB RAM

The framework is meant to be able to handle anything from 10 to 10,000 clients. The more clients that join the framework the more efficient it will be assuming the hardware is updated correspondingly to match the demands. This means that if the framework is functional with 10 clients it is also functional with 10,000 clients. Because of limited resources the tests carried out in this case study only has 10 clients.

6.3.1 Functional Requirement Test

In the following section each case study will test the prototype for a specific situation but also for a number of functional requirements specified in chapter 2. All graphs in the following sections are made based on data stored by logging production information from the simulation program. Each time a client update, change power level or price in its production it is saved in the database. This is one of the functional requirements of the project. All functional requirements will be tested and in fact most are used in each case study. In the first case study of the both price signal and bid model there is an additional description after the study that notes which functional requirements are tested and how.

6.4 Case studies and proof of concept

The purpose of a case study is the examination of an object in a given situation. The case studies in this project seek to discover if the principle of the framework prototype is functioning as expected when the prototype is deployed in a (simulation of) network of power producers.
The case studies demonstrates if the framework as a feasible solution to improve efficiency and economy for a network of power producers.
The case studies may also show that some parts of the prototype is not functioning correctly as described in the functional requirement specification of chapter 2 or the use case development in chapter 3. The tests may also discover situations where the prototype is vulnerable.
The result of the case studies should show if the proof of concept can be approved or should be dismissed. Proof of concept is a statement indicating that the basic functionalities of a software prototype have been demonstrated and it is working as expected. The proof of concept verifies that the principles of prototype can be exploited in a useful way.

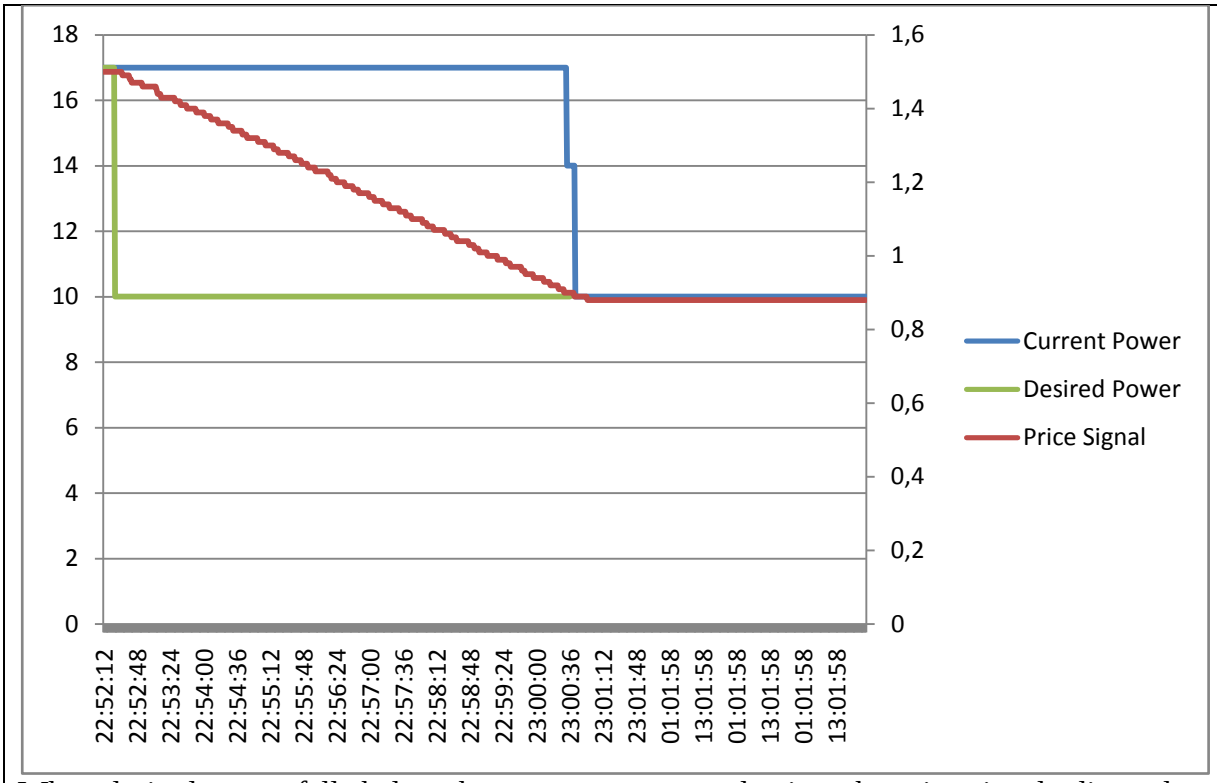
6.4.1 Case 1: Price signal

The price signal is tested by using the simulation program. The signal will be tested in an optimal situation, worst-case scenario, and by a number of tests where some variables are changed to see the result. The default values of the variables are listed in Table 22.

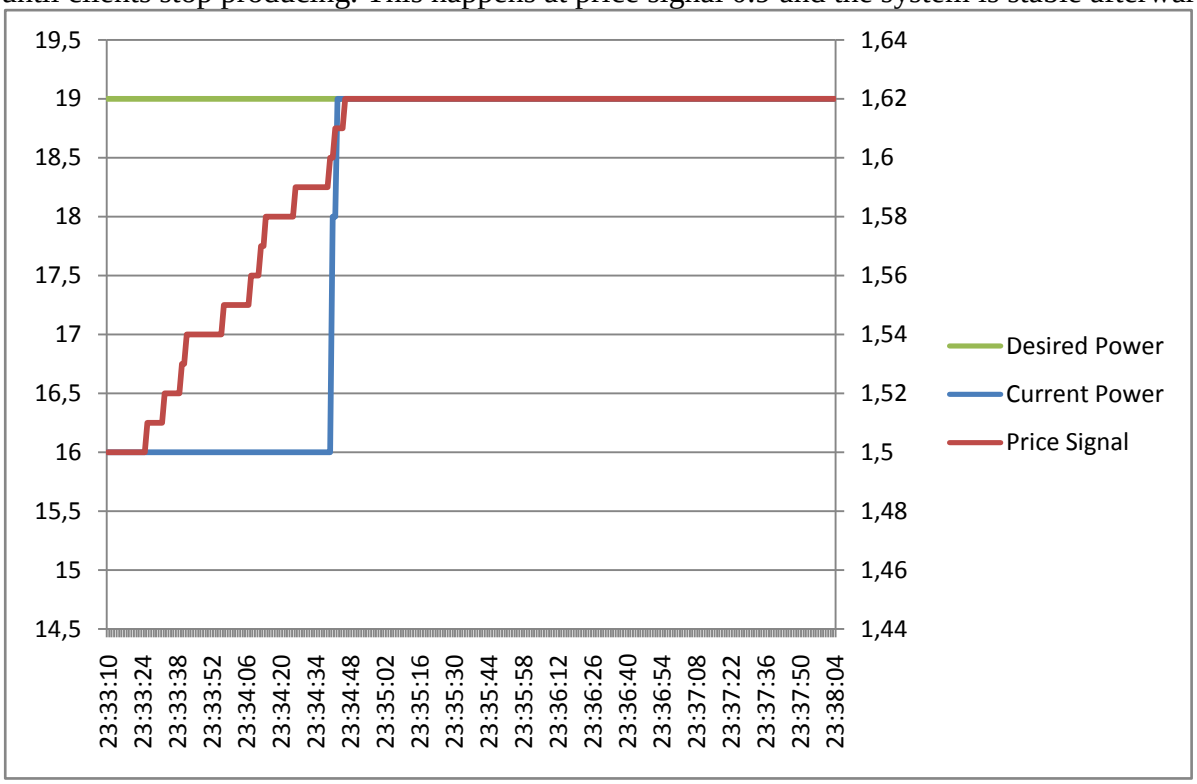
Price Signal (value at start up)	1.50
Precision	1.00
Regulating Index	0.01
Desired Power	10
Min Price	0.00
Max Price	2.00

Table 22 default values of price signal test variables

Test No	1
Purpose	Testing price signal
Alternative variables	-
Description	The test is expected to show that the signal will adjust to fit the production with no problems.
No. of performed tests	3
Stabilizing time	511, 98 and 43 seconds
Test remarks	The price signal stabilizes perfectly but in one test case the majority of the clients had a lower start up price than the price signal, which meant that the signal had to adjust downwards to stop clients from producing. It is important to note that simulated clients have start-up price of 1.0-2.0 and stop-prices of 0.0-1.0. It might be more realistic to use a stop price just a bit smaller than start price. The time of stabilizing the price signal is very much dependant on the update interval of both server and clients and also the regulating index (0.01).
Graph of price and power	



When desired power falls below the current power production, the price signal adjusts downwards until clients stop producing. This happens at price signal 0.9 and the system is stable afterwards.

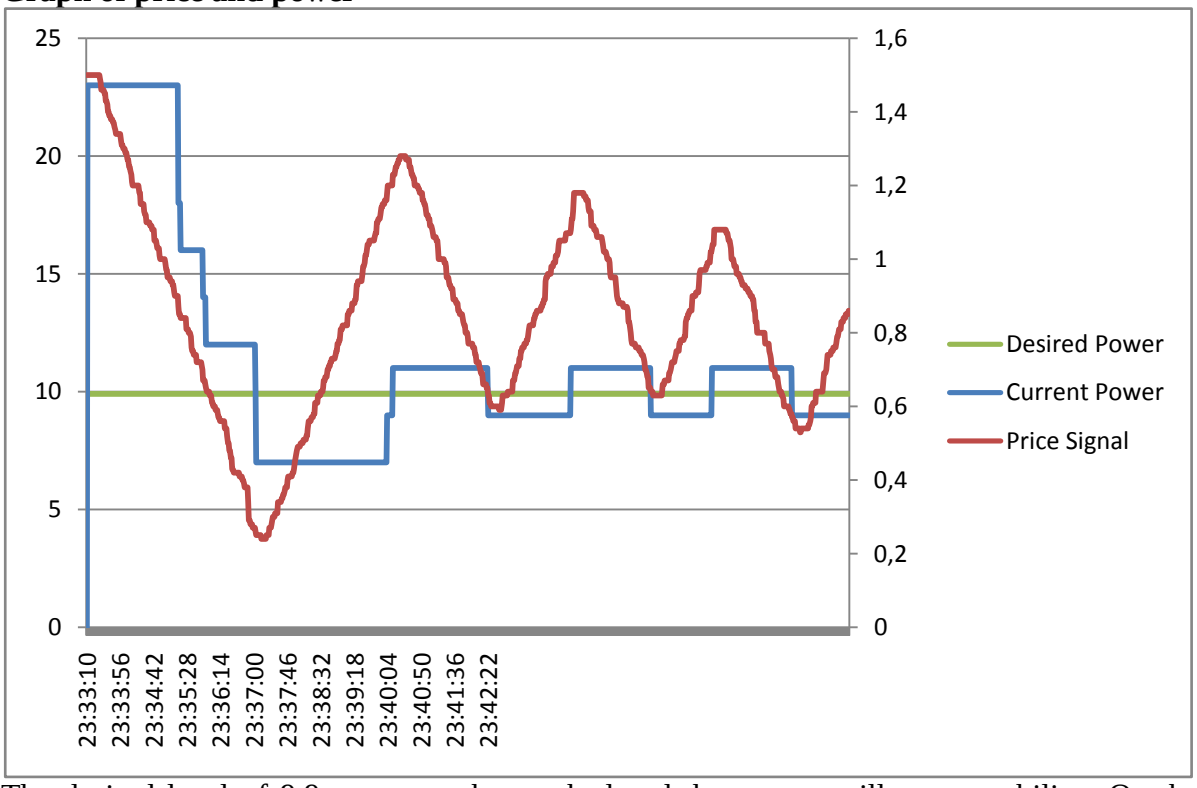


The price signal adjusts upwards to encourage more clients to produce. When the price signal reaches 1.6 more clients start production and the system stabilizes. Actually it can be seen that the price signal reaches 1.62 before the desired power level equals the actual production. This is a result of the server has updated the price signal 2 times before some clients update and register their

production in the database.
In this case most of the functional requirements are tested – numbers in the following text refer to the requirement's specific number in chapter 2. The frameworks makes it possible for clients to trade automatically (1), In the first graph the control user interface is used like an administrator would (4) to change the desired power production level from 17 to 10 (2). The price signal automatically adjust (5) to the new wanted power level, which the clients check (6) and finally the simulated clients decide individually and automatically to either start, continue or stop production of power (7).

Test No	2
Purpose	Test of stabilization with no marginal.
Alternative variables	Precision = 0.0, Desired power = 9.9
Description	The test is expected to show that the price signal will not stabilize unless it is in the situation where the number of clients matches exactly the desired level of power.
No. of performed tests	3
Stabilizing time (avg)	Indefinite
Test remarks	As expected the price signal did not stabilize and kept adjusting between a too high production and a too low production.

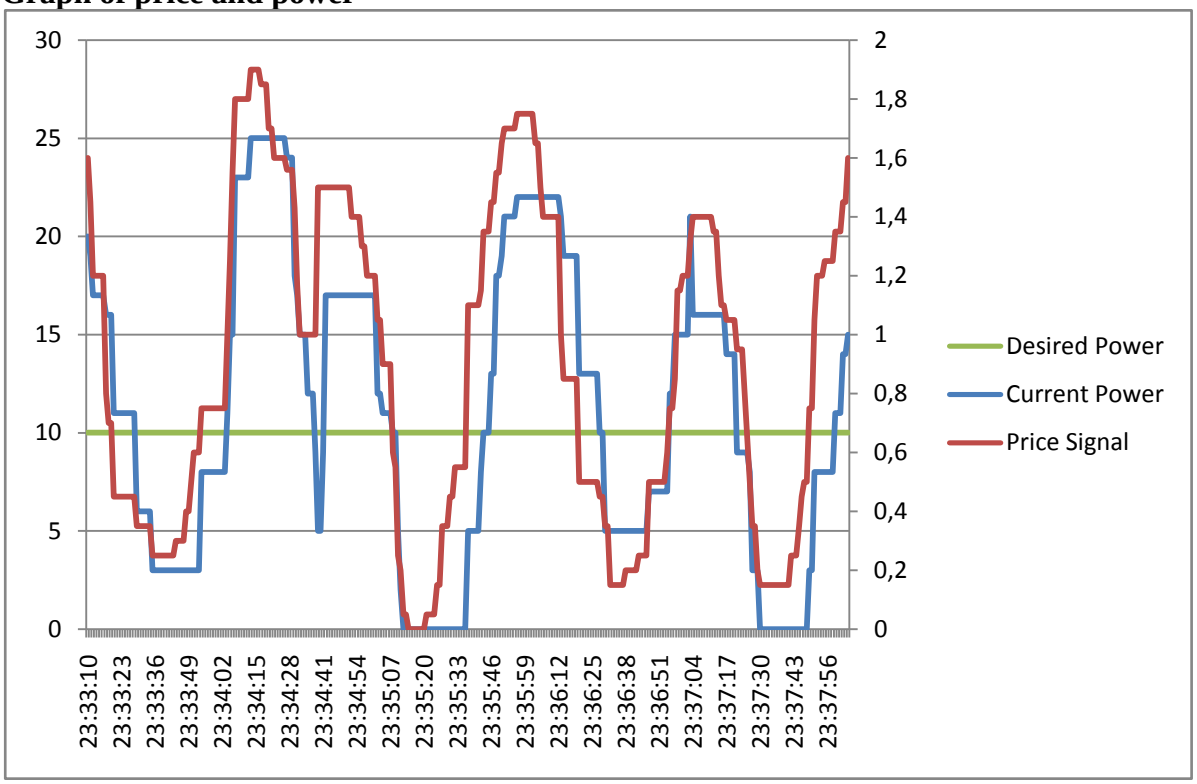
Graph of price and power



The desired level of 9.9 can never be reached and the system will never stabilize. On the other hand the average power production seems to fit the desired level once the system has gone into a frequency mode.

Test No	3
Purpose	Test of price signal with relative large regulating index
Alternative variables	Regulating index = 0.5
Description	The test is expected to show that the price signal will adjust much faster to changes in production, but also more rough making it harder to stabilize the signal.
No. of performed tests	3
Stabilizing time (avg)	32 seconds, 703 seconds, 2368 seconds
Test remarks	The test shows that it is hard to stabilize production and price signal if the regulating index is not adequate. The signal changes so abruptly that many clients have to start and stop their production several times. This is not optimal because it is destabilizing production and a gentler regulating index (like in the test of default values) would be recommended. The reason for the large differences in stabilizing times is that clients are created randomly and it is like an instance of luck that the signal stabilizes more than adjusting to fit a production.

Graph of price and power



The regulating index causes major changes in both price signal and power production. If the system is stabilizing it is a matter of luck. Several times the power production reaches 11 or 9 which both are within the precession marginal of +/- 1 from 10, but it is for a very short period of time. The reason that the signal changes immediately afterwards is that at least one client is in the process of starting or stopping production and the delay of registering log information about production causes the production and price signal to adjust once again.

6.4.2 Price signal remarks

The price signal functions as expected and will en realistic scenario be a good solution for trading power. The price signal will not be a good solution for power producers with a relative long start up period because the trading circumstances can change from favorable to not favorable before the production start up has occurred. The price signal model is good for producers who have a varying production both in power level and power cost because the price signal is so flexible. Clients with relative low marginal cost will benefit from this model when the price signal goes up and simply stop production when the signal falls below their marginal stop-price.
The test also showed that many variables are used and everything depends on how the framework is set up. It is possible to create a lot of situations where the framework will perform poorly, but it is possible to make a stable system.

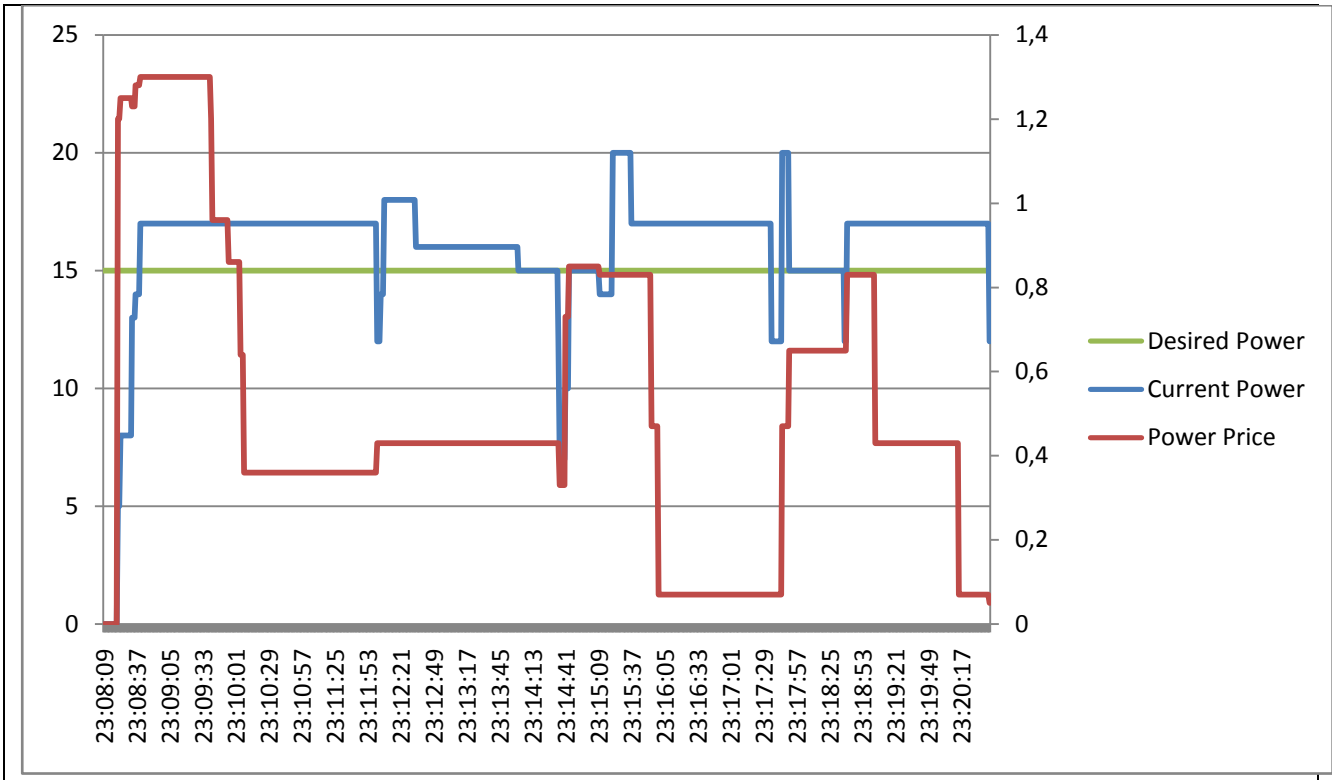
6.4.3 Case 2: Bid control

The bid model will be tested with different values of variables to simulate a realistic situation. The default variables are shown in Table 23.

Variables	**Values**
Desired Power	10.0
Min Price	0.00
Max Price	2.00
Client Update Time Span	10 seconds
Client Startup Time Span	Average 30 seconds

Table 23 default values for bid model tests

Test No	1
Purpose	Testing bid model with default values
Alternative variables	-
Description	The test is expected to show that the server will accept bids to cover desired production and every time the current production falls below the desired a new bid will be accepted.
No. of performed tests	3
Test remarks	Bids are accepted when power level falls below desired production.
Graph of power level	



The graph shows the desired power level of 15 (green) the actual power production (blue) and the average price of production (red). In the beginning there are just few bids and lots of power is needed. That is why even the high bids are accepted causing the average price to rise dramatically. After a couple of minutes several bids have been produced and they are about to expire, but the functionality has allowed the clients to send bids for continuing their production. To mark the clear difference between start-up price and stop-price the stop-price is on average 1 amount lower. That is why the average price falls to about 0.4. At about 23:12 the first dip occurs, which means a bid has expired and no production were continued. A new bid was accepted immediately after to ensure production level of at least 15 and this caused the price to rise. The pattern repeats throughout the graph.

Functional requirement 9 and 10 was tested in this case, which illustrated that the system makes it possible for each client to give bids to the server and check if they have been accepted. The server accepts bids when more power is needed, which also is seen in the graph. This is what happens when each client start up a production.

6.4.4 The price bid remarks

The price bid model is functioning as expected. The case study shows that the server accepts the cheapest bids when power level falls below the desired level. The bid model will with 10 clients rarely be on exactly the desired amount. It will be easier and easier to hit this level as more clients join. The bid model is favorable for clients that need a certain amount of production time and has large expenses concerned with starting and stopping production. If a client only is able to produce for a little while during a longer period it will be easier for the client to gamble and send higher bids for an increased income.

6.5 Conclusion

The chapter has tested the prototype for fulfilling the requirement specified in chapter 2. A case study has investigated the GFPT prototype's behavior in different situations.

The prototype is able to trade power automatically by using both a price signal and a bid model as power controlling mechanisms. The prototype is functioning like expected and as stated in the functional requirement specification. The case studies are therefore a proof of concept that the framework prototype is working. The framework has many variables and it is possible to make many different setups. If the framework is set up to match the clients' it is possible to make an optimal solution.

The price signal model is good for producers who have a varying production both in power level and power cost because the price signal is so flexible. Clients are not guaranteed a period of production, but whenever the signal is higher than the marginal cost the client will make an additional income compared with the bid model, since all clients are paid the same (value of price signal).

The case study shows that the bid model accepts the cheapest bids when power level falls below the desired level. The bid model is favorable for clients that need a certain amount of production time and has large expenses concerned with starting and stopping production. If a client only is able to produce for a little while during a longer period it will be easier for the client to gamble and send higher bids for an increased income.

Both power control models are feasible solutions for different types of power producers. This case study therefore recommends that both models are used if the prototype will be further developed in the future. Each client must be able to use the model that fits its units and production schedule best.

7 Conclusion

This chapter will give a short summary of what have been achieved in this thesis, how
The chapter will also include concluding remarks from each chapter of the report and finally a description of recommendations for future improvements.

7.1 Achievements

This thesis has dealt with the problems of trading power in a generic framework. The analysis of the power market and problem area shows that the power production can optimized if all power producers get a chance to trade power and if the trading is done automatically and in real-time.
The vision of this project was to model a state-of-the-art GFPT, which improves trade opportunities for power producers and also could be beneficial for vendors of producing units and improve overall efficiency and economy for the society as well. This has been done by analyzing power market, producing problems and challenges. A prototype GFPT was designed and developed using the newest available technologies such as WCF. The prototype was tested in the case study chapter by a client simulation program for the basic functionality of a GFPT and the 2 power control models price signal and bid control was examined.
The case study showed that it is indeed possible to construct a functioning GFPT solution which enables automatic trading for power producers. The investigation of power control models showed that the price signal is very useful for clients with varying production while the bid model is useful for clients with some sort of scheduled production.

7.2 Thesis remarks

The thesis has analyzed the power markets and power producing problems and found that some of the problems can be improved by developing a GFPT as suggested in the problem definition. The analysis showed that the power market could be both more effective and faster responding when regulating system balance if smaller production units were used due to their fast start-ups. A virtual power plant could obtain an effective and fast regulating ability if trading could be done automatically in real-time. The major problems and challenges of the power market were identified:

- Enabling small producers to trade power in a virtual power plant
- Automatic trade of power to minimize human resources.
- Real-time trade of power optimizing balance regulating.
- Controlling power production level.

One solution is to implement a GFPT where power producers of all sizes can join in a virtual power plant. A server functions as the broker agent that trades power in external markets and upholds an optimized production inside the framework to fit the traded energy. The framework server has a module that regulates power level in the framework by some production control logic. This logic can be designed in many different ways. In this project 2 models have been analyzed:

- Price signal
- Bid model

The price signal is used to regulate a price for power production and each client decides if they want to produce at this price. The server regulates the signal either up and down if more or less clients are needed to produce power.
The bid model makes use of bids from clients. The framework has a function that enables clients to automatically send bids by a custom-set interval as well as price, production period etc. When the server needs more power it accepts the lowest bids available.
A prototype was implemented using the newest state-of-the-art development tools WCF, WPF and .Net Framework. The implemented prototype is the core of the framework. The framework is

only in the first stage of development with basic functionality, but can easily be extended to a full-scale solution. The price signal and bid model are just 2 ways of controlling power production and other control models could also be added.
The case studies showed that both the price signal and the bid model are feasible power control solutions. Both models adjust power level in production successfully to a stabilized framework and fulfill the requirements. Clients with the lowest prices are the ones that actually get to produce.
The thesis recommend that in a further development and deployment of a GFPT both the price signal and the bid model is used to accommodate the different power producers in the best possible way.
The framework is the first of its kind to allow power producers to trade power automatically using service oriented architecture, and the project could be the foundation of further development and deployment of a framework taking advantage of the models, which have already been designed and build in the prototype.

7.3 Future improvements

The framework is in the first stage of development and is only the basic functioning core. The framework can be extended to full deployable solution based on the modeling and design of this project. Some of the improvements which can be made are the following:

- The power control model. This project has used a price signal and a bid round. There is many different ways of modeling this and although both models have proved worthy of functioning well in the case study more research could be done to optimize the model to the specific clients who are going to produce. If it is a group of wind power plant their focus should be based more on an unreliable production methodology. Another model could be a reverse price signal, where clients each give their price signal and the server decides which clients to pick. This solution also resembles the bidding round although no duration of production is given. A model where the server distributes the power production equal between clients could also be mentioned.
- Automatic start-up of clients. All producing units could be directly started and stopped by the server. This leaves all decisive logic with the server and the clients have no control of the production. On the other hand the clients have no expenses on making the logic in their own software to set start-up values and stop price etc.
- The interface to the external markets could be automated. The regulating market already has an automatic software interface and the framework could be designed to automatically send bids for the regulating market. This might be possible with other markets as well.
- Considering a future where every household would have a small combined heat and power unit a larger framework would be necessary to service the many thousands homes. Here it would be an advantage if both producers and consumers could operate in the framework. This means that bids in the bid model should not only be for producing power but also for buying power. A result could be that the framework might not need to communicate with the external markets if enough producers and consumers exist in the framework to make it exactly self sufficient.
- Geographical optimization. Producers who are geographically close to consumers could be paid a bonus or increased price. This would mean that power would have to travel less distance, thus resources can be saved.
- Forecasting can be done using statistics, clients own production plans, weather forecast etc. This is an advanced problem, but if accurate forecasting is done, the framework would be more efficient and make a larger profit.

- Security. No security has been implemented in this project. This is a major issue which involves safe communication, integrity and breakdown handling.

8 Bibliography

8.1 Papers

Is System Control Entirely by Price Feasible?
Fernando L. Alvarado, Fellow, IEEE

OASIS Reference Model for Service Oriented Architecture 1.0
C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz
http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

Introduction to Design Patterns in C#
Cooper J., IBM T J Watson Research Center, 2002

8.2 Web pages

Energinet.dk
http://www.energinet.dk

Forskrift A – principper for elmarkedet
http://www.energinet.dk/NR/rdonlyres/3E2CA7D4-A4D0-4D38-A846-CF4788247813/0/01**Forskrift**APrincipperforelmarkedet.pdf

Om energi no. 5 2006:
http://www.energinet.dk/NR/rdonlyres/3E2CA7D4-A4D0-4D38-A846-CF4788247813/0/01**Forskrift**APrincipperforelmarkedet.pdf

Seminar for Power Traders
http://www.energinet.dk/NR/rdonlyres/26530AE7-046C-4543-A62F-922F0AB2BB9C/0/Seminaromelmarkedoverheads6306.pdf

New Energy no. 6 2006
http://www.newenergy.info/index.php?id=1350

Elforsyningslov dkvind
http://www.dkvind.dk/nyheder/foer_2005/ny_elforsyningslov_kort.htm

PBA information from Fjernvarmen
http://www.fjernvarmen.dk/upload/scanenergi.pdf

Variations in Wind Power
http://www.windpower.org/en/tour/grid/index.htm

Energi Danmark A/S
http://www.energidanmark.dk/Produkter/Decentral+Kraftvarme/

Nord Pool "The common Nordic power market"
http://www.nordpool.com/organisation/common_market.html#danish

Law no. 375 from 02/06/1999. Elforsyningsloven. Lov om elforsyning.
http://www.energitilsynet.dk/import/arkiv/71/

Technical Overview and Benefits of the IEC61850 Standard for Substation Automation.
R. Mackiewicz, SISCO Inc.
http://www.sisconet.com/downloads/IEC61850_Overview_and_Benefits_Paper_General.pdf

Press announcement: "Start vaskemaskinen når det blæser – og spar penge".
http://www.energinet.dk/da/menu/Nyheder/Nyhedsartikler/energiudsigt.htm

The Microsoft Developer Network
http://msdn.microsoft.com

International Electrotechnical Commission
http://www.iec.ch/

Critique of the governments energy policy by the Danish Social Democrats
http://socialdemokraterne.dk/default.aspx?func=article.view&id=176164

The Danish Social Democrats energy policy 2006
http://socialdemokraterne.dk/download.aspx?docId=167995 ,

Dong Energy
http://www.dongenergy.com

EC Power
http://www.ecpower.dk

Vestas
http://www.vestas.com

8.3 Phone References

Thomas Eriksen, Energi Danmark: tbe@energidanmark.dk
Thomas stated that only producers with a capacity above 150kW were interested in being BRP, because it involved a lot of work. Installation, maintenance of software and the daily plans for energinet.dk.

8.4 Magazines

Kraftvarme NYT, august 2006
Foreningen Danske Kraftvarmeværker

8.5 Books

Framework Design Guidelines
Krzysztof Cwalina and Brad Abrams

Addison Wesley, 2006, 0-321-24675-6

Windows Communication Foundation Unleashed
Craig McMurtry, Marc Mercuri, Nigel Watling and Matt Winkler
Sams Publishing, 2007, 0-672-32948-4

Objekt Orienteret Analyse og Design
Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, Jan Stage
Marko, 1998, 87-7751-129-8

Design Patterns, Elements of Reusable Object-Oriented Software
Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
Addison-Wesley, 1995, 02-0163-361-2

8.6 White papers

Jøsang, R. Ismail and C. Boyd. A survey of Trust and Reputation Systems for
Online Service Provision. Decision Support Systems, 2007.

T. R. Hansen. Analysis and Design of An expert based fuzzy control scheme for a mini Combined Heat and Power unit. Department of Electronics, Engineering College of Copenhagen. 1999

9 Appendix A

9.1 Dictionary and terms

WCF - Windows Communication Foundation
WWF - Windows Workflow Foundation
WPF - Windows Presentation Foundation
SOA - Service Oriented Architecture
SQL - Structured Query Language
TDD - Test-driven development
API - Application Programming Interface
GFPT – Generic Framework for Power Trading

10 Appendix B: Installation guide

Installing and working with the framework applications requires knowledge about Microsoft SQL Server 2005 and Visual Studio 2005.
This installations guide briefly describes the required steps in order to get the system up and able to compile and run the solution.

10.1.1 Visual Studio 2005

Visual Studio 2005 does not support the .NET framework 3 by default. The following updates and add-ons available from the plug-in folder on the CD must be installed.

1. Visual Studio 2005 Service Pack 1
2. Microsoft® Windows® Software Development Kit for Windows Vista™ and .NET Framework 3.0 Runtime Components
3. Visual Studio 2005 Extensions for .NET Framework 3, November 2006
4. Visual Studio 2005 Extensions for Windows Workflow Foundation

10.1.2 SQL Server 2005

The SQL server needs to be updated to SP1 or newer. This SP is available from Microsoft and not available on the CD because of the different service packs for the different versions of the SQL Server.

1. Create a new database: “Power Server”
2. Restore the database from the PowerSystem\Database\PowerServer.bak file.

10.1.3 Configuration

Update the connection string in the PowerService.cs file to match the logon criteria for the SQL server in use.