ALGEBRAIC SOFTWARE STRUCTURING

Jean-Pierre Le Pabic

PREAMBLE

It is now a common thing to come up against a software malfunction of a commonly used device, such as DECT telephones, GPS navigators, video recorders. Some cars loaded with electronics have even made the headlines from time to time because the driver no longer had control of his vehicle. Of course, armies of programmers always end up fixing bugs when the stakes are high, like in a car.

This poor quality of current software led me to bring out an old method, which does not benefit from any notoriety but which has proven itself in terms of quality, development times and scalability. It has been tested in complex systems with strong real-time constraints such as public or private telephone systems.

The origin of the method dates back to the time of the first computer-controlled public telephone exchanges. It took about 40 people over 2 years to develop such software with the resources available at the time. But each new order involved a certain number of modifications which themselves led to others and so on, which ultimately resulted in the complete rewriting of the software. This situation was obviously not tenable and it was decided to develop a method which avoids all these problems. It is this method that I present in the following. It was simply redesigned to apply to any type of software development and I called it "Algebraic Software Structuring".

INTRODUCTION

There is no easier way to understand the interest of the Algebraic Software Structuring than to take as an example the elementary arithmetic problems that are posed to children at school. Even for an adult, they are not always simple because the arithmetic imposes to apprehend the problem in a more or less global way. But as soon as we apply the techniques of algebra, these same problems are solved with extreme ease because algebra makes it possible to concentrate successively on each fundamental characteristic of the problem independently of the others. The same is true when applying Algebraic Software Structuring to software development.

Although multiple methods of analysis have emerged, they are still much more akin to arithmetic than algebra. We build an organization chart as general as possible which we then cut out to distribute the tasks. This flowchart is generally very complicated and the difficulties in mastering it intellectually are sources of errors sometimes discovered late during integration tests.

The Algebraic Structuring of Software consists on the contrary in identifying each elementary function which can be of course an execution program but also a sequence logic, a data management program or an interface program with hardware or software elements (sensors, actuators, displays, data transmission, etc.).

In this, it should be noted that the Algebraic Structuring of Software is closely related to the analysis of value. It is observed that during development, useless functionalities are implemented unconsciously, which increases the cost of the product. The value analysis generates a reflection that makes it possible to determine the functionality of each element, thus eliminating what is useless and ultimately reducing costs.

It will be noted in particular that the program linking logic is no longer disseminated in different programs but on the contrary entirely contained in one or more specific programs for which this is the raison d'être. In other words, the chaining logic is not scattered in the execution logic.

Once this step is completed, each elementary function thus identified is processed in a good quality program (short, one function, one input, one output except possibly for switching programs).

Programs handling data of a similar nature are then grouped ("algebra" means "meeting") into modules. Each module is known to the other modules only by its entry points and therefore constitutes a subset whose accesses are perfectly controlled.

Thus the responsibility for the development of the different modules can be entrusted to different engineers or even to different subcontractors. This responsibility typically covers the following phases: analysis, programming, unit testing and documentation.

The method produces a text-type document both directly usable by a programmer and understandable by people whose job is not IT. The overall structuring of the software is thus understandable by business managers and sales representatives, which means that any modification or addition of functionality can be sufficiently understood by them to obtain a quick estimate for a potential decision to make or a customer quote.

But in the same way as for the analysis of the value, the Algebraic Structuring of Software requires an accompaniment until the method is familiar, without which, one quickly notes a drift which makes lose all the advantages of it. Indeed, the method goes against the usual reflexes of analysis to the point that beginners do not understand its interest until the structure of the software is fairly well established.

BASIC PRINCIPLES

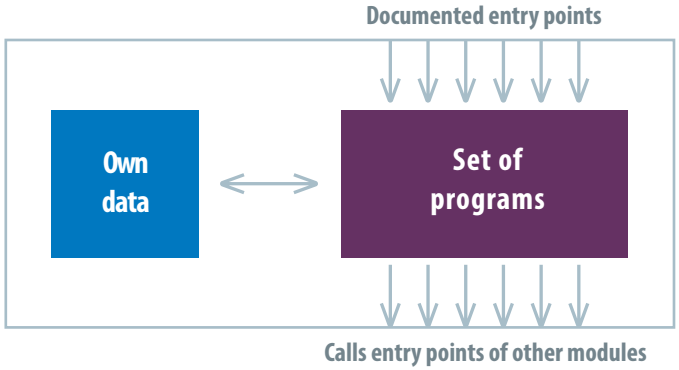
The fundamental rule is that the software is made up of watertight modules built around specific data of the same nature.

Only the module programs have access to this data.

The module is known to other modules only through its documented entry points.

The module itself can call the documented entry points of other modules.

As the Algebraic Structuring of Software is closely related to the analysis of the value, one will endeavor to determine beforehand the function of each program or set of programs to carry out the division in module. The pre-cut described below takes into account the different types of basic functionality found in any software.



THE DIFFERENT TYPES OF MODULES

There are four types of modules :

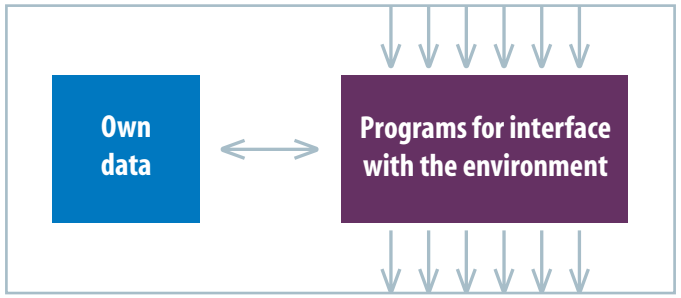
1. interface modules with the environment
2. operating data modules
3. execution modules
4. coordination modules

- 1 -
Interface modules with the environment

Interface modules have the role of isolating the core software from the outside world.

There is a **first level of decoupling** when the information is modified only in its form. In a car, it could be reading an exterior, interior, engine temperature, etc.), lighting a light (interior lighting, flashing, high beam, dipped beam, etc.). In these examples, it will simply be a question of translating a read value which does not necessarily have any intrinsic meaning into a value in the usual unit such as the degree centigrade.

A **second level of decoupling** will be used when the information must be processed to be usable, in one direction or the other. This case corresponds in particular to interfaces with display systems (HMI) or with standard communication programs. Another example is smart sensors in the agile factory.



- 2 -
Operating data modules

Operating data is that which is specific to each installed system, which has an influence on its operation and which is rarely or never modified..

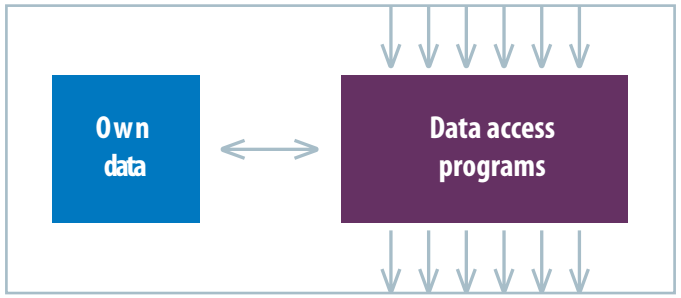
The following examples can be cited:

In a GPS navigator, these are in particular the home address, the addresses used regularly and stored in the memory, the choice of voice, etc.

In a DECT telephone, these are in particular the list of pre-recorded numbers, the choice and ringtone level.

In a business telephone system, this includes the list of internal numbers, whether 2, 3 or 4 digits, speed dialing, etc.

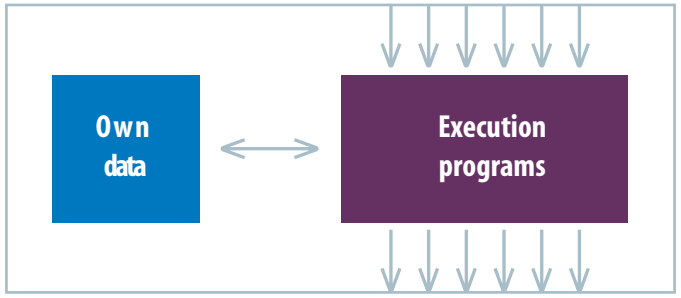
In a heating system, these are the minimum and maximum temperatures of the different zones when defined by human-machine interface.



- 3 -
Runtime modules

These modules contain the programs performing additional functions of the same nature. These are usually programs available on the market.

In business telephony, they are used to decode the numbers dialed from the digitized acoustic signal generated by the telephone set or to decode the caller's number from the modulation received on the external line.



- 4 -
Coordination modules

These modules contain all of the system's sequencing logic.

They launch the execution of programs according to the event, their own logic and possibly the data contained in the operating data modules.

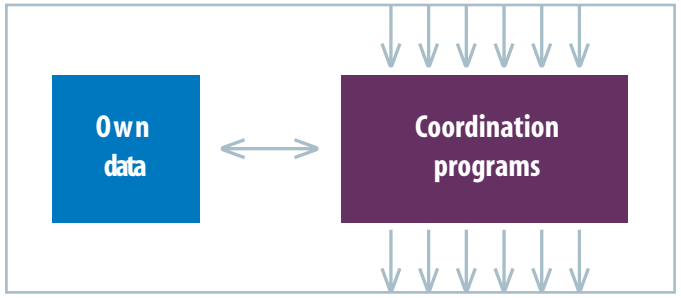
The event can be a physical event or the expiration of a timer, which will result in two types of coordination modules that frequently co-exist.

Indeed, a computer system must often manage time constraints. They can be classified into two categories.

In the first where the system must have completed a number of tasks within a specified time, the computer system is considered to be fast enough to deliver the response well in time. In general, therefore, there is not too much to worry about. Otherwise, it will be necessary to check that the execution time of the program sequence does not exceed the specified value.

> Take the example of telephone systems. In the days of dial telephones, the dialed number was sent in the form of successive line openings/closings, ten per second. It was then necessary to read the state of the line approximately every 10 ms, which was a very strong real-time constraint, especially since the computer systems of the time were much slower than today and that hundreds of subscribers could dial at any given time. Today, pulses of two frequencies with a duration of 40 ms have replaced line openings/closings and decoding can be done by hardware devices. There is therefore no longer any real real-time constraint. It is also considered with reason that the user can possibly wait a few hundred milliseconds for his call to be processed.

In the second category, the system must respond within a specified time, no more and no less. This period is not necessarily short. It can be milliseconds or tens of seconds and even hours or days. An additional constraint resides in the precision of the moment of processing. These constraints are solved by timing programs, located in a specific coordination module.



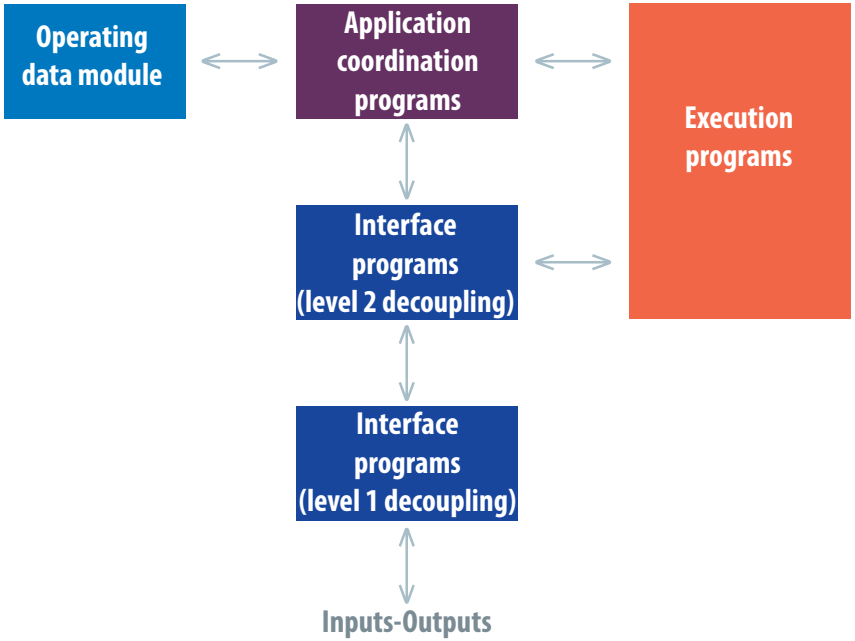
THE SUPER-FUNCTIONS

The set of programs relating to a functionality of the software constitutes a super-function. The most common are :

1. APPLICATION
2. OPERATION
3. MAINTENANCE
4. AUDITS
5. COUNTS/OBSERVATIONS

Diagram of a super-function

Each super-function includes at least coordination programs and interface programs.



- 1 -
The APPLICATION super-function

The super-function APPLICATION contains the minimum set of programs distributed in the different modules and which allow the operation of the software. **This is the only super-function that is essential**.

The following examples can be given :

In a telephone system, processing calls in such a way as to connect the caller and the called party.

In a heating system: manage the available power so that the requested temperature is effective.

In a GPS navigation system: determine the route to follow and changes in direction during the journey.

In an autonomous car: manage the different parameters to control the direction and the speed.

- 2 -
The OPERATION super-function

A computer system can be installed in several environments, at different customers. The specific requirements for each of these facilities may vary from one to another. And the data relating to each installation can be found, as mentioned above, in the operating data module.

The programs responsible for managing this data constitute the OPERATION super function.

It is not only a question of having read and write access to this data, but of possibly managing access rights, modification rights, etc.

- 3 -
The MAINTENANCE super-function

The proper functioning of a system is linked to the proper functioning of the physical equipment that composes it. For this reason and to obtain additional data to those normally used, sensors or even actuators are often added which are of no use in the context of standard operation but provide information on the correct operation of an element.

The launch of anomaly search programs originates in the specific coordination module. The anomaly search programs themselves reside in the environment interface modules and locally control the equipment to which they have access.

Significant results are transmitted to the MAINTENANCE super-function.

In a car, these sensors are plethora to the point of sometimes providing false information threatening engine failure: temperatures of different points of the engine, levels, etc.

- 4 -
The AUDITS super-function

Even if everything that can be physically controlled gives satisfactory results, it can happen that inconsistencies occur in the software, inconsistencies which are generally due to software defects. But they can also be due to an external influence in the case of connected systems.

The consequences can be serious. We will remember the destruction of the Iranian centrifuges a few years ago because of a software inconsistency generated by a program coming from outside.

A less serious consequence may simply be the "crash" of the system.

The launch of the inconsistency search programs, in the same way as the anomaly search programs, originates in the specific coordination module. The programs for finding inconsistencies themselves are located in the various modules and locally check the data therein.

The significant results are sent to the AUDITS super-function.

By way of example, one could cite the case where a heating system continues to heat when the required temperature is reached. This does not happen because the software of such a system is simple and not connected. In complex and connected systems, an equivalent situation is not excluded.

- 5 -
The COUNTS / OBSERVATIONS super-function

This super-function makes it possible to store data inherent to the operation of the software, in particular the number of events of each type, the cumulative duration of certain states, etc.

In a telephone system, these will be, among other things, the number of outgoing and incoming calls, the duration of occupation of each line, etc.

In a heating system, this will be the number of ignitions, the cumulative heating time of each of the circuits, etc.

IMPLEMENTATION

1. **FIRST STEP : Physical interfaces**
2. **SECOND STEP : Logical interfaces**
3. **THIRD STEP : Operating data**
4. **FOURTH STEP : Process coordination**
5. **FIFTH STEP : External software**

The APPLICATION super-function is the only one that necessarily exists. It must be written first independently of others which may be added or supplemented later.

This ability alone shows the originality of the method. It is indeed generally prescribed to do a complete analysis of the system before starting to program for fear of having to re-write entire sections of the software.

I can cite an anecdote on this subject, which dates back to when I was a young engineer in a company that developed large public telephone exchanges. It was the era of new technologies and it was necessary to demonstrate this quickly to the upper hierarchy. In this case, it was necessary to demonstrate a telephone call. This resulted in the rapid writing of a (large) piece of software that was going to be thrown away because it was not reusable in a truly operational system. Which made an engineer older than me say: "for the hierarchy, it has to sound" with the inconvenience of wasting the time of a whole development team. The method I recommend was not yet developed, otherwise this quickly written software could have been reused.

This characteristic of the method will appeal to developers who also like to have concrete results quickly and not having to wait for all the additional programs to be written in order to be able to test the main functions.

The other super-functions exist or do not exist although some are unavoidable.

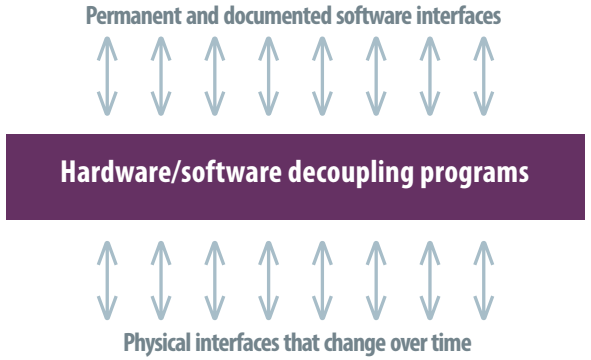
Their addition results in requests for additional entry points in each module concerned. The module manager then writes the corresponding program and updates the document listing the entry points of his module.

- FIRST STEP -
Physical interfaces

First of all, it is essential to identify the physical interfaces, ie all the sensors for which the software can read the value(s) as well as the actuators that the software can control. These physical interfaces are then grouped together by nature in different modules. This grouping is not subject to any particular rules and is left to the initiative of the developer. It is then necessary to write the translation programs between the permanent and documented software interfaces and the physical interfaces that can evolve over time.

A temperature sensor equipped with a thermistor will give a resistance value that will have to be transformed into degrees Celsius. It is a very simple program but it will probably have to be re-written in the event of a sensor change. On the physical side, we will have the information; "Physical address of the sensor + "Read value". On the software side, we will have the information: "Identity of the sensor" + "Temperature". The corresponding entry point will only indicate the parameter: "Identity of the sensor".

All of these programs do no processing. Their sole purpose is to isolate the core software from the outside world. Once the programs relating to all these inputs-outputs have been written, it remains for the person responsible for the module(s) to document all the entry points.



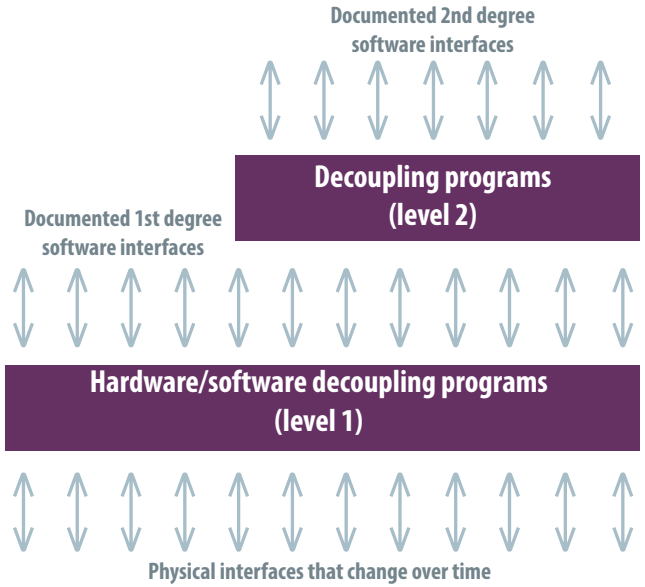
- SECOND STEP -
logical interfaces

In many cases, the physical environment does not provide directly usable information. It is necessary to process a certain number of successive data to obtain meaningful information.

Let's take the example of multi-frequency signals. A certain number of "speech" samples are needed for the decoding software to be able to determine the frequency or frequencies of the signal and therefore the number that has been dialed.

It may also happen to have to count pulses.

All these actions require a mini coordination software which will result in a table of programs selected by a state/event pair.



- THIRD STEP -
Operating data

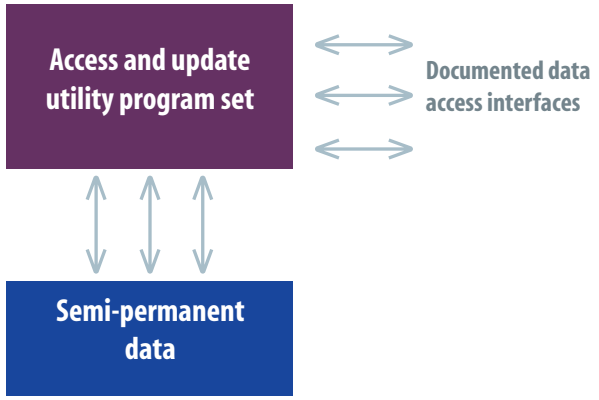
Operating data (or semi-permanent data) is data that differs from one installation of the system to another but does not change or very rarely changes on a given installation.

In a business phone system, this extensive data includes everything related to system size, users, outside lines, and so on.

In a GPS navigation system, the base map will vary from country to country. The home address and any preferences vary from system to system.

In a heating system, these will be the required temperatures.

These different examples show that semi-permanent data can represent large or very small volumes depending on the application.



- FOURTH STPE -
Process coordination

The process coordination programs are fed by all the events, pre-processed or not, that come from the outside world. These different events are associated with the state of the system to determine the actions to be executed, the last of these actions always being the update of the state of the system.

We thus obtain a table with two dimensions, one being the event, the other the state of the system.

In some cases, the number of states can be very large and it is then necessary to orient the processing according to the value of a certain number of data so as to reduce the size of the array to something reasonable.

Some systems have to manage several entities. The typical example is the telephone system which has to manage all the users of the system. This introduces a third dimension in the table which is the number of the entity from which an event originates.

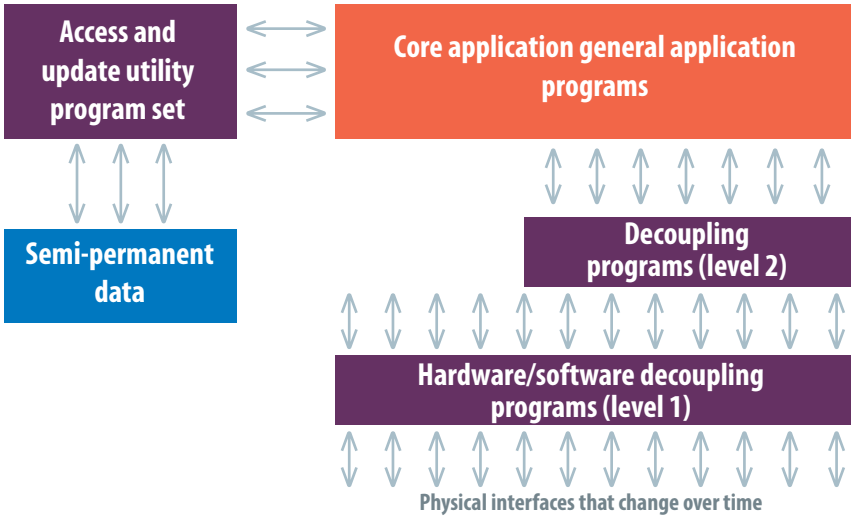
There can also be several types of entities. To take the example of the business telephone system, which is particularly rich, we find the users whose management will obey certain rules and the external lines which will obey others. There will therefore be an event/state table relating to users and a similar table relating to external lines.

The programs of each table can generate calls towards the other table, which constitutes for this one complementary events. In a sense, if an outside phone line rings, there will need to be a call to the user panel to ring a phone extension. In the other direction, if a user wishes to make an external call, the table of external lines must be called to allow him to dial on the public network.

An important additional rule is that a program performs processing only relative to a given entity. If several entities are involved in a process, an internal table entry must be used to perform the processes relating to the second entity. That is to say that, in addition to the "external event/state" pairs, there will be

"internal event/state" pairs. The "events" dimension of the table will therefore be increased accordingly.

A specific coordination program is often added to the previous one which manages the physical events. It supports time constraints. The simplest example is the program that needs to be run at regular intervals. This type of program can be masked or unmasked depending on external constraints. There are also programs which will only be executed after the priority programs if there is free time before the next deadline. These are, for example, audit or maintenance programs.



External software

Throughout the analysis, the need for complementary software, "off the shelf" and ready to use, will appear. These are, for example, USB communication software, digital filtering software making it possible to decode the requested number or that of the caller in telephony, human-machine interface software, IP-type communication software, etc.

The persons in charge will group all this software in one or more modules of which they will document the entry points.

APPLICATION TO A HEATING SYSTEM

1. **FIRST STEP :** The minimal system
2. **SECOND STEP :** Oil or gas central heating
3. **THIRD STEP :** Taking into account the thermal inertia of the system
4. **FOURTH STEP :** Taking into account an additional domestic hot water (DHW) circuit

PWhy choose a heating system for an example of software implementation using algebraic structuring?

The reasons are as follows:

- The features that come into play are quite easy to design.

- Depending on the sophistication of the system, the software can range from very simple to quite complicated, thus demonstrating the easy evolution from a basic version to a sophisticated version. Examples of commonly used structures are also implemented.

- The reader will not need to dive into the detail of the analysis, it is the functionality of each entry point that matters and it is specified each time in the preamble. It should also be noted that the resulting analysis has not been tested and may therefore contain some errors. If you discover one, you will have no trouble correcting it.

The simplest heating systems include a simple thermostat that directly controls on or off. We are going to suppose that this very simple system contains a microprocessor which can read the state of the thermostat and which can consequently control the starting or stopping of the heating.

The more sophisticated systems use a heat transfer fluid to distribute the energy in different places and manage several heating circuits: space heating, domestic hot water, heated towel rail in particular. They take into account the outside temperature and count the operating times relating to each circuit.

- STEP 1 -
The minimal system

First, we will consider that the heating element works on electricity and does not need flame ignition. Furthermore, the calories are transmitted directly to the ambient air without going through a heat transfer fluid as in the case of central heating.

It is quite obvious that for such a simple problem, the Algebraic Structuring of Software is of no immediate interest. But it allows to apprehend the way of thinking that is specific to it. In addition, the resulting analysis will serve as a basis for the implementation of more complex systems.

A classic program, not taking into account the Algebraic Structuring of Software, could be the following:

```
StartingPoint
    EnergieEmise = 0
    Write AdresseEnergie = Arret /*Writing to the energy emission control hardware address, of the value corresponding to the emission stoppage*/
Bouclage
    Read AdresseThermostat /*thermostat status reading: F = heating request; A = off*/
    If EtatThermostat = A GoTo ArretEnergie
EmissionEnergie
    If EnergieEmise = 1 GoTo Bouclage
    Write AdresseEnergie = Marche /*Writing to the energy emission command hardware address, of the value corresponding to the emission*/
    EnergieEmise = 1
    GoTo Bouclage
ArretEnergie
```

If EnergieEmise = 0 GoTo **Bouclage**

Write AdresseEnergie = Arret /*Writing to the energy emission control hardware address, of the value corresponding to the emission stoppage*/

EnergieEmise = 0

GoTo **Bouclage**

In a ASSO (Algebraic Structuring of SOftware) program, there will be 4 modules:

The IEN environment interface module, the DEV event detection module, the COO coordination module and the GET time management module.

In this very simple example, event detection and time management are anecdotal, but it is bad practice to mix programs with different functionalities in the same module. This is a first illustration of the fact that the Algebraic Structuring of Software presents strong analogies with the analysis of value.

We thus obtain the basic architecture of more complex systems.

Moreover, one of the first consequences of the existence of modules containing their own data, is that it is the modules themselves which carry out their initialization.

INTERFACE WITH THE ENVIRONMENT : IEN

The IEN environment interface module manages the room temperature sensor (thermostat) and the heating control.

Here, in addition to the initialization entry, three entry points will be needed, the names of which will begin with Ixx_:

I00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

Write AdresseEnergie = Arret /*Writing to the energy emission control hardware address, of the value corresponding to the emission stoppage*/

EnergieEmise = 0 /*Indicateur d'émission d'énergie*/

Return

I01_Thermostat

CALLED BY : COO

RETURNED PARAMETERS : ValeurThermostat (On => Chauffage; Off => ArretChauffage)

This entry point is used to know if the ambient temperature is above or below the demand.

Description of input parameters : none

Read AdresseThermostat /*reading thermostat status*/

Return

I02_ControleEmissionEnergie (OnOff)

CALLED BY : COO

RETURNED PARAMETERS : NONE

This entry point is used to start or stop the emission of energy.

Description of input parameters :

OnOff. Démarrage = 1; Arrêt = 0

Case OnOff = 1

Write AdresseEnergie = M /*Writing to the energy emission command hardware address, of the value corresponding to the emission*/

Return

Case OnOff = 0

Write AdresseEnergie = A /*Writing to the hardware address controlling the energy emission, the value corresponding to the emission

stop*/
Return
Case OnOff = other
Call Audit(I02_ControleEmissionEnergie, OnOff) /*For the record: call of an audit program following inconsistency with transmission of input values*/.
Return

EVENT DETECTION MODULE : DEV

This module makes it possible to detect the events which will be the basis of the processing carried out. Entry points start with Dxx_.

D00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

ValThermostat = 0 /*ambient temperature higher than requested temperature*/

Return

D01_ControleTemperatureAmbiante

CALLED BY : GET

RETURNED PARAMETERS :

This entry point is used to detect the crossing of the requested temperature threshold and to launch the appropriate actions.

Description of input parameters : none

Call I01_Thermostat

Case ValeurThermostat = Off /*stop heating*/

Case ValThermostat = 1 /*Previous state : ValeurThermostat = On*/

Call C01_Traitements(Evenement =0) /*passage of the temperature threshold on the rise*/

ValThermostat = 0 /*ValThermostat update*/

Return

Case ValThermostat = 0 /*Previous state : ValeurThermostat = Off*/

Return

Case ValeurThermostat = On /*heating start-up*/

Case ValThermostat = 1 /*Previous state : ValeurThermostat = On*/

Return

Case ValThermostat = 0 /*Previous state : ValeurThermostat = Off*/

ValThermostat = 1 /*ValThermostat update*/

Call C01_Traitements(Evenement =1) /*passage of the temperature threshold downwards*/

Return

COORDINATION MODULE: C00

This module is at the origin of the actions taken and generates the launch of the corresponding programs. The entry points, two in number, begin with Cxx_.

C00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

Etat = 0 /*initialization of the process state. No energy emission*/

Return

C01_Traitements (Evenement)

CALLED BY : DEV

RETURNED PARAMETERS : NONE

This entry point is the origin of all tasks performed by the system. There are two states: State 0: no energy emission; State = 1: energy emitted.

Description of input parameters :

Evenement = 0 : passage of the temperature threshold on the rise

Evenement = 1 : passage of the temperature threshold downwards

Case Etat = 0 /*No energy emission*/

Case Evenement = 0

Return

Case Evenement = 1

Call I02_ControleEmissionEnergie (OnOff=1)

Etat = 1 /*énergie émise*/

Return

Case Etat = 1 /*energy emitted*/

Case Evenement = 0

Call I02_ControleEmissionEnergie (OnOff=0)

Etat = 0

Return

Case Evenement = 1

Return

Case Etat = autre

Call Audit(C01_Traitements, Evenement) /*For the record: call of an audit program following inconsistency with transmission of input values*/

Return

MANAGEMENT OF TIME : GET

This module includes all the programs which make it possible to launch the tasks (only one in this example) whose occurrence is regular or depends on the availability of the processor. It is controlled by a clock interrupt. Entry points start with Gxx_

G00_SystStartingPoint

***** CALLED BY :**

***** RETURNED PARAMETERS : NONE**

This entry point is the starting point of the system at power-up. It allows the time management module to initialize its own data as well as those of the other modules. The following loop launches the tasks unmasked by the interrupt program

Description of input parameters : none

Mask ClockInterrupt /*masking the clock interrupt to avoid interruption during the initialization phase*/

Call I00_Initialisation /*Initialization of the interface module with the environment*/

```
Call C00_Initialisation /*Coordination module initialization*/
Call D00_Initialisation /*Initialization of the event detection module*/
Write ClockCounter = 0 /*software clock initialization*/
MaskD01 = 0 /*masking task D01*/
UnMask ClockInterrupt
AdresseBoucle /*Start address of the loop of tasks to be executed*/
Case MaskD01 = 1
   Call D01_ControleTemperature /*the treatments are carried out with a
   recurrence of 1s*/
   MaskD01 = 0 /*masking task D01*/
GOTO AdresseBoucle /*The only GOTO of the program!*/
```

G01_ClockInterrupt

Although other solutions are possible, I recommend that a 1ms interrupt be generated. This gives more flexibility to subsequent additions by notably allowing the creation of timer programs. The tasks to be executed are unmasked based on their execution period.

```
*** CALLED BY : CLOCK INTERRUPT
*** RETURNED PARAMETERS : NONE
```

This program is launched by the interruption at 1ms

```
Description of input parameters : none
Reset Interrupt
ClockCounter = ClockCounter +1
Case ClockCounter >= 1000
   Write ClockCounter = 0
   MaskD01 = 1 /*unmasking of task D01 executed every second */
   Return
Case ClockCounter < 1000
   Return
```


- STEP 2 -
Oil or gas central heating

The previous programs can be used directly for electric heating. But in the case of using a system operating on gas that does not include a pilot light or a system operating on fuel oil, it is not only necessary to control the emission of fuel (gas or domestic fuel oil) but also to proceed when this fuel is ignited. In all cases, it must be checked that ignition has taken place, which is done in the conventional way by a thermocouple or a flame detector. This check must be carried out regularly thereafter.

This check should be carried out after a few seconds and then repeated regularly. Fuel emission is cut off in the event of several successive negative checks. All values associated with these features are parameterized. This is not semi-permanent data because it does not change from one installation to another.

In the case of central heating, it is also necessary to take into account the temperature of the heat transfer fluid, the water which circulates in the radiators. Initially, we consider this to be a user setting. As the low thermal inertia of this fluid, associated with the heating power, causes its temperature to rise and fall quickly depending on whether there is combustion or not, it is necessary to provide a differential between the stopping temperature and the starting temperature of the combustion.

Time constraints make it necessary to expand a little on the module responsible for managing tasks based on time, i.e. launching tasks at the specified time. We will assume that no task requires precision or recurrence less than a second.

Let's consider the additions to be made, module by module.

In the interface module with the environment IEN , it will be necessary to add an entry point for reading the temperature of the heat transfer fluid which we will call I03_TemperatureCaloporteur, an ignition activation/stop control entry point which we will call I04_Allumage. It can be assumed that in some systems

a simple activation of the ignition is necessary. We will consider here that a deactivation is also necessary.

There will also be a flame presence control entry point which we will call I05_ ControleFlamme.

Finally, a circulator control command will be required, which will be turned on as long as the ambient temperature is below the requested temperature. It will be I06_Circulateur.

The DEV event detection module must include a heat transfer fluid temperature control program: D02_ControleTemperatureCaloporteur.

In the COO coordination module, there will be no entry point to add but the event/state table will have more entries.

In the GET time management module, entry points must be provided for masking or unmasking the launch of ignition tasks G02_MiseEnRouteAllumage and G03_ArretProcedureAllumage.

Furthermore, the temperature of the heat transfer fluid must not exceed a certain value. Most of the time, this value will be stored by operator. This is therefore operating data. An additional module will therefore have to be created: the DEX operating data module whose entry point name will begin with Xxx_. We will therefore have X00_Initialisation as well as X01_TemperatureMaximumCaloporteur

OPERATING DATA MODULE

X00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : None

This entry point is used to initialize module-specific data.

Description of input parameters : none

Return /*No data to initialize in this configuration.*/

X01_TempératureMaximumCaloporteur

CALLED BY : DEV

RETURNED PARAMETERS : TemperatureMaximumCaloporteur

This entry point is used to obtain the maximum value in degrees centigrade of the heat transfer fluid.

Description of input parameters : none

Read AdresseTemperatureMaximumCaloporteur /*Memory reading of the maximum temperature of the heat transfer fluid*/

Return

INTERFACE MODULE WITH THE ENVIRONMENT: IEN

The interface module with the environment includes the following additions:

I03_TempératureCaloporteur

CALLED BY : DEV

RETURNED PARAMETERS : TempératureCaloporteur

This entry point is used to obtain the value in degrees centigrade of the heat transfer fluid.

Description of input parameters : none

Read AdresseTemperatureCaloporteur /*reading the thermistor value and translating it into temperature value*/

Return

I04_Allumage(OnOff)

CALLED BY : COO

RETURNED PARAMETERS : NONE

This entry point is used to start or stop the fuel ignition process. If it is an electric arc for example, it is necessary to establish the voltage necessary for the arc and then to suppress it after a certain delay.

Description of input parameters :

OnOff : 1 octet. Allumage = 1; Arrêt = 0

Case OnOff = 1

Write AdresseAllumage = M /*Switching on the ignition. Writing to the ignition control hardware address, the value corresponding to the ignition*/

Return

Case OnOff = 0

Write AdresseAllumage = A /*Writing to the ignition control hardware address, of the value corresponding to the ignition stopping */

Return

Case OnOff = other

Call Audit(I04-Allumage, OnOff) /*For the record: call of an audit program following inconsistency with transmission of input values*/

Return

I05_Controleflamme

CALLED BY : GET

RETURNED PARAMETERS : PresenceFlamme

PresenceFlamme = 1 flamme; PresenceFlamme = 0 pas de flamme

This entry point is used to know if the flame is present or not

Description of input parameters : none

Read AdresseCapteurFlamme /*reading at the sensor address of the value provided by the sensor and translation into "presence/absence"*/

Return

I06_Circulateur(OnOff)

CALLED BY : COO

RETURNED PARAMETERS : NONE

This entry point is used to start or stop the heat transfer fluid circulator

Description of input parameters :

OnOff : 1 octet. circulator start-up = 1; Arrêt circulateur = 0

Case OnOff = 1

Write AdresseCirculateur = 1 /*Starting the circulator. Writing to the circulator control hardware address of the value corresponding to start-up*/

Return

Case OnOff = 0

Write AdresseCirculateur = 0 /*Stopping the circulator. Writing to the circulator control hardware address of the value corresponding to the stop*/

Return

Case OnOff = other

Call Audit(I06_Circulateur, OnOff) /*For the record: call of an audit program following inconsistency with transmission of input values*/

Return

EVENT DETECTION MODULE : DEV

The event detection module includes the following additions :

D00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

ValThermostat = 0 /*ambient temperature higher than requested temperature*/

Call X01_TemperatureMaximumCaloporteur

TemperatureUsageCaloporteur = TemperatureMaximumCaloporteur /*The current temperature of the heat transfer fluid is initialized to the maximum value*/

Return

D02_ControleTemperatureCaloporteur

CALLED BY : GET

RETURNED PARAMETERS :

This entry point makes it possible to detect the crossing of the requested temperature threshold and to launch the appropriate actions. It is necessary to memorize 3 temperature levels: the low level "B" where the temperature is lower than the maximum temperature minus the differential, the intermediate level "I" where the temperature is higher than the maximum temperature minus the differential and lower than the maximum temperature and finally the high level "H" where the temperature is higher than the maximum temperature.

Description of input parameters : none

Call I03_TempératureCaloporteur

Case TempératureCaloporteur =< TemperatureUsageCaloporteur - Différentiel

Case SeuilCalo = B /*Previous state: heat transfer fluid temperature < TemperatureUsageCaloporteur - differential*/

Return

Case SeuilCalo = I /*Previous state: heat transfer fluid temperature = intermediate temperature*/

Call C01_Traitements (Evenement =2) /*passing the heat transfer fluid temperature threshold downward*/

SeuilCalo = B /*Threshold update*/

Return

Case SeuilCalo = H /*Previous state: heat transfer fluid temperature > maximum temperature*/

Call C01_Traitements (Evenement =2) /*passing the heat transfer fluid temperature threshold downward*/

SeuilCalo = B /*Threshold update*/

Return

Case TempératureCaloporteur < TemperatureUsageCaloporteur

Case SeuilCalo = B /*Previous state: heat transfer fluid temperature < TemperatureUsageCaloporteur -- differential*/

SeuilCalo = I /*Threshold update/

Return

Case SeuilCalo = I /*Previous state: heat transfer fluid temperature = intermediate temperature*/

Return

Case SeuilCalo = H /*Previous state: heat transfer temperature > maximum temperature*/

SeuilCalo = I /*Threshold update*/

Return

Case TempératureCaloporteur >= TemperatureUsageCaloporteur

Case SeuilCalo = B /*Previous state: heat transfer fluid temperature < TemperatureUsageCaloporteur - differential*/

Call C01_Traitements (Evenement =3) - /*passage of the temperature threshold of the heat transfer fluid upwards.*/

SeuilCalo = H /*Threshold update*/

Return

Case SeuilCalo = I /*Previous state: heat transfer fluid temperature = intermediate temperature*/

Call C01_Traitements (Evenement =3) - /*passage of the temperature threshold of the heat transfer fluid on the rise*/

SeuilCalo = H /*Threshold update*/

Return

Case SeuilCalo = H /*Previous state: heat transfer fluid temperature > maximum temperature*/

Return

COORDINATION MODULE : COO

Some initializations must be added, while the event processing program becomes significantly more complex.

C00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

Etat = 0 /*initialization of the state of the coordination process*/

STA = 1 /*Ambient temperature indicator >= requested temperature*/

STC =1 /*Coolant temperature indicator >= maximum temperature*/

Return

C01_Traitements (Evenement)

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is the origin of all the tasks carried out by the system, even if the launching of some is subcontracted to time management.

We can now consider the following states:

0: No energy emission

Only temperature control is performed

1: energy emission, flame control in progress.

Rather than planning for the cases "energy emission before ignition", "energy emission after ignition and before flame control", we will consider that these will be two phases of the same state.

2: Ignition carried out

3: Fault. We can only exit this state by human intervention (IHM) not covered here.

Description of input parameters : Evenement

Evenement = 0 : passage of the ambient temperature threshold upwards

Evenement = 1 : passing the ambient temperature threshold downward

Evenement = 2 : passage of the temperature threshold of the heat transfer fluid downwards

Evenement = 3 : passage of the temperature threshold of the heat transfer fluid upwards

Evenement = 4 : Flame detected

Evenement = 5 : No flame detection

Case Etat = 0 /*No energy emissions*/

Case Evenement = 0 /*passage of the temperature threshold upwards*/

STA = 1 /*Update indicator: ambient temperature >= requested temperature*/

Return

Case Evenement = 1 /*passage of the temperature threshold downwards*/

STA = 0 /*Update indicator: ambient temperature < requested temperature*/

Case STC = 0 /*heat transfer fluid temperature <maximum temperature - differential*/

Call I02_ControleEmissionEnergie (OnOff=1) /*Fuel emission*/

Call I06_Circulateur(OnOff = 1) /*circulator start-up*/

Call G02_MiseEnRouteAllumage

Etat = 1 /*energy emitted + uncontrolled flame*/

Phase = 0 /*Energy emission phase before ignition*/

Compteur = 0

Return

```
    Case Evenement = 2 /*Passage of the heat transfer fluid temperature threshold downwards*/
        STC = 0 /*UPDATE: heat transfer temperature <maximum temperature - differential*/
        Case STA= 0 /*ambient temperature < requested temperature*/
            Call I02_ControleEmissionEnergie (OnOff=1) /*Fuel emission*/
            Call I06_Circulateur(OnOff = 1) /*circulator start-up*/
            Call G02_MiseEnRouteAllumage
            Etat = 1 /*circulator start-up*/
            Phase = 0 /*Energy emission phase before ignition*/
            Compteur = 0
            Return
    Case Evenement = 3 /*Passage of the thermal fluid temperature threshold upwards*/
        STC = 1 /*UPDATE: heat transfer temperature >= maximum temperature*/
        Return
    Case Evenement = autre /*Cases 4 and 5+ are inconsistent*/
        Call Audit(C01, Etat = 0, Evenement : x) /*For the record: call of an audit program following inconsistency with transmission of input values*/
        Return
Case Etat = 1 /*energy emitted + flame control in progress*/
    Case Evenement = 0
        Call I02_ControleEmissionEnergie (OnOff=0)
        Call I06_Circulateur(OnOff = 0) /*circulator stop*/
        Call G03_ArretProcedureAllumage
        STA = 1 /*ambient temperature >= requested temperature*/
        Etat = 0
        Return
    Case Evenement = 1
        Return
```



```
    Case Evenement = 2
        Return
    Case Evenement = 3
        Call I02_ControleEmissionEnergie (OnOff=0)
        Call I06_Circulateur(OnOff = 0) /*circulator stop*/
        Call G03_ArretProcedureAllumage
        STC = 1 /*heat transfer fluid temperature >= maximum temperature*/
        Etat = 0
        Return
    Case Evenement = 4 /*Flamme détectée*/
        Etat = 2
        Return
    Case Evenement = 5 /*Pas de flamme*/
        Call I02_ControleEmissionEnergie (OnOff=0)
        Call I06_Circulateur(OnOff = 0) /*circulator stop*/
        Call G03_ArretProcedureAllumage
        Etat = 3 /*Mistake*/
        Call Alarme : Le programme appelé, non détaillé ici, génère l'affichage d'une information signalant le problème. La sortie de l'état 3 ne peut se faire que suite à une commande manuelle.
        Return
Case Etat = 2 /*flamme contrôlée*/
    Case Evenement = 0
        Call I02_ControleEmissionEnergie (OnOff=0)
        Call I06_Circulateur(OnOff = 0) /*circulator stop*/
        Call G03_ArretProcedureAllumage /*Nécessaire pour désactiver le contrôle régulier de la flamme*/
        STA = 1 /*ambient temperature >= requested temperature*/
        Etat = 0
        Return
```

```
Case Evenement = 1
    Return
Case Evenement = 2
    STC = 0 /*température caloporteur < température maximale moins différentiel*/
    Return
Case Evenement = 3
    Call I02_ControleEmissionEnergie (OnOff=0)
    Call I06_Circulateur(OnOff = 0) /*circulator stop*/
    Call G03_ArretProcedureAllumage /*Nécessaire pour désactiver le contrôle régulier de la flamme*/
    STC = 1 /*heat transfer fluid temperature >= maximum temperature*/
    Etat = 0
    Return
Case Evenement = 4 /*Event not normally expected*/
    Call Audit(C01, Etat = 2, Evenement : 4): /*For the record: call of an audit program following inconsistency with transmission of input values*/.
    Return
Case Evenement = 5 /*Unintentional stopping of combustion*/
    Call I02_ControleEmissionEnergie (OnOff=0)
    Call I06_Circulateur(OnOff = 0) /*circulator stop*/
    Call G03_ArretProcedureAllumage /*Needed to disable regular flame control*/
    Etat = 3
    Return
Case Etat =3 /*Mistake*/
  Return /*no treatment whatever the event*/
```


TIME MANAGEMENT MODULE : GET

The only programs that are time-constrained are those for ignition and flame control.

G00_SystStartingPoint

***** CALLED BY :**

***** RETURNED PARAMETERS : NONE**

This entry point is the system startup point upon power-up. It allows the time management module to initialize its own data as well as that of other modules. The following loop launches the tasks unmasked by the interrupt program.

Three timing indicators will be used to determine the current phase: timing before ignition (IndicateurTempo1EnCours), ignition duration timing (IndicateurTempo2EnCours), timing before flame detection (IndicateurTempo3EnCours).

It can be assumed that a few seconds are needed after the fuel is emitted to ignite the flame. Depending on the type of ignition, we assume that it must also last a few seconds to be certain of the result. Finally, it seems reasonable to wait a little after lighting before checking the flame. All these time constraints will be satisfied by a timeout program. If you use an operating system, timing services are available but it is so simple to generate them yourself. The ideal is to make yourself as independent as possible of the operating systems to obtain a much higher processing capacity and to be independent of software that you do not control.

Description of input parameters : none

```
Mask ClockInterrupt /*masking of the clock interrupt to avoid interruption during the initialization phase*/
Call I00_Initialisation /*Initialization of the interface module with the environment*/
Call C00_Initialisation /*Coordination module initialization*/
Call D00_Initialisation /*Initialization of the event detection module*/
Call X00_Initialisation /*Initializing the operating data module*/
Write ClockCounter = 0 /*software clock initialization*/
MaskD01 = 0 /*masking task D01*/
MaskD02 = 0 /*masking task D02*/
```

```
IndicateurTempo1EnCours = 0 /*Invalidation of fuel emission delay before ignition*/
IndicateurTempo2EnCours = 0 /*Ignition duration time delay invalidation*/
IndicateurTempo3EnCours = 0 /*Invalidation of time delay before flame detection*/
IndicateurTempo4EnCours = 0 /*Invalidation of regular flame control timer*/
UnMask ClockInterrupt
AdresseBoucle /*Start address of the loop of tasks to be executed*/
/*******************/
Case MaskD01 = 1
  Call D01_ControleTemperature /*the treatments are carried out with a recurrence of 1s*/
  MaskD01 = 0 /*masking task D01*/
/************************/
Case MaskD02 = 1
  Call D02_ControleTemperatureCaloporteur /*the treatments are carried out with a recurrence of 1s*/
  MaskD02 = 0 /*masking task D02*/
/*******************************/
Case IndicateurTempo1EnCours = 1 /*Fuel emission delay before ignition*/
  ClockCounterT1 = ClockCounterT1+1 /*fuel emission duration counter before ignition*/
  Case ClockCounterT1 = T1 /*end of timout*/
    Call I04_Allumage(OnOff = 1) /*starting the ignition*/
    IndicateurTempo2EnCours = 1 /*Activation of ignition duration delay*/
    IndicateurTempo1EnCours = 0 /*Deactivation of fuel emission delay before ignition*/
    ClockCounterT2 = 0
/************************************/
```



```
Case IndicateurTempo2EnCours = 1 /*Ignition duration delay*/
  ClockCounterT2 = ClockCounterT2+1 /*ignition duration counter*/
  Case ClockCounterT2 = T2 /*end of ignition*/
    Call I04_Allumage(OnOff = 0)
    IndicateurTempo2EnCours = 0 /*Disable ignition duration delay*/
    IndicateurTempo3EnCours = 1 /*Delay activation before flame detection*/
    ClockCounterT3 = 0
/************************************/
Case IndicateurTempo3EnCours = 1 /*Delay before flame detection*/
  ClockCounterT3 = ClockCounterT3+1 /*time counter before flame check*/
  Case ClockCounterT3 = T3
    ClockCounterT'4 = 0
    IndicateurTempo3EnCours = 0 /*Deactivation of delay before flame detection*/
    Call I04_Controle_Flamme
      Case PrésenceFlamme = 1
        Call C01_Traitements(Evenement = 4) /*Flame detected*/
        IndicateurTempo4EnCours = 1 /*Activation of regular flame check delay*/
        Essai = 0 /*reset of the counter for the maximum number of negative checks for the presence of flame*/
      Case PrésenceFlamme = 0
        Essai = Essai + 1
        Case Essai < Max
          IndicateurTempo2EnCours = 1 /*restarting the ignition procedure*/
        Case Essai >= Max
        Essai = 0 /*resetting the counter of the maximum number of negative flame presence checks*/
        Call C01_Traitements(Evenement = 5) /*No flame detected*/
```

```
        /*********************************/
    Case IndicateurTempo4EnCours = 1 /*Regular flame control time delay*/
      ClockCounterT4 = ClockCounterT4+1 /*time counter before each flame
      check*/
      Case ClockCounterT4 = T4
        Call I04_Controle_Flamme
          Case PrésenceFlamme = 1
            CompteurPresenceFlamme = 0
          Case PrésenceFlamme = 0
            CompteurPresenceFlamme = CompteurPresenceFlamme +1
            Case CompteurPresenceFlamme = MaxCPF
              Call C01_Traitements(Evenement = 5) /*No flame detected*/
              IndicateurTempo4EnCours = 0
    GOTO AdresseBoucle /*The only GOTO of the program!*/
```

G01_ClockInterrupt

CALLED BY : CLOCK INTERRUPT

RETURNED PARAMETERS : NONE

This program is started by the interrupt at 1 ms. It allows you to unmask event detection tasks every second.

Saving the data contained in the registers used by the interrupt program is not necessary here.

Description of input parameters : none

```
ClockCounter = ClockCounter +1
Case ClockCounter >= 1000
  Write ClockCounter = 0
  MaskD01 = 1 /*unmasking task D01*/
  MaskD02 = 1 /*unmasking task D02*/
Return
```


G02_MiseEnRouteAllumage

CALLED BY : C01

RETURNED PARAMETERS : NONE

This entry point allows GET to trigger the sequence of actions necessary for ignition (not to be confused with fuel emission!). For greater readability, a counter will be assigned to each timing phase.

Description of input parameters : none

```
IndicateurTempo1EnCours = 1 /*Tfuel emission delay before ignition*/
ClockCounterT1 = 0
Return
```

G03_ArretProcedureAllumage

This program stops the ignition procedure at any stage

CALLED BY : C01

RETURNED PARAMETERS : NONE

Description of input parameters : none

```
IndicateurTempo1EnCours = 0 /*Fuel emission delay before ignition*/
IndicateurTempo2EnCours = 0 /*Ignition duration delay*/
IndicateurTempo3EnCours = 0 /*Delay before flame detection*/
IndicateurTempo4EnCours = 0 /*Regular flame control time delay*/
Return
```

- STEP 3 -
Taking into account the thermal inertia of the system

When there is a heat transfer fluid, the energy it contains will continue to heat the room even after combustion has stopped. Heating can thus become excessive, leading to an unwanted rise in the ambient temperature. The ideal is that its temperature is the minimum temperature to maintain the desired room temperature. We can imagine several algorithms that make it possible to determine this temperature. The simplest is to lower the temperature of the heat transfer fluid until the time interval where the ambient temperature remains above the required temperature is short, for example 5 minutes. To not complicate things too much, we will assume that the difference between the temperature of the heat transfer fluid and the ambient temperature must be proportional to the difference between the ambient temperature and the outside temperature.

Strictly speaking, it would also be necessary to take into account a time lag in the effect of a variation in exterior temperature on the heat loss of the home. But this is about giving an example of structuring, not optimizing heating system software.

It will be necessary to create a new OBC observation/counting module in order to know the combustion duration and the combustion cessation duration. This module will be powered by the on/off program.

In the interface module with the IEN environment, I02 will need to be completed to count the heating times in the OBC module and add three additional entry points; one to have access to the outside temperature: I08_TemperatureExterieure, the second to adjust the temperature of the heat transfer fluid: I09_AjustementTemperatureCaloporteur and the third to obtain the requested ambient temperature: I10_TemperatureDemandee. The latter does double duty with reading the thermostat to decide whether to turn the heating on or off. It is nevertheless necessary to determine the coefficient K as explained below.

If the duration of combustion shutdown is greater than a given value, the value of the maximum temperature of the heat transfer fluid will be reduced until it is

stable. The proportionality coefficient K between (heat transfer fluid temperature - ambient temperature) and (ambient temperature - outside temperature) will then be calculated.

Finally, in the Time Management module, you will need to create a clock allowing COB to calculate the combustion shutdown times.

INTERFACE AVEC L'ENVIRONNEMENT : IEN

I02_ControleEmissionEnergie (OnOff)

CALLED BY : COO

RETURNED PARAMETERS : NONE

This entry point is used to start or stop the emission of energy.

Description of input parameters :

OnOff : 1 octet. Start = 1; Stop = 0

```
Case OnOff = 1
    Write AdresseEnergie = M /*Writing to the hardware address controlling the energy emission, the value corresponding to the emission*/
    Call 001_DureeEmissionEnergie(MarcheArret = 1
    EnergieEmise = 1
    Return
Case OnOff = 0
    Write AdresseEnergie = A /*Writing to the hardware address controlling the energy emission, the value corresponding to the emission stop*/
    Call 001_DureeEmissionEnergie(MarcheArret = 0)
    EnergieEmise = 0
    Return
Case OnOff = other
    Call Audit(I02_ControleEmissionEnergie, OnOff) /*For the record: call of an audit program following inconsistency with transmission of input values*/.
```



```
Return
```

I08_TempératureExterieure

CALLED BY : OBC

RETURNED PARAMETERS : TempératureExterieure

This entry point is used to obtain the value in degrees centigrade of the outside temperature.

Description of input parameters : none

```
Read AdresseTempératureExtérieure /*reading the resistance value and translating it into a temperature value*/
Return
```

I09_AjustementTempCaloporteur(Ajustement)

CALLED BY : OBC

RETURNED PARAMETERS :

This entry point is used to adjust the temperature of the heat transfer fluid.

Description des paramètres d'entrée : Ajustement. Ajustement =1 : increase of one notch in the temperature of the heat transfer fluid; Ajustement =0 : reduction of one notch in the temperature of the heat transfer fluid. The notch value is a system parameter.

```
Case Ajustement = 1
    TemperatureUsageCaloporteur = TemperatureUsageCaloporteur + Cran
Case Ajustement = 0
    TemperatureUsageCaloporteur = TemperatureUsageCaloporteur - Cran
    Return
```

I10_TempératureDemandee

CALLED BY : OBC

RETURNED PARAMETERS : TempératureDemandee

This entry point is used to obtain the value in degrees centigrade of the requested temperature. This entry point, combined with the ambient temperature reading, could replace heating control by the thermostat. It is necessary to determine the coefficient K.

Description of input parameters : none

Read AdresseTempératureDemandee /*reading the resistance value and translating it into a temperature value*/

Return

TIME MANAGEMENT MODULE : GET

In G01_ClockInterrupt, you will need to add some instructions to obtain times in hours, minutes and seconds:

G01_ClockInterrupt

CALLED BY : CLOCK INTERRUPT

RETURNED PARAMETERS : NONE

Description of input parameters : none

```
ClockCounter = ClockCounter +1
Case ClockCounter > 1000
    Write ClockCounter = 0
    Write Secondes = Secondes + 1
    Case Secondes > 60
        Secondes = 0
        Write Minutes = Minutes + 1
        Case Minutes > 60
            Minutes = 0
            Heures = Heures + 1
    MaskD01 = 1 /*unmasking task D01*/
    MaskD02 = 1 /*unmasking task D02*/
    Return
```

And you need an entry point for reading the time:

G04_LectureHorloge

This program returns the current time

CALLED BY : 001

RETURNED PARAMETERS : Heures, Minutes, Secondes

Description of input parameters : none

```
Read Heures
Read Minutes
Read Secondes
Return
```

OBSERVATIONS/COUNTS MODULE : OBC

In the new OBC observation/counting module, a first initialization entry point 000 will be required, an entry point 001_DureeEmissionEnergie intended to update the counters for the duration of operation and stoppage of the emission of energy, an entry point intended to count the heating and heating stop times 002_DureeHeating (different from the emission of energy which stops when the maximum temperature of the heat carrier is reached), as well as a point of consultation entry for the various counters, not taken into account here. 002_DureeChauffage is called by the C01_Traitements program of which it is the only update that will only appear in the next step.

The heating times will be accumulated, which is interesting to know for consumption. On the other hand, only the last digit of the interval between two fuel emissions will be kept. Its value will be used to reduce the temperature of the heat transfer fluid if it is higher than the configured value, increase the temperature if it is lower than this value.

Finally, we will determine the value of the coefficient K which makes it possible to change the temperature of the heat transfer fluid as a function of the outside temperature. This value will be kept in non-volatile memory and initialized at the factory.

We will also observe that with this very simple algorithm, knowing the outside

temperature does not add much: the system adjusts itself. However, this is a possibility offered by all boilers on the market. We will therefore keep it.

O00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : None

This entry point is used to initialize module-specific data.

Description of input parameters : none

Return /*No data to initialize in this configuration.*/

O01_DureeEmissionEnergie (MarcheArret)

CALLED BY : IEN

RETURNED PARAMETERS : None

This entry point is used to update the run time counter.

Description of input parameters : Start of energy emission : MarcheArret = 1, Energy emission shutdown : MarcheArret = 0

Call G04_LectureHorloge

Case MarcheArret = 1 /*Start of energy emission*/

Store HeureMiseEnMarche /*save start time*/

Case MarcheArret = 0 /*Energy emission shutdown*/

Write HeureArret

DuréeMarche = HeureArret - HeureMiseEnMarche

DureeTotaleMarche = DureeTotaleMarche + DureeMarche

Return

O02_DureeChauffage (MarcheArret)

CALLED BY : DEV

RETURNED PARAMETERS : None

This entry point is used to count the times during which space heating is active. We will take into account the duration during which it is inactive to calculate the coefficient K.

Description of input parameters : Start of energy emission : MarcheArret = 1, Energy emission shutdown : MarcheArret = 0

Call G04_LectureHorloge

Case MarcheArret = 1 /*Start of heating*/

Store HeureDebutChauffage /*save heating start time*/

DureeArretChauffage = HeureDebutChauffage - HeureArretChauffage

Case DureeArretChauffage >= DureeOptimale /*DureeOptimale is a construction parameter*/

Call I09_AjustementTempCaloporteur(Ajustement = 0)

Case DureeArret < DureeOptimale /*DureeOptimale is a construction parameter*/

Call I09_AjustementTempCaloporteur(Ajustement = 1)

Case MarcheArret = 0 /*Heating stop*/

Write HeureArretChauffage

DuréeMarcheChauffage = HeureArretChauffage - HeureDebutChauffage

DureeTotaleMarcheChauffage = DureeTotaleMarcheChauffage + DureeMarcheChauffage

Call I03_TempératureCaloporteur

Call I10_TemperatureDemandee

Call I08_TemperatureExterieure

K = (TemperatureCaloporteur - TemperatureDemandee)/(TemperatureDemandee - TemperatureExterieure)

Store K /*save K in non-volatile memory*/

Return

EVENT DETECTION MODULE : DEV

In order to start with a correct value of the temperature of the heat transfer fluid after switching off, it is necessary to replace in the initialization the consideration of the maximum value by a calculation of this value from the coefficient K which is constantly adjusted. during operation.

D00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

```
Write AdresseEnergie = A /*Writing to the hardware address controlling
the energy emission, the value corresponding to the emission stop*/
EnergieEmise = 0 /*Energy emission indicator*/
Call I10_TemperatureDemandee
Call I08_TemperatureExterieure
TemperatureUsageCaloporteur = TemperatureDemandee + K/(Tempera-
tureDemandee - TemperatureExterieure)
Return
```


- STEP 4 -
Taking into account an additional domestic hot water circuit (DHW)

This step will illustrate the management of several entities by the system. In certain cases, these entities can be differentiated only by the data which concerns them (operational data) or by a specific mode of operation. It is this last case which will be taken into account here.

However, there remain strong similarities:

- the requested DHW temperature which is that of the water in the hot water tank instead of that of the ambient air;

- the temperature of the heat transfer fluid which will be different from that used for heating the ambient air;

- the process of switching the burner on and off. This organ being common, priority will be given to domestic hot water which can thus interrupt the heating of the premises.

It will be necessary to plan:

- a program for reading the requested DHW temperature,

- a program for reading the DHW temperature.

- a program for controlling a second circulator whose role is to direct the hot water leaving the boiler to the hot water tank exchanger.

The maximum temperature of the heat transfer fluid will be that of the requested ECS temperature increased by DeltaCalECS. Arriving at this maximum temperature, the burner will be interrupted until the temperature has decreased by VarCal. This sequence will continue until the DHW temperature has reached the requested temperature. It will resume when the DHW temperature is the requested temperature minus VarECS.

In the DEX operating data module, it will be necessary to plan to repatriate the maximum value of the heat transfer fluid which differs depending on the use.

In IEN, you will need to modify I06 which indicates the maximum temperature of the heat transfer fluid. In fact, this temperature is now different depending on whether you heat the ambient air or the water in the hot water tank. I06 must also provide the temperature differential to be applied depending on the case. This data is semi-permanent and stored in the operational data module. To choose one or other of the sets of values, it will use the HeatingMode indicator located in the coordination module.

OPERATING DATAMODULE : DEX

X01_TempératureMaximumCaloporteur(ModeChauffage)

CALLED BY : DEV

RETURNED PARAMETERS : Maximum temperature of the heat transfer fluid, Differentiel

This entry point is used to obtain the maximum value in degrees centigrade of the heat transfer fluid depending on the use (local heating or DHW) with the differentials to be applied.

Description of input parameters : ModeChauffage (premises heating : 0; DHW : 1)

Read AdresseTemperatureMaximumCaloporteur, index = ModeChauffage /*The maximum temperatures of the heat transfer fluid as well as the differentials are located in the operating data module*/

Return

INTERFACE MODULE WITH THE ENVIRONMENT : IEN

We will also have the following additional programs:

I11_TempératureDemandeeECS

CALLED BY : DEV

RETURNED PARAMETERS : TempératureDemandeeECS

This entry point is used to obtain the value in degrees centigrade of the requested DHW temperature.

Description of input parameters : none

Read AdresseTempératureDemandeeECS /*reading the resistance value and translating it into a temperature value or reading the stored data*/

Return

I12_TempératureECS

CALLED BY : DEV

RETURNED PARAMETERS : TempératureECS

This entry point is used to obtain the value in degrees centigrade of the DHW temperature.

Description of input parameters : none

Read AdresseTempératureDemandeeECS /*reading the resistance value and translating it into a temperature value*/

Return

I13_CirculateurECS(OnOff)

CALLED BY : COO

RETURNED PARAMETERS : NONE

This entry point is used to start or stop the circulator directing the hot water leaving the boiler to the hot water tank exchanger.

Description of input parameters :

OnOff : 1 octet. DHW circulator start-up = 1; DHW circulator stop = 0

Case OnOff = 1

Write AdresseCirculateurECS = M /*Switching on the DHW circulator. Writing to the DHW circulator control hardware address of the value corresponding to start-up*/

Return

Case OnOff = 0

Write AdresseCirculateurECS = A /*Stopping the DHW circulator. Writing to the DHW circulator control hardware address, the value corresponding to the stop*/

Return

Case OnOff = other

Call Audit(I12_CirculateurECS, OnOff) /*For the record: call of an audit program following inconsistency with transmission of input values*/

Return

EVENT DETECTION MODULE : DEV

Input point D01 uses the state of a thermostat to detect crossing the ambient temperature threshold. Regarding the DHW temperature, we have a minimum temperature and a maximum temperature. It is therefore necessary to add the ad hoc program. This entry point will now be called: D01_ExtremityTemperatureControl, the end being able to be ambient air or domestic hot water.

D01_ControleTemperatureExtremite

CALLED BY : GET

RETURNED PARAMETERS :

This entry point makes it possible to detect the crossing of the requested temperature threshold and to launch the appropriate actions.

Description of input parameters : none

Call C02_ModeChauffage /*premises heating or domestic hot water*/

Case ModeChauffage = 0 /*premises heating*/

Call I01_Thermostat

Case ValeurThermostat = Off /*heating stop*/

Case ValThermostat = 1 /*Previous state: Thermostat value = On*/

Call C01_Traitements(Evenement =0) /*passage of the temperature threshold upwards*/

ValThermostat = 0 /*ValThermostat update*/

Return

Case ValeurThermostat = 0 /*Previous state: Thermostat Value = Off*/

Return

Case ValeurThermostat = On /*heating start-up*/

Case ValThermostat = 1 /*Previous state: Thermostat Value = On*/

Return

Case ValThermostat = 0 /*Previous state: Thermostat Value = Off*/

ValThermostat = 1 /*ValThermostat update*/

Call C01_Traitements(Evenement =1) /*passage of the temperature threshold downwards*/

Return

Case ModeChauffage = 1 /*DHW heating*/

Call I12_TempératureECS

Call I11_TemperatureDemandeeECS

Case TempératureECS =< TemperatureDemandeeECS - VarECS

Case SeuilECS = B /*Previous state : températureECS < TemperatureDemandeeECS -- VarECS*/

Return

Case SeuilECS = I /*Previous state : température ECS = température intermédiaire*/

Call C01_Traitements (Evenement =1) /*passage of the DHW temperature threshold downwards*/

SeuilECS = B /*Threshold update*/

Return

Case SeuilECS = H /*Previous state : température ECS > température demandée*/

Call C01_Traitements (Evenement =1) /*passage of the DHW temperature threshold downwards*/

SeuilECS = B /*Threshold update*/

Return

Case TempératureECS < TemperatureDemandeeECS

Case SeuilCalo = B /*Previous state : température ECS < Temperature-DemandeeECS -- VarECS*/

SeuilECS = I /*Threshold update/

Return

Case SeuilCalo = I /*Previous state : température ECS = température intermédiaire*/

Return

Case SeuilCalo = H /*Previous state : température ECS > température demandée ECS*/

SeuilCalo = I /*Threshold update*/

Return

Case TempératureECS >= TemperatureDemandeeECS

Case SeuilECS = B /*Previous state : température ECS < Temperature-DemandeeECS - VarECS*/

Call C01_Traitements (Evenement =0) /*passage of the DHW temperature threshold upwards*/

SeuilCalo = H /*Threshold update*/

Return

Case SeuilECS = I /*Previous state: heat transfer temperature = intermediate temperature*/

Call C01_Traitements (Evenement =0) /*passage of the DHW temperature threshold upwards*/

SeuilECS = H /*Threshold update*/

Return

Case SeuilECS = H /*Previous state: heat transfer temperature > maximum temperature*/

Return

The entry point D02_ControleTemperatureCaloporteur does not include any modification.

COORDINATION MODULE : C00

The initialization program is completed by the heating mode. An additional entry point will also be necessary to determine which heating mode we are in: C02_HeatingMode.

It will also be necessary to replace the data specific to space heating by variable data depending on the heating method. As this data is in the same module, we have direct access to it. We will therefore replace "ambient temperature" with "end temperature" which could be either the ambient temperature or the DHW temperature. The STA indicator will be at 1 when the end temperature is higher than the requested temperature.

C00_Initialisation

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is used to initialize module-specific data.

Description of input parameters : none

ModeChauffage= 0 /*Premises heating*/

Etat = 0 /*initialization of the state of the coordination process*/

STA = 1 /*end temperature indicator >= requested temperature*/

STC =1 /*heat transfer fluid temperature indicator >= maximum temperature*/

Return

The C01_Treatments entry point varies little.

C01_Traitements (Evenement)

CALLED BY : GET

RETURNED PARAMETERS : NONE

This entry point is the origin of all tasks performed by the system. There is no saving

of data used by the interrupted program because all tasks are performed for a duration well below a millisecond.

In this new step, we add calls to the OBSERVATION/COUNTS OBC module at each start or stop of the circulator corresponding to the heating of the premises, in order to count the heating durations and especially the stops to calculate the coefficient K.

We can now consider the following states:

0 : No energy emissions

Only temperature control is performed

1 : energy emission, flame control in progress.

Rather than providing for the cases "emission of energy before ignition", emission of energy after ignition and before flame control", we will consider that these will be two phases of the same state

2 : Ignition completed

3 : Mistake. We can only exit this state by human intervention (IHM) not covered here.

Description of input parameters : Evenement

Evenement = 0 : passage of the end temperature threshold upwards

Evenement = 1 : passage of the end temperature threshold downwards

Evenement = 2 : passage of the heat transfer fluid temperature threshold downwards

Evenement = 3 : passage of the heat transfer fluid temperature threshold upwards

Evenement = 4 : Flame detected

Evenement = 5 : No flame detection

```
Case Etat = 0 /*No energy emissions*/
  Case Evenement = 0 /*passage of the temperature threshold upwards*/
    STA = 1 /*Update: température extrémité >= température demandée*/
    Return
Case Evenement = 1 /*passage of the temperature threshold downwards*/
  STA = 0 /*Update : température extrémité < température demandée*/
  Case STC = 0 /*heat transfer temperature <maximum temperature - differential*/
    Call I02_ControleEmissionEnergie (OnOff=1) /*Fuel emission*/
    Case ModeChauffage = 0 /*Premises heating*/
      Call I06_Circulateur(OnOff = 1) /*starting local heating circulator*/
      CallI002_DuréeChauffage(MarcheArret = 1) /*accounting of space heating time*/
    Case ModeChauffage = 1 /*DHW heating*/
      Call I12_CirculateurECS(OnOff = 1) /*DHW circulator start-up*/
    Call G02_MiseEnRouteAllumage
    Etat = 1 /*Energy emitted + uncontrolled flame*/
    Phase = 0 /*Energy emission phase before ignition*/
    Compteur = 0
    Return
Case Evenement = 2 /*Passage of the heat transfer fluid temperature threshold downwards*/
  STC = 0 /*Update : température caloporteur <température maximale - différentiel*/
  Case STA= 0 /*température extrémité < température demandée*/
    Call I02_ControleEmissionEnergie (OnOff=1) /*Fuel emission*/
   Case ModeChauffage = 0 /*Premises heating*/
      Call I06_Circulateur(OnOff = 1) /*circulator start-up*/
      CallI002_DuréeChauffage(MarcheArret = 1) /*accounting of premises heating time*/
   Case ModeChauffage = 1 /*DHW heating**/
```

```
            Call I12_CirculateurECS(OnOff = 1) /*circulator start-up*/
        Call G02_MiseEnRouteAllumage
        Etat = 1 /*Energy emitted + uncontrolled flame*/
        Phase = 0 /*Energy emission phase before ignition*/
        Compteur = 0
        Return
    Case Evenement = 3 /*Passage of the heat transfer fluid temperature threshold upwards*/
        STC = 1 /*update : heat transfer fluid temperature >= maximum temperature*/
        Return
    Case Evenement = autre /*Cases 4 and 5+ are inconsistent*/
        Call Audit(C01, Etat = 0, Evenement : x) /*For the record: call of an audit program following inconsistency with transmission of input values*/
        Return

Case Etat = 1 /*energy emitted + flame control in progress*/
    Case Evenement = 0
        Return
    Case Evenement = 1
        Call I02_ControleEmissionEnergie (OnOff=0)
        Case ModeChauffage = 0 /*Premises heating*/
            Call I06_Circulateur(OnOff = 0) /*circulator stop*/
            CallI002_DuréeChauffage(MarcheArret = 0) /*accounting of space heating time*/
        Case ModeChauffage = 1 /*DHW heating*/
            Call I12_CirculateurECS(OnOff = 0) /*circulator stop*/
        Call G03_ArretProcedureAllumage
        STA = 1 /*end temperature >= requested temperature*/
        Etat = 0
        Return
```



```
Case Evenement = 2
    Return
Case Evenement = 3
    Call I02_ControleEmissionEnergie (OnOff=0)
    Case ModeChauffage = 0 /*Premises heating*/
        Call I06_Circulateur(OnOff = 0) /*circulator stop*/
        CallI002_DuréeChauffage(MarcheArret = 0) /*accounting of space heating time*/
    Case ModeChauffage = 1 /*DHW heating*/
        Call I12_CirculateurECS(OnOff = 0) /*circulator stop*/
    Call G03_ArretProcedureAllumage
    STC = 1 /*heat transfer fluid temperature >= maximum temperature*/
    Etat = 0
    Return
Case Evenement = 4 /*Flame détected*/
    Etat = 2
    Return
Case Evenement = 5 /*No flame*/
    Call I02_ControleEmissionEnergie (OnOff=0)
    Case ModeChauffage = 0 /*Premises heating*/
        Call I06_Circulateur(OnOff = 0) /*circulator stop*/
        CallI002_DuréeChauffage(MarcheArret = 0) /*accounting for space heating duration*/
    Case ModeChauffage = 1 /*DHW heating*/
        Call I12_CirculateurECS(OnOff = 0) /*circulator stop*/
    Call G03_ArretProcedureAllumage
    Etat = 3 /*Mistake*/
    Call Alarme /*The called program, not detailed here, generates the display of information indicating the problem. Exiting state 3 can only be done following a manual command.*/
    Return
```

Case Etat = 2 /*controlled flame*/

Case Evenement = 0

Call I02_ControleEmissionEnergie (OnOff=0)

Case ModeChauffage = 0 /*Premises heating*/

Call I06_Circulateur(OnOff = 0) /*circulator stop*/

CallO02_DuréeChauffage(MarcheArret = 0) /*accounting for space heating duration*/

Case ModeChauffage = 1 /*DHW heating*/

Call I12_CirculateurECS(OnOff = 0) /*circulator stop*/

Call G03_ArretProcedureAllumage /*Necessary to deactivate regular flame control*/

STA = 1 /*end temperature >= requested temperature*/

Etat = 0

Return

Case Evenement = 1

Return

Case Evenement = 2

STC = 0 /*heat transfer fluid temperature < maximum temperature minus differential*/

Return

Case Evenement = 3

Call I02_ControleEmissionEnergie (OnOff=0)

Case ModeChauffage = 0 /*chauffage des locaux*/

Call I06_Circulateur(OnOff = 0) /*circulator stop*/

CallO02_DuréeChauffage(MarcheArret = 0) /*caccounting for space heating duration*/

Case ModeChauffage = 1 /*chauffage ECS*/

Call I12_CirculateurECS(OnOff = 0) /*circulator stop*/

Call G03_ArretProcedureAllumage /*Nécessaire pour désactiver le contrôle régulier de la flamme*/

STC = 1 /*heat transfer fluid temperature >= maximum temperature*/

Etat = 0

Return

Case Evenement = 4 Évènement non attendu normalement

Call Audit(C01, Etat = 2, Evenement : 4): /*Pour mémoire : appel d'un programme d'audit suite à incohérence avec transmission des valeurs d'entrée*/.

Return

Case Evenement = 5 /*Arrêt intempestif de la combustion*/

Call I02_ControleEmissionEnergie (OnOff=0)

Case ModeChauffage = 0 /*Premises heating*/

Call I06_Circulateur(OnOff = 0) /*circulator stop*/

CallO02_DuréeChauffage(MarcheArret = 0) /*accounting for space heating duration*/

Case ModeChauffage = 1 /*DHW heating*/

Call I12_CirculateurECS(OnOff = 0) /*circulator stop*/

Call G03_ArretProcedureAllumage /*Necessary to deactivate regular flame control*/

Etat = 3

Return

Case Etat =3 /*Mistake*/

Return /*no treatment whatever the event*/

C02_ModeChauffage

CALLED BY : DEV, IEN, OBC

RETURNED PARAMETERS : ModeChauffage (0 : Premises heating; 1 : DHW heating)

This entry point is used to determine the heating mode. DHW heating has priority

Description of input parameters : none

Call I11_TemperatureDemandeeECS

Call I12_Temperature ECS

```
Case Temperature ECS =< TemperatureDemandeeECS - VarECS
  Case ModeChauffageAnterieur =0 /*Premises heating in progress*/
    Call C01_Traitement (Evenement = 0) /*Local heating shutdown if necessary*/
    ModeChauffage = 1
    Return
Case Temperature ECS >= TemperatureDemandeeECS
  Case ModeChauffageAnterieur =1 /*DHW heating in progress*/
    Call C01_Traitement (Evenement = 0) /*DHW heating stop*/
    ModeChauffage = 0
    Return
Return
```

TIME MANAGEMENT MODULE : GET

This module is not affected by the addition of DHW heating.

OBSERVATION/COUNTING MODULE : OBC

In this module, we will add the meter relating to domestic hot water heating. The additions are to be made at the heart of the program.

O01_DureeEmissionEnergie (MarcheArret)

CALLED BY : IEN

RETURNED PARAMETERS : None

This entry point is used to update the different counters. We do not take into account the case where the maximum temperature of the heat transfer fluid is never reached because this would then be an undersizing of the necessary heating power.

Description of input parameters : Start of energy emission : MarcheArret = 1, Stopping energy emission : MarcheArret = 0


```
Case MarcheArret = 1 /*Start of energy emission*/
  Call G04_LectureHorloge
  Call C02_ModeChauffage /*premises or domestic hot water heating */
  Case ModeChauffage = 0 /*Premises heating*/
    Write HeureMiseEnMarche /*save start time*/
    DureeArret = HeureMiseEnMarche - HeureArret
    Case DureeArret >= DureeOptimale /*DureeOptimale is a construction parameter*/
      Call I09_AjustementTempCaloporteur(Ajustement = 0)
    Case DureeArret < DureeOptimale /*DureeOptimale is a construction parameter*/
      Call I09_AjustementTempCaloporteur(Ajustement = 1)
  Case ModeChauffage = 1 /*DHW heating*/
    Write HeureMiseEnMarcheECS /*save start time DHW heating*/
    DureeArret = HeureMiseEnMarcheECS - HeureArretECS
Case MarcheArret = 0 /*Energy emission stop*/
  Call G04_LectureHorloge
  Call C02_ModeChauffage /*premises or domestic hot water heating*/
  Case ModeChauffage = 0 /*Premises heating*/
    Write HeureArret
    DuréeMarche = HeureArret - HeureMiseEnMarche
     DureeTotaleMarche = DureeTotaleMarche + DureeMarche
  Case ModeChauffage = 1 /*Chauffage ECS*/
     Write HeureArret
     DuréeMarcheECS = HeureArretECS - HeureMiseEnMarcheECS
     DureeTotaleMarcheECS = DureeTotaleMarcheECS + DureeMarcheECS
Return
```

CONCLUSION

This example should have allowed you to get a first idea of the advantages of "Software Algebraic Structuring". Additional structures that would have taken too long to detail make it possible to simply manage the execution of tasks according to their recurrence, their priority, to hide and unmask them, to distribute them over time so that there are no elementary cycles overloaded while others are empty. There was no need for it in this relatively simple example, but this is no longer the case as soon as the number of tasks and their respective constraints exceed a certain threshold.

We must also not forget that, in the same way as for value analysis, the algebraic structuring of software requires support which initially remains irreplaceable. Even the seasoned analyst will go wrong from time to time, but experience will allow him to get back on track without it taking a lot of time.