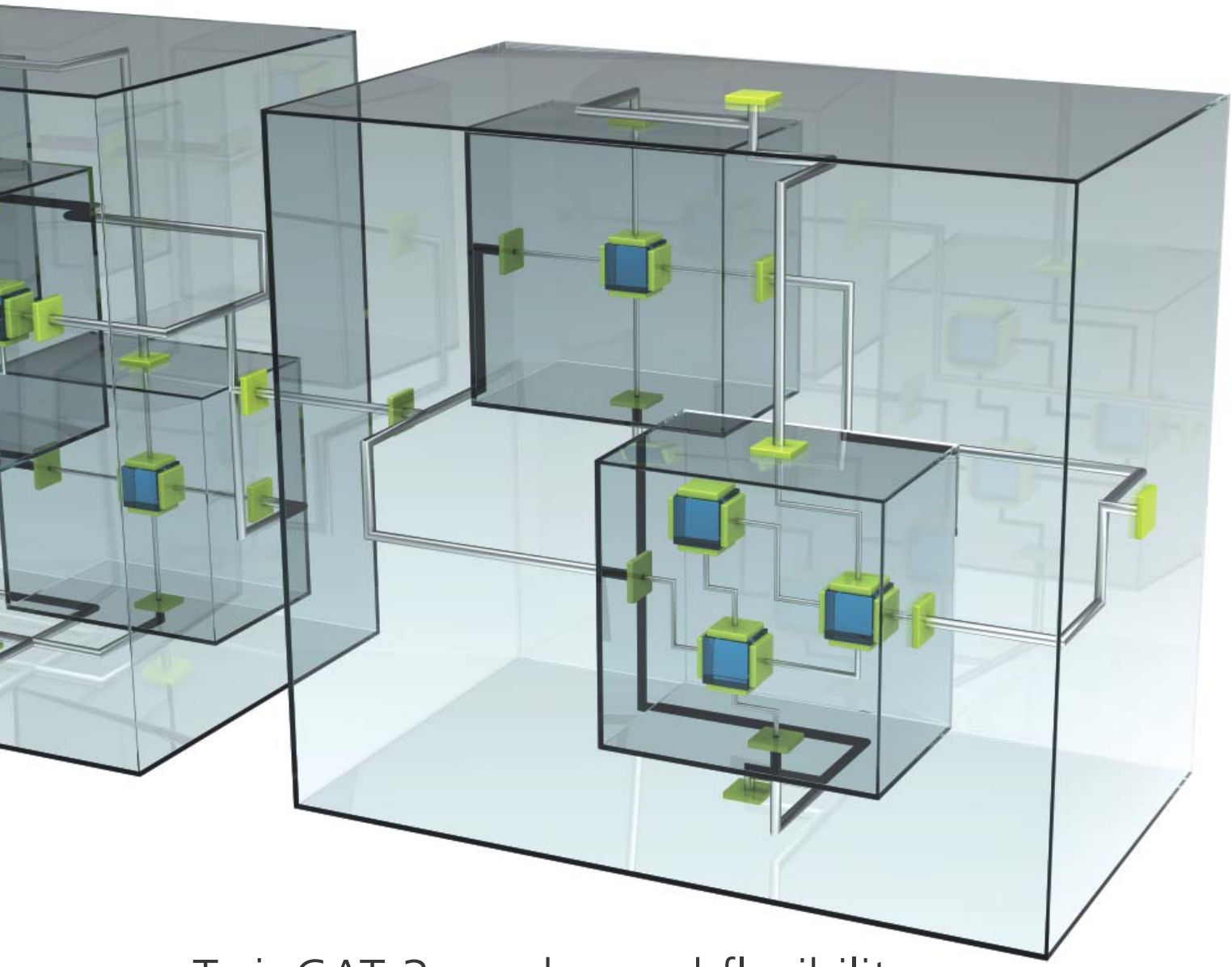
eXtended Automation Engineering (XAE): Modular engineering using the object-oriented IEC 61131-3 extensions



TwinCAT 3 – enhanced flexibility through object-oriented extensions

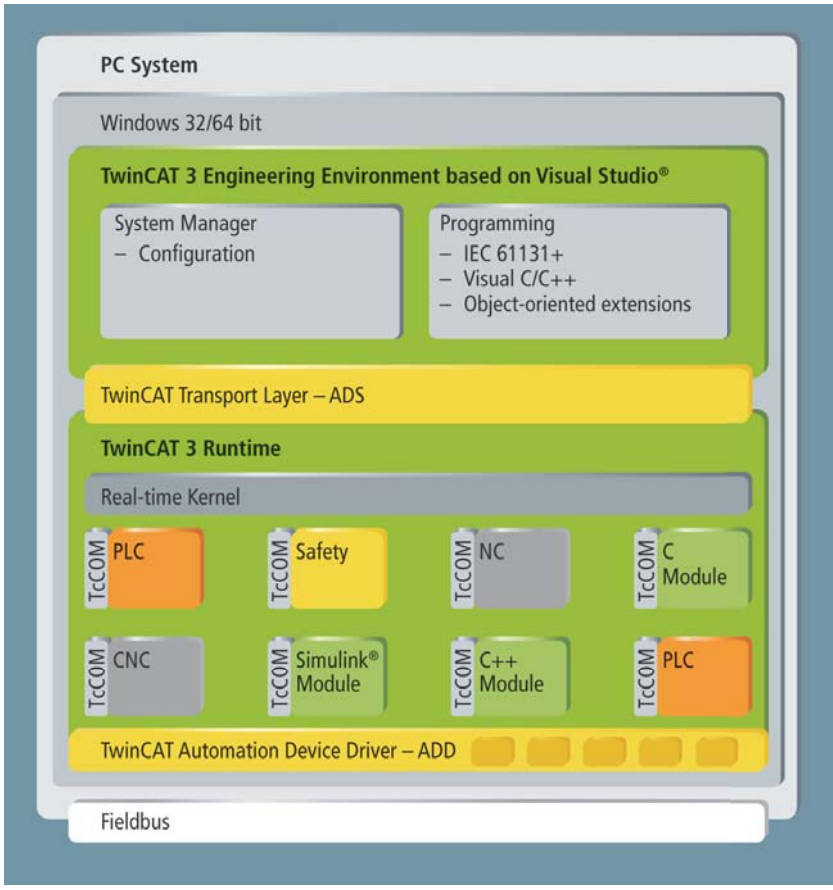
Beckhoff now uses Microsoft Visual Studio® with the new software generation TwinCAT 3. All major programming languages, such as C/C++, a Matlab®/Simulink® connection and the IEC 61131-3 languages, including the object-oriented IEC 61131-3 extensions, are integrated into a single development environment. These enable, among other things, the use of object-oriented techniques such as single inheritance, interfaces, methods and attributes, which significantly increase both the reusability and the quality of the control code.

New methods and tools are required in order to master the constantly increasing requirements for the productivity, flexibility and availability of a plant, while at the same time reducing the costs for the engineering process, commissioning and maintenance of the plant. One way to achieve this is to use a mechatronic approach. The plant is thereby divided into mechatronic modules (objects), which map individual functions of the total process of a plant. If these modules are not only functionally encapsulated, but the interfaces between the modules are also defined, then a kind of building kit is available from which one can utilize for the project engineering of new plants.

The individual modules of the building kit thereby completely map their respective subfunctions. In other words: they cover all facets of a module, from the mechanical, to the electrical, to the control functions, which do not necessarily run on their own controller.

In order to do justice to these object-oriented concepts in control technology, the language used must be extended by these possibilities. However, the goal in doing so cannot be to introduce a new object-oriented language, but rather to make these object-oriented extensions available in the standard languages of control technology. This is where TwinCAT 3 comes into play.

TwinCAT 3 software is the result of consistent development of TwinCAT 2 automation software, which has been well-known for many years through a wide range of applications worldwide. The goal of this further development, besides other important points such as the integration of the Beckhoff PLC Control and System Manager engineering tools into one environment and the use of the globally-recognized Microsoft Visual Studio® 2010 software environment as a framework, was also the adaptation of the TwinCAT software environment to the constantly increasing requirements in the field of machine and plant engineering. In order to, at the same time, do justice to the new module and object-oriented ways of thinking, the IEC 61131-3 language was extended by ten new keywords, whose meaning and application are described below.



TwinCAT 3 – development environment architecture

New keywords in TwinCAT 3

EXTENDS

Using the keyword EXTENDS, function blocks can be derived from other function blocks. The derived function blocks have the same methods and properties as the basic function block, but these operate on the data of the derived function block.

INTERFACE

Interfaces are virtual classes that define methods that must instance objects derived from them, so that they can be compiled without error. They can thus be used in order to define the structure or interfaces of object types. Beyond that, however, they make it possible, by means of pointers to the interface, to directly "address" all function blocks derived from them with precisely these interface pointers.

IMPLEMENTS

The keyword IMPLEMENTS implements an interface in a function block.

METHOD

Methods are kinds of functions that are assigned to a function block. In other words, they are not self-contained POUs (Program Organization Units), but rather a component of the function block and operate on the data of the function block instance that contains them.

PROPERTY

Properties are characteristics that can be assigned to a function block or a program. These properties can each contain a Get and a Set method and they serve as a type of read or write protection

for internal variables. In addition to the actual assignment, it is also possible to implement further instructions in these Get or Set methods in order to make the necessary value available or to pre-process the assigned value.

| **THIS**

With the aid of the This pointer, it is possible to point to the instance of the function block itself.

| **SUPER**

The Super pointer can be used to point to methods of the function block from which inheritance takes place.

| **FB_init/FB_reinit/FB_exit**

Based on constructors and destructors, these methods are called when initializing, copying and exiting the function block instance.

These new keywords are available in all IEC 61131-3 languages. The application of the new keywords will now be presented on the basis of a small example.

A plant in which several kinds of cylinders are used serves as the basis for this example. The task is now to make the control code for this plant so flexible that the actual set sequence of the plant is independent of the type of cylinder used. What all of these cylinder types have in common is that they have two positions (left and right) that can be detected by means of limit switches.

As a solution for this example, it is now defined that, in accordance with the object-oriented concept, the function of each of these cylinders is mapped entirely in a function block. Beyond that, it is defined that all cylinders possess a state machine, which should be supplied with the required position of the cylinder or should return the current position of the cylinder by means of inputs or outputs respectively.

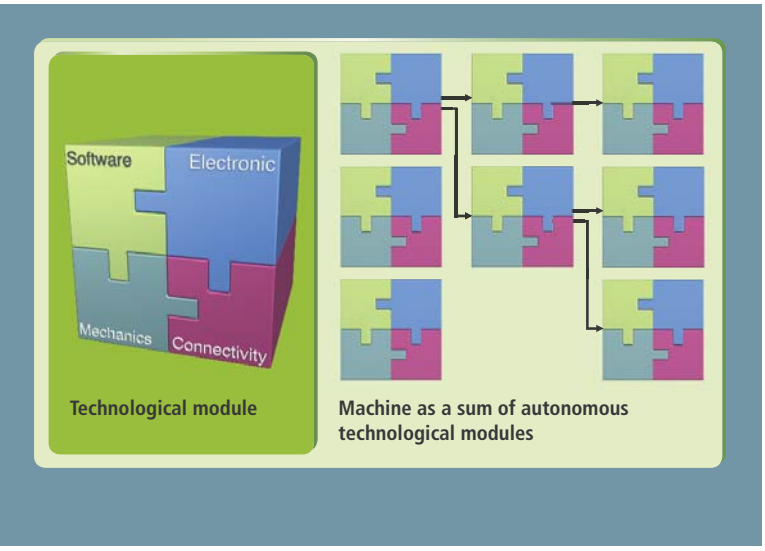


Figure 1: Mechatronic structure of the overall system

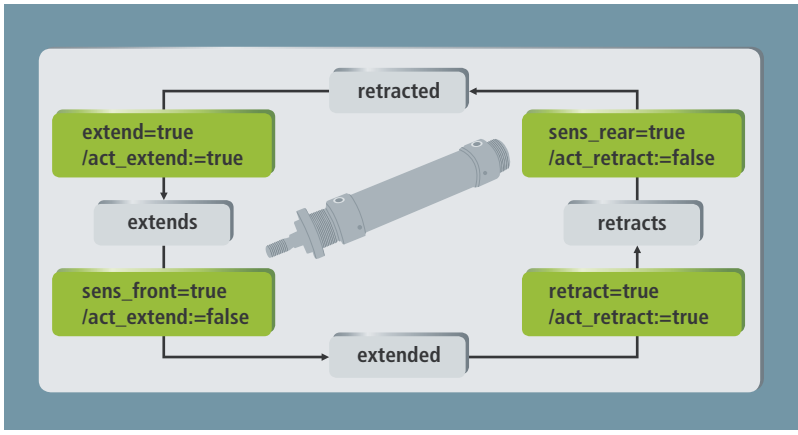


Figure 2: Example cylinder with state machine

Since, according to the task, the respective type of cylinder used in the plant should be flexible, the interface via which the cylinders are addressed must be the same. From this, the first thing to be defined is an interface, iCylinder, which must be implemented by all cylinder function blocks. This interface possesses a method, mStateMachine, which maps the state machine of the cylinders. This method has the inputs bPosLeftReq and bPosRightReq as well as the outputs bActPosLeft and bActPosRight, which represent the signals of the limit position switches. These inputs and outputs are all of the type BOOL (see Figure 3).

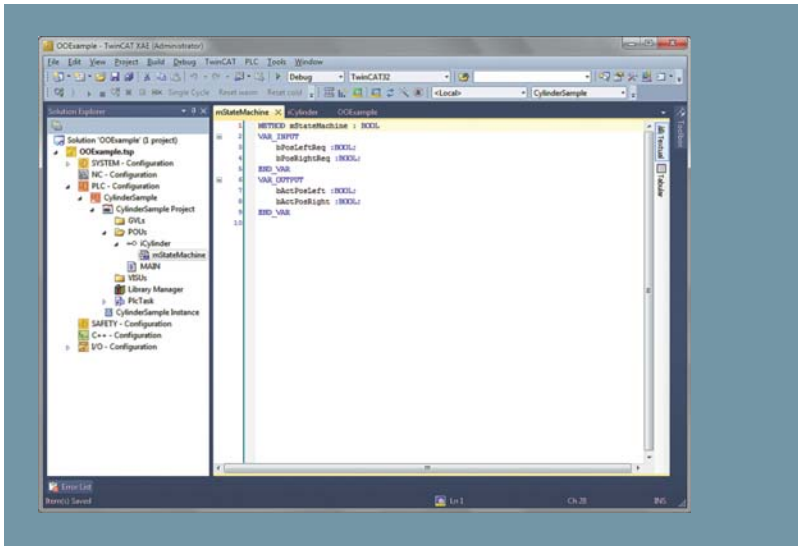


Figure 3: Interface iCylinder with the method mStateMachine

Based on this interface, a basic function block, fbBaseCylinder, can now be created in which the state machine of the cylinder is implemented. If, at the time of creating the function block, one selects that

the interface iCylinder is to be implemented, the method mStateMachine will be created automatically along with all of its inputs and outputs and then only needs to be programmed. The interfaces that a function block implements are defined with the help of the keyword IMPLEMENTS (see Figure 4). The derivation of a function block from several interfaces is thereby possible.

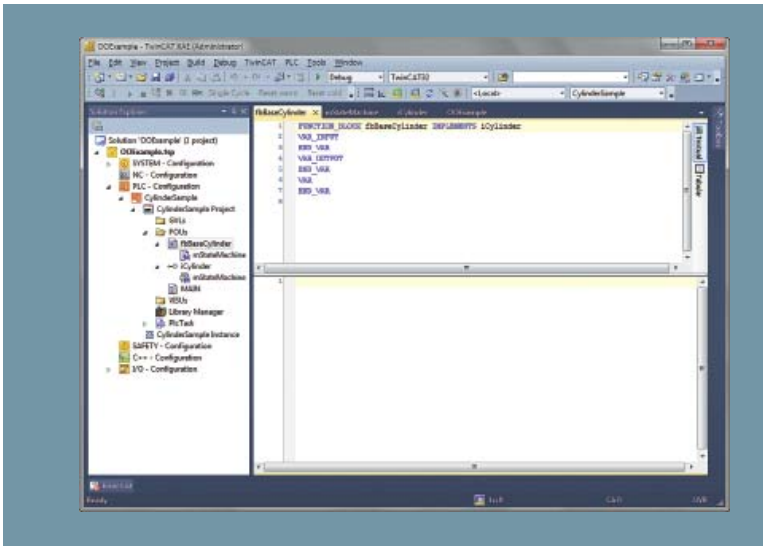


Figure 4: Basic function block of the cylinder

Once the basic function block has been programmed, the function blocks of the different cylinder types can be derived from it. This is done using the keyword EXTENDS.

The derived function blocks now inherit all methods and properties of the basic block. In our example, this means that the function blocks of the derived cylinders inherit the methods of the mStateMachine from the basic function block. The advantage here is that extensions in the method mStateMachine, which apply to all cylinder function blocks, only have to be updated in the basic function block and are then immediately available in all derived function blocks. If the state machine of a derived cylinder exhibits a different behavior to those of the basic function block, it is possible to overwrite this method. This takes place by means of the derived function block implementing a method with the same name as that of the function block of the basic class.

The task in this example was to find a solution in which the set sequence of the plant is independent of the cylinder type that is actually installed in the plant. In order to show that the task is fulfilled by this solution, two function block types, Cylinder_TypA and Cylinder_TypB, are now derived from the basic function block and programmed accordingly. In doing so, additional, partly differing inputs and outputs are added to oth types of function blocks. One instance of each type is created in the program Main (see Figure 5). Since access to these function blocks is to take place via the interface, an interface pointer is also created that points to the interface iCylinder.

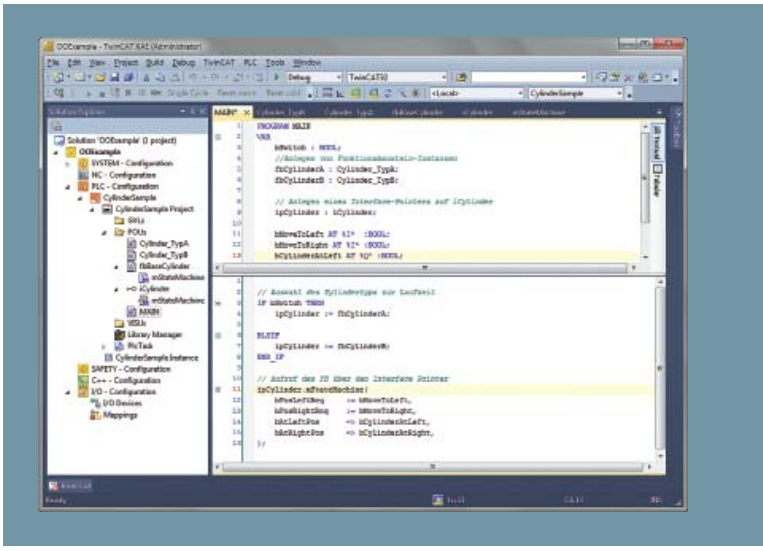


Figure 5: Calling the cylinder function blocks

As can be seen in the illustration, our state machine is actually called via the call of the method mStateMachine of the interface pointer. By assigning a function block instance to the interface pointer, it can now be decided at runtime which method mStateMachine is actually called. This is called 'late linking.'

An additional nice possibility for the use of interfaces is, for example, switching the mode of operation of the modules in a plant. If it is ensured that all modules in a plant (e.g. also our cylinder) implement an interface containing the basic interface for switching the mode of operation, then the switching of the mode of operation of all plant components can be accomplished by means of just one loop via an array of this interface type.

Conclusions

If we now summarize once more what has been shown in this example, it can be seen that the introduction of these new keywords gives rise to new possibilities not only to functionally encapsulate control code, but also to structure it. The goal is therefore to increase not only the reusability of the control code, which leads to considerable savings in time and costs, but also its readability and extendibility. This results in control code of a considerably higher quality.

www.beckhoff.com/TwinCAT3