D5.3 - Load management methods and prototypes

Energy management strategies and supervisory control of building loads

Madsen, Per Printz; Andersen, Palle

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

ENCOURAGE

Embedded INtelligent COntrols for bUildings with Renewable generAtion and StoraGE

Grant Agreement No.: 269354

D5.3 – Load management methods and prototypes

Document Number	D5.3
Document Title	Load management methods and prototypes
Version	1.0
Status	Final
Work Package	WP5
Deliverable Type	Report
Contractual Date of Delivery	M28
Actual Date of Delivery	M28
Responsible Unit	AAU - WP5
Contributors	AAU, ISA, ENORD, ATOS, GNERA, ESVAL
Keyword List	Energy management strategies and supervisory control of building loads
Dissemination level	CO

Amendment History

Version	Date	Author	Description
0.1	08/02/2012	AAU	Initial Draft – Task roles
0.2	24/09/2012	AAU, ISA, ENORD, ATOS, GNERA, ESVAL	Internal structure and function description
0.3	1/09/2013	AAU	Development of control strategies for building level energy management
0.4	1/9/2013	AAU	The lighting control module
0.5	9/9/2013	AAU	Merging contributions
0.6	29/9/2013	ISA, ISEP	Internal review
1.0	29/9/2013	AAU	Finishing

Table of Contents

1. Executive Summary

The target for the ENCOURAGE energy management system is to design and test strategies for supply and demand side energy management of a microgrid. This will be done by designing a supervisory controller to manage energy flows so that generated power in the microgrid is mainly consumed by local consumers and the power trade between the microgrid and the grid is shrunk to minimum.

Buildings' role in providing enough flexibility to the supervisory controller is huge as they account for 41% of total final energy consumption in Europe, followed by transport (32%), industry (24%), and agriculture (2%) (ODYSSEE-MURE project, 2012). When all building loads are considered as *critical* there would be no opportunity to optimize both the electricity sawing and consumption cost. Optimization of building loads based on electricity price signal includes shedding, shifting or rescheduling the power consumption pattern. To this aim, loads will be characterized by specific flexibility patterns. For instance *shiftable* loads like HVAC systems will be characterized by the amount of energy that can be shifted in time. This would be, however, at the cost of a wider thermal tolerance that users give permission for. The wider the thermal tolerance is, the more flexibility will be provided to the supervisory controller. Load management strategies will be devised such that thermal comfort and other user-predefined preferences will be satisfied.

A dedicated language was developed to apply the supervisory controller to different houses. This language is used for implementing a glue layer between the load management controller and the middleware lager. Beside this interconnection, between the advanced top level controllers, the dedicated language also handles the curtailable load e.g. the lighting system. Here the main target is not to enhance the flexibility but to lower the energy consumption.

This deliverable reports description of load types, a strategy for energy management at building level, designated supervisory controller for the buildings load management and results of simulation studies.

The concept of microgrid control together with interplaying generation and consumption units is shown in Figure 1.

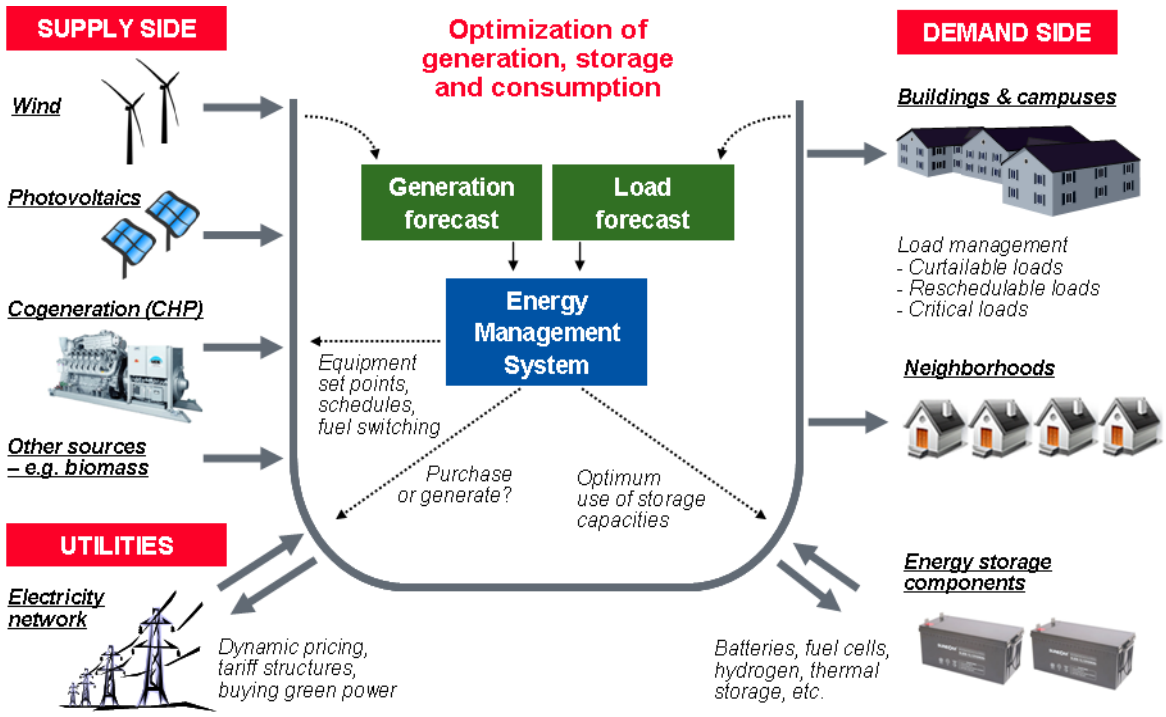


Figure 1. Application domain of the ENCOURAGE project: buildings, campuses, neighbourhoods connected in local microgrids with renewable generation and storage devices

1.1. WP5 Objectives

Embedded intelligent controls for buildings with renewable generation and storage (ENCOURAGE) aims to develop embedded intelligence and integration technologies that will directly optimize energy use in buildings and enable active participation in the future smart grid. The target energy saving for a network of buildings composed of distributed energy consumption, production and storage units is 20% via design of supervisory control schemes that coordinate among interplaying energy devices and buildings (Arne Skou, 2012).

As part of ENCOURAGE, we are going to design a supervisory controller that integrates and manages all energy units in the microgrid. The objectives are as follows:

- Energy needs of the microgrid are, as much as possible, to be provided by local generation units, which are photovoltaic (PV) cells in our case study. The purpose is to minimize dependency to the grid power.
- The other objective is to minimize electricity consumption costs of individual households.
- The energy manager i.e. a supervisory controller is supposed to work with the existing single loop controllers in the building, for instance heating thermostats.

However, the first two objectives might be conflicting, in which case priority would be with individuals' benefit. For example, there might be time intervals during which power demand of a house exceeds its production. Assuming that power is provided by the grid at a lower price rate than the neighbouring production units in island, power would be purchased from the grid. On the other hand, policies could be enacted to promote trade of power mostly within the microgrid in order to minimize dependency to the grid. For instance, price of the locally produced power could be kept always lower than the grid electricity price.
The above mentioned objectives are to be fulfilled via design of a hierarchical control structure that is shown in Figure 2. The hierarchy is explained later in the report in more details.

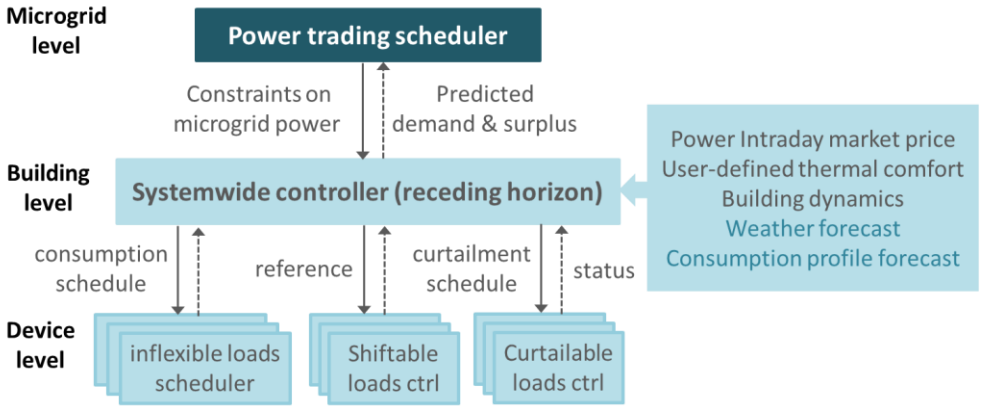


Figure 2. Supervisory controller hierarchy

1.2. WP5 Subtasks

According to ENCOURAGE project proposal, WP5 will develop control strategies for optimal operation of energy generation, consumption, and storage devices connected in a network. The tasks of this work package include:

Task 5.1 addresses the supply side of the system by developing appropriate monitoring and control concepts for local generation elements based either on conventional or renewable energy sources.

Task 5.2 aims to develop optimized control strategies for management of internal loads in the building (demand side). This will include shifting and/or shedding of specific loads.

Task 5.3 develops system optimization strategies integrating both supply and demand sides. The algorithms will calculate optimal schedules (e.g. start/stop times of individual pieces of equipment) and set points (e.g. target volume of energy to be generated in a boiler). Another result will be the optimized use of energy storage over time. The most typical solution interval will be one day, but the solution will be scalable to both shorter and longer intervals.

This deliverable documents the work done in task 5.2.

2. Introduction

The global movement is toward power production mostly using renewable energy resources rather than using fossil fuels, which are environmentally polluting and are being depleted very fast. These renewable energy resources, for instance solar, wind, biomass and geothermal are, by their nature, highly distributed compared to large concentrated nuclear and fossil-fuel power stations. Regaining power balance and allocation of resources in such a diverse and distributed energy market will be two big challenges. Smart grid, as a newly emerging concept to be built upon the existing infrastructure of power grid, is to facilitate the coordination among all the contributing production, consumption and storage units. In this scheme, a single small-scale power consumer will be no longer an inactive component, but potentially will contribute to energy management of the smart grid by providing flexibility. Making use of local power generation units and storage devices increase flexibility of the grid nodes.

Several world-wide studies have been conducted recently to propose new market, communication, and control layouts for the emerging large scale distributed energy system. Encourage, NeogridEU, iPower and FlexPower are examples of many on-going European and Danish projects that are going to develop methodologies with different approaches to overcome imbalances of the future smart grid.

Embedded intelligent controls for buildings with renewable generation and storage (ENCOURAGE) aims to develop embedded intelligence and integration technologies that will directly optimize energy use in buildings and enable active participation in the future smart grid. The target energy saving for a network of buildings composed of distributed energy consumption, production and storage units is 20% via design of supervisory control schemes that coordinates among interplaying energy devices and buildings (Arne Skou, 2012).

2.1. WP5 in ENCOURAGE Architecture

ENCOURAGE aims to develop embedded controls, intelligent hardware devices, and open service-oriented platform that will allow end-users to achieve energy savings by orchestrating various energy generation, consumption, and storage devices in non-residential buildings, campuses, and neighbourhoods, and also enabling the possibility of exchanging energy surplus with other entities. The project has been structured into 8 work packages in order to achieve this

Figure 3. ENCOURAGE structure

main goal.

WP5 will develop control strategies for optimal operation of energy generation, consumption, and storage devices connected in a network. In order to ensure integration between work packages, WP7 is dedicated to integrate the developed strategies by the other work packages. WP7 conducts tests on a number of microgrid case studies under real conditions. Interactions between WP5 and the rest of system setup are via exchange of information through middleware layer as shown in Figure 4.

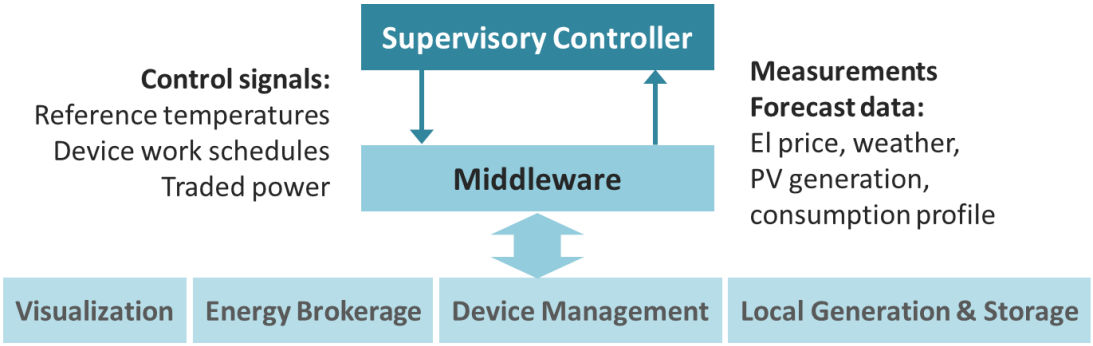


Figure 4. Data exchange between SC and other modules via middleware

2.2. Demonstration Sites

Among different demonstration scenarios of ENCOURAGE project, the focus of work package 5 is dedicated to Denmark demonstration site, which is a residential area. However, developed control strategies are general and can be applied to other case studies. In this section we summarize the main characteristics of the concerned microgrid case study.

One of the demonstration sites of the Encourage project that we focus on in this study is a network of eight residential buildings i.e. detached houses located in Gistrup area, Northern Denmark. Each house is equipped with photovoltaic (PV) cells with capacity of producing 4kW of electricity. Thus, electricity need of an individual house is provided partly by solar cells and the remainder could be purchased from both other producers in the microgrid or from the electricity grid, depending on the energy price provided by each energy source. Indoor air is heated by electrical floor heating in the houses. Measurements show that electrical space heater, electric water heater, appliances, and lighting respectively account for highest to lowest power consumption in a building. A satellite view of the houses taken from Google map is depicted in Figure 5. *Aalborg demonstration site. Houses denoted with red circles are equipped with PV cells.*

Figure 5. Aalborg demonstration site. Houses denoted with red circles are equipped with PV cells.

All the houses are similar and very well insulated. The houses are occupied by different types of family i.e. young couples, families with children and retired people who are couple or single. The chosen occupancy diversity allows testing different consumption profiles for load control and energy exchange between the houses, which is the normal case in a medium to large scale power island. Some houses are occupied mostly in evenings and weekends, while the others consume power at all times during a week. A power island is a specific part of the power grid that has both production and consumption. Some power islands can, if necessary, generate, the right amount of power, even though electrical grid power from the electric utility is no longer present. In our case, the houses will always be connected to the grid, so it is not necessary to handle the situation with a disconnection from the grid.

The microgrid is always connected to the grid. Therefore the islanded-mode is never imposed to the microgrid physically. However, the strategies should be enacted in order to make it as independent as possible from the grid. Therefore, the power island can purchase and sell electricity from/to the grid at any time. However, the objective is to make the trade flow in only one direction at a time, meaning that as long as there is a power demand in the microgrid, no power will be sold to the grid.

The microgrid local power generators are renewable, non-dispatchable sources. There is no specific energy storage device to store energy for a later use. However, the building thermal mass is a dynamic energy buffer that can be charged in a controlled way, but the discharge is not controllable, although is predictable. The stored energy is in thermal form. Thus, this storage capacity makes the heating and cooling loads flexible to a certain degree, as determined by the building dynamics.

2.3. Literature Survey

There are two mainstream approaches for energy consumption/production management towards a smarter electric grid, i.e. direct and indirect control. The former relates to a setup where an energy node in the grid informs the aggregator of its potential flexibility on consumption or production. The load flexibility is to be provided by means of some storage facilities. In return, the aggregator controls the unit based on the predicted flexibility within the limits and costs agreed upon in

advance (Biegel, 2012). In the latter approach, price incentives are sent to distributed energy resources in order to encourage individual units for example detached houses, residential or office buildings to consume electricity when energy surpluses in the grid by shifting their power demands, and use local energy resources or the stored energy when there is power congestion or deficit in the grid (Pinson, 2012), (Moslehi, 2010). A further classification of load control policies for demand-side management in smart buildings is proposed in (Gehrke, 2013) based on the reaction to external signals, participation in markets, topology, decision making mechanisms and fault handling.

The direct control concept is inspired from method for optimal use of a power plant portfolio; see for instance (K. Edlund, 2011). This is also reflected in the terminology where an aggregator is assumed to control a group of consumers as a Virtual Power Plant, (Nerea Ruiz, 2009), (A. Gomes, 2007) , (Gong, Xie, & Zhang, 2011). Optimal operation of a portfolio of consumers often involves solving a non-convex optimization problem, for which many approaches may be taken, (Gomes & Martins, 2007) uses evolutionary algorithms, (Parisio, 2011) use Mixed Integer Linear Programming (MILP). An agent based solution is used in PowerMatcher, (Hommelberg, Warmer, Kamphuis, & Kok, 2007), while a sorting algorithm, which match a certain formulation of the problem has been discussed in (Trangbæk & Bendtsen, 2011), and a similar method is used in (Biege, Andersen, Pedersen, Nielsen, & Stoustrup, 2013).

The concept of indirect control within the smart grid is conceptually studied and classified in two main categories in (Heussen K. a., 2012). One type of indirectness involves not direct control command but only an incentive. Operation of electrical power systems based on nodal price control was firstly addressed in the studies conducted by Fred Schweppe which is summarized in (Schweppe, 1988). Many researches were conducted ever since studying different aspects of market-oriented approach for the electrical power system sector (Jokic, 2009), (Alvarado F. , 1999), (Alvarado F. , 2003), (Alvarado F. a., 2003). Resent work in (Mitter, 2010), (Mardavij Roozbehani, 2012) and (Juelsgaard, 2013) conclude that passing on the real-time wholesale electricity prices to the end consumers can lead to increased volatility, lack of robustness and instability. In (Madsen G. D., 2013) a methodology is described allowing estimating in advance the potential response of flexible end-consumers to price variations. This response is subsequently embedded in an optimal price-signal generator, and prices are estimated and broadcast once a day for the following one, for households to optimally schedule their consumption. (Bosch, 2009) and (Annaswamy, 2011) suggest price based schemes which ensure economically optimal operation while also respecting grid constraints. Examples of schemes allowing a consumer to optimally exploit real-time prices can be found in (Jørgense, 2012) , (Glielmo, 2011) and (Frauke Oldewurtel, 2010).

A novel generalized framework for modeling a storage node in the grid is proposed in (Heussen K. a., 2012). It models any type of interactions among the energy generators/consumers and storage devices, energy leakages in transmission lines and due to energy conversions via definition of a generic power node. It exploits Model Predictive Control (MPC) in combination with Mixed Integer Linear Programming (MILP) (Bemporad, 1999). Load shifting based on price incentives for households in a microgrid is addressed in recent studies using optimal controller in (Pedersen, Andersen, Nielsen, & Staermose, 2011). MPC was previously addressed in (Tahersima F. a., 2012)

for heating load management of a single residential building. Also, (Tahersima F. a., 2011) suggests an assistant chart that quantifies energy flexibility of households.

3. Building Loads Classification

Depending on the degree of user intervention, various building services can be classified in two main categories i.e. controllable and non-controllable loads. In the first category loads can be controlled automatically or manually. Our focus from now on will be on the controllable loads with both automatically and manually controllable features. Heating, ventilation and air conditioning (HVAC) systems are usually controlled automatically. Lighting is in a middle category i.e. partly automatically-manually controlled. Appliances reside in a third class i.e. manually controlled by the user. A 2006 statistical results reveal that around fifty % of the total energy consumption in a typical Danish household relates to manually controlled loads as shown in Figure 6 (Røpke, 2010).

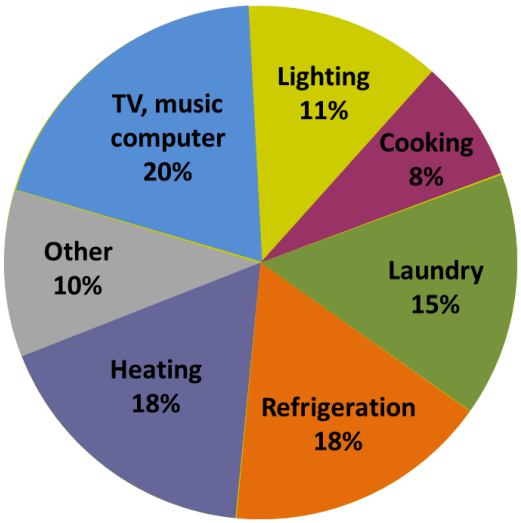


Figure 6. Danish household energy consumption in 2006 ((Røpke, 2010)

Loads that are adjusted manually by the users do not contribute to the flexibility automatically with the currently available infrastructure. However, inflexible loads that are not necessarily critical can contribute to increasing flexibility by engaging the users' manual adjustment. For instance, a schedule for using this type of non-critical devices can be prescribed by the energy management system to shift consumption according to the grid's condition.

The other type of loads, for instance lighting can be shed automatically by partial home automation. For example motion-based actuators can be used to turn on/off the lights very efficiently based on the room's occupation. However, this action is not a pre-planned procedure to be incorporated in the loads flexibility, but it will often result in a significant energy saving.

Control of the HVAC systems is normally performed automatically in order to accomplish some user-specified references for instance indoor temperature and humidity. All buildings can potentially benefit from their thermal mass storage depending on the type of insulations and also the type of HVAC systems. For example, hydronic floor heating pipes casted into a thick layer of concrete floor release the buffered heat in several hours such that it can be regarded as a thermal storage like a hot water tank. The flexibility provided by this category is the most reliable.

Table 1 shows a classification of different load types from the perspective of directly or indirectly accessed i.e. reliable or unreliable flexible loads, respectively.

Table 1. Taxonomy of a residential building loads

	Automatic	Partly automatic/manual	Manual
Appliances			x
HVAC	x		
Lighting		x	
Multimedia			x
Consumption	18%	11%	61%

Please see Appendix A for a thorough classification of building loads.

4. Indirect Energy Management Strategy

In this work, the focus is on indirect control strategies of households' energy consumption. In this approach, consumption would be shifted using energy storage facilities of the building or shed provided that power consumption should not be critical at the time. Both shifting and shedding of the loads are decided based on an *a priori* known electricity price signal for a future time interval such that cost of electricity consumption will be minimized. In order to devise a strategy based on energy flexibility, we need to know potential flexibility or inflexibility of a building.

We have categorized various controllable energy loads in the building to automatically and manually controllable loads. In each category two types can be realized i.e. shiftable loads and curtailable loads. Examples of manually controllable loads are appliances and multimedia devices that are directly interfered with by the user. The two types of shiftable and curtailable loads are controllable, although different depending on the type of the variable being controlled and residents' expectation of comfort. To give an insight, let's compare heating with lighting; a heater is controlled normally to maintain a specific thermal comfort criterion that is often specified either by a profile of reference temperature or two upper and lower boundaries. In either case, time interval of heating possibly can be shifted depending on inherent heat capacity of the building and thermal tolerance degree of the building's residents. For example, the larger heat capacity of the building and the higher tolerance of the occupants, flexibility in time of heating is larger. On the other hand, light is a non-storable and is an instant load. Therefore, it would only be possible to cut down or dim the light when it is not needed, for instance when daylight is available or motion is not detected in the room. For a taxonomy on different type of loads please see (Petersen, 2013), (Fatemeh Tahersima, 2013).

In this work the focus is on indirect control strategy of household's energy consumption. A model predictive controller (MPC) was formulated that systematically finds the energy consumption pattern of flexible loads provided that knowledge about other loads and productions and the building dynamics are available. The flexible loads are mainly the shiftable loads and to some extents the curtailable loads. A cost is formulated based on power consumption and its price which is minimized by the designed controller. In the proposed scheme electricity can also be sold to the grid and consumption can be curtailed if convenient. In contrary to the available literature, a hierarchical controller rather than a centralized one is proposed. The first advantage is that it exploits the existing stand-alone single-loop controllers which are usually available in the buildings. The new integrating and optimizing layer connects to the lowest layer by commanding a general reference signal to the single loop controllers. The system-wide controller is designed in a receding horizon fashion in order to incorporate building energy flexibilities based on a dynamical model, future preferences and disturbances. Figure 7 shows the logical structure of the building level control system. The glue layer, also called Device Level Controllers, takes care of the interconnection between the high level MPC controller and the low level devices. All the house specific control logic is placed in this glue layer. This means the MPC controller are a general propose controller and because of that are the same in all houses.

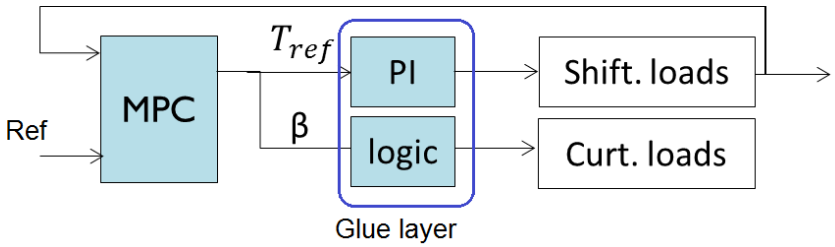


Figure 7: The structure of building level control

The MPC controller outputs are actuation signals to the HVAC, hot water tank, lighting, etc…, and consumption profile to the appliances and other non-critical inflexible loads in the building. Its inputs are measurements like indoor temperature, hot water tank temperature, etc… and forecasts of power consumption in the building, weather, electricity intraday market price inside and outside the microgrid, and electricity generation of PVs. In the sequel we describe the controller structure in more details.

4.1. Hierarchical Supervisory Controller for Microgrid Energy Management

The control hierarchy as shown in Figure 2 composes three different layers. The task of each layer and the connection between the layers are described in the following.

- Device layer at the bottom of hierarchy comprises single loop controllers for controllable (shiftable and curtailable) loads, controllable generation units (not available) and storage devices (not available). It is responsible for maintaining a set point, light adjustment, etc… using an on/off or proportional integral (PI) control action.
- Building level at the middle of hierarchy includes a system-wide controller that keeps the economy and comfort in balance. It minimizes the cost of electricity consumption while maintaining the comfort levels determined by the user. A priori knowledge about building dynamics, comfort preferences, weather changes, power generation and price of electricity are needed as inputs to the controller at this level. This layer receives measured status data from device-level controllers i.e. heating/cooling thermostats and provides them with reference signals.
- Microgrid level at the top is responsible for distribution of locally generated energy among households with energy demands. It receives predictions of power surplus profile (for sale) and power needs profile (to be purchased) from the system-wide controllers in the middle. Based on these inputs, it predictively assigns surplus power in the microgrid among the demanding houses. The system-wide controller is designed such that the power produced by PVs is consumed by the producing house at the first place. The excess is distributed by the power trading/scheduler among the other houses with power deficit. It predictively determines the constraints on the amount of buying energy and selling energy for each house in the microgrid.

5. Device Level Controllers

The primary energy elements in one house in the basic scenario are:

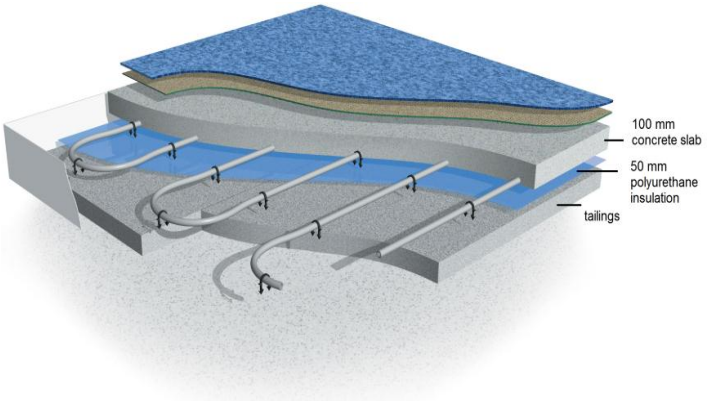
- PV cells
- Loads categorized mainly in controllable (curtailable, shiftable, schedulable) and non-controllable
- Interactions with the utility grid
- Interaction with the micro grid's other solar cell modules

We do not consider any electrical storage device in simulation scenarios; nor are the schedules to the manually controllable loads generated in the simulations. Among the household loads, an HVAC system and a hot water tank are categorized as controllable (shiftable), lighting (curtailable) and home appliances (schedulable) loads. Electricity generated by PVs cannot be curtailed.

Flexibility type of building loads and control options for the loads are classified in tables in Appendices B and C respectively.

5.1. HVAC Systems Control

HVAC systems in Denmark demonstration sites are electrical/ hydronic floor heating systems embedded into a thick layer of concrete floor. The concrete floor can be regarded as a heat buffer in the system as the heat will be transferred to the surface and thereafter to the air gradually during several hours. Knowing the dynamics of the system, this buffering capacity can be exploited to shift the power consumption provided that a certain temperature boundary will be respected.



At this level, a single loop controller is designed to maintain a specific temperature reference. This temperature reference is determined by an upper layer controller at the building level which orchestrates the power consumption of the whole building. The building-level controller determines the reference such that the cost of heating will be minimized.

A proportional integral (PI) controller is designed to maintain the temperature reference. The output of this PI controller can easily be translated to on/off commands suited to relay actuators. Formulation of the PI controller is given:

$$\dot{x} = k_1(T_{ref} - T)$$

$$Q_{heat} = k_2(T_{ref} - T) + x$$

In the formula, x is integration state, T_{ref} is the temperature reference, T is the indoor temperature, Q_{heat} represents the transferred heat to the floor surface, and parameters k_1 and k_2 are determined based on a step response of the system.

In the case of a hydronic floor heating system, there are two control variables for adjusting Q. They are mass flow rate and forward temperature of water i.e. $Q = c_w q(T_{for} - T_{ret})$, in which the parameters are specific heat of water, flow rate, forward and return temperature of water. The flow rate is usually controlled based on each zone's specific temperature preferences. Water temperature is determined centrally for the whole hydronic system.

5.2. Lighting Control

The task for the lighting control systems is to control the room lighting so the right light level is maintain without using more energy than needed. (DiLouie, 2008). The lighting control is typically based on motion sensors (PIR sensors), switches, timers and photocells. Beside of that, lighting control often makes use of time schedules, states of the house and time of the year.

The lighting system can't be used for enhancing the flexibility because it doesn't contain any storage capacity. On the other hand it is a very important system for minimizing the energy consumption.

There is a number of lighting control system available on the marked. One example is the Philips Dynalite system[1]: this system can be used in e.g. houses, offices, hospitals among other. This system is a distributed system based on special load controllers (dedicated hardware). The controllers are connected to devices and communicate with other controllers through a proprietary protocol Dynet based on RS-485.

Another interesting initiative, regarding standardization of communication, is the DALI working group set up by leading manufacturers and institutions in the field of digital lamp/luminaire control to promote DALI technology and applications.[2] DALI stands for **D**igital **A**ddressable **L**ighting **I**nterface and is a protocol set out in the technical standard IEC 62386.

All these awardable lighting control systems are based on proprietary protocols and/or dedicated hardware. They are, because of that, not directly useable in the ENCOURAGE project.

In ENCOURAGE it has been decided to build a new software control module that was able to run on the core platform. This control module shall be able to handle the house specific controlling i.e. the glue layer. This means that it shall take care of the lighting control and, if needed, the local PI feedback controllers. It is very important that the system is easy to program and setup because the control requirement and the devices differ from house to house.

[1] Source: http://www.lighting.philips.com/main/subsites/dynalite/index.wpd

[2] Source: http://dali-ag.org/

The fact, that it shall be easy to setup and program the control system, requires that the system shall be based on a dedicated way to express the control task and to setup the interface to the rest of the ENCOURAGE system.

In the following, a dedicated control language for the ENCOURAGE system will be designed. This language is called the **ELL** language (**E**ncourage **L**ogical **L**anguage). The language shall be able to implement the so called glue layer.

5.3.1. The ELL language

The task for the Light control sub module is to secure the right light level when needed and on the other hand minimize the energy consumption for room lighting by dimming the light or by switching off the light when the room are empty. The developed language shall not only be useable for the light control module but shall be general enough to handle all the necessary logical control in an ordinary house. Take this into account, the main task for the system is to increase the comfort and at the same time to lower the consumption of recourses.

The main problem is: If you want to increase the comfort and at the same time want to lower the consumption of recourses you must have an intelligent control system, which can interconnect the different subsystems in an intelligent way. For instance, when you leave the house and lock the door, then all the light should switch off and the heating system should settle on a lower level and maybe the ventilation system should ventilate the house if the humidity is too high. Of course all these control actions shall be executed in a system, which can communicate with the door lock (the security system), the light, the heat and the ventilation system. In addition to that it is clear that the control actions depend on the state of the environment, the system itself and the people living in the house. For instance the control actions depend on whether it is day or night, summer or winter, if people are home, if they are physically active, reading, watching TV, sleeping and so on.

To overcome these problems it is necessary to have a general and easy to use tool for setting up the control function of each house. The control function shall be flexible enough so that it can handle, not only the lighting system, but also all other sub-systems in the house, like the heating system and the ventilation system if needed.

The lighting system will in the following be the base for specifying the requirement to the ELL language.

This light control is based on:

- The state of the room e.g. is it empty or is it occupied.
- Motion sensors.
- Time of the day, day of the week or time of the year etc.
- The outside light intensity.
- State of the users of the house (if available). E.g. at work, slipping, making foot, watching TV and so on.
- And other signals that are telling something about the need for light.

(Meyer, 2004) and (Krogh, 2006) describe different ways to program home control systems especially finite state machines (FSM). Based on these and the about description the following requirements for the language can be defined.

Requirements for the tool/control language

- It should be possible to implement finite state machines. (FSM)
- It should be possible to handle logical expressions.
- It should be possible to handle timers and timer logic
- It should be possible to handle hysteresis. (Bang-bang controller).
- It should be possible to include dates and real time in the logical expressions.
- It should be executed on the core platform.
- It should communicate with the IO devices through the ENCOURAGE middleware.
- It should be possible to program for a non-technician with only a short introduction course.

Existing language

Normally logical control tasks are handled by PLC's (Programmable Logical Controllers). PLC is programmed by different manufacture specified programming language. Many of these PLC languages are different kinds of ladder logic.

Windldr[3] are an example of a common ladder logic language. This language can be used of programming a group of different PLC's. Ladder logic is a general propose logical control language, and, because of that, can be used for programming the light system. Programming Ladder logic requires a technician that understands traditional hard wires relay implantation. Besides that, it is difficult to implement FSM's in Ladder logic.

Another way of programming PLC's is by State Logic. State Logic utilizes a natural language, known as ECLiPS or English Control Language Programming Software. This language allows the programmer to have the freedom of writing commands in natural words. (Dr. John R. Wright, 1999). FSM's can be handled by State Logic.

These different control language are not directly designed for our purpose and will be difficult to implement on the ENCOURAGE platform.

Madsen (Madsen, 2007) describes an easy to use language for this kind of control tasks. This language is hardware independent, easy to learn and has a simple and intuitive communication interface. The language consists of logical statement, based on input, states and time. The control structure is designed for describing FSM. ELL will be based on the ideas from (Madsen, 2007).

The ELL (Encourage Logical Language)

An ELL program consists of two parts:

[3] http://www.idec.com/language/english/manual/WindLDRTutorial.pdf

1. The Description of the interface to the middleware and predefined constants.
2. The control algorithm, or the program part, consisting of a number of FSM and statements.

5.3.2. Interface to the middleware

The middleware interface consists of four sections, one section for each of the four different types of I/O values i.e.:

- ON/OFF input signals, also called digital inputs '**DIGIINPUT**'
- ON/OFF output signals, also called digital inputs '**DIGIOUTPUT**'
- Analog input signals, also called digital inputs '**ANAINPUT**'
- Analog output signals, also called digital inputs '**ANAOUTPUT**'

Each of these signals is defined by a symbolic name **LightSwitch1_Beedroom** and the MacroCellID, the CellID and the DeviceID e.g.:

MacroCellID: **Jadevej**

CellID: **nr3**

DeviceID: **device7**

```
DIGIINPUT
  LightSwitch1_Beedroom          DEV : Jadevej nr3 device7;
  DoorBell                       DEV : Jadevej nr3 device10;
  Window1Open_Beedroom           DEV : Jadevej nr3 device2;

DIGIOUTPUT
  lamp1_Levingroom               DEV : Jadevej nr3 device13;

ANAINPUT
  _RoomTemperature_Kitchen       DEV : Jadevej nr7 device7;

ANAOUTPUT
  _LightDimmer_Beedroom          DEV : Jadevej nr7 device2;
```

The symbolic name is a self-defined variable name. This name shall give an easy understandable name to each IO signal. This name is used in the rest of the program. The scaling and unit of the variable are defined by the middleware.

5.3.3. The Program

After the interface part comes the program part. These two parte are separated by the word **PROGRAM**.

The program part consists of zero to N FSM and of zero to N statements, where N is only limited by the amount of memory in the runtime system (The ENCOURAGE core platform).

5.3.3.1. Finite state machine (FSM)

To illustrate the syntax for a FSM, a small example FSM, called **LivingRoom,** is shown below. This FSM has three states namely **INIT**, **Empty** and **Occupied**. It uses three variables from the middleware **LightSwitchOn**, **LightSwitchOff** and **PirSensor**.

The INIT state will be the first state to enter when the execution of the program are started. Figure: 9 show a state diagram that illustrates the function of this FSM. It is required that all FSMs have a INIT state, the naming and the number of all other stats can be chosen freely. In this example the FSM will, just after starting the program, switch to the empty state.

The two expressions i.e. **PirSensor OR LightSwitchOn** and **DELAY(NOT PirSensor,60*15) OR LightSwitchOff** are governing the transaction from one state to the other. E.g. if the FSM are in the Empty state and the PIR sensor are on or the light switch are pushed then the FSM will go to **Occupied** state.

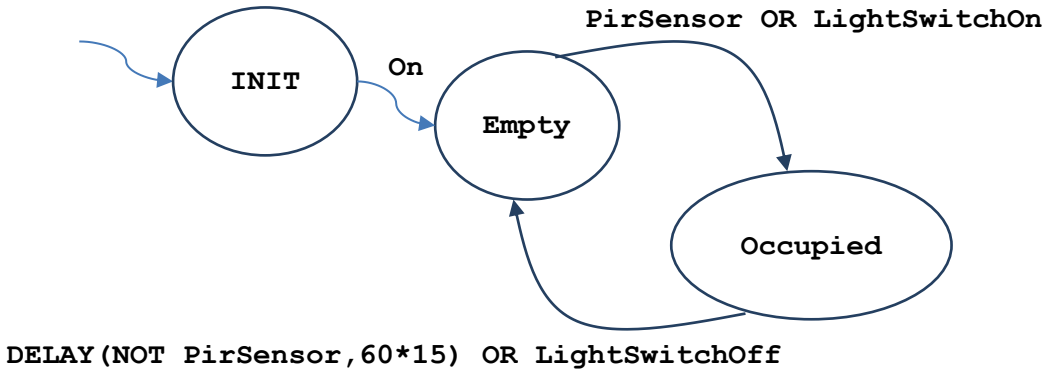


Figure 9: Simple FSM for determine if the living room are occupied or not

The program for this FSM will be like this:

```
FSM: LivingRoom
  STATE: INIT
    GO TO Empty WHEN ON;
  END
  STATE: Empty
    LivingRoomLight= OFF;
    LivingRoomHeating= OFF;
    GO TO Occupied WHEN PirSensor OR LightSwitchOn;
  END
  STATE: Occupied
    LivingRoomLight= ON;
    LivingRoomHeating= Heating.Running;
    GO TO Empty WHEN DELAY(NOT PirSensor,60*15) OR LightSwitchOff;
  END
END
```

if the FSM is in the state **Empty** then the statements inside the state are active, which means that the living room light are off, the living room heating are off and **PirSensor** OR **LightSwitchOn** are check and if true(ON) then the state machine will switch to state **Occupied**.

There can be as many statements, inside a state, as needed. These statements can be ordinary statements e.g.:

```
Light= lightSwitch AND NOT dayLight;
Ventilator= Light OR NOT DELAY(NOT Light, 60);
```

And as many GO TO statements as needed:

```
GO TO Empty WHEN DELAY(NOT PirSensor,60*15) OR LightSwitchOff;
```

Modern houses are equipped with a number of devices. These are for instance light switches, lamps, motion sensors, temperature sensors, humidity sensors, radiators, floor heating, ventilators, heat producing systems and so on. The control of all these devices interacts with each other. For instance the light, the heat and the ventilation system shall cooperate to save energy and to increase comfort. Furthermore the control depends on the state of the house, the time of the year (State of year), the state of the people, living in the house and so on.

This lead to the fact that a house control program, in general, consist of cooperating FSMs.

Requirement to the FSMs, implemented in the ELL-language can therefore be extended with these requirements:

- It shall be easy to synchronize multiple FSMs.
- It shall be possible to operate with nested FSMs.

Synchronization of multiple FSMs

The ELL language is a modified version of the DL-language defined in (Madsen, 2007). The main structure for a FSM is like this:

```
FSM: LivingRoom
  RESET= reset;
  HOLD= OFF;

  STATE: INIT
    ..
  END
  STATE: Empty
    ..
  END
  STATE: Occupied
    ..
  END
END
```

The FSM is named **LivingRoom** in the example above. This name is used, by the FSM itself or by other parts of the control system to determine in witch state the FSM is. For instance **LivingRoom.INIT** is true if the FSM is in the INIT state, otherwise it is false. After the naming, two statements are possible, a reset statement and a hold statement. If the **RESET** is set to **ON** (true) then the FSM will go to the INIT state and stay there, no matter what happens. If the **HOLD** statement is **ON** then the FSM stays in the current state unless the **RESET** is set to **ON**. The Hold statement can be used for synchronizing different state machines, but it is seldom used because there are more straight forward ways to synchronize FSM's.

As mentioned earlier: **LivingRoom.Empty** is ON if the state machine **LivingRoom** is in the **Empty** state and **LivingRoom.Occupied** is ON if the state machine **LivingRoom** is in the **Occupied** state. In both cases otherwise OFF. This will allow the programmer to synchronize different FSMs by using these in-state signals in the state transition statement e.g. **GO TO lightOn WHEN LivingRoom.Occupied**.

```
FSM: HeatingSystem
.
.
  STATE: Hearing
    _TempRoom1Ref= 21.0;
     GO TO Wait WHEN Ventilationsystem.Ventilate;
  END
  STATE: Wait
    _TempRoom1Ref= 0.0;
     GO TO Hearing WHEN NOT Ventilationsystem.Ventilate;
  END
.
.
END

_HeadRoom1 = PID(_TempRoom1Ref,_TempRoom1, _p,_i,_d,_st,ON,0,_db);

FSM: Ventilationsystem
.
.
  STATE: StopVentilate
    Fane= OFF;
    GO TO Ventilate WHEN StartVent;
    GO TO Ventilate WHEN _Humidity > _MaxHumidity;
  END

  STATE: Ventilate
    Fane= ON;
    GO TO StopVentilate WHEN DELAY(ON,300);
    GO TO StopVentilate WHEN _Humidity < _MaxHumidity-5;
  END
.
.
END
```

In this example there are two FSMs, **HeatingSystem** and **Ventilationsystem,** and a PID-controller, which are controlling the heat inlet to the room. The **HeatingSystem** sets the wanted temperature in the room **_TempRoom1Ref**. The **Ventilationsystem** FSM controls the ventilation. The ventilation starts when the logical value **StartVent** is set or when the humidity gets higher than **_MaxHumidity** and stops again after 300 sec. or when the humidity gets lover than **_MaxHumidity-5**. The PID controller is a function that takes the parameters: **_TempRoom1Ref,_TempRoom1**, the proportional factor, the Integral factor, the Derivative factor and the sampling time **, _p,_i,_d,_st**. The three last terms are preset, a preset value and a dead band, **ON,0,_db**. Preset and the preset value are used for manual operation and for cascade coupling of the controller. The dead band is used for handling noisy measurements.

Nested FSM's:

When designing a program for controlling a complex system it is often based on FSM, where state contains other FSM (nested FSMs). It is not possible in the ELL-language to define FSM inside FSM, because it will result in large FSMs that are difficult to maintain. The solution here is to use the reset statement. The **RESET** statement is a sort of activating/deactivating the FSM. This enables one FSM to activate/deactivate another FSM.

```
FSM: Season
.
.
  STATE: Winter
    Heating = ON;
    GO TO Spring WHEN (3 <= TIME.MONTH) AND (TIME.MONTH <= 5);
  END
END

FSM: HeatingSystem
  Reset= NOT Heating:
.
.
END
```

This results in a number of relatively small FSMs and thereby the FSMs will be easier to survey and maintain.

5.3.3.2. Expressions

All actions are defined by statements. Statements can be inside a state or outside the FSM's. If placed inside a state they are execute when the FSM are in that particular state otherwise not. If the statements are placed outside the FSM's then they are executed all the time and in parallel with all the active states.

There are three kinds of statements:

- Logical assignment: **Light= lightSwitch AND NOT dayLight;**
- Analog assignment: **_temperatureError= _RoomTempSetpoint - _Roomtemp;**
- GoTo: **GO TO Empty WHEN DELAY(NOT PirSensor,900) OR LightSwitchOff;**

The last part of the logical statement e.g. **lightSwitch AND NOT daylight**; are the logical expression.

Expressions can be ether logical or analog, depending on the type of the assignment. E.g. for assignment of digital output logical expressions are used, and for assignment for analog output analog expressions are used.

A logical expression consists of digital signals, analog signals, logical operators, comparison operators, functional operators and brackets.

Logical operators:

- **AND**
- **OR**
- **NOT**

Comparison operators:

- **>**
- **<**
- **>=**
- **<=**
- **=**

Functional operators:

- **DELAY**
- **FF**
- **PID**
- **TIME.MINUTE**
- **TIME.HOUR**
- **TIME.DAY**
- **TIME.WEEK**
- **TIME.WEEKDAY**
- **TIME.MONTH**
- **SQR**
- **FLASH**
- **QFLASH**

An analog expression consist of analog signals, functional operators, floating point operations and brackets.

Functional operators:

- **PID**
- **TIME.MINUTE**
- **TIME.HOUR**

- **TIME.DAY**
- **TIME.WEEK**
- **TIME.WEEKDAY**
- **TIME.MONTH**
- **SQR**

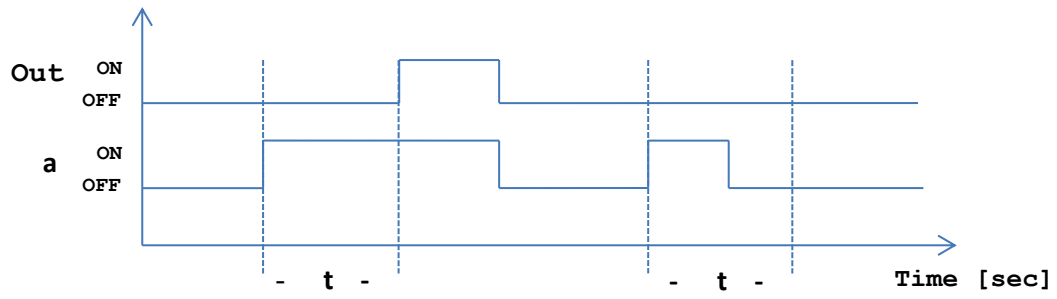
Floating point operations:

- **+**
- **-**
- *****
- **/**

5.3.3.3. Description of the functional operators

Timer function (DELAY)

This function is used whenever a timer is needed. It takes two arguments are; a logical expression and an analog expression. The logical expression defines the logical value to be delayed, and the analog expression defines the amount of time, in sec., the logical value has to be delayed.



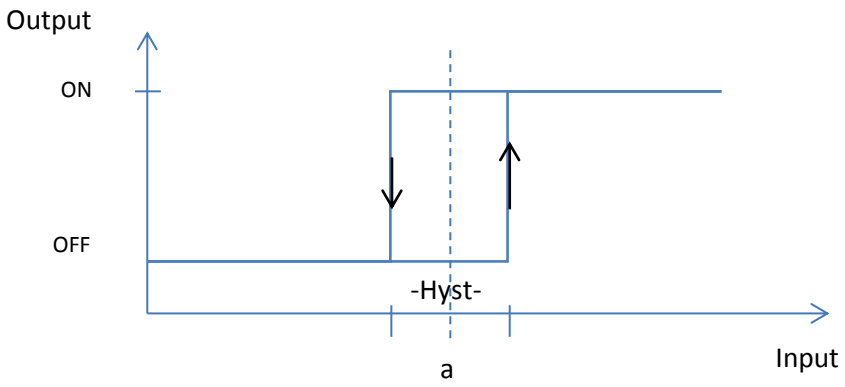

```
Out= DELAY(a,t);
```

Bang-Bang controller

Bang–bang controller (BBCTRL) also called an on–off controller, are based on a hysteresis element that switches between two states e.g. ON and OFF. This state is the output from the controller. This controller can not only be used as a feedback controller but also when a level signal is needed. It takes two arguments. These are two analog expressions. Where the first argument is the centre of the hysteresis and the second is the width of the hysteresis.

```
Output= BBCTRL(_Input – _a, _Hyst);
```

Time [sec]



Flip-flop

The Set-Reset flip-flop with toggle function is a so-called JK latch. This type of flip-flop is commonly used in control for handling ON/OFF states based on push-button. It takes two arguments that are two logical expressions.

JK latch truth table			
Set	Reset	Out	Comment
0	0	Out	No change
0	1	0	Reset
1	0	1	Set
1	1	Not Out	Toggle

```
out=FF(set,reset);
```

PID-controller:

The most common feedback controller is the PID-controller. In the absence of knowledge of the underlying process, a PID controller has historically been considered to be the best controller. (Bennett, 1993).

This equation shows how the output is calculated.

$$Out = K_p e(t) + K_i D_t \sum_{i=0}^{t} e(i) + K_d \frac{e(t)-e(t-1)}{D_t}$$

Where:

K_p: Proportional gain, a tuning parameter

K_i: Integral gain, a tuning parameter

K_d: Derivative gain, a tuning parameter

e: Error. The difference between the set point and the measured value.

D_t: The Time between each sample.

Beside of this equation a PID controller contains a number of features that are necessary in real applications. These are:

- Anti-integral windup on saturation.
- Bump-less transferee between auto and manual.
- Bump-less tuning.
- Deadband handling.

```
_Out= PID(_sp, _mv, _kp, _ki, _kd, _st, Auto, _Preset, _db);
```

Where:

_sp:	Are the set point.
_mv:	Are the measured value.
_kp:	Proportional gain.
_ki:	Integral gain.
_kd:	Derivative gain.
Auto:	ON/OFF signal that switch between auto and manual.
_Preset:	If manual then _out= _Preset;
_db:	If the absolute value of the error is less then _db then _out will not change;

5.3.3.4. Program example

A simple ELL program that controls the light in a living room is described below. The light is dimmed when the outside light is above a specific level. All the lights are switched off when the room is empty.

```
#-----------------------------------------------------
DIGIINPUT
  LivingRoomLightButtonOn        DEV : Jadevej nr3 device7;
  LivingRoomLightButtonOff       DEV : Jadevej nr3 device3;
  LivingRoomMotionDetect         DEV : Jadevej nr3 device4;

DIGIOUTPUT
  LivingRoomAllLight             DEV : Jadevej nr3 device2;

ANAINPUT
  _OutdoorLight                  DEV : Jadevej nr3 device6;

ANAOUTPUT
  _LivingRoomLightDimmer         DEV : Jadevej nr3 device5;

ANACONST
  _LightLevel          = 80;
  _LightP              = 3.2;

PROGRAM

FSM: LivingRoomLightBimmer
  STATE: INIT
    _LivingRoomLightDimmer = 100;
    GO TO dimme WHEN NOT BBCTRL(_OutdoorLight - _LightLevel, 5.0)
                     AND LivingRoom.Occupied;
  END
  STATE: dimme
    _LivingRoomLightDimmer= (100 - _OutdoorLight)* _LightP;
    GO TO INIT WHEN BBCTRL(_OutdoorLight - _LightLevel, 5.0);
    GO TO INIT WHEN LivingRoom.Empty;
  END
END


FSM: LivingRoom
  STATE: INIT
    GO TO Empty WHEN ON;
  END
  STATE: Empty
    LivingRoomAllLight = OFF;
    GO TO Occupied WHEN LivingRoomLightButtonOn;
  END
  STATE: Occupied
    LivingRoomAllLight = ON;
    GO TO Empty WHEN DELAY(LivingRoomMotionDetect,60*15)
                     OR LivingRoomLightButtonOff;
  END
END
```

5.3. Appliances Control

One of the load types that can contribute dramatically to energy saving and cost minimization are manually controllable loads including home appliances and multimedia devices. The system-wise controller can be designed to recommend a certain time period for using a specific device. The idea is that the controller has access to the pattern of power consumption of an electrical device during a day.

Here we have described the idea of controlling such electrical devices. Consider three devices with different pattern as shown in Figure 10. The power consumption can be shifted in batch and not continuously during a certain time period determined by the user.

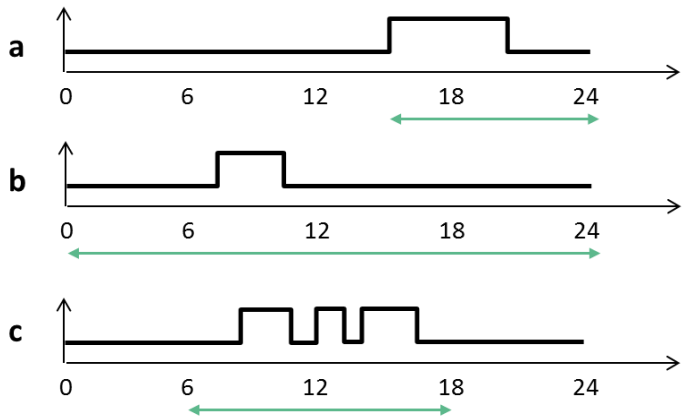


Figure 10. Power consumption pattern of three manually controlled loads. The consumption can be shifted in batch within the time horizon specified in green.

Therefore, the information needed from each load is its consumption pattern during a day, the pattern of consumption each time the device turns on until it switches off again and the time horizon during which the consumption can be shifted.

This part of design is not included yet in the simulation results even if, if this feature is desired to be embedded into the energy management controller, the above mentioned data would be needed for each device.

6. Building Level Controller

Block diagram of the hierarchical supervisory controller at the building level is depicted in figure 11. A Model Predictive Controller (MPC) was chosen as the system-wide controller (Maciejowski, 2012). The reason for this choice is that all the system disturbances and future references can systematically be incorporated into the MPC. On the other hand, the middleware has to provide a foreseen estimate of surplus and demand power to the power scheduler at the top layer. This feature would already be embedded in the system-wide controller if we choose a receding horizon controller. At the bottom layer we have designed Proportional Integral (PI) controller for heating loads and light curtailment is done based on inputs from sensors measuring light or detecting motion. This will be implemented using the ELL language.

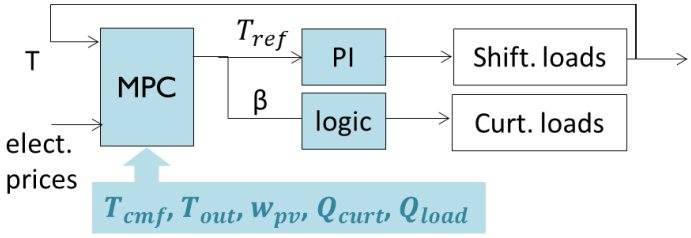


Figure 11. Block diagram of the hierarchical supervisory controller. Inputs to the MPC are: intraday market price of electricity, pre-set user-defined comfort temperature T_{cmf}, forecast of outdoor temperature T_{out}, Prediction of electricity generation by PVs W_{PV}, Predicted consumption profile of curtailable Q_{curt} and non-flexible loads Q_{load}. Output control signals are: a reference temperature T_{ref} to HVAC system controller and a coefficient of curtailment β to the lighting system controller.

6.1. Model Predictive Controller

MPC is based on iterative, finite horizon optimization of a plant model. At time *k* the current plant state is sampled and a cost minimizing control strategy is computed (via a numerical minimization algorithm) for a relatively short time horizon in the future: $[k, k+N]$. Specifically, an online or on-the-fly calculation is used to explore state trajectories that emanate from the current state and find (via the solution of Euler-Lagrange equations) a cost-minimizing control strategy until time *k+N*. Only the first step of the control strategy is implemented, then the plant state is sampled again and the calculations are repeated starting from the now current state, yielding a new control and new predicted state path. The prediction horizon keeps being shifted forward and for this reason MPC is also called receding horizon control. Although this approach is not optimal, in practice it has given very good results. A discrete MPC scheme is shown in Figure .

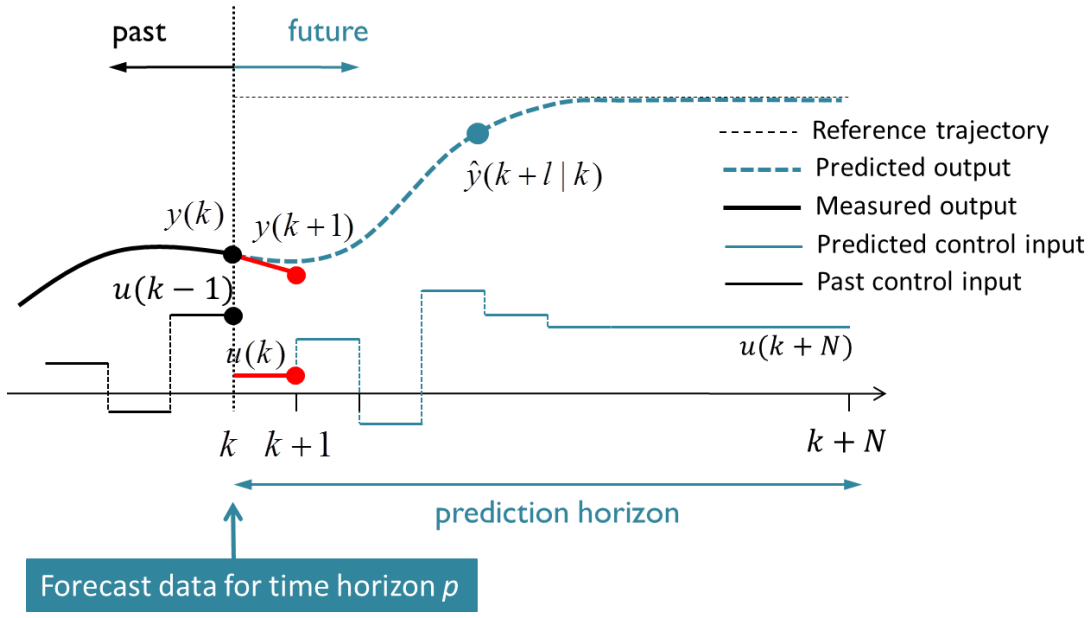


Figure 12. Discrete MPC Scheme

6.2. Problem Formulation

In this section the formulation of a model-based predictive controller as the building-level load controller is described.

6.2.1 System Modelling

Dynamics of a house and its heating loads are governed by the following first order model. This model is accurate enough for control design purposes in practice.

$$C\dot{T} = UA(T_{out} - T) + Q_{heat}$$

Where:

- U: thermal transmittance $[kW/m^2\ ^\circ C\ ]$
- A: surface area $[m^2]$

T is the room temperature and T_{out} is the outdoor temperature. C is the heat capacity of the floor material and the houses air, envelope, and furniture. Q_{heat} is dissipated heat from floor heating system or any other system to the room's air. In the case of electric floor heating Q_{heat} is equal to the electric power put to the floor. Heat flow from an electric floor heating system is controlled using a PI controller to regulate the building temperature. The PI controller in state space form is given:

$$\dot{\xi} = \frac{K_p}{T_{int}}(T_{ref} - T)$$

$$Q = K_p(T_{ref} - T) + \xi$$

Where:

- K_p: Proportion gain.
- T_{int}: Integration time.
- T_{ref}: Are the reference temperature. [$^{\circ}C$].
- ξ: Integral state.

Considering the sampling time, $h = t_{k+1} - t_k$, closed-loop discrete time system is:

$$T(k+1) = a_{11}T_k + a_{12}\xi(k) + b_1 T_{ref}(k) + e_1 T_{out}(k)$$
$$\xi(k+1) = a_{21}T(k) + a_{22}\xi(k) + b_2 T_{ref}(k)$$
$$Q_{heat}(k) = c_1 T(k) + c_2 \xi(k) + d T_{ref}(k)$$

Where:

- $a_{11} = 1 + \frac{h}{C}UA$
- $a_{12} = \frac{h}{C}$
- $a_{21=} h\frac{K_p}{T_{int}}$
- $a_{22} = 1$
- $b_1 = \frac{h}{C}K_p$:
- $b_2 = h\frac{K_p}{T_{int}}$
- $c_1 = -\frac{h}{C}K_p$
- $c_2 = \frac{h}{C}$:
- $d = \frac{h}{C}K_p$
- h: Sampling time.
- C: thermal capacitance $[kJ/kg\ ^{\circ}C]$

Parameters of the above equation are given in the following matrix form:

$$\begin{bmatrix} T(k+1) \\ \xi(k+1) \end{bmatrix} = \begin{bmatrix} 1 + \frac{h}{C}UA & \frac{h}{C} \\ h\frac{K_p}{T_{int}} & 1 \end{bmatrix} \begin{bmatrix} T(k+1) \\ \xi(k+1) \end{bmatrix} + \begin{bmatrix} \frac{h}{C}K_p & \frac{h}{C}UA \\ h\frac{K_p}{T_{int}} & 0 \end{bmatrix} \begin{bmatrix} T_{ref}(k) \\ T_{out}(k) \end{bmatrix}$$

$$Q_{heat} = \begin{bmatrix} -\frac{h}{C}K_p & \frac{h}{C} \end{bmatrix} \begin{bmatrix} T(k+1) \\ \xi(k+1) \end{bmatrix} + \begin{bmatrix} \frac{h}{C}K_p \end{bmatrix} T_{ref}(k)$$

6.2.2. Optimization Problem

Optimization Problem is formulated in a receding horizon framework. An economic solution is achieved by penalizing purchase from the utility grid in the cost function. Also, discomfort i.e. deviation from a comfort temperature profile is penalized. The other term in the cost function is related to curtailment penalty.

$$\min_{T_{ref},\beta,w_{sell},w_{mg}} \sum_{k=1}^{N} \rho_{discmf}\left|T(k) - T_{cmf}(k)\right| + \rho_{buy}(k)w_{buy}(k) - \rho_{sell}(k)w_{sell}(k) + \rho_{curt}(k)\beta_{curt}(k)Q_{curt}(k)$$

$$s.t. \quad T(k+1) = a_{11}T_k + a_{12}\xi(k) + b_1 T_{ref}(k) + e_1 T_{out}(k)$$

$$\xi(k+1) = a_{21}T(k) + a_{22}\xi(k) + b_2 T_{ref}(k)$$

$$Q_{heat}(k) = c_1 T(k) + c_2 \xi(k) + d T_{ref}(k)$$

$$\left|T(k) - T_{cmf}(k)\right| \le T_{tol}(k)$$

$$0 \le Q_{heat}(k) \le Q_{max}$$

$$0 \le \beta_{curt}(k) \le 1$$

$$0 \le w_{sell}$$

$$w_{buy}(k) = Q_{heat}(k) + Q_{load}(k) + \left(1 - \beta_{curt}(k)\right)Q_{curt}(k) + w_{sell}(k) - w_{pv}(k)$$

In which *k* is the time instant and *N* is the prediction horizon.

ρ_{discmf} and ρ_{curt} are coefficients of penalty for thermal discomfort and power curtailment of the appertaining curtailable loads, respectively.

Control variables are curtailment coefficient $\beta_{curt}(k)$, the selling power to the grid $w_{sell}(k)$ and the reference temperature of the building $T_{ref}(k)$.

Predicted signals and system disturbances include comfort temperature profile $T_{cmf}(k)$, the buying and selling price from the grid $\rho_{buy}(k)$ and $\rho_{sell}(k)$, the discomfort penalty ρ_{discmf}, the curtailment penalty ρ_{curt},, the curtailable and inflexible loads Q_{curt} and Q_{load}, and electricity generation of PV cells w_{pv}, all for the next 24 hours.

Boundaries on building temperature T_{tol} and maximum heat flow Q_{max} are the known parameters.

7. Simulation results

Results of two simulation scenarios for energy management of one building are presented and discussed in this section. Parameters of the building dynamics are chosen based on data from a low energy building i.e. very similar to the houses in Jadevej case study. The sampling time is one hour, which is equal to the time interval of variations in predicted price profile. Predicted signals are assumed to be available one day ahead. This is specifically important for the price profile which is settled in an hourly basis a day ahead in the Elspot trading system. The power price is determined by balance between supply and demand and fixed from 12:45 CET each day to be applied from 00:00 CET the next day [19]. Therefor the prediction horizon for MPC is chosen as 1 day. Price signals are taken from the Nordpool database for a period of one week in February 2013. Weather data is also taken from Danish Meteorological Institute (DMI). PV cells production data is achieved from Jadevej case study.

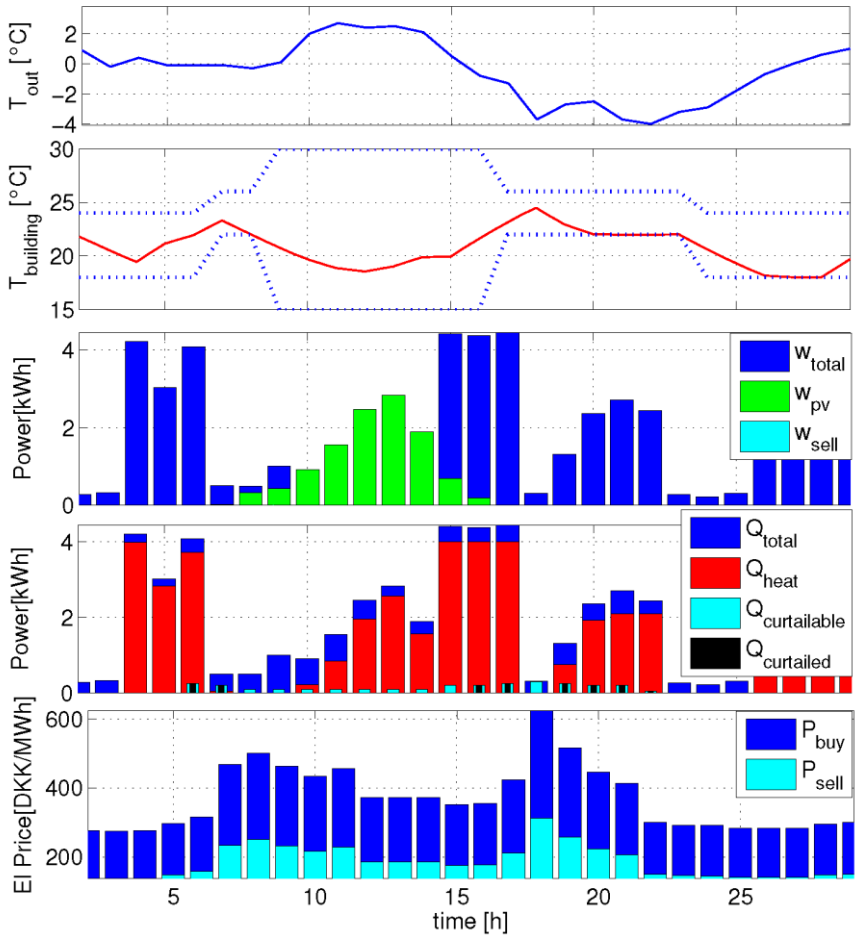


Figure 13. Energy management of one building in a day. Building temperature is limited between two temperature profiles which are customized based on the user

preferences. $Q_{curtailable}$ ***is the total predicted power need of curtailable loads and*** $Q_{curtailable}$ ***is the curtailed power i.e.*** $(1-\beta)Q_{curtailable}$***. At the peak of price, the flexible loads are zero and only inflexible load is consuming the expensive power.***

In the simulations, there is not considered electricity trading among the buildings yet possible. Figure 13 depicts energy management by shifting and shedding based on electricity price, weather data, and prediction of non-flexible loads.

In the simulation, discomfort and curtailment costs in the cost function are chosen such that they are highest in two time horizon, one from 6:00 to 8:00 AM and the other from 17:00 to 22:00. It is obvious in figure 13 that during the two time horizons, comfort is mostly fulfilled comparing daytime or over midnight.

Simulation result over one week is depicted in figure 14. Simulations show that the proposed controller is 33% more economic compared to an energy minimizing MPC which considers a constant price in whole one day. Compared to the MPC that only optimizes comfort, the proposed controller saves app. 50% in electricity consumption cost. However, these savings are excluding energy tax which in Denmark costs 0.82DKr=kWh.

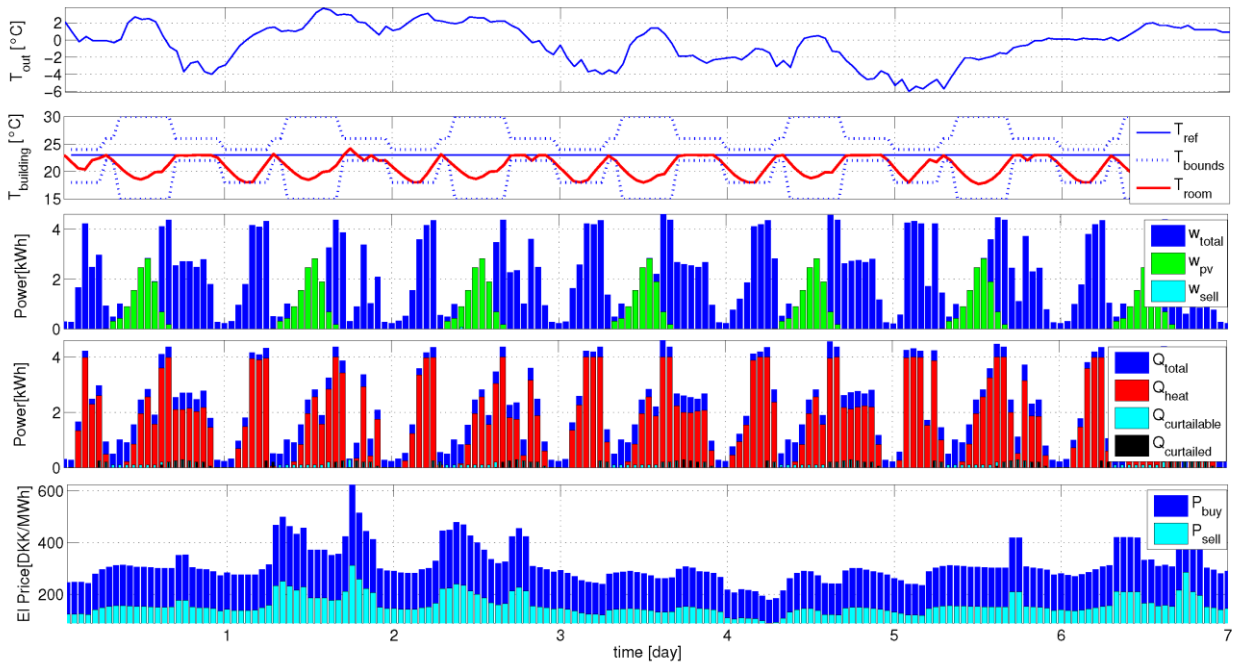


Figure 14. Energy management of one building during one week.

A building with a higher flexibility potential could benefit from lower electricity consumption costs in the future smart grid. Energy demands of the flexible loads can be curtailed or shifted to consume electricity at lower price rates. Therefore the more a building increases the flexibility the more it can save in electricity bill. Figure 15 shows the amount of savings in electricity bill against flexibility, which is defined as the boundaries around room temperature.

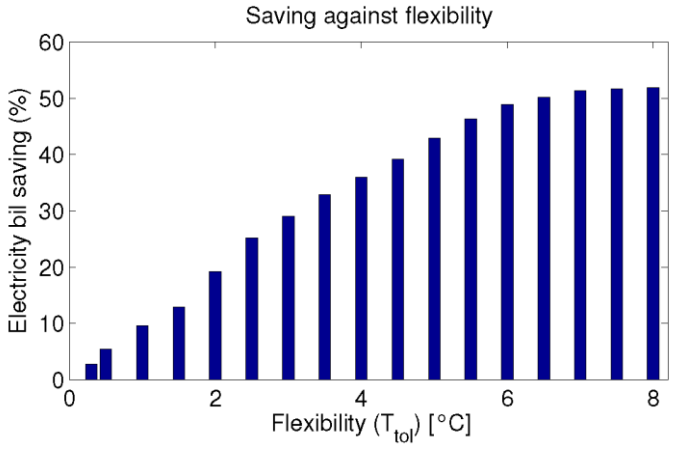


Figure 15. Percentage of consumption cost saving against flexibility. Saving is calculated based on consumption cost when thermal comfort is maximum i.e. Ttol very close to zero.

8. Conclusion

This work defines a framework for energy management on building level. Deliverable D5.4 Energy management system will describe the total microcell based energy management system. Here the focus is on the two bottom layers of the proposed supervisory controller and formulated a MPC as the cell-level controller. The MPC works in combination with device layer controllers by supplying them with control references. The results show that with reliable price predictions substantial savings in energy costs are obtainable for a consumer which implements a predictive controller.

The achieved results will be used in the microcell energy management system, where we add other similar building modules and also the top layer energy balancer. This level of controller will control energy trade among the buildings and also the utility grid.

The saving depends strongly on how flexible the people living in the house are. Flexibility here means the size of the acceptable temperature interval around the comfort temperature. If this interval size are zero then the system has on room for optimizing the consumption, and because of that it perform like an ordinary control system. On the other hand if the people living in the house accept a given degree of discomfort then the sawing potential is up to 50 %.

A dedicate language (ELL) for easy deployment and setup of the building level control system has been developed. A prototype of an integrated development environment for the ELL language (ellIDE) is implemented. The documentation of ELL and ellIDE are available in the report: ELL (ENCOURAGE Logical Language) Manual.

Bibliography

A. Gomes, C. H. (2007). A Multiple Objective Approach to Direct Load Control Using an Interactive Evolutionary Algorithm. *IEEE Transactions on Power Systems*.

Alvarado, F. (1999). The Stability of Power System Markets. *IEEE Transactions on Power Systems*, 505-511.

Alvarado, F. (2003). Is system control entirely by price feasible? *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*.

Alvarado, F. a. (2003). Stability analysis of interconnected power systems coupled with market dynamics. *Power Systems, IEEE Transactions on*, 695 -701.

Annaswamy, A. K. (2011). Wholesale energy market in a smart grid: Dynamic modeling, and stability. *IEEE Conference on Decision and Control (CDC).*

Arne Skou, e. a. (2012, November). *Embedded Intelligent Controls for Buildings with Renewable Generation and Storage*. Retrieved from http://www.encourage-project.eu/

Bemporad, A. a. (1999). Control of Systems Integrating Logic, Dynamics, and Constraints. *Automatica*, 407-427.

Bennett, S. (1993). A history of control engineering.

Biege, B., Andersen, P., Pedersen, T. S., Nielsen, K. M., & Stoustrup, J. a. (2013). Smart grid dispatch strategy for on/off demand-side devices. In *Proceedings of the European Control Conference 2013.* IEEE Pres.

Biegel, B. (2012). *Flexibility Interface - Information Modeling for Direct Control.* Aalborg, Denmark: iPower: http://www.ipower-net.dk/Publications.aspx.

Bosch, A. J. (2009). Real-time control of power systems using nodal prices. *International Journal of Electrical Power and Energy Systems*.

DiLouie, C. (2008). *Lighting controls handbook.* Lilburn, Ga. [u.a.]: Fairmont Press [u.a.] .

Dr. John R. Wright, J. (1999). The Debate Over Which PLC Programming Language is the State-of-the-Art. *Journal of Industrial Technology*, Volume 15, Number 4.

edersen, T. S. (2011). Using Heat Pump Energy SStorage in the Power Grid. *IEEE International Conference on Control Applications.*

Fatemeh Tahersima, P. P. (2013). An Intuitive Definition of Demand Flexibility in Direct Load Control. *IEEE Conference on Control Applications.* Hyderabad, India.

Frauke Oldewurtel, A. U. (2010). Reducing Peak Electricity Demand in Building Climate Control using Real-Time Pricing and Model Predictive Control. *49th IEEE Conference on Decision and Control.*

Gehrke, A. M. (2013). An Overview of Load Control Policies in Buildings for Smart Grids. *IEEE International Conference on Smart Energy Grid Engineering.*

Glielmo, A. P. (2011). Energy efficient microgrid management using Model Predictive Control. *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC).*

Glielmo, A. P. (2011). Energy efficient microgrid management using Model Predictive Control. *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC).*

Gomes, A., & Martins, C. H. (2007). A Multiple Objective Approach to Direct Load Control Using an Interactive Evolutionary Algorithm. *IEEE Transactions on Power Systems*.

Gong, J., Xie, D., & Zhang, C. J. (2011). Multiple Objective Compromised Method for Power Management in Virtual Power Plants. *Energies*.

Heussen, K. a. (2012). Indirect Control for Demand Side Management - A Conceptual Introduction. *3rd IEEE PES Innovative Smart Grid Technologies (ISGT).* Berlin, Germany.

Heussen, K. a. (2012). Unified System-Level Modeling of Intermittent Renewable Energy Sources and Energy Storage for Power System Operation. *IEEE Systems Journal*, 140-151.

Hommelberg, M., Warmer, C., Kamphuis, I., & Kok, J. A. (2007). Distributed Control Concepts using Multi-Agent technology and Automatic Markets: An indispensable feature of smart power grids. *2007 IEEE Power Engineering Society General Meeting.*

Jokic, A. a. (2009). Real-Time Control of Power Systems Using Nodal Prices. *International Journal of Electrical Power and Energy Systems*, 552-530.

Jørgense, R. H. (2012). Economic Model Predictive Control for Building Climate Control in a Smart Grid. *Innovative Smart Grid Technologies (ISGT).*

Juelsgaard, M. A. (2013). Stability Concerns for Indirect Consumer Control in Smart Grids. *Proceedings of European Control Conference (ECC).*

K. Edlund, J. a. (2011). Hierarchical model-based predictive control of power plant portfolio. *Control Engineering Practice*.

Krogh, G. E. (2006). Programming Discrete Control Systems Using State Machine Templates. *The 8th International Workshop on Discrete Event Systems.*

Maciejowski, J. (2012). *Predictive Control with Constraints.* Harlow, UK: Prentice Hall.

Madsen, G. D. (2013). Chance-Constrained Optimization of Demand Response to Price Signals. *IEEE Transactions on Smart Grid.*

Madsen, P. (2007). Dedicated Programming Language for Small Distributed Control Devices. *Second IEEE Conference on Industrial Electronics and Applications. (ICIEA 2007).*

Mardavij Roozbehani, M. A. (2012). Volatility of Power Grids Under Real-Time Pricing. *IEEE Transactions on Power Systems.*

Meyer, G. (2004). *Smart Home Hacks: Tips & Tools for Automating Your House.* O'Reilly.

Mitter, M. R. (2010). On the Stability of Wholesale Electricity Markets under Real-Time Pricing. *49th IEEE Conference on Decision and Control.*

Moslehi, K. a. (2010). A Reliability Perspective of the Smart Grid. *IEEE Transactions on Smart Grid*, 57-64.

Nerea Ruiz, I. C. (2009). A Direct Load Control Model for Virtual Power Plant Management. *IEEE Transactions on Power Systems*.

ODYSSEE-MURE project. (2012). *Energy Efficiency Trends in Buildings in the EU.* Europe: Intelligent Energy Europe.

Parisio, A. a. (2011). Energy Efficient Microgrid Management Using Model Predictive Control. *50th Conference on Decision and Control.*

Pedersen, T. S., Andersen, P., Nielsen, K. M., & Staermose, H. L. (2011). Using Heat Pump Energy SStorage in the Power Grid. *IEEE International Conference on Control Applications.*

Petersen, M. H. (2013). A Taxonomy for Flexibility Modeling and a Computationally Efficient Algorithm for Dispatch in Smart Gerids. *American Control Conference.* Washington, DC, US.

Pinson, P. (2012). *Indirect Control by Prices - Basic Concepts, Applications and Requirements.* Copenhagen, Denmark: iPower:http://www.ipower-net.dk/Publications.aspx.

Røpke, I. a. (2010). Information and Communication Technologies - A New Round of Household Electrification. *Energy Policy*, 1764-1773.

Schweppe, F. C. (1988). *Spot Pricing of Electricity.* Springer.

Tahersima, F. a. (2011). Contribution of Domestic Heating Systems to Smart Grid Control. *IEEE Conference on Decision and Control.*

Tahersima, F. a. (2012). Economic COP Optimization of a Heat Pump with Hierarchical Model Predictive Control. *IEEE Conference on Decision and Control.*

Trangbæk, K., & Bendtsen, J. D. (2011). Hierarchical Control for Smart Grids. *Proceedings of the 18th IFAC World Congress.*

Appendix A. Building Load Types

Ordinary houses contain a large number of electricity consuming products. All These products are divided into a number of categories these are:

- Lighting
- Heating and ventilation
- Washing
- Fridge/freeze
- Cooking
- IT and electronic
- Others

In the following each group is described in a more details.

- Coverage [quantity/household]: The number of entities per household
- Consumption [kWh/year]: The average consumption per year
- On [Watt]: The consumption when switch on and if important the standby consumption

A.1. Lighting

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
LED	0,95	8	4
Incandescent Bulb	4,34	394	43
Halogen	5,93	76	34,19
Crystal Buld	0,26	3	4
Light chain	0,59	26	25
Light sensor	0,48	11	1
Fluorescent tubes	1,40	246	35
Compact fluorescent lamps	8,18	28	8

A.2. Heating and ventilation

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
Circulating Pump, self-regulating		263	
Circulating Pump, step regulated step 3		525	60
El towel dry	0,41		68
El radiator	0,10		371

El Water heater per person	0,10	850	
Floor heating (best insulated housing)		7000 (140m2)	
Floor heating (new houses)		10500 (140m2)	
Floor heating (old houses)		11900 (140m2)	
Heat pump. Air to air	0,02		1460
Heat pump. Air to water	0,03		2400[4]
Heat pump: Water to water.	0,01		2800[5]
Ventilation without heat recovery			
Ventilation with heat recovery			
Ventilation with heat pump.			500
Dehumidifier		96	100,00

A.3. Washing

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
Dishwasher		290[6]	1500
Dryer	0,67	218	2500
Washing Machine		333	2000

A.4. Fridge/freeze

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
Ice Machine			
Chest Freezer	0,67	312	140
Fridge with freezer box	0,34	240	300
Fridge without freezer box	0,22	197	
Upright Freezer	0,57	142	

[4] http://www.toshiba-aircon.jp/press/2009/img/090731/cat_estia.pdf

[5] http://www.jordvarme.dk/produkter/jordvarme-dc.html

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

A.5. Cooking

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
Blender		4	300
Water boiler	0,65	64	2000
El oven	0,85	11	2367
Exhaust Hood	0,71	76	90
Espresso	0,83	75	184
Food processor	0,12	9	500
Coffee Grinder	0,03	1	250
Coffee Machine	0,21	69	1250
Boiling Plates	0,62	96	1347
Small Oven	0,91	126	1653
Microwave Oven	0,10	96	1500
Food mixer	0,76	19	972
waffle iron	0,40	28	1029
Quooker	0,02	50	3000
Toaster		27	900
Stove, induction hot plates incl. Standby		215	
Stove, ceramic hot plates incl. Standby		223	
Stove and oven incl. standby		156	

A.6. IT and electronic

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
Blue ray player			7
Clock radio	0,65	16	5
DAB radio	0,36	22	7
TV	2,04	171	139
DVD	0,86	6	11
Ghettoblastere	2,04	35	139
Chanel selector	0,38		
Parabola	0,33	48	10

Projector	0,13	18	10
Gaming consoles	0,05	96	80
Home audio	0,57	107	124
Surround	0,90	42	14
Video	0,32	56	137
Mobile phone		10	30
Burglar alarm	0,14	35	5
Laptop	0,49		50
Desktop computer	0,49		225
Printer			370
Monitor			150
Decoder incl. Standby		144	17
ADSL		88	10
Fax incl. Standby		88	10

A.7. Other

Appliances name	Coverage [quantity/household]	Consumption [kWh/year]	On [Watt]
Sauna			9000
Solarium	0,02		
Spa bath	0,03		
Vacuum Cleaner	0,92	31	2067
Swimming pool	0,01		2500
Answering machine	0,08		3
Terrace Heater	0,06		1500
Aquarium 54 litres incl. Heating, pump and light		200	
Electric razor		14	10
Battery charger		less than 1	3
Motion detector		9	1
Drilling machine		9	750
Drain pump		23	250
Elevation bed		4	

Electric hedge trimmer		3	1000
Heated towel rail		183	50

Sources:

http://www.oksolar.com/technical/consumption.html

http://www.goenergi.dk/forbruger/alt-om-energiforbrug/elforbrug/hvor-bruger-du-mest-stroem

http://www.toshiba-aircon.jp/press/2009/img/090731/cat_estia.pdf

http://www.jordvarme.dk/produkter/jordvarme-dc.html

http://www.goenergi.dk/

Appendix B. Flexibility Type of Building Loads

The flexibility of the appliances in households is assessed based on the following parameters:

- Hours of delay in the start of the operation of the device
- The length of the intermediate interruptions during the operation admitted by the devices

The degree of flexibility has been assessed based on the following table, which shows the hours of delay in the start of operation, the duration of interruption and the possibility of temporarily storing energy offered by the devices.

DEGREE OF FLEXIBILITY	HOURS OF DELAY IN THE START OF OPERATION	INTERMEDIATE INTERRUPTIONS	TEMPORARILY STORAGE OF ENERGY
High	4-8	Long	High
Medium	1-4	Short	Medium
None	0	None	Low

The following subsections analyse the flexibility of the most common appliances in households.

B.1. Lighting

In general, lighting has no flexibility, as it must be operational when needed.

APPLIANCES NAME	FLEXIBILITY
LED	None
Incandescent Bulb	None
Halogen	None
Crystal Bulb	None
Light chain	None
Light sensor	None
Fluorescent tubes	None
Compact fluorescent lamps	None

B.2. Heating and ventilation

Heating and ventilation allow, in general, a certain degree of flexibility, as they can have some short periods of interruption or short delay in the start of operation.

APPLIANCES NAME	FLEXIBILITY
Circulating Pump, self-regulating	Medium
Circulating Pump, step regulated step 3	Medium
El towel dry	Medium
El radiator	Medium
Hot Water	Medium
El Water heater per person	Medium
Floor heating (best insulated housing)	Medium
Floor heating (new houses)	Medium
Floor heating (old houses)	Medium
Sun heating	None
Sun heating for hot water	None
Heat pump. Air to air	Medium
Heat pump. Air to water	Medium
Heat pump: Water to water.	Medium
Ventilation without heat recovery	Medium
Ventilation with heat recovery	Medium
Ventilation with heat pump.	Medium
Dehumidifier	Medium

B.3. Washing

Appliances related to washing are the ones which most flexibility permit, as their start can be delayed.

APPLIANCES NAME	FLEXIBILITY
Dishwasher	High
Dryer	High
Washing Machine	High

B.4. Fridge/Freeze

In general, fridge and freeze allow short periods of interruption during their operation, as well as temporary storage of energy.

APPLIANCES NAME	FLEXIBILITY
Ice Machine	Medium
Chest Freezer	Medium
Fridge with freezer box	Medium
Fridge without freezer box	Medium
Upright Freezer	Medium

B.5. Cooking

Generally speaking, cooking appliances do not allow flexibility, as they must be in operation when needed.

APPLIANCES NAME	FLEXIBILITY
Blender	None
Water boiler	None
El oven	None
Exhaust Hood	None
Espresso	None
Food processor	None
Ice Cube Machine	None
Coffee Grinder	None
Coffee Machine	None
Boiling Plates	None
Small Oven	None
Microwave Oven	None
Food mixer	None
Waffle iron	None
Quooker	Medium
Toaster	None
Stove, induction hot plates incl. Standby	None
Stove, ceramic hot plates incl. Standby	None
Stove and oven incl. standby	None

B.6. TV, Video and Audio

These appliances do not offer any degree of flexibility, as their start cannot be delayed.

APPLIANCES NAME	FLEXIBILITY
Blue ray player	None
Clock radio	None
DAB radio	None
TV	None
DVD	None
Channel selector	None
Parabola	None
Projector	None
Gaming consoles	None
Home audio	None
Surround	None
Video	None
El lawn mower	None
Garden Pool	Medium
Garden fountain	Medium
Hand Vacuum Cleaner	None
Hair dryer	None
Antenna amplifier	None
Decoder incl. Standby	None
ADSL	None
Fax incl. Standby	None

B.7. Other

APPLIANCES NAME	FLEXIBILITY
Sauna	Medium
Solarium	Medium
Spa bath	Medium
Vacuum Cleaner	None
Swimming pool without heating	Medium
Swimming pool	Medium
Answering machine	None

APPLIANCES NAME	FLEXIBILITY
Terrace Heater	Medium
Wireless phone	None
Burglar alarm	None
Laptop	None
Desktop computer	None
Printer	None
Monitor	None
Aquarium 54 litres incl. Heating, pump and light	Medium
Shaving	Medium
Battery charger	High
Motion detector	None
Drilling machine	Medium
Drain pump	Medium
Elevation bed	None
Electric hedge trimmer	None
Heated towel rail	Medium

Appendix C. Requirements

The requirements for SCM are defined in WP2 deliverable 2.2. There are 30 requirements for the SCM. These requirements are divided into three Priorities: 12 of them have Must Priority, 14 have Should Priority and 4 have Could Priority.

SCM consists of three parts according to the ENCOURAGE architecture. These parts are Energy manager, Load manager and Local generation control.

In the following the requirement are listed, and it is specified which part of SCM each requirement are related to.

The 12 'Must' requirements are

1.

ID	ATOS1.1.1
Rationale (The reason)	Improvement of consume of energy
Supervisory control part	Energy manager + Load manager
Action	Improve the consumption using the platform in buildings

2.

ID	ENO1.1.19a + ESV 1.1.1 + EZM 1.1.44
Rationale (The reason)	The system shall monitor real-time disaggregated consumption
Supervisory control part	Energy manager
Action	Real-time data plus recorded data

3.

ID	ADV.1.1.6
Rationale (The reason)	System should provide basic control (ON/OFF) for individual appliances
Supervisory control part	Energy manager + Load manager
Action	switch on/off

4.

ID	ADV.1.1.9
Rationale (The reason)	System has to react to events sent from the middleware (configurable alarms)
Supervisory control part	Energy manager + Load Manager + Local generation control
Action	React according rules, e.g. send an alarm to BI & EB or a message to a user

5.

ID	EZM.1.1.30 +EZM.1.1.5 + ENO 1.1.33 + ENO 1.1.34
Rationale (The reason)	Supervisory control will define algorithms to control in house devices (example increase/decrease temperature, floor heating, water tank temp, home appliances, UPS etc.... send notification)
Supervisory control part	Energy manager + Load manager
Action	Control Action Device

6.

ID	EZM.1.1.41
Rationale (The reason)	All data and user information gathered should be protected with reference to Data Information Directives / Regulations. It is important that no information can be attained by unauthorized parties.
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Arch requirement

7.

ID	EZM.1.1.42
Rationale (The reason)	All data transmission must be carried out over a secure connection
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Arch requirement

8.

ID	EZM.1.1.36 + ENO1.1.6
Rationale (The reason)	System needs to monitor external influences (external climate)
Supervisory control part	Energy manager
Action	External sensors active

9.

ID	ENO1.1.34
Rationale (The reason)	To make sure the local surplus is used locally
Supervisory control part	Energy manager
Action	By communication between the local house controls, a program makes sure that the local surplus is used local

10.

ID	ENO1.1.37
Rationale (The reason)	To store energy in the heat pump
Supervisory control part	Energy manager + Load manager
Action	When there is surplus in the grid or financial advantage for the residents, the house control should tell the heat pump to raise the temperature in the water storage

11.

ID	ENO1.1.4
Rationale (The reason)	Energy Manager has to control the heating of the house
Supervisory control part	Load manager
Action	Wireless temperature transmitters sending room temperature to the house control

12.

ID	ENO1.1.5
Rationale (The reason)	Energy Manager has to control the set point of heating in the house
Supervisory control part	Energy manager + Load manager
Action	Connection to the heating system with information about raising or lowering the temperature

The 'Must' requirement can be merged into three descriptive requirements.

- Requirements 1, 3, 5, 11 and 12: ***Sawing energy.***
    - The energy consumption shall be improved. This improvement shall be done by controlling the appliance, the light and the heating of the houses. Improvement means lowering the consumption with atlases 5% without compromising the living comfort.
- Requirements 2, 9, 10, 11 and 12: ***Local use of local production.***
    - SCM shall minimize the mismatch between the locally produced energy and the locally used energy e.g. by controlling the heating of the houses through controlling heat pumps and/or setting the set points for room temperature in the individual houses. The locally production and consumption shall be monitored and logged in real-time.
- Requirements 4, 6, 7 and 8: ***Data handling and Security.***
    - SCL shall handle data and event from the middleware in a secure manner. This means that all data transmission must be carried out over a secure connection so that no information can be attained by unauthorized parties. SCM shall, through the middleware, send the necessary signals to the EB&BI module. SCM shall send message to a user through a user interface.

The 14 'Should' requirements are

1.

ID	ENEL1.1.6
Rationale (The reason)	Involve users in efficiency targets
Supervisory control part	Energy manager + Load manager + Local generation control
Action	The Encourage system should improve the energy efficiency of the building

2.

ID	ENO1.1.15
Rationale (The reason)	To turn off lights using motion sensors and/or sound detectors to make sure the consumers do not use unnecessary energy
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Shut down the light in the room.

3.

ID	ENO1.1.24 + EZM 1.1.15
Rationale (The reason)	The system should be capable of sending event / rules driven emails, Text message. Private twitter feeds or IM should also be considered. E.g. Triggered when consumption is over budget
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Notification Sent

4.

ID	EZM.1.1.3 + EZM.1.1.41
Rationale (The reason)	Well recognized industry standards and protocols should be used throughout
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Identified what defines types and common protocol

5.

ID	ENO1.1.32
Rationale (The reason)	To give consumers ideas how to save energy
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Post ideas with how to save energy regarding the consumption

6.

ID	EZM.1.1.34
Rationale (The reason)	The system shall monitor real-time multiple generation (co-generation, wind, solar, grid) in order to create predictive forecast
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Real-time data recorded

7.

ID	EZM.1.1.45
Rationale (The reason)	The system should allow for certain devices to define their own constraints / critical set points
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Apply a rule relating to critical devices

8.

	ENEL.1.1.36
Rationale (The reason)	To control customers' inertial loads to achieve peak shifting.
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Transform high level decision into device orders (e.g. switch on/off devices)

9.

ID	ENO1.1.1
Rationale (The reason)	To raise the temperature in the house in case of surplus production or financial advantage. Otherwise and in case that solar cells are not producing electricity, the temperature can be lowered
Supervisory control part	Energy manager + Load manager + Local generation control
Action	If surplus or financial advantage the house control should raise the temperature in the house

10

ID	ENO1.1.40
Rationale (The reason)	To use surplus energy to recharge an UPS system
Supervisory control part	Energy manager + Load manager + Local generation control
Action	To recharge an UPS system

11.

ID	ENO1.1.7
Rationale (The reason)	To use the washer and dryer when the electricity is produced
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Wireless relay on wire to the washer

12.

ID	ENO1.1.8
Rationale (The reason)	To lower the temperature of fridge and freezer when surplus production
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Wireless regulator to control the temperature in the fridge and freezer

13.

ID	ENO1.1.9
Rationale (The reason)	To raise the temperature of the hot water tank when electricity is produced
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Wireless regulator and wireless communication to existing regulator to the hot water tank to raise the water temperature

14.

ID	ENO1.1.41
Rationale (The reason)	To save energy when the residence is empty or residents are asleep
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Lover the temperature in the house when the house is in stand by for a longer period like during nights, holidays etc.

The 'Should' requirement can be merged into three descriptive requirements.

- Requirements 1, 3, 4 and 5: ***Communication with the user.***
- Requirements 2, 7, 8, 9, 10, 11, 12, 13 and 14: ***Controlling.***
- Requirements 6: ***Forecasting.***

The 4 'Could' requirements are

1.

ID	ENO1.1.36
Rationale (The reason)	To find out if the households are using more or less energy than other similar households
Supervisory control part	Energy manager + Load manager + Local generation control
Action	"Make a questionnaire that the residents can fill out, with following questions: • Size of the residence

	• Number of inhabitants • Construction year • Heating system

2.

ID	ENO1.1.11
Rationale (The reason)	Decisions must be performed by a rule-based engine, e.g.: To Avoid bacterial growth, the temperature in the hot water tank has to be raised to over 80°C once a week
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Execute control law: Wireless regulator and wireless communication to existing regulator to the hot water tank to raise the water temperature

3.

ID	ENO1.1.28
Rationale (The reason)	To give prosumers opportunity to buy external supervision of solar panels and/or heat pumps
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Sending status data from heat pump and solar panels to service provider

4.

ID	ENEL.1.1.40
Rationale (The reason)	Consumers can be represented also by associations and municipalities.
Supervisory control part	Energy manager + Load manager + Local generation control
Action	Encourage architecture must provide different level of aggregation (hierarchy) in order to allow associations of users or municipalities to assume the role of energy manager of groups of buildings and/or users

Appendix D. BNF for the ELL language

Tokens

```
<DEFAULT> SKIP : {
" "
| "\r"
| "\t"
| "\n"
}
```

```
<DEFAULT> SPECIAL : {
<SINGLE_LINE_COMMENT: "#" (~["\n","\r"])* ("\n" | "\r" | "\r\n")>
}
```

```
<DEFAULT> TOKEN : {
<TRUE: "ON">
| <FALSE: "OFF">
| <TRUE2: "TRUE">
| <FALSE2: "FALSE">
| <ANALOG: "ANACONST">
    1     | <DIGITAL: "DIGICONST">
| <ANAIN: "ANAINPUT">
| <ANAOUT: "ANAOUTPUT">
| <DIGIIN: "DIGIINPUT">
| <DIGIOUT: "DIGIOUTPUT">
| <BLINK: "FLACH">
| <FBLINK: "QFLACH">
| <DELAY: "DELAY">
| <FF: "FF">
| <PID: "PID">
| <SQR: "SQR">
| <SEKVENS: "FSM">
| <DFREG: "DFREG">
| <BBREG: "BBCTRL">
| <RESET: "RESET">
| <HOLD: "HOLD">
```

```
| <TILSTAND: "STATE">
| <INIT: "INIT">
| <GAA: "GO">
| <TIL: "TO">
| <NAAR: "WHEN">
| <SLUT: "END">
| <START: "PROGRAM">
| <MIN: "TIME.MINUTE">
| <HOUR: "TIME.HOUR">
| <DAY: "TIME.DAY">
| <WEEK: "TIME.WEEK">
| <WEEKDAY: "TIME.WEEKDAY">
| <MDR: "TIME.MDR">
}



<DEFAULT> TOKEN : {
<LPAREN: "(">
| <RPAREN: ")">
| <SEMICOLON: ";">
| <COMMA: ",">
| <COLON: ":">
| <SA: "@">
}



<DEFAULT> TOKEN : {
<PLUS: "+">
| <MINUS: "-">
| <MULTIPLY: "*">
| <DIVIDE: "/">
| <OG: "AND">
| <ELLER: "OR">
| <IKKE: "NOT">
| <EQL: "<=">
| <EQG: ">=">
| <EQU: "=">
}



<DEFAULT> TOKEN : {
<REALVAL: <INTVAL> ("." <INTVAL>)? | "." <INTVAL>>
```

| <INTVAL: (<DIGIT>)+>
| <DEVICEID: "DIV:" <LETTER> (<LETTER> | <DIGIT> | "_" | "." | "/")*>
| <IDENTIFIER: <LETTER> (<LETTER> | <DIGIT> | "_" | ".")*>
| <AIDENTIFIER: "_" (<LETTER> | <DIGIT> | "_")*>
| <#LETTER: ["$","A"-"Z","a"-"z","\u00c0"-"\u00d6","\u00d8"-"\u00f6","\u00f8"-"\u00ff","\u0100"-"\u1fff","\u3040"-"\u318f","\u3300"-"\u337f","\u3400"-"\u3d2d","\u4e00"-"\u9fff","\uf900"-"\ufaff"]>
| <#DIGIT: ["0"-"9","\u0660"-"\u0669","\u06f0"-"\u06f9","\u0966"-"\u096f","\u09e6"-"\u09ef","\u0a66"-"\u0a6f","\u0ae6"-"\u0aef","\u0b66"-"\u0b6f","\u0be7"-"\u0bef","\u0c66"-"\u0c6f","\u0ce6"-"\u0cef","\u0d66"-"\u0d6f","\u0e50"-"\u0e59","\u0ed0"-"\u0ed9","\u1040"-"\u1049"]>
}

Non-terminals

program	::=	(erklaering)* <START> (statement)* <EOF>
erklaering	::=	aconstsektion
	|	dconstsektion
	|	aisektion
	|	aosektion
	|	disektion
	|	dosektion
statement	::=	simpelstatement
	|	sequence
aconstsektion	::=	<ANALOG> (aconstspes)*
dconstsektion	::=	<DIGITAL> (dconstspes)*
aisektion	::=	<ANAIN> (aikanalspes)*
aosektion	::=	<ANAOUT> (aokanalspes)*
disektion	::=	<DIGIIN> (dikanalspes)*
dosektion	::=	<DIGIOUT> (dokanalspes)*
aconstspes	::=	<AIDENTIFIER> <EQU> <REALVAL> <SEMICOLON>
dconstspes	::=	<IDENTIFIER> <EQU> <TRUE> <SEMICOLON>

	|	<IDENTIFIER> <SA> <FALSE> <SEMICOLON>
aikanalspes	::=	<AIDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
realvv	::=	<REALVAL>
	|	<MINUS> <REALVAL>
aokanalspes	::=	<AIDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
dikanalspes	::=	<IDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
dokanalspes	::=	<IDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
simpelstatement	::=	<IDENTIFIER> <EQU> logical <SEMICOLON>
	|	<AIDENTIFIER> <EQU> anaval <SEMICOLON>
sequence	::=	<SEKVENS> <COLON> sekstart sekvensctrl inittilstand (tilstand)* <SLUT>
sekstart	::=	<IDENTIFIER>
sekvensctrl	::=	(resetsek)? (holdsek)?
resetsek	::=	<RESET> <EQU> logical <SEMICOLON>
holdsek	::=	<HOLD> <EQU> logical <SEMICOLON>
inittilstand	::=	stateinitstart handlingsdel (gaatil)* <SLUT>
tilstand	::=	statestart handlingsdel (gaatil)* <SLUT>
stateinitstart	::=	<TILSTAND> <COLON> <INIT>
statestart	::=	<TILSTAND> <COLON> <IDENTIFIER>
handlingsdel	::=	(simpelstatement)*
gaatil	::=	<GAA> <TIL> <IDENTIFIER> <NAAR> logical <SEMICOLON>
	|	<GAA> <TIL> <INIT> <NAAR> logical <SEMICOLON>
logical	::=	lterm ((<ELLER>) lterm)*
lterm	::=	lexp ((<OG>) lexp)*
lexp	::=	<IKKE> lelement
	|	lelement
lelement	::=	<IDENTIFIER>

	|	<TRUE>
	|	<FALSE>
	|	<BLINK>
	|	<FBLINK>
	|	lfunction
	|	<LPAREN> logical <RPAREN>
	|	anacomp
anacomp	::=	[2] anaval (">" anaval | "<" anaval | <EQG> anaval | <EQL> anaval | <EQU> anaval)
lfunction	::=	<DELAY> <LPAREN> logical <COMMA> anaval <RPAREN>
	|	<FF> <LPAREN> logical <COMMA> logical <RPAREN>
	|	<BBREG> <LPAREN> anaval <COMMA> anaval <RPAREN>
anaval	::=	term ((<PLUS> | <MINUS>) term)*
term	::=	exp ((<MULTIPLY> | <DIVIDE>) exp)*
exp	::=	<MINUS> element
	|	element
element	::=	<AIDENTIFIER>
	|	<REALVAL>
	|	<LPAREN> anaval <RPAREN>
	|	function
	|	<MIN>
	|	<HOUR>
	|	<DAY>
	|	<WEEK>
	|	<WEEKDAY>
	|	<MDR>
function	::=	<SQR> <LPAREN> anaval <RPAREN>

| <DFREG> <LPAREN> anaval <COMMA> anaval <RPAREN>

| <PID> <LPAREN> anaval <COMMA> anaval <COMMA> anaval <COMMA> anaval <COMMA> anaval <COMMA> logical <COMMA> anaval <RPAREN>

Appendix E. ELL Manual

ENCOURAGE

Embedded Intelligent Controls for Buildings with Renewable Generation and Storage

Grant Agreement No.: 269354

WP5: ELL (ENCOURAGE Logical Language).

Manual

ENCOURAGE

Deliverable D5.3 (Prototype)

Table of Contents

1. Introduction

This manual describes the IDE tool and the control language special develop for ENCOURAGE. This language is called ELL (Encourage Logical Language).

2. The IDE tool

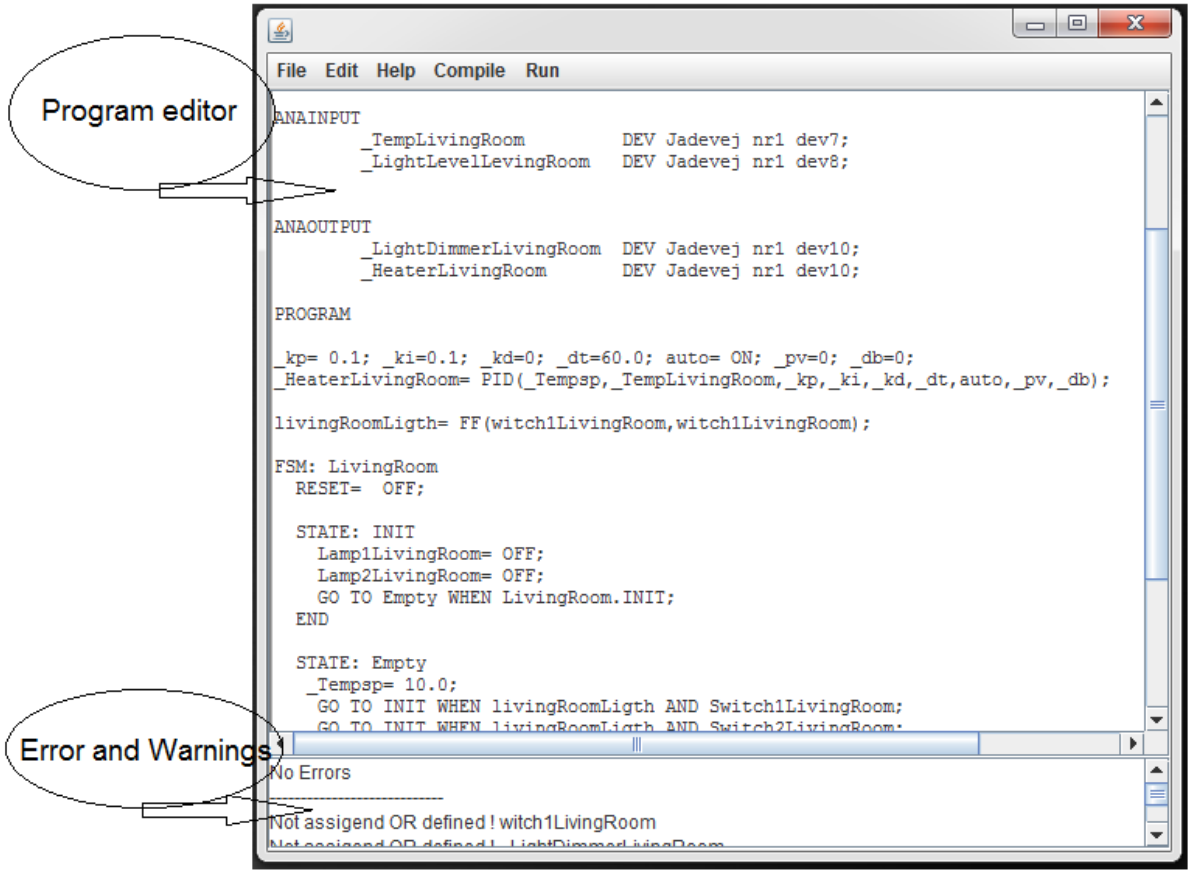


Figure 7: The IDE tool

The IDE tool is a dedicated development tool for the ELL language. This tool includes the necessary and only the necessary features for programming, compiling, executing and debugging the ELL code. Figure 1 shows the tool. The tool is started by clicking on the **ellIDE.jar** file or by using the Java interpreter directly in a command prompt, like this:

```
..> java –jar ellIDE.jar
```

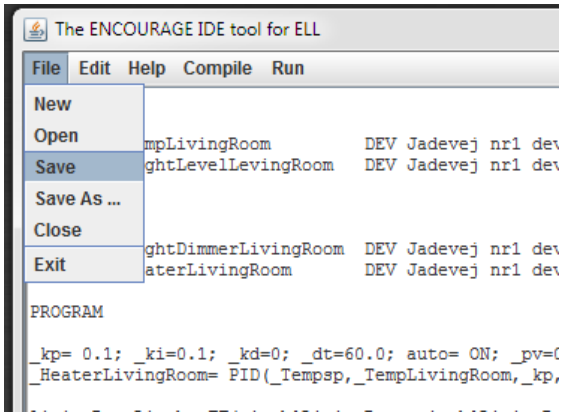
The window consists of three fields:

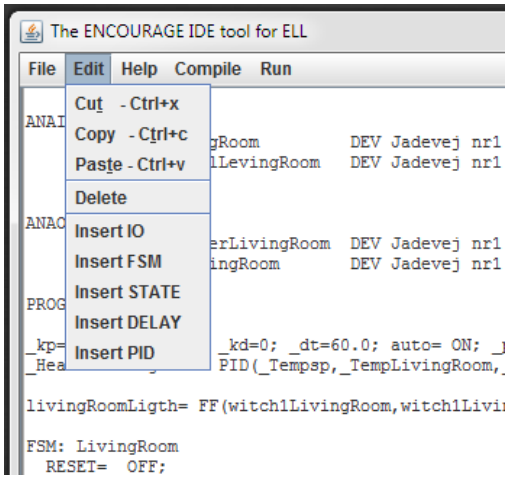
1. A menu field
2. An editor frame

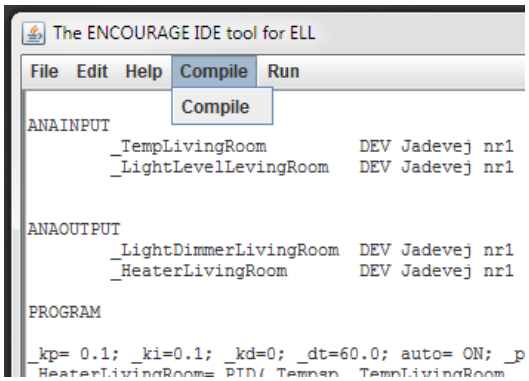
3. An Error and Warning frame

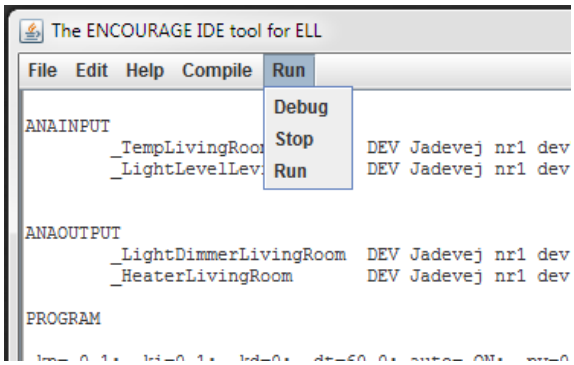
The menu field contains a:

- **File menu**: for handling files e.g. Open, save, save as, new, close and exit where exit close down the IDE tool itself.
- **Edit menu** with cut, copy, paste and delete. Beside of that it also contain a number of shorthand's for programming, like inserting IO sections, a FSM's, PID controllers and so on.
- **Help menu** with only one point: Help. (Not implemented in the prototype).
- **Compile menu** with only one point: Compile. If clicking on compile then the program in the edit frame will be compiled to interpretable code, if no error is found. The code can be executed on the core platform by a dedicated interpreter.
- **Run menu**: containing a Debug, stop and Run menu point. Debug will start the interpreter and display all variables in real-time in a separate window. Stop will stop the execution and Run will start the execution without the debugging window.









The normal key board combinations: Ctrl-x, Ctrl-c and Ctrl-v can be used for cut, copy and paste.

Figure 2 shows have the Error are handled by the compiler. The ELL language is case sensitive. Here the word STATE is spelt wrongly; namely STaTE. It shall be mention here, that the prototype compiler can't find all errors. As long as the ELL program can be compile to something that can be executed, then the compiler won't necessarily find the errors. Especially doublet definitions will not be catch by this compiler prototype.

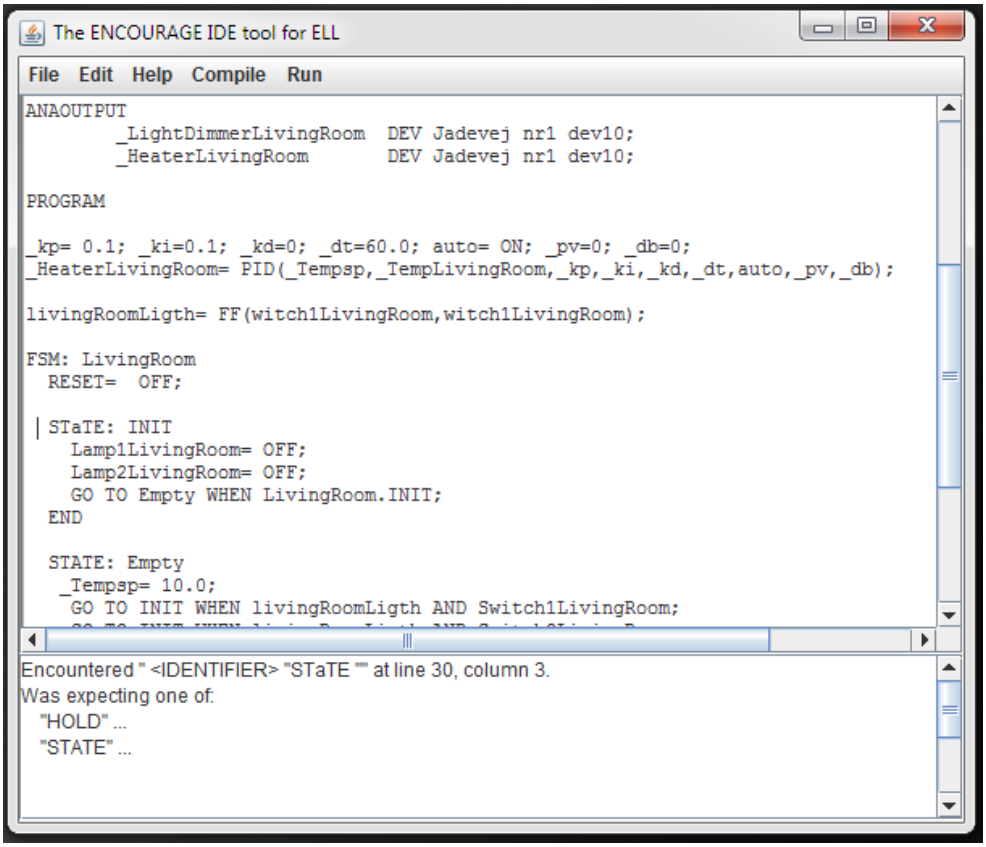


Figure 8: The compiler has found an Error (STaTE should be STATE)

Figure 3 shows an example without any errors. Here there are two warnings of the type: Not assigned OR defined and Not used. These warnings a very useful when finding runtime errors in the program. For instance the warning in this example:

```
Not assigned OR defined ! _LightDimmerLivingRoom
```

This warning tells the programmer that the variable `_LightDimmerLivingRoom`, which is an analog output, has not been assigned. This again tells the programmer that the program might be uncompleted.

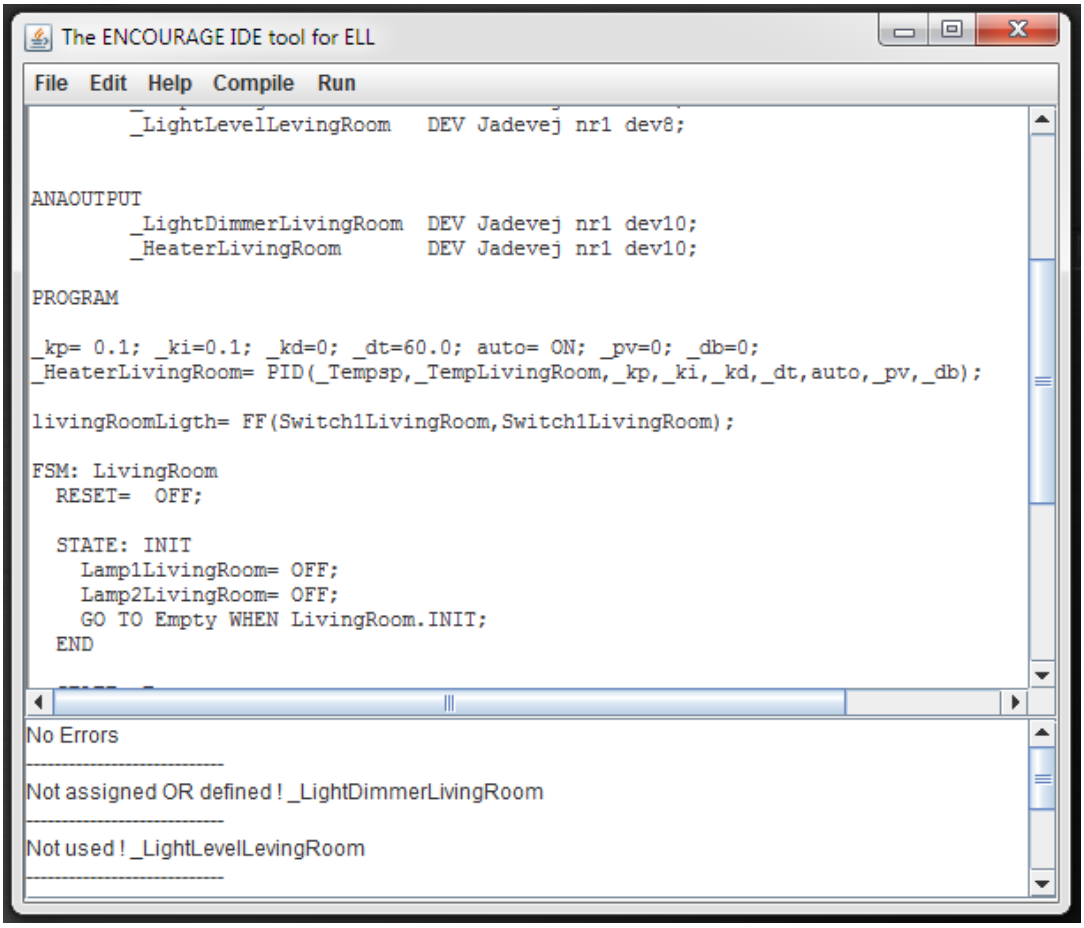


Figure 9: A program with no errors, but there is a number warnings.

The output from the IDE is the interpretable code. This code is put in a file in the same directory as the IDE jar file. If the program is new and hasn't got a name i.e. it hasn't been saved under a specific name, then the code is named: **fo.ell.bin.** Otherwise the name will be program name extended by **.bin .** This interpretable code file contains instructions for a virtual stack machine called the ELL interpreter

3. The ELL interpreter

A standalone version of the ELL interpreter is named **EllRun.jar.** This file is a runnable jar file which means it can be executes from a command prompt like this:

```
..> java –jar EllRun.jar 'the interpretable code name'
```

The code file from the IDE tool, shall be placed in the same directory as the interpreter **EllRun.jar.**

The interpreter implements a virtual stack machine. This stack machine operates on two different stacks one for the logical statements, a Boolean stack, and another for the analog statements, a floating point value stack.

The instructions for the logical stack are:

LOD #add : Load memory address #add on to the boolean stack
OUT #add : Move the top of the stack to mem. add. #add.
OR : OR the two top values on the stack and replace them with the result.
AND : Same as OR except using an AND instead of an OR.
NOT : Invert the top of the stack.

And for analog stack:

ALOD #add : Load memory address #add on to the analog stack.
AOUT #add : Move the top of the analog stack to #add in mem.

SUM : Add the two top values of the stack and replace them with the result.
SUB, MUL, DIV : Same as SUM except using subtraction, multiplication and division.
MINUS : Negate the top value on the analog stack.
ACONST #Value : Load a constant on top of the analog stack.

For some instructions both stack are in play. These are:

GRA : If Aanstak[top-1] > ANAstach[top] then put true on top of the Boolean stack.
EGRA : If Aanstak[top-1] >= ANAstach[top] then put true on top of the Boolean stack.
LES, ELES : Same as GRA, EGRA except using < and <=.

Where ANAstack[top] is the top element of the analog stack.

Beside this there are a number of instructions for special functions. These are DELAY for implementing timers, PID for PID-controllers and FF for Flip-Flop functions, BBCTRL for implementing bang-bang controllers.

The following example shows how the analog and the Boolean stacks are used in combination to perform a logical control statement. The control action is: Switch on a fan motor when the humidity is too high and the button is pushed. Switch off the fan when the humidity has gone down 10 % or the button is pushed again:
In logical terms it could be written like this, which is exactly the same as in the ELL language:

```
FanM= FF(_Humidity > _MaxHumidity AND StartVent,
       _Humidity < _MaxHumidity-10 OR StartVent);
```

Where: **FanM** is a ventilation fan motor.
FF is filpflop with a syntax like: **FF(Set,Reset)**.
_Humidity is the measurement in percent (%).
_MaxHumidity is analog value E.g. 80 %
StartVent is a logical start/stop value. E.g. input from a push button.

This small example will result in the following sequence of instructions to the interpreter:

```
ALOD 247
ALOD 253
GRA
LOD 241
ANDN
ALOD 247
ALOD 253
ACONST 10.0
SUB
LES
LOD 241
ORN
FF 0
OUT 240
```

This means load add. 247 and 253 on top of the analog stack. Perform a greater than operation on these two values, throw them away and place the result on the Boolean stack. Load add. 241 on top of the Boolean stack. Make an AND on the two values on the Boolean stack. Then again load add. 247 and 253 on top of the analog stack and now load the value 10 on top of the analog stack, make a subtraction and a less than operation, and thereby remove all three values from the analog stack and place the result on the Boolean stack. These instructions calculate this part **_Humidity < _MaxHumidity-10.** The next part is the OR operation. After this instruction, ORN, the result of the

first logical expression **_Humidity > _MaxHumidity AND Ventilate** is next to the top of the stack and the result of the last logical expression **_Humidity < _MaxHumidity-10 OR Ventilate** is on the top of the stack. The FF instruction uses these two values for set and reset and then they are deleted from the stack. The result of the FF instruction is placed on the stack. And lastly the top of the Boolean stack is written out to add. 240 and deleted from the stack.

Besides this dual stack machine the interpreter also have special instructions for implementing finite state machines (FSM).

These instructions are:

SEK : Mark the start of FSM code block.
ESK : Mark the end of FSM code block.
RES : If the top value of the Boolean stack is TRUE then the FMS transfers to the init state.
HOLD : If the top value of the Boolean stack is TRUE then the FMS stays in its current state.

STAI #add : Mark the start of the init state. If the mem. Cell given by #add is true then the instructions in the state are executed.
STA #add : Mark the start of a normal state. #add is used in the same way as for STAI.
GO #add : Sets the state value given by #add to the top value of the Boolean stack.

Here is an ELL-language example that illustrates how a FSM are handled by the interpreter. ON and OFF is used for simplicity, of course these values can be exchanged by general logical expressions. Here there are two states, the INIT state and the st1 state. The FSM always starts in the INIT state. Each state can contain as many statements as needed. Here each state only contains one statement, the GO TO statement.

```
FSM: SM1
   RESET= OFF;
   HOLD= OFF;
   STATE: INIT
      GO TO st1 WHEN ON;
   END
   STATE: st1
      GO TO INIT WHEN ON;
   END
END
```

The corresponding interpreter instructions look like this:

```
SEK
LODF
RES
LODF
HOLD
STAI 0
LODT
GO 2
STA 2
LODT
GO 0
ESK
```

The first five instructions indicate where the FSM starts and tells the interpreter if the FSM is in reset mode or in hold mode.

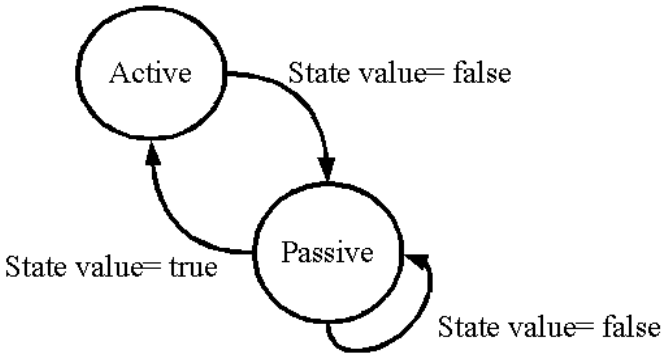


Figure 4: The two states of the interpreter.

The next six instructions are the implementation of the two states. The addresses after the STAI and STA instructions (0 and 2) are the address of the two state values. Figure 4 shows what happens when entering a state in this part of the code. If the state value is true it always switches to the active state and if the state value is false it stays in, or switches to, the passive state. When in the active state the interpreter execute the instructions as normal. If the interpreter is in the passive state it only reads the instructions and examine the state values. When reaching the end of the FSM, indicated by ESK, the interpreter switches to the active state. The reset and hold mode of the interpreter govern the manipulation of the state values. If the interpreter is in reset mode, the state values related to STAI instructions are set to true and all other state values, inside the specific FSM, are set to false. This has the highest priority. If the interpreter is in hold mode, the possibility for GO to set a state value is disabled.

4. The ELL language

An ELL program consists of two parts separated by the word **PROGRAM**:

3. The Description of the interface to the middleware and predefined constants.
4. The control algorithm, or the program part, consisting of a number of FSM and statements.

```
# Part one: The interface to the middleware and predefined constants

PROGRAM

# Part two: The control algorithm consisting of a number of FSM and statements
```

4.1. Interface to the middleware

The middleware interface consists of four sections, one section for the four different types of I/O values i.e.:

- ON/OFF input signals, also called digital inputs '**DIGIINPUT**'
- ON/OFF output signals, also called digital inputs '**DIGIOUTPUT**'
- Analog input signals, also called digital inputs '**ANAINPUT**'
- Analog output signals, also called digital inputs '**ANAOUTPUT**'

Each of these signals is defined by a symbolic name **LightSwitch1_Beedroom** and the MacroCellID, the CellID and the DeviceID e.g.:

MacroCellID: **Jadevej**

CellID: **nr3**

DeviceID: **device7**

```
DIGIINPUT
  LightSwitch1_Beedroom          DEV : Jadevej nr3 device7;
  DoorBell                       DEV : Jadevej nr3 device10;
  Window1Open_Beedroom           DEV : Jadevej nr3 device2;

DIGIOUTPUT
  lamp1_Levingroom               DEV : Jadevej nr3 device13;

ANAINPUT
  _RoomTemperature_Kitchen       DEV : Jadevej nr7 device7;

ANAOUTPUT
  _LightDimmer_Beedroom          DEV : Jadevej nr7 device2;
```

N.B: All analog values shall start with an **_.**

4.2. Predefined constants

The predefined constants, gives the programmer the opportunity to use symbolic names for specific constants and thereby make it easier to maintain the program.

```
ANACONST
  _LightLevel          = 80;
  _KP                  = 3.2;
  _KI                  = 0.67;
  _KD                  = 0;


DIGICONST
  true                 = ON;
  false                = OFF;
```

4.3. The control algorithm

After the interface part comes the control algorithm part. These two parte are separated by the word **PROGRAM**.

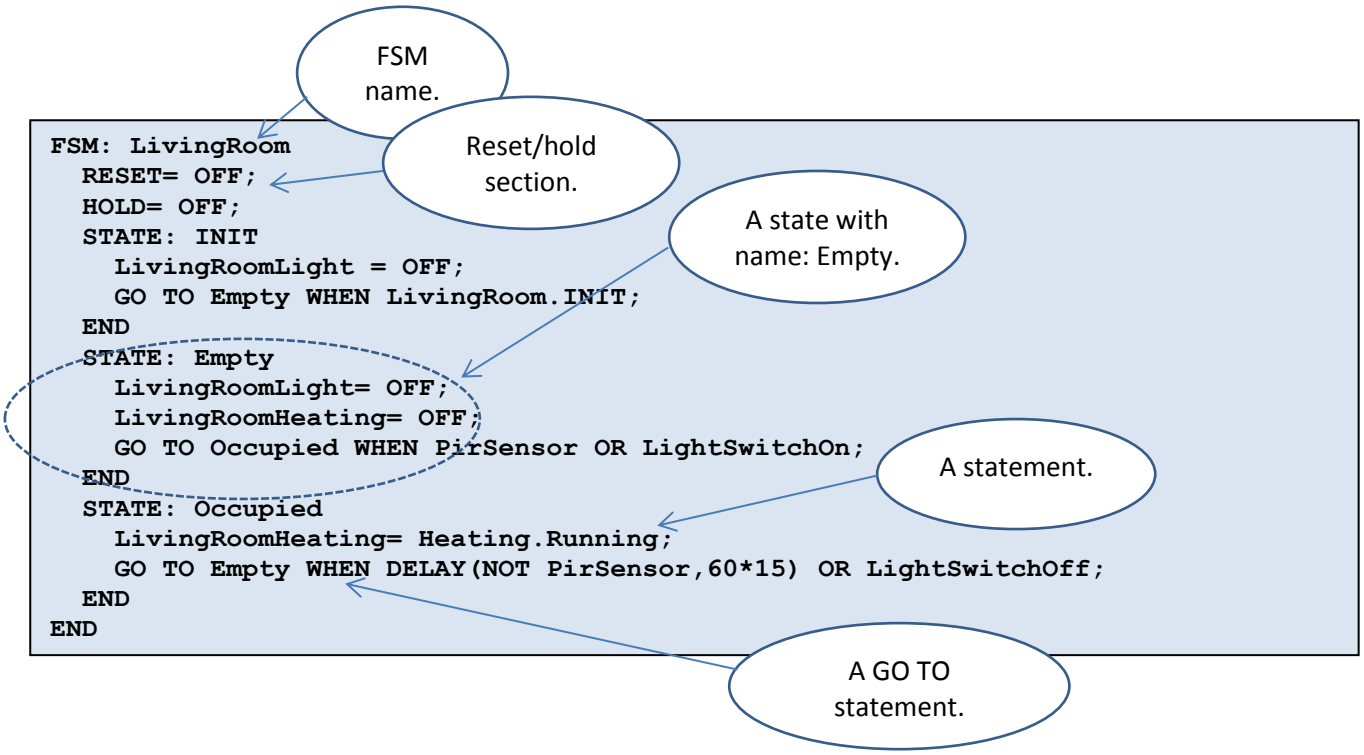
The control algorithm part consists of zero to N FSM and on zero to N statements, where N only is limited by the amount of memory in the runtime system (The ENCOURAGE core platform).

4.3.1. Finite state machine (FSM)

A FSM starts with the word FSM and after that the name of the FSM.

Then there can be a reset/hold section which controls the execution of the FSM.

The remaining part of the FSM consists of a number of states. The first state shall be named INIT. This state is the start state. Here, the FSM, will start when the execution are started. The rest of the stated can be named arbitrarily by the programmer.



If the FSM is in the state **Empty** then the statements inside the state are active, or in other word the statements and GO TO statements are executed in parallel and are running until the state gets passive. This means that **PirSensor** OR **LightSwitchOn** are check and if true(ON) then the state machine will switch to state **Occupied**.

You can have as many statements and GO TO statements inside a state as you like.

4.3.2. Statements

All actions are defined by statement. Statement can by inside state or outside the FSM's. If placed inside a state they are execute when the FSM are in that particular state other vice not. If the statements are placed outside the FSM's then they are executed all the time and in parallel.

- There are three kinds of statements:
- Logical assignment: **Light= lightSwitch AND NOT dayLight**;
- Analog assignment: **_temperatureError= _RoomTempSetpoint - _Roomtemp;**

- GoTo: **GO TO Empty WHEN DELAY(NOT PirSensor,900) OR LightSwitchOff;**

The last part of the logical statement e.g. **lightSwitch AND NOT daylight**; are the logical expression

Expressions can be ether Logical or analog, depending on the type of the assignment. E.g. for assignment of digital output logical expressions are used, and for assignment for analog output analog expressions are used.

4.3.2.1. Logical expressions

A logical expression consists of digital signals, analog signals, logical operators, comparison operators, functional operators and brackets.

Logical operators:

- **AND**
- **OR**
- **NOT**

These three operators are combined like this:**(logical expression) AND (logical expression).** where the result are a **(logical expression).**

This means that it is possible to make arbitrary sequence of digital symbols, logical operators and brackets, as long the brackets fits together like traditional logical algebra.

Comparison operators can be used for compering analog expressions. The result is a logical expression.

Comparison operators:

- **>**
- **<**
- **>=**
- **<=**
- **=**

These five operators can be used like this:

E.g. **(analog expression) > (analog expression).** Where the result are a **(logical expression).**

Functional operators:

- **DELAY**
- **FF**
- **PID**
- **BBCTRL**
- **TIME.MINUTE**

- **TIME.HOUR**
- **TIME.DAY**
- **TIME.WEEK**
- **TIME.WEEKDAY**
- **TIME.MONTH**
- **SQR**
- **FLACH**
- **QFLACH**

4.3.2.2. Analog expressions

An analog expression consist of analog signals, functional operators, floating point operations and brackets.

Functional operators:

- **PID**
- **TIME.MINUTE**
- **TIME.HOUR**
- **TIME.DAY**
- **TIME.WEEK**
- **TIME.WEEKDAY**
- **TIME.MONTH**
- **SQR**

Floating point operations:

- **+**
- **-**
- *****
- **/**

These four operators are combined like this:**(analog expression) + (analog expression).** Where the result are a **(analog expression).**

This means that it is possible to make arbitrary sequence of analog symbols, analog operators and brackets, as long the brackets fits together like traditional algebra.

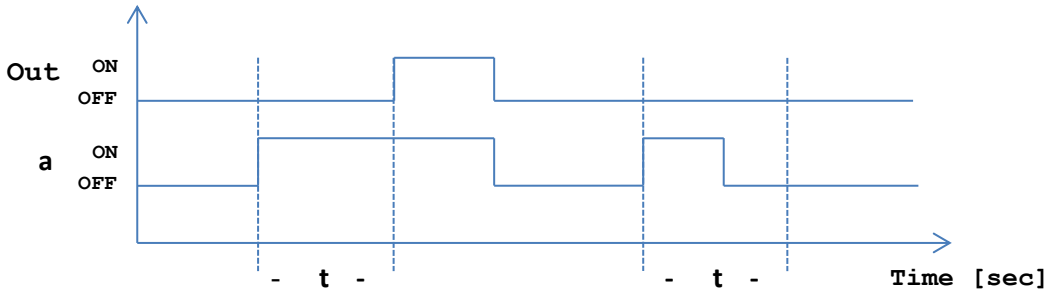
4.3.3. Description of the functional operators

4.3.3.1. Timer function (DELAY)

(logical value)= DELAY((logical expression), (analog expression));

This function is used whenever a timer is needed. It takes two arguments a logical expression and an analog expression. When the logical expression goes to ON the output value will go to ON after n seconds, where n is given by the analog expression.

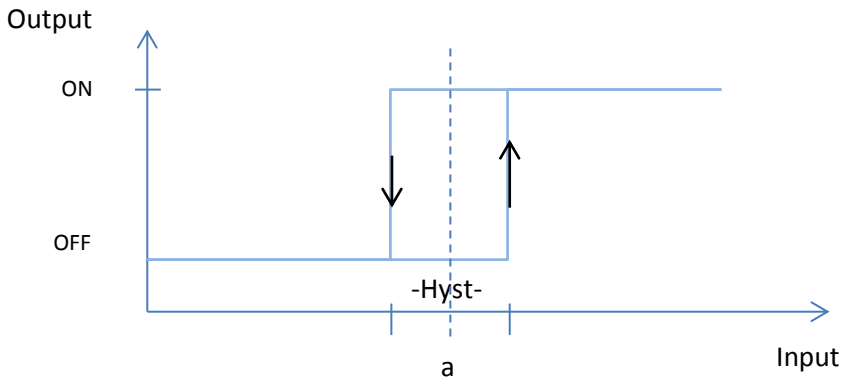
```
Out= DELAY(a,t);
```



4.3.3.2. Bang-Bang controller (BBCTRL)

Bang–bang controller (**BBCTRL**) also called on–off controller, are based on a hysteresis element that switches between two states e.g. ON and OFF. This state is the output from the controller. This controller can not only be used as a feedback controller but also when a level signal is needed. It takes two arguments. These two inputs are both analog expressions.

```
Output= BBCTRL(Input – a,Hyst);
```



4.3.3.3. Flip-flop (FF)

The Set-Reset flip-flop with toggle function is a so-called JK latch. This type of flip-flop is commonly used in control for handling ON/OFF states based on push-button. It takes two arguments. These are two logical expressions.

JK latch truth table			
Set	Reset	Out	Comment
0	0	Out	No change
0	1	0	Reset
1	0	1	Set
1	1	Not Out	Toggle

```
out=FF(set,reset);
```

4.3.3.4. PID-controller (PID)

The most common feedback controller is the PID-controller. In the absence of knowledge of the underlying process, a PID controller has historically been considered to be the best controller. (Bennett, 1993).

This equation shows how the output is calculated.

$$Out = K_p e(t) + K_i D_t \sum_{i=0}^{t} e(i) + K_d \frac{e(t)-e(t-1)}{D_t}$$

Where:

K_p: Proportional gain, a tuning parameter

K_i: Integral gain, a tuning parameter

K_d: Derivative gain, a tuning parameter

e: Error. The difference between the set point and the measured value.

D_t: The Time between each sample.

Beside of this equation a PID controller contains a number of features that are necessary in real applications. These are:

- Anti-integral windup on saturation.
- Bump-less transferee between auto and manual.
- Bump-less tuning.
- Deadband handling.

```
_Out= PID(_sp, _mv, _kp, _ki, _kd, _st, Auto, _Preset, _db);
```

Where:

_sp:	Are the set point
_mv:	Are the measured value
_kp:	Proportional gain
_ki:	Integral gain
_kd:	Derivative gain
Auto:	ON/OFF signal that switch between auto and manual
_Preset:	If manual then _out= _Preset
_db:	If the absolute value of the error is less then **_db** then **_out** will not change

4.3.3.5. Square root (SQR).

Square root is the only mathematic function that is available in the ELL language.

_Out= SQR (_input);

4.3.3.6. Flash function (FLASH).

Flash generate an alternating value with a frequency of 0.5 Hz. The ON time is 0.8 sec. and the OFF time is 1.2 sec. This function is mostly used for alert flash.

AllertLamp= FLASH AND alert;

Here the alert lamp will flash if the alert signal is ON.

4.3.3.7. Quick Flash function (QFLASH).

Quick Flash generates an alternating value with a frequency of 1 Hz. The ON time is 0.4 sec. and the OFF time is 0.6 sec. This function is fast alternative to **FLASH**.

AlarmLamp= QFLASH AND alarm;

4.3.3.8. Time function (MINUTE, HOUR, DAY, WEEK, WEEKDAY, MONTH)

There are a number of real-time functions in ELL. These functions are used for handling time dependent control. For instance; if the light has to switch ON between 6 o'clock and 10o'clock each Monday then the ELL code would look like this:

Light= (6<=TIME.HOUR) AND (TIME.HOUR<=10) AND (TIME.WEEKDAY = Monday);

Where Monday are a constant set to 2. Sunday is the first day of the week.

The available timing functions are:

- **TIME.MINUTE :** are the current minute [0-59].
- **TIME.HOUR :** are the current hour [0-23].
- **TIME.DAY :** are the current minute [1-31].
- **TIME.WEEKDAY :** are the current weekday [1-7].
- **TIME.WEEK :** are the current week [1-52].
- **TIME.MONTH :** are the current month [0-11].

5. Program example

A simple ELL program that controls the light in a living room is shown below. The light is dimmed when the outside light is above a specific level. All the light is switched off when the room is empty.

```
#-----------------------------------------------------
DIGIINPUT
  LivingRoomLightButtonOn        DEV : Jadevej nr3 device7;
  LivingRoomLightButtonOff       DEV : Jadevej nr3 device3;
  LivingRoomMotionDetect         DEV : Jadevej nr3 device4;

DIGIOUTPUT
  LivingRoomAllLight             DEV : Jadevej nr3 device2;

ANAINPUT
  _OutdoorLight                  DEV : Jadevej nr3 device6;

ANAOUTPUT
  _LivingRoomLightDimmer         DEV : Jadevej nr3 device5;

ANACONST
  _LightLevel          = 80;
  _LightP  = 3.2;

PROGRAM

FSM: LivingRoomLightBimmer
  STATE: INIT
    _LivingRoomLightDimmer = 100;
    GO TO dimme WHEN NOT BBCTRL(_OutdoorLight - _LightLevel, 5.0)
                    AND LivingRoom.Occupied;
  END
  STATE: dimme
    _LivingRoomLightDimmer= (100 - _OutdoorLight)* _LightP;
    GO TO INIT WHEN BBCTRL(_OutdoorLight - _LightLevel, 5.0);
    GO TO INIT WHEN LivingRoom.Empty;
  END
END


FSM: LivingRoom
  STATE: INIT
    GO TO Empty WHEN ON;
  END
  STATE: Empty
    LivingRoomAllLight = OFF;
    GO TO Occupied WHEN LivingRoomLightButtonOn;
```

```
  END
  STATE: Occupied
    LivingRoomAllLight = ON;
    GO TO Empty WHEN DELAY(LivingRoomMotionDetect,60*15)
                     OR LivingRoomLightButtonOff;
  END
END
```

Appendix A. BNF for the ELL language

Tokens

```
<DEFAULT> SKIP : {
" "
| "\r"
| "\t"
| "\n"
}
```

```
<DEFAULT> SPECIAL : {
<SINGLE_LINE_COMMENT: "#" (~["\n","\r"])* ("\n" | "\r" | "\r\n")>
}
```

```
<DEFAULT> TOKEN : {
<TRUE: "ON">
| <FALSE: "OFF">
| <TRUE2: "TRUE">
| <FALSE2: "FALSE">
| <ANALOG: "ANACONST">
    3     | <DIGITAL: "DIGICONST">
| <ANAIN: "ANAINPUT">
| <ANAOUT: "ANAOUTPUT">
| <DIGIIN: "DIGIINPUT">
| <DIGIOUT: "DIGIOUTPUT">
| <BLINK: "FLASH">
| <FBLINK: "QFLASH">
| <DELAY: "DELAY">
| <FF: "FF">
| <PID: "PID">
| <SQR: "SQR">
| <SEKVENS: "FSM">
| <DFREG: "DFREG">
| <BBREG: "BBCTRL">
| <RESET: "RESET">
```

```
| <HOLD: "HOLD">
| <TILSTAND: "STATE">
| <INIT: "INIT">
| <GAA: "GO">
| <TIL: "TO">
| <NAAR: "WHEN">
| <SLUT: "END">
| <START: "PROGRAM">
| <MIN: "TIME.MINUTE">
| <HOUR: "TIME.HOUR">
| <DAY: "TIME.DAY">
| <WEEK: "TIME.WEEK">
| <WEEKDAY: "TIME.WEEKDAY">
| <MDR: "TIME.MONTH">
}


<DEFAULT> TOKEN : {
<LPAREN: "(">
| <RPAREN: ")">
| <SEMICOLON: ";">
| <COMMA: ",">
| <COLON: ":">
| <SA: "@">
}


<DEFAULT> TOKEN : {
<PLUS: "+">
| <MINUS: "-">
| <MULTIPLY: "*">
| <DIVIDE: "/">
| <OG: "AND">
| <ELLER: "OR">
| <IKKE: "NOT">
| <EQL: "<=">
| <EQG: ">=">
| <EQU: "=">
}
```

```
<DEFAULT> TOKEN : {
<REALVAL: <INTVAL> ("." <INTVAL>)? | "." <INTVAL>>
| <INTVAL: (<DIGIT>)+>
| <DEVICEID: "DIV:" <LETTER> (<LETTER> | <DIGIT> | "_" | "." | "/")*>
| <IDENTIFIER: <LETTER> (<LETTER> | <DIGIT> | "_" | ".")*>
| <AIDENTIFIER: "_" (<LETTER> | <DIGIT> | "_")*>
| <#LETTER: ["$","A"-"Z","a"-"z","\u00c0"-"\u00d6","\u00d8"-"\u00f6","\u00f8"-"\u00ff","\u0100"-
"\u1fff","\u3040"-"\u318f","\u3300"-"\u337f","\u3400"-"\u3d2d","\u4e00"-"\u9fff","\uf900"-"\ufaff"]>
| <#DIGIT: ["0"-"9","\u0660"-"\u0669","\u06f0"-"\u06f9","\u0966"-"\u096f","\u09e6"-"\u09ef","\u0a66"-
"\u0a6f","\u0ae6"-"\u0aef","\u0b66"-"\u0b6f","\u0be7"-"\u0bef","\u0c66"-"\u0c6f","\u0ce6"-
"\u0cef","\u0d66"-"\u0d6f","\u0e50"-"\u0e59","\u0ed0"-"\u0ed9","\u1040"-"\u1049"]>
}
```

Non-terminals

program	::=	(erklaering)* <START> (statement)* <EOF>
erklaering	::=	aconstsektion
	|	dconstsektion
	|	aisektion
	|	aosektion
	|	disektion
	|	dosektion
statement	::=	simpelstatement
	|	sequence
aconstsektion	::=	<ANALOG> (aconstspes)*
dconstsektion	::=	<DIGITAL> (dconstspes)*
aisektion	::=	<ANAIN> (aikanalspes)*
aosektion	::=	<ANAOUT> (aokanalspes)*
disektion	::=	<DIGIIN> (dikanalspes)*
dosektion	::=	<DIGIOUT> (dokanalspes)*

aconstspes	::=	<AIDENTIFIER> <EQU> <REALVAL> <SEMICOLON>
dconstspes	::=	<IDENTIFIER> <EQU> <TRUE> <SEMICOLON>
	|	<IDENTIFIER> <SA> <FALSE> <SEMICOLON>
aikanalspes	::=	<AIDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
realvv	::=	<REALVAL>
	|	<MINUS> <REALVAL>
aokanalspes	::=	<AIDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
dikanalspes	::=	<IDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
dokanalspes	::=	<IDENTIFIER> <SA> <DEVICEID> <SEMICOLON>
simpelstatement	::=	<IDENTIFIER> <EQU> logical <SEMICOLON>
	|	<AIDENTIFIER> <EQU> anaval <SEMICOLON>
sequence	::=	<SEKVENS> <COLON> sekstart sekvensctrl inittilstand (tilstand)* <SLUT>
sekstart	::=	<IDENTIFIER>
sekvensctrl	::=	(resetsek)? (holdsek)?
resetsek	::=	<RESET> <EQU> logical <SEMICOLON>
holdsek	::=	<HOLD> <EQU> logical <SEMICOLON>
inittilstand	::=	stateinitstart handlingsdel (gaatil)* <SLUT>
tilstand	::=	statestart handlingsdel (gaatil)* <SLUT>
stateinitstart	::=	<TILSTAND> <COLON> <INIT>
statestart	::=	<TILSTAND> <COLON> <IDENTIFIER>
handlingsdel	::=	(simpelstatement)*
gaatil	::=	<GAA> <TIL> <IDENTIFIER> <NAAR> logical <SEMICOLON>
	|	<GAA> <TIL> <INIT> <NAAR> logical <SEMICOLON>
logical	::=	lterm ((<ELLER>) lterm)*

lterm	::=	lexp ((<OG>) lexp)*
lexp	::=	<IKKE> lelement
	|	lelement
lelement	::=	<IDENTIFIER>
	|	<TRUE>
	|	<FALSE>
	|	<BLINK>
	|	<FBLINK>
	|	lfunction
	|	<LPAREN> logical <RPAREN>
	|	anacomp
anacomp	::=	[4] anaval (">" anaval | "<" anaval | <EQG> anaval | <EQL> anaval | <EQU> anaval)
lfunction	::=	<DELAY> <LPAREN> logical <COMMA> anaval <RPAREN>
	|	<FF> <LPAREN> logical <COMMA> logical <RPAREN>
	|	<BBREG> <LPAREN> anaval <COMMA> anaval <RPAREN>
anaval	::=	term ((<PLUS> | <MINUS>) term)*
term	::=	exp ((<MULTIPLY> | <DIVIDE>) exp)*
exp	::=	<MINUS> element
	|	element
element	::=	<AIDENTIFIER>
	|	<REALVAL>
	|	<LPAREN> anaval <RPAREN>
	|	function
	|	<MIN>
	|	<HOUR>

	| <DAY>
	| <WEEK>
	| <WEEKDAY>
	| <MDR>
function	::= <SQR> <LPAREN> anaval <RPAREN>
	| <DFREG> <LPAREN> anaval <COMMA> anaval <RPAREN>
	| <PID> <LPAREN> anaval <COMMA> anaval <COMMA> anaval <COMMA> anaval <COMMA> anaval <COMMA> logical <COMMA> anaval <RPAREN>

Appendix B. The ELL interpreter instruction set

LOD #add	: Load memory address #add on to the boolean stack
OUT #add	: Move the top of the stack to mem. add. #add.
OR	: OR the two top values on the stack and replace them with the result.
AND	: Same as OR except using an AND instead of an OR.
NOT	: Invert the top of the stack.
ALOD #add	: Load memory address #add on to the analog stack.
AOUT #add	: Move the top of the analog stack to #add in mem.
SUM	: Add the two top values of the stack and replace them with the result.
SUB, MUL, DIV	: Same as SUM except using subtraction, multiplication and division.
MINUS	: Negate the top value on the analog stack.
ACONST #Value	: Load a constant on top of the analog stack.
GRA	: If Aanstak[top-1] > ANAstach[top] then put true on top of the Boolean stack.
EGRA	: If Aanstak[top-1] >= ANAstach[top] then put true on top of the Boolean stack.
LES, ELES	: Same as GRA, EGRA except using < and <=.
SEK	: Mark the start of FSM code block.
ESK	: Mark the end of FSM code block.
RES	: If the top value of the Boolean stack is TRUE then the FMS transfers to the init state.
HOLD	: If the top value of the Boolean stack is TRUE then the FMS stays in its current state.
STAI #add	: Mark the start of the init state. If the mem. Cell given by #add is true then the instructions in the state are executed.
STA #add	: Mark the start of a normal state. #add is used in the same way as for STAI.
GO #add	: Sets the state value given by #add to the top value of the Boolean stack.
BLK	: Sets the value at the top of the boolean stack an alternating value. Frequency: 0.5 Hz.
FBLK	: Sets the value at the top of the boolean stack an alternating value. Frequency: 1 Hz.
BB #BBnr	: Run the Bang-Bang controller giver by #BBnr and Sets the value at the top of the boolean stack to the output.
FF #FFnr	: Run the Flip-Flop giver by #FFnr and Sets the value at the top of the boolean stack to the output.
PID #PIDnr	: Run the PID controller giver by #PIDnr and Sets the value at the top of the analog stack to the output. To calculate the output it uses 7 elements from the analog stack and one from the Boolean stack. i.e. Astack[top - 6], Astack[top - 5], Astack[top - 4], Astack[top - 3], Astack[top - 2], Bstack[top], Astack[top - 1], Astack[top] where: Astack[top - 6] are the set point, Astack[top - 5] are the measurement, Astack[top - 4] are P, Astack[top - 3]are I, Astack[top - 2] are D, Bstack[top]are the auto/manual signal, Astack[top - 1]are the preset value, Astack[top] are the deadband.

MIN : Put the real-time minute value on to the analog stack.
HOUR : Put the real-time hour value on to the analog stack.
DAY : Put the real-time day value on to the analog stack.
WEEK : Put the real-time week value on to the analog stack.
WEEKDAY : Put the real-time weekday value on to the analog stack.
MDR : Put the real-time month value on to the analog stack.