Alternative configured objects towards optimal control using ABB 800xa

Yousef Iskandarani, Hamid Reza Karimi, Erik Andre Nerhus, Bjornar Haeras and Henrik Johan Ask
Department of Engineering
University of Agder
Grimstad, Norway
Email: {yousef.iskandarani, Hamid.R.Karimi}@uia.no

Abstract—Object Oriented Programming (OOP) is a programming paradigm which be highlighted using the 800xa system. This work will investigate the use of OOP programming for designing industrial application control based on a soft controller. The different types of objects which can be defined in different classes, declaration and implementation using the 800xa will be explored, thus highlighting the use of static, dynamic and embedded objects.

In the next phase, an illustration on how to map and use these objects in ABB 800xA for efficient control, hence, providing a systematic approach for accessing any of the plant aspects using a unique plant object.

Keywords: Object Oriented Programming, Dynamic object, Static Object, Embedded Object, plant mapping, 800xa system, Industrial IT

I. INTRODUCTION

In time, the control system required to run large plants and applications has only grown more complex and time consuming to make. Users expect and demand more functionality from automation systems than ever before. The number of I/O’s are increasing in more complex and large plants. The concepts of modularity and re-usability are the basis of the Object-Oriented approach [5].

Object-oriented programming is a way to manage the complexity of a control system and concentrate the focus on specific tasks. Object-oriented programming is based on objects that represent a part in the control system. The part can be all from a tank to a sensor inside the tank. When using objects, the structure of the program used to control the plant, will be much more understandable for an engineer. They can focus more on the engineering part and not do the job of a programmer. The ABB 800xa Industrial It system is used in as the OOP industrial IT study case, to navigate through the programs, HMI, controllers and all the applications which this system offers will be introduced.

SYSTEM 800XA OVERVIEW

The industrial IT System 800XA gives the engineers and operators an excellent overview over the plant. The declared objects represents real objects, such as tanks pumps and valves. Objects can represent a whole production line if wanted.

Process graphics and logs can be assigned to the objects in the system, to get an even greater overview of the whole prose. The objects can be placed in different structures. The structures are used to navigate through the different parts of the project depending on the desirable job "control properties, HMI, Library, historical data, etc..". The structures are located in my ePlant as shown in Figure [1].

In the following paper, two main different types of structures will be highlighted thoroughly, the structures are:

Control Structure: Shows the base of the program. Here the engineers build libraries and objects or use the already declared libraries and objects which can be found in the libraries and objects structure respectively, moreover, the program control code is setup here using single control module and control modules. Afterward all connections to hardware is distributed from this structure.

Functional Structure: Shows the plant from the process point of view. Here are the HMI built up.

There are several of structures to choose from, but this is the only ones we need for this rapport. When we uses this two structures, the data created in one of them, can also being used in the other one. Objects can jump from structure to structure, if we want to use the object in the functional structure, we just calls it up there. Then all the data that we made in the control structure follows it to the functional structure as the controller is setup here.
II. OBJECT TYPES CONCEPTUALIZATION

The structure pyramid as shown in Figure 2 is used to illustrate the hierarchy mapping of object oriented programming. It is noted that the whole program which include the object, control modules and libraries is included in the application. In most of the control projects as the one used for this case study (controlling the candle making process), the main object will naturally be the plant. From there the programmer can move down one step at the time until reaching the bottom layer, which only contains aspects. The point of mapping it this way, is so facilitate the backtrack of the logical steps and be able to trace the path from top to bottom and vice versa. For simple example as demonstrated by the pyramid shows the flow from the Plant $\rightarrow$ Static Object $\rightarrow$ Static Object $\rightarrow$ Aspects.

![Fig. 2. The OOP pyramid based approach show the different possible layers of objects](image)

Hierarchy as shown in Figure [3] is the way to simplify how the control sequence should be executed, with one logic having a higher priority than another, and a possibility of enabling different logics to start and stopping another function as can be done using a certain expressions

![Fig. 3. Hierarchy mapping enables accessing any aspect in the control logic through a single plant object](image)

This is an example of a typical plant structure. A normal pointer in the program would for example be Candle_presse.Temp_sens2.active as shown in Figure [4]. When doing it this way it is easy to backtrack the location of the aspect in the process. Therefore, making it easier to locate an error if that would arise.

![Fig. 4. 800xA mapping](image)

III. STATIC OBJECTS

Static objects as shown in Figure [5] contain only aspects, which are the lowest building blocks in the object oriented (oo) program as already introduced in Figure [2]. Every aspect represent a property of the object. Static objects have no boundaries, therefore it can be sufficient for components that do not need any form of control or may go out of control.

![Fig. 5. static object structure](image)

It is important to implement all the aspects that is needed in the basic object when it is declared. The main reason is the fact the the same object will be used several times by just calling it using the already declared object structure data type which is made for this object, hence, specifying exactly which sensor to be reached for example. Aspects can make their data available as a set of named properties with values of simple types (String, Real, Boolean, etc).

This can for example be a temperature sensor as shown in Figure [6]. A temperature sensor will only measure a temperature, therefore it will only need few aspects to have the control over the readout.

![Fig. 6. Temperature sensor structure](image)

Since there is no expressions used in a static objects, all of the variables can be changed independently of each other. That means that if someone need to control a motor, the motor can be turned on and off at the same time. In most cases that will not be a sufficient solution, so static objects will only be used where the control task has no limitations.

Here the temperature sensor is a static object, it has only one purpose, to read the temperature. The level sensors is also a static object who will give a signal when the level is reached, no signal when not.
IV. DYNAMIC OBJECTS

The process of modeling a complex plant can be supported and simplified by employing standard modules in the used library. Each standard module represents the dynamic behavior of a real world object [7]. The dynamic objects as shown in Figure [7] is controlled by functions 'expressions' characterizing the relations between its aspects unlike for static objects where aspects act independent of each other.

Here, the parameters are controlled by the function, structured text is used often for defining this function. For example as shown in Figure [8], If the S5 block gets a signal, it will execute the function, set the parameter and go to the next step.

In the auto mode, it is a requirement to measure the fluid flow which is getting pass the pump, so the flow is basically the same flow we are using in the manual mode but it is linked to the same data type in order to get the same measuring with the same tool of hardware.

Moreover, a timer function block is used to define how long the pump is going to run. Under the timer function a Boolean data type called level sensor. This in hardware is a sensor that gives a signal when the tank is full (level sensor), so the pump will stop even if the timer is still running.

V. EMBEDDED OBJECTS

The object is defined as embedded object as shown in figure [10] when one or more objects are nested in an object. The top level in our hierarchy mapping 3, the plant or application is also an embedded object, containing all of the other objects in the control project. An embedded object serves a certain purpose to the control system. Which means that a valve is not an embedded object unless it has some another object in it, for example a temp sensor. Now the valve is an embedded object, because the temp sensor is a object that has been embedded to the valve.

VI. LIBRARIES IN 800XA

Libraries add functionality as well as utilize the use of objects when building a control project. A library may contain
functions, data types, function block types and control module types. The programmer can define his/her own library. This library will contain different project specifics, like a special tank or a typical valve type which will be used many times in the project.

For example a typical plant that may contain hundreds of valves, and it is likely that they are all of the same type. Then the programmer can define one valve type which can be reused whenever one is required in the program a new instance of that type is created.

There are several standards that come with the ABB package. For example the BasicLib. This library contains basic data types and function blocks. This library has definitions for many useful functions like, counters, timers, time and date. These pre-defined standards can be used to speed up the progress of the project. The programmer can also access other libraries, like control libraries. In these control libraries there are functions like PID loops, lead, lag and other control functions. When using libraries like that, the delivery time of control systems decrease significantly.

VII. OBJECT DECLARATION USING 800XA

The first window that we are opening in ABB is my ePlant. From there we navigate us through the different structures until we get to the control structure. Here we have an overview of all programs within the control network. We go into one of the main areas of control structure, which is the control properties. There is the main program we use to build up the control, which is the control builder. There are three main parts in the control builder, libraries, applications and controllers. It is in the library we are providing standard objects that we need in the process. We choose from the functionality of the object if we need a static or dynamic module.

The static objects that we will create is built up like a datatype. A new datatype is created with all the aspects in it, and we call the datatype for example temp fig[11]. And then another datatype is made. This datatype contains all of the type of components that we have in the process. It is also an embedded object, cause of several static objects in another object. All the components in this object will have the datatype temp_sens.

When you are making a dynamic object in ABB 800xa, you start off by making a data type. In this example we are making a pump, as shown above. In the data type you define the aspects needed to run a pump, like active and inactive. You then make a Control module type for pump_standar, this will define one standard for all pumps you decide to add in the control modules. For this example we only needed one pump, pump_standar1.

When you are creating the control module type: pump_standar, you define the parameters you need to run the program, in our case: auto, semi_auto and manual. Instead of making each of these parameters a bool, you connect them to our own data type: pump, giving us the opportunity to call up active and inactive from our library. You are now able to create a function in the program making it unable to run all modes at the same time, thus making the object dynamic.

VIII. ONLINE MODE

When the system is online, the main program (single control module) is often used as a startup function for other dynamic
objects that is running in a parallel sequence. This parallel sequence could be started in a state. This is done by setting a condition to true, that will let the parallel sequence pass the first transition. Now the dynamic object is running its function, while the main program is waiting to pass the next transition. The condition of this transition will be set to true in the end of the program where the dynamic object is running. When the parallel sequence is finished, the main program will then continue to its next step. The main program can also start multiple parallel sequences at once, if we want it to. Or we can choose to have our main program to start a parallel sequence, and continue running itself. We have also the opportunity to let the main program start a parallel sequence, while the main program is waiting for a transition to pass. And then the parallel program could be used to start another parallel program in its program, or when it is finished.

This picture 14 shows the main program in front, it stands in the position where it starts the mixture pump as a parallel process, and it is now waiting for the mixture pump to finish with its process before it goes to the next state. The window behind the main program shows the process of the mixture pump. It is now running with a timer, and it is waiting for a signal to stop, or if the timer goes out before the signal, we will start the alarm sequence. Both of them affects the main program, but in different ways. If the signal comes, the mixture pump will stop, and the main program will continue to the next state, and the alarm will not start. But if the timer goes out and the alarm sequence starts, the main program will just be forced to the state "not started" while the alarm is running its own sequence. Then we have to start the system all over again if we want it to run.

This is how it looks like when we run the program online. All the temperature sensors is in a folder called TEMP_SENSORS. It is the same objects that we created
earlier 11. It is a good way to organize the structure in the program, because all the same components are in the same folder. It is easy to navigate through the different temp sensors and read the temp that is wanted.

CONCLUSION

When using object-oriented programming you get a good and clear structure of the program. Objects makes it possible to provide standardized components. The mapping structure makes it easy for other to understand the program code. Static and dynamic are both equally useful in a program. Static is needed where there are several aspects without any function, while dynamic have built in functions.

ACKNOWLEDGMENT

The authors would like to thank the technical staff in the engineering department at the University of Agder for providing the means which advanced this work. Moreover, special thanks to the Industrial IT 800xa developers for making it possible to implement the object oriented algorithm using the Industrial IT software.

REFERENCES