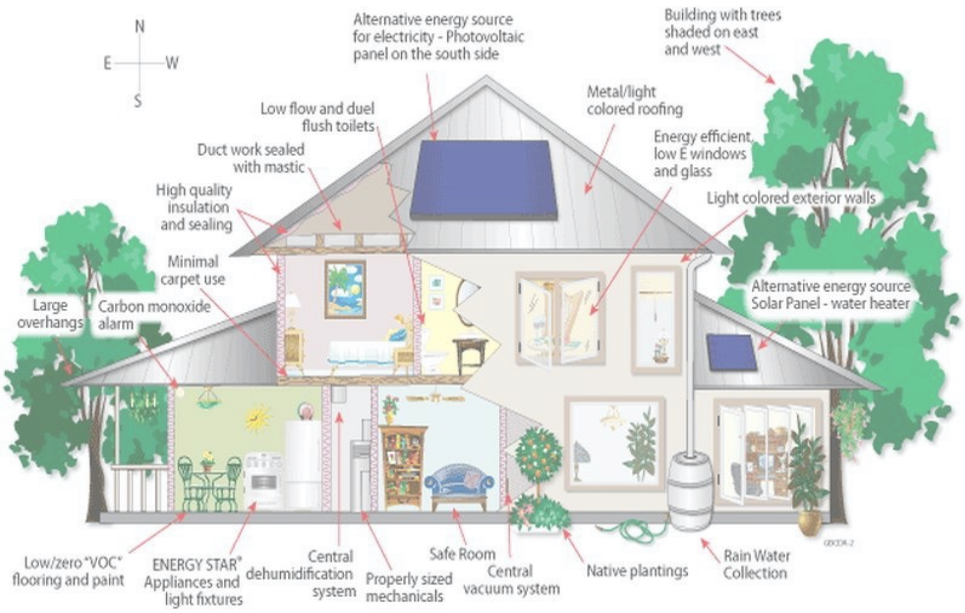
FORMALIZATION OF A SMART HOUSE WITH TRIO

Green building



FILIPPO PELLOLIO

ANDREA ROTTIGNI

1.1 Overall System Description

The system to be model is a smart house. A smart house is composed by a pull of devices, which are wired connected to the electrical system and communicate with the EMS (energy manager system) through a wireless network called HAN (home area network). The devices can be divided into two main categories:

- Smart Devices: exchange information through the HAN with the EMS
- Legacy Devices: can only be controlled by smart plugs, which provide remote metering and remote control for legacy devices.

Smart devices can be divided into two categories:

- Periodically permanent
- Programmed

Smart devices execute task scheduled by an external event. There two types of task:

- May: smart operation that can be delayed and does not have tight time requirements
- Must: smart operation that cannot be shifted or interrupted after its start

Each activity has three different functionalities to allow energy management:

- Load Shifting
- Load Balancing
- Load Shedding

In our case, when a task is ready to start a message is sent through the HAN to the EMS which decides whether the task can start or not and, if it is necessary, it promotes shifting or balancing or shedding. In case of a must task when a request is sent to the EMS, the EMS must start the activity, and can't delay it or promotes balancing, shifting or shedding.

The energy manager takes care of measuring the power consumption and in case of overload decides which tasks can be shifted, shed or balanced. In case of overload a warning message is sent to the task that

is executed not in the normal condition. If the EMS is not able to resolve the overload within a certain amount of time, there is a blackout. A manual restore is necessary after a blackout, and all the tasks that were executing before this event are resumed. The EMS also measures the power produced by the “green” power suppliers and stores it in the batteries if it is not used.

The, so called, HEM (home electricity meter) provides the current amount of energy that is bought from the network. It has the switch to restore from a blackout.

The green power suppliers produce energy with an unpredictable behavior. In our proposed system there are three power suppliers:

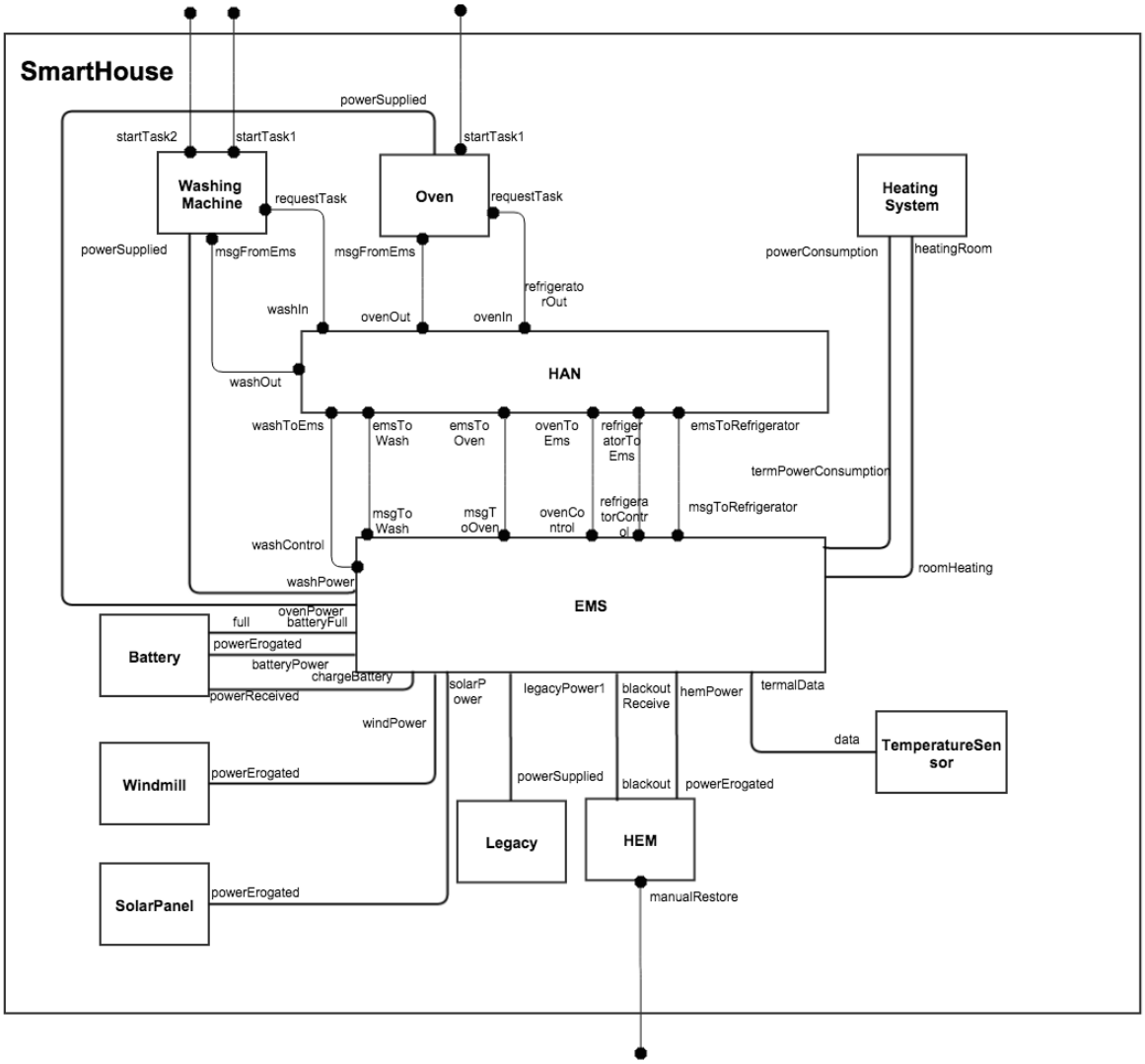
- Windmill
- Solar panel
- Batteries

Our system guarantees the following properties:

- liveness of all the devices which are requiring power supply and their start can not be infinitely postponed
- a device which is raising a request for power supply must be served
- in case of overload it is possible to disconnect the device that allows load shifting and shedding in order to avoid a blackout
- a device requiring to execute a task will eventually terminates

1.2 FORMAL MODEL OF THE SMART HOUSE SYSTEM

Figure 1.1 shows the TRIO class diagram of the proposed system derived from the description of the previous section. We decided to simplify the model and eliminate all the human interfaces, replacing them with events coming from the environment.



As already said, a smart house is composed by a pull of devices. In our proposed model we have modeled a class Device, which contains all the states, events and formulae that are common for all the smart devices. Then, we modeled a washing machine, an oven, a refrigerator and the heating system. These are basically the smart devices described above,

even if the heating system is slightly different with respect to the other ones, as we will see in the following sections. We modeled these 4 devices because they allow us showing all the possible combination of may, must task and periodically permanent, programmed devices.

The class legacy device basically defines the functionality of a smart plug. We decide to introduce only a legacy device because they are all identical.

We introduced only a temperature sensor because, in our proposed smart house, other kind of sensor would be useless.

After this brief description of the assumptions that we have made, we can now describe a possible flow of actions in our system:

- **A task is scheduled:** when an event startTask(t) is received, after the specified delay, the device sends a message to EMS. The HAN introduces only some delay, assuming that no packet is lost. After that the EMS has received the message containing all the information about the task and the device it checks if can execute it without causing an overload. If this is the case then a start message is sent to the device, which can start its task. The EMS keeps track all the tasks in execution. In case of overload there are several possibilities. If the requested task is a “MUST” task, a starting message is sent to the device. Then if there are some “MAY” task in execution the EMS promotes balancing, shedding or shifting to avoid a blackout. If this is not possible a blackout will occur. If it is a “MAY” one, the EMS can delay it or promote balancing and shedding.
- **The temperature is under/above a certain threshold:** the temperature sensor always measures the temperature in each room and it is wired connected to the EMS. When the EMS receives a temperature above or under a certain threshold it starts heating or cooling the room. In a sense the heating/cooling activity is “MAY” task because the amount of power provide to do such an activity can be balanced.
- **Blackout:** if the EMS measure for a certain interval that the amount of power absorbed from the network is above the maximum threshold it enters in a blackout state: all power consumption and production are set to 0. After a manual restore event (in the HEM) all the executing tasks are resumed.

- **Legacy device task:** when a legacy device start a task the EMS can only monitor the amount of power used by such a device. The EMS can't control the device.
- **Windmill and Solar Panel power production:** they are wired connected to the EMS and they communicate the amount of power that they are producing. This power can be used immediately, if it is necessary, or can be stored in the batteries. We have assumed that if the power suppliers produce enough power to satisfy the devices needs no power is bought from the network. We made the assumption that, if the batteries are full, the exceeding power is lost.
- **Batteries charging and discharging:** the EMS, based on the current consumption, decides whether charging the batteries or draining power from them. The EMS can only charge the batteries if they aren't full and can only drain power if they aren't empty.

1.2.1 The Washing Machine and Oven Modules

These two modules are very similar; they inherit from the device class. They both receive input from the environment with the startTask event. The major difference between the two modules is that the washing machine can schedule both may and must task, while the oven can schedule only must task, because it seems a reasonable assumption that the oven doesn't stop a task during the execution.

When a task is scheduled with the startTask(t) event, where t is the delay, after the delay a request to execute the task is sent to the EMS. A request is sent only if the device is not executing another task. If this is the case, the request to the EMS is sent after the previous task has been completed.

This is the code of the classes:

```
CLASS WASHINGMACHINE( CONST MAXPOWER,CONST MAXSLOTS,CONST
MAXDELAY)

INHERIT DEVICE:

SIGNATURE:

VISIBLE:

STARTTASK1,STARTTASK2;

AXIOMS:

/*(DEFINES THE SPECS OF TASK1 AND TASK2)*/

END
```

```
CLASS OVEN(CONST TASKPOWER, CONST TASKTIME)

INHERIT DEVICE:

SIGNATURE:

VISIBLE: STARTTASK1;

ITEMS:

EVENT STARTTASK1(NATURAL);

AXIOMS:

/* AXIOM THAT DEFINES TASK1 */

END
```

1.2.2 The Home Area Network

The Home Area Network (HAN) is the component in charge of the communication between the smart devices and the EMS.

Its implementation is quite simple, it simply repeats the requests received from the devices to the EMS, introducing a small delay due to the communication protocol.

Follows the signature of the class, the axioms are not reported due to their triviality:

```
class HAN(const MAX_DELAY):

signature:

visible:
washIn,washOut,ovenIn,ovenOut,refrigeratorIn,refrigeratorOut

emsToWash,emsToOven,washToEms,ovenToEms;

domains:

Bool:0..1;

TaskType: {May,Must};

Resp: {Go,Warn};

event
washIn(integer,TaskType,natural,real,natural,natural,Bool,Bool
);

event
ovenIn(integer,TaskType,natural,real,natural,natural,Bool,Bool
);

event washOut(integer,Resp);

event ovenOut(integer,Resp);
```

```
EVENT EMSTOWASH(INTEGER,RESP);

EVENT EMSTOOVEN(INTEGER,RESP);

EVENT
WASHTOEMS(INTEGER,TASKTYPE,NATURAL,REAL,NATURAL,NATURAL,BOOL,B
OOL);

EVENT
OVENTOEMS(INTEGER,TASKTYPE,NATURAL,REAL,NATURAL,NATURAL,BOOL,B
OOL);

AXIOMS:...(SIMPLY ROUTE THE MESSAGES)

END
```

1.2.3 DEVICE

The class device defines the behavior of any smart devices, both oven and Washing machine inherit from it.

The class receives inputs from the EMS and send messages to the HAN directed to the EMS in this way:

```
VISIBLE:

INPUTS

MSGFROMEMS,POWERSUPPLIED,

OUTPUTS

REQUESTTASK;
```

Where `POWERSUPPLIED` is a `STATE` that represents the amount of power currently received from the electrical system.

Now let’s talk about the axioms that regulate the device class.

The first one, TaskIdDef, defines the id of the tasks performed by the device, it is different for every new task and it is incremental (not necessarily needed but useful for aesthetic purposes).

NotRequestWhileWorking: It states that, if a task is requested, the previous task active on that device must have been completed.

NoConsecutiveRequests: If a request has been sent and is waiting for a response from the EMS no other tasks can send requests.

MustInsurance: Ensures that if a must request has been sent the task will be executed. The task will start after MAX_DELAY from the request, that is, after the network delay. Since a must task can't be shed, shift or balanced, the power is supplied to the device executing the task for exactly t time units, i.e., the time necessary to complete the task. The only exception is in case of a blackout; in that case a warning message is sent to the device.

MustEndIfNotSheddable: If a task is not sheddable then it must be completed sooner or later.

ResponseGeneratedByRequest: Ensures that responses from the EMS are generated only after a task request with the same id has been sent.

ResponseStartSupply: ensures that, if a response from the EMS has arrived and it’s positive then the device is powered.

StartAfterBlackout: Models the behavior of a device when the power is restored after a blackout.

PowerIfExecuting: Ensures that the device is only powered if it is performing a task.

WarningMessage: Ensures that the power is toggled from the device after a Warning message from the EMS

PERFORMINGONLYIFREQUEST: Defines the state PERFORMINGTASK as the task currently in execution.

TIMETOLIVEDEFINITION: It defines the time to live of a task that is contained in the state PERFORMINGTASK(I,T)

REMAININGENERGYDEFINITION: it defines how much more energy is needed to complete the current task.

SHIFTDEFINITION: Defines the behavior of a task that is shiftable.

NONBALANCEABLEDEF: If a task is not balanceable then the power supplied to it must always be either 0 or the power needed by it; this axioms ensures this property.

```
CLASS DEVICE( CONST MAXPOWER,CONST MAXSLOTS,CONST MAXDELAY):
SIGNATURE:
VISIBLE:
MSGFROMEMS,POWERSUPPLIED,
REQUESTTASK;
DOMAINS:
SLOTSAMOUNT: 1..MAXSLOTS;
POWERAMOUNT: 1..MAXPOWER;
DELAYAMOUNT: 1..MAXDELAY;
BOOL:0..1;
TASKTYPE: {MAY,MUST};
RESP: {START,WARN};
ITEMS:
EVENT MSGFROMEMS(INTEGER,RESP);
STATE POWERSUPPLIED(POWERAMOUNT);
```

```
EVENT
requestTask(integer,TaskType,PowerAmount,natural,SlotsAmount,D
elayType,Bool,Bool);

STATE performingTask(integer,real);

STATE remainingEnergy(integer,real);

STATE stopCounter(integer,natural);

AXIOMS:

/*axioms of the class*/

END
```

1.2.4 Windmill and Solar Panel

The classes regarding this generators are almost empty, the only output is the power generated, there are no axioms since their behavior is almost random and we can't predict the wind or the sunlight with trio.

```
CLASS Windmill:

SIGNATURE:

VISIBLE: powerErogated;

ITEMS:

STATE powerErogated(real);

END
```

```
CLASS SOLAR:

SIGNATURE:

VISIBLE: POWEREROGATED;

ITEMS:

STATE POWEREROGATED(REAL);

END
```

1.2.5 TEMPERATURE SENSORS

They are almost the same as the power generators, we can't describe their output. They simply perceive the temperature in the room and blindly send it to the EMS through the `STATE DATA`.

```
CLASS SENSOR:

SIGNATURE:

VISIBLE: DATA;

ITEMS:

STATE DATA(NATURAL,INTEGER);

END
```

1.2.6 SMART HOUSE

This is the parent class that contains all the others, it's work is just creating the various items and connecting them using the CONNECTIONS.

```
CLASS SMARTHOUSE():

IMPORT:

EMS,HAN,DEVICE,WASHINGMACHINE,OVEN,WINDMILL,TEMPERATURESENSO
R,HEATING,SOLAR,HEM,BATTERY;

MODULES:

WM : WASHINGMACHINE[MAXPOWER IS 1500,MAXSLOTS IS 5,MAXDELAY
IS 500];

OV : OVEN[MAXPOWER IS 2000];

LEG: LEGACY[MAXPOWER IS 2000];

HT: HEATING

BAT: BATTERY[CAPACITY IS 20000000]

EMS : EMS[MAXFROMHEN IS 20000];

HAN : HAN[MAX_DELAY IS 5];

HEM : HEM[MAX_POW IS 20000,MAX_OVF IS 1.3,MAX_TIME IS
300 ];

TS : TEMPERATURESENSOR

WINDMILL : WINDMILL;

SOLARPANEL : SOLAR;


CONNECTIONS:

//OVEN

(DIRECT OV.POWERSUPPLIED, EMS.OVENPOWER);
```

```
(direct OV.requestTask, HAN.ovenIn);
(direct OV.msgFromEMS, HAN.ovenOut);
//washing machine
(direct WM.powerSupplied, EMS.washPower);
(direct WM.requestTask, HAN.washIn);
(direct WM.msgFromEMS, HAN.washOut);
//heating
(direct HT.powerConsumption, EMS.termPowerConsumption);
(direct HT.heatingRoom, EMS.roomHeating);
//batteries
(direct BAT.full, EMS.batteryFull);
(direct BAT.powerErogated, EMS.batteryPower);
(direct BAT.powerReceived, EMS.chargeBattery);
//windmill
(direct Windmill.powerErogated, EMS.windPower);
//solar
(direct SolarPanel.powerErogated, EMS.solarPower);
//legacy
(direct LEG.powerSupplied, EMS.legacyPower1);
(direct LEG.powerRequired, EMS.legacyReq1);
//HEM
(direct HEM.blackout, EMS.blackoutReceived);
(direct HEM.powerErogated, EMS.hemPower);
//temperature sensor
(direct TS.data, EMS.termalData);
```

```
//HAN-EMS
(DIRECT EMS.WASHCONTROL, HAN.WASHTOEMS);
(DIRECT EMS.MSGTOWASH, HAN.EMSTOWASH);
(DIRECT EMS.OVENCONTROL, HAN.OVENTOEMS);
(DIRECT EMS.MSGTOOVEN, HAN.EMSTOOVEN);
END
```

1.2.7 LEGACY DEVICE

This class represents all the devices that have no internal hardware to deal with the EMS, they just absorb power, just like old devices.

The class has only one axiom: DEFINITIONOFPOWERCONSUMPTION , it defines the device behavior: either it is switched off and it is not absorbing any power, or it is working and it's absorbing all the power it needs.

```
CLASS LEGACYDEVICE (CONST MAXPOWER)
SIGNATURE:
VISIBLE: POWERREQUIRED, POWERSUPPLIED;
TEMPORAL DOMAIN: REAL
DOMAINS:
POWERAMOUNT: 1..MAXPOWER;
ITEMS:
STATE POWERREQUIRED(POWERAMOUNT);
STATE POWERSUPPLIED(POWERAMOUNT);
AXIOMS:
```

1.2.8 HEATING

Heating is a sort of special device that is switched on or off by the EMS.

Every time the EMS realizes that the temperature in one of the rooms is beneath the threshold, it switches on the heating in that room only.

The only axioms in the class states that the Heating in a room only absorbs power if the room is cold and needs to be heated.

```
CLASS HEATING:

SIGNATURE:

VISIBLE HEATINGROOM, POWERCONSUMPTION;

ITEMS:

STATE POWERCONSUMPTION(REAL);

STATE POWERPERROOM(NATURAL,REAL)

AXIOMS:

/* AXIOMS OF THE CLASS */

END
```

1.2.9 EMS

The EMS is probably the most important part of all the system. You can find below the skeleton of the class.

There are four events, which basically regulate the exchange of messages between the EMS and the smart devices through the HAN network. Two of these events, washControl and ovenControl, represent a message received from the washing machine or the oven. The arguments associated to washControl and ovenControl represents the data associated with each device request and these parameters are:

- The identifier of the task that the device would like to execute, which is of type integer

- The type of task, i.e., May or Must
- The normal amount of power necessary to execute the task
- The amount of time necessary to execute the task, assuming that p is constant.
- The maximum number of slot in which the task can be divided
- The maximum amount of time that an idle phase can last
- Information about the possibility to shed the task, which is of type Bool
- Information about the possibility to balance the task, which is of type Bool

The other two events, msgToWash and msgToOven, represent a message sent by the EMS to the devices. The arguments associated to these two are:

- The identifier of the task that previously a device asked to execute which is of type integer
- The response of the EMS, which can be Start or Warn. A Warn message is sent both when there is no sufficient power to start the task, and when there is a blackout.

Then, there are a lot of states that can be divided in some categories:

- State for the devices: legacyReq1, legacyPower1, ovenPower, washPower. The argument of all these states is the amount of power absorbed by the device at that instant.
- State for the power suppliers: solarPower, windmillPower. The argument of each state is the amount of power produced.
- States for the batteries: batteryPower, batteryFull, chargeBattery. The batteryFull state doesn't have any parameter. The other two states have as parameter the amount of power drained from the batteries (batteryPower) or the amount of power provided to the batteries (chargeBattery)
- States for the HEM: blackoutReceive, hemPower. The first one doesn't have any parameter and it represents the fact that a blackout is in progress. The other one has as argument the amount of power currently bought
- States for the heating system and the temperature sensor: roomHeating, termPowerConsumption, termThreshold, termalData. The id of the rooms is the argument of roomHeating and represents which rooms are currently heated. For what concerning termThreshold and termalData the arguments are the id of the room and the temperature.

- Not visible state: washState, ovenState, washStopCounter, max, consumption and overflow. The arguments of washState and ovenState are the id of the task that the device is executing, the amount of time remaining to execute the task, the amount of energy remaining to complete the task. Instead, washStopCounter, has as parameters the id of the task and the number of time that the task has been stopped so far. It is used to check if the task can be shifted anymore. The argument of max and consumption is, respectively, the maximum amount of consumable power and the current power consumption. Overflow doesn't have any parameter and represents that the current consumption is above the max.

```
CLASS EMS(CONST MAX_FROM_HEM):

SIGNATURE:

VISIBLE:

WASHCONTROL, MSGTOWASH, MSGTOOVEN, OVENCONTROL,
TERMPOWERCONSUMPTION, ROOMHEATING, TERMALDATA, HEMPOWER,
BLACKOUTRECEIVE, LEGACYPOWER1, SOLARPOWER, WINDMILLPOWER,
CHARGEBATTERY, BATTERYPOWER, BATTERYFULL, LEGACYREQ1;

DOMAINS:

RESP: {START,WARN};

ITEMS:

EVENT
WASHCONTROL(INTEGER,TASKTYPE,POWERAMOUNT,NATURAL,SLOTSAMOUNT,D
ELAYTYPE,BOOL,BOOL);

STATE WASHPOWER(REAL);

EVENT MSGTOWASH(INTEGER,RESP);

EVENT
OVENCONTROL(INTEGER,TASKTYPE,POWERAMOUNT,NATURAL,SLOTSAMOUNT,D
ELAYTYPE,BOOL,BOOL);

STATE OVENPOWER(REAL);
```

```
EVENT MSGTOOVEN(INTEGER,RESP);

STATE LEGACYPOWER1(REAL);

STATE LEGACYREQ1(NATURAL);

STATE WINDMILLPOWER(REAL);

STATE SOLARPOWER(REAL);

STATE OVERFLOW;

STATE HEMPOWER(REAL);

STATE CONSUMPTION(REAL);

STATE MAX(REAL);

STATE BLACKOUTRECEIVE;

STATE TERMALDATA(NATURAL,INTEGER);

STATE TERMTHRESHOLD(NATURAL,INTEGER);

STATE TERMPOWERCONSUMPTION(REAL);

STATE ROOMHEATING(NATURAL);

STATE BATTERYPOWER(REAL);

STATE BATTERYFULL;

STATE CHARGEBATTERY(REAL);

STATE WASHSTATE(INTEGER, REAL, REAL);

STATE OVENSTATE(INTEGER, REAL, REAL);

STATE WASHSTOPCOUNTER(INTEGER, NATURAL);

AXIOMS:

/** AXIOMS OF THE CLASS **/

END
```

ConsumptionDefinition: Defines the consumption, the production, the max amount of power that can be consumed,

PowerUnicity: Defines the uniqueness of several states such as washPower,ovenPower, windmillPower, solarPower, batteryPower, legacyPower1, hemPower, legacyReq1, washState, ovenState, washStopCounter

ResponsesFromEMS: Defines all the possibile responses of the ems according to the current conditions.

WashStartMsgConditions: Defines that if a starting message is sent to the washing machine than a certain amount of power greater than 0 is provided to the device. If the request is to execute a must task, or in the past there was a request to execute that must task, but was interrupted for a blackout, than the power provided to the device is exactly the required one.

OvenStartMsgConditions: Defines that if a starting message was sent to the oven than there is a request to execute that task or there was a request to execute that task, but was interrupted because of a blackout, than the power provided to the device is exactly the required one.

PowerIfRequested: Defines that power is provided to a device only it has previously sent a request

ResponseEnsurance: Defines that to every request corresponds a reponse.

PowerBalance: Defines the consumption at any instant is less than or equal to the sum of the power produced, bought and received from the batteries.

If the amount of power consumed is equal to the power produced by the windmill and the solar panels then the power received from the network and from the batteries is equal to 0.

If the power received from the network is greater than the maximum amount of power that can be bought, then we are in overflow.

If the power produced is less than the power produced and the batteries are not full, then the exceeding power is used to charge the batteries.

OverflowMeasuresShed: Defines that if there is an overflow, and a task that can be shed is in execution, than this task is interrupted (power set to 0) and a warning message is sent to the device.

OverflowMeasuresShift: Defines that if there is an overflow, and a task that can be shifted is in execution and it wasn't stop more than the number of allowed stop, than this task is interrupted (power set to 0) for at most the maximum amount of time specified in the initial request.

OverflowMeasuresBalance: Defines that, if an overflow is in progress, and it is in execution a task that can be balanced and exist a power that allows to not overcome the power threshold, then the task is executed using this power. In our system only task scheduled by the washing machine and the heating system can be balanced

WashStopCounterDefinition: Defines a counter that counts the number of time a task is stopped. Firstly it specifies that if a counter is counting the number of stop of certain task, than the device is executing that task (washState keeps track of the current task in execution). Secondly, it specifies that the counter start from zero. Thirdly it defines how the counter is incremented, that is: if a task is in execution and up to now a certain amount of power is provided to the device and there isn't a blackout in progress and now the amount of power provided to the device is 0, than the counter should be incremented by one unit. Lastly it defines that if the counter is true for a certain task and for a certain number of stops greater than zero, than the task must be a "May" task.

WashStateDefinition: Defines the state washState that keeps track of the current task in execution. The state washState has three arguments, the task id, the amount of time remaining for the completion of the task and the amount of energy necessary to complete the task. It specifies that energy and time will become 0 in the future except when the task allows load shedding. It then defines the monotonicity of the energy. Lastly it defines how the energy value and the time parameter change and how the energy and time parameters are correlated.

OvenStateDefinition: Defines the state ovenState that keeps track of the current task in execution. The state ovenState has three arguments, the task id, the amount of time remaining for the completion of the task and the amount of energy necessary to complete the task. It specifies that energy and time will become 0 in the future except when the task allows load shedding. It then defines the monotonicity of the energy. Lastly it defines how the energy value and the time parameter change and how the energy and time parameters are correlated.

OnBlackout: Defines that if a blackout is in progress the consumption is set to 0 and a warning message is sent to the devices that are connected through the HAN.

RestoreFromBlackout: Defines that if a blackout has just finished, and some tasks were executing before the blackout then a message to all the devices with pending task is sent, in order to resume the execution.

TermalAlarm: Defines that if there exist a sensor that perceives a temperature below a certain treshold in a room than we start to heat that room.

Some final considerations must be done for what concerning load balancing, shedding and shifting. In our model we didn't define a hierarchy among the devices and among the three policies. For example if there is a power overload and the washing machine is executing a task that allows balancing, shedding and shifting, there isn't any axiom that specifies what policy should be chosen. We decided to make this choice because we think that inserting a formula for all the possible combinations would have only added a lot of axioms without adding any relevant feature.

1.2.10 BATTERY

A Battery has the unique property of being both a generator and a device: it can both absorb and give power.

Here is the signature of the class:

```
CLASS BATTERY(CONST CAPACITY):

SIGNATURE:

VISIBLE:

FULL, POWEREROGATED, POWERRECEIVED;

ITEMS:

STATE FULL;

STATE EMPTY;

STATE POWEREROGATED(REAL);

STATE POWERRECEIVED(REAL);

STATE BATTERYLEVEL(REAL);

AXIOMS:

/* THE AXIOMS OF THE CLASS */

END
```

Full and empty States are Booleans and are useful since a full battery can't receive any power and an empty battery can't give any.

Battery has some axioms:

FULLDEFINITION, EMPTYDEFINITION : They trivially define the behavior of full and empty states.

CHARGINGDEFINITION, DRAININGDEFINITION: They define how the charge level change with respect to the power absorbed and given to the electrical system.

StateUnicityDefinition: It states that the states are true for only one value in every moment.

PositiveValue: Trivial axiom that states that the electrical power is always a positive value.

1.2.11 HEM

The home electricity meter is in charge of regulating the absorption of power from the network. It measures the amount of power that is currently bought from the network and, if this amount is greater than the maximum power that can be bought, then it causes a blackout. Actually the blackout is immediate when the amount of power is greater than the maximum multiplied with a certain threshold, which is a sort of security measure. Otherwise it generates a blackout when the maximum amount of power is overcome for more than the maximum amount of time. These properties are basically the axioms of the class.

```
CLASS HEM(CONST MAX_POW,CONST MAX_TIME,CONST MAX_OVF):

SIGNATURE:

VISIBLE POWEREROGATED, BLACKOUT, MANUALRESTORE;

ITEMS:

STATE POWEREROGATED(REAL);

STATE BLACKOUT;

EVENT MANUALRESTORE;

AXIOMS:

/* AXIOMS OF THE CLASS */

END
```