Implementing Function Block Adapters

Torsten Heverhagen, Rudolf Tracht
University of Essen, Germany
FB 12, Industrial Automation
Torsten.Heverhagen@uni-essen.de, Rudolf.Tracht@uni-essen.de

1. Introduction and Motivation
Programmable Logic Controllers (PLCs) are widely used for controlling industrial manufacturing systems. The programming of PLCs is normally done in special languages defined in the IEC 61131-3 standard [2]. The increasing complexity of the controlling software for manufacturing systems leads to the need for more powerful specification languages. Latest developments in object oriented technology like UML-RealTime (successor of ROOM [4]) face this need [1]. But in most cases it is not possible to completely substitute PLCs in existing plants with object oriented systems. Therefore, our approach is to integrate object oriented technology (UML-RealTime) into an existing PLC-environment in the case of extending a manufacturing system with new components without throwing away the PLC. New components can be for example an Industrial Personal Computer (IPC) which is connected over a fieldbus system to the PLC. We assume, that the IPC program is then designed with UML-RealTime.

In [5] we introduced a new UML stereotype, the Function Block Adapter (FBA), which is responsible for the connection of UML-RealTime capsules and function blocks of the IEC 61131-3 standard. FBAs contain an interface to capsules as well as to function blocks and a description of the mapping between these interfaces. For this description a special FBA-language is provided. The FBA-language is easy to understand both to UML-RealTime and to IEC 61131-3 developers, so they can unambiguously express the interface mapping. An important advantage of the FBA-language is the possibility to use it at an early design state of the UML-RT system.

In this paper we discuss implementation issues of FBAs. We show that different hardware solutions force highly hardware-dependent implementation models. In most cases a FBA is implemented in two programming languages – an object oriented and a IEC 61131-3 programming language. While object oriented programs mostly implement an event-driven execution semantic, PLC programs are executed cyclically. Especially this heterogeneous implementation environment was the motivation for developing Function Block Adapters.

Section 2 gives a short introduction into Function Block Adapters. A general statechart which explains execution semantics of FBAs is given in section 2.3. In section 3 two possible hardware solutions are discussed in principle. A hardware solution with a fieldbus of type Profibus-DP and a PLC of type S7-300 is discussed in section 4. Section 5 closes this paper with a summary.

2. Function Block Adapters
Assuming that there is a PLC on which runs a function block called MyFB like shown in Figure 1.

Furthermore we assume that there is a new application being developed which is designed in UML-RealTime. This application contains a capsule called MyCapsule like shown in Figure 2. MyCapsule has to send and receive the signals of protocol MyProtocol to and from the function block MyFB. Figure 3 shows a possible interaction between an instance of MyCapsule called capsuleInst and an instance of MyFB called myFBInst. At first the signal called sig1 is sent from capsuleInst to myFBInst. Sig1 contains an instance of the data class MyData.
myData.attr1 = 4711;
myData.attr2 = 4712.

Of course the function block MyFB cannot receive the UML-signal without a translation into a FB-signal. The legend which is attached to sig1 in Figure 3 shows the assignments of the function block variables, which are needed to give the information of UML-signal sig1 into the function block MyFB. MyFB reads the values of attr1 and attr2 as a sequence in the input variable A. Input variable B is used to signal MyFB that valid data is assigned to variable A. With the output variable F MyFB acknowledges the inputs of variables A and B.

The second signal sig2 is sent synchronous. This means that the sender (myFBinst) waits for an acknowledgement (sig3). Graphically a asynchronous message is displayed by a single sided arrow and a synchronous message by a double sided arrow.

The data of sig2 is given in the output variable D of MyFB. With output variable E MyFB tells that the assignment of D is valid. In input variable C MyFB awaits the acknowledgement.

The translation of the timing diagrams into UML-signals is done within Function Block Adapters. Sections 2.1 and 2.2 explain the structure and the behavior of the FBA called MyFBA.

2.1 Structure

A FBA is a stereotype of UML which contains all properties of a capsule. A FBA uses ports to establish connections to other capsules.

Additionally FBAs define interface variables for the communication with function blocks. A FBA can graphically displayed in an extended structure diagram (Figure 4). The FBA MyFBA contains a port ~port1 which is connected to port1 of MyCapsule. Interface variables of MyFB which are input
variables like $A$, $B$, and $C$ are output variables of $MyFBA$. Interface variables of $MyFB$ which are output variables like $D$, $E$, and $F$ are input variables of $MyFBA$.

The class symbol of $MyFBA$ is given in Figure 5. The declaration of the interface variables has the same syntax like in IEC 61131-3 for function blocks. Ports are displayed like ports of normal capsules. Connections to other capsules or function blocks are only shown in the extended structure diagram.

2.2 Behavior

The behavior of FBAs describe how the translation between the function block interface and the capsule interface is done. For this a special language is provided – the FBA-Language.

The FBA-Language defines operations which are called when signals arrive from a port or from the Function Block. We distinguish between operations for the translation from UML-Signals to Function-Block-Signals (FB-Signals) and operations for the translation from FB-signals to UML-Signals.

In operations of the first category two functions are needed. $\text{Delay}(\text{time})$ is a function that delays the execution of following commands for the $\text{time}$ given as a parameter. $\text{WaitFor}(\text{bool}, \text{time})$ is a function that delays the execution of following commands until the Boolean expression given as first parameter evaluates from false to true. The second parameter is a timeout, which assures that the FBA is not able to hang up. Additionally to these two functions we only need assignments. In assignments access to properties of the FBA class and used data classes is possible. Properties of UML classes are Attributes, Operations, and AssociationEnds. An example operation for the translation of the UML-Signal $\text{sig1}$ to the FB-Signal specified in Figure 3 is given in Figure 6.

Next we show an operation of the second category for the translation of FB-Signals into UML-Signals.

For operations like this additional functions $\text{SendSync}($port, $\text{send\_signal}, \text{receive\_signal}, \text{timeout})$ and $\text{SendAsync}($port, $\text{send\_signal})$ are needed, which send asynchronous or synchronous messages through ports of the FBA. Furthermore declarations of instances of signals are added which are used in calls of the functions $\text{SendSync}$ and $\text{SendAsync}$. $\text{SendAsync}$ sends an asynchronous message $\text{send\_signal}$ through port $\text{port}$. This asynchronous sending of

<table>
<thead>
<tr>
<th>Figure 5</th>
</tr>
</thead>
</table>

| 1 | ON UML-Signal sig1 PORT port1 |
| 2 | PRECONDITION $F = false$ |
| 3 | BEGIN |
| 4 | $B := false$; |
| 5 | $C := false$; |
| 6 | $A := \text{sig1.data.attr1}$; |
| 7 | $\text{Delay}(2\text{ms})$; |
| 8 | $B := true$; |
| 9 | $\text{WaitFor}( F, 1\text{s})$; |
| 10 | $B := false$; |
| 11 | $A := 0$; |
| 12 | $\text{WaitFor}( F = false, 1\text{s})$; |
| 13 | $A := \text{sig1.data.attr2}$; |
| 14 | $\text{Delay}(2\text{ms})$; |
| 15 | $B := true$; |
| 16 | $\text{WaitFor}( F, 1\text{s})$; |
| 17 | $B := false$; |
| 18 | $A := 0$; |
| 19 | $\text{WaitFor}( F = false, 1\text{s})$; |
| 20 | END |
| 21 | POSTCONDITION $F = false$ |

| Figure 6. Translation operation for sig1 |

| 1 | ON FB_Signal E |
| 2 | PRECONDITION $D <> 0$ |
| 3 | SIGNALS |
| 4 | sig2: MyProtocol.sig2; |
| 5 | sig3: MyProtocol.sig3; |
| 6 | BEGIN |
| 7 | $C := false$; |
| 8 | sig2.data := D; |
| 9 | $\text{SendSync} (~\text{port1}, \text{sig2}, \text{sig3}, 60\text{s})$; |
| 10 | $C := true$; |
| 11 | $\text{Delay}(1\text{ms})$; |
| 12 | $C := false$; |
| 13 | END |
| 14 | POSTCONDITION $E = false$ |

| Figure 7. Translation operation for FB-signal E |
signal `send_signal` takes no time. If `SendSync` is used instead and `receive_signal` is given as an incoming signal of port `port` and a timeout is set, the function at first sends `send_signal` and then waits for `receive_signal`. An example of an operation of the second category is given in Figure 7.

The two operations explained above are typical examples for translation operations of FBAs. All operations consist in their body of the following elements:

- assignments to variables of the associated Function Block
- access to properties of data classes of signals
- calls of the functions
  - `Delay(time)`
  - `WaitFor(bool_expression, timeout)`
  - `SendAsync(port, send_signal)`
  - `SendSync(port, send_signal, receive_signal, timeout)`

The main purpose of the FBA-Language is to give developers of both UML-RT and IEC 61131-3 a common language for the specification of adapters between components of their models. The FBA-Language is not designed to specify behavior of Function Blocks or of capsules. This means that a FBA does not specify what happens after a signal is translated and sent to a capsule or to a Function Block. This is the reason why we left control structures like `IF THEN ELSE` and loops out of the FBA-Language. If an UML-Signal is such complex that the FBA-Language is not sufficient for the translation to FB-Signals, we prefer to redesign the UML-RT interface instead of extending the language. The reason for this is, that the UML-RT system is applied to an existing system. The UML-RT developer should try to keep his design as conform as possible to the design of the existing system.

2.3 A General Statechart for FBAs

The statechart of Figure 8 explains how a FBA is executed. If nothing has to be translated the FBA is in state `Idle`. In this state the transition `synchronize` is periodically fired. Within this transition the interface variables are synchronized with the function block variables and for a FB-signal evaluated. If the FBA is implemented in two parts (section 3) the evaluation of function block variables can be done at the PLC-part within the FBA.

If a FB-signal has occurred, a special UML-signal is generated which triggers one the transitions `FBSignal1` to `FBSignal<m>`. The transitions `UMLSignal1` to `UMLSignal<n>` are triggered by UML-signals of the interface ports of the FBA. Because the translation operations of section 2.2 contain wait states, the processing of this translations is done in states and not within transitions.

The statechart of Figure 8 shows, that only one translation operation can be processed at the same time.

3. Hardware Solutions

When implementing a FBA the following points have to be considered:

a) How are the Function Block variables with the FBA variables synchronized?
b) How are the translation operations of FBAs invoked?
   The problem here is the invocation of the translation operations of FB-Signal. UML-Signals are
   triggers for transition of FBAs. To this transitions the necessary operations for the translation of
   UML-Signals can be added.

c) How are the functions Delay, WaitFor, SendAsync, and SendSync implemented?
   Answers to this questions depend very on the hardware connecting the PLC and the IPC. There is no
   standard way of connecting a PLC and an IPC. Some general examples of doing this are the following:

3.1 Hardware solution 1
   If the PLC interface is very simple, then digital inputs and outputs are sufficient. In most cases the IPC
   must be extended with a digital I/O card. In this solution the FBA is implemented completely at the
   IPC.

![Figure 9. Hardware solution 1: The FBA is only at the IPC](image1)

**About a)** How are the Function Block variables with the FBA variables synchronized?

The Function Block variables can be read and written with the digital I/O card. This can be done by
polling or by interrupt techniques.

**About b)** How are the translation operations of FBAs invoked?

Every time a polling function or an interrupt function was invoked, the Boolean expressions of the FB-
Signals must be evaluated. If a FB-Signal becomes true, the associated translation operation is
invoked.

**About c)** How are the functions Delay, WaitFor, SendAsync, and SendSync implemented?

All functions are implemented and used in the same programming language and environment within
the IPC.

3.2 Hardware solution 2
   The PLC uses some kind of serial communication over an industrial fieldbus or simply a serial
   interface like for example RS232 to communicate with an IPC. The IPC uses its existing serial
   interface or must be extended with a fieldbus interface. The implementation of the FBA then consists
   of two parts. One part resides at the IPC and the other part at the PLC. Between the two parts a
   communication protocol must be established within the FBA.

![Figure 10. Hardware solution 2: The FBA is distributed over PLC and IPC](image2)
About a) How are the Function Block variables with the FBA variables synchronized?

The FBA variables which must be synchronized with Function Block variables reside at the PLC part of the FBA. So the FBA can access the Function Block directly.

About b) How are the translation operations of FBAs invoked?

The part of the FBA which is implemented in the PLC is executed in every cycle of the PLC. Each cycle the Boolean expressions of the FB-Signals are evaluated. If a signal becomes true, a message containing all necessary information is sent to the IPC.

Also every cycle the FBA part of the PLC must check if the FBA part of the IPC wishes to send a message.

About c) How are the functions Delay, WaitFor, SendAsync, and SendSync implemented?

Delay and WaitFor are implemented completely at the PLC part of the FBA in IEC 61131-3 languages. SendAsync and SendSync are implemented in the IPC part of the FBA.

4. Example for the Fieldbus Profibus-DP and the PLC S7-300

In this section we discuss the hardware solution of section 3.2 realized by a fieldbus of type Profibus-DP and a PLC of type S7-315-DP. The communication between the IPC and the PLC is done over the Profibus-DP. For this the IPC uses a communication processor (CP) called Profibus-CP 5412. The PLC of type CPU315-DP already contains a Profibus-CP.

Like mentioned above, the FBA is implemented in two parts – the capsule part resides at the IPC and the function block part (FB-part) resides at the PLC.

4.1 How are the Function Block variables with the FBA variables synchronized?

Because both the IPC and the PLC are active nodes we need a master-master protocol for communicating over the Profibus. A suitable fieldbus protocol is the FDL (Field Data Link) protocol [6].

At the IPC the FDL programming interface is provided by a C library with function calls like SCP_send and SCP_receive. At the PLC the two functions AG_SEND and AG_RECV are used for FDL-connections. With the FDL-protocol messages can be received asynchronous and synchronous. The configuration, initialization, and parameter setting of FDL-connections is out of the range of this paper.

As mentioned in section 2.3 the synchronize action is implemented in two parts. The C++ code fragment in Figure 12 belongs to the capsule-part. Each time the transition synchronize of Figure 8 is fired, it calls the function SCP_receive to check if the function block part of the FBA wants to send a message with the call of AG_SEND. The FB-part of the FBA checks (call of function AG_RECV) in every PLC cycle if the capsule part of the FBA wants to send a message with the call of SCP_send. The next section explains this aspect again but in more detail.

4.2 How are the translation operations of FBAs invoked?

The translation operations of UML-signals are invoked by the UML-signals itself.

The FB-signals at first have to be recognized. Then the data of the FB-signal is transferred to the capsule part by an FDL-connection. At the capsule part an internal UML-signal is generated which triggers the transition which is responsible for the FB-signal (section 2.3).

The recognition of the FB-signal is done within the FB-part. The key mechanism is the edge recognition of Boolean expressions. In our example of Figure 7 the Boolean expression consists only of the variable E. For edge recognition a function block called R_TRIG is provided in [2]. A code
fragment of the FB-part of the FBA written in Structured Text [2] is given in Figure 11. A code fragment of the capsule part of the FBA written in C++ is given in Figure 12.

```
(1) FUNCTION_BLOCK MyFBA
(2) VAR
(3)   my_trig: R_TRIG;
(4) END_VAR
(5) ...
(6) my_trig( E );
(7) IF my_trig.Q THEN
(8) ...
(9) AG_SEND( ... D ... );
(10) ...
(11) END_IF
(12) ...
(13) END_FUNCTION_BLOCK
```

This code fragment shows the implementation of the action attached to the synchronize transition of Figure 8. If the function SCP_receive() returns true, an internal message is sent which fires transition FBSignal1 of Figure 8.

For the explanation of Figure 11 we outline the execution behavior of a PLC in Figure 13. PLC functions are executed in every PLC cycle. At the beginning of a cycle the input variables are red. Line (6) of Figure 11 evaluates in every cycle, if the value of E has changed from false to true. This happens in cycle 6 of Figure 13. Only in this cycle the output variable my_trig.Q is true. Then the function AG_SEND gives the data of variable D to the Profibus system which sends the data over a FDL connection to the Profibus-CP of the IPC.

The time for sending of FDL messages can be greater than the cycle time of a PLC. Furthermore the time interval in which the synchronize action is fired, is in most cases greater than the cycle time of the PLC. This must be considered when FBAs are implemented.

During the design of FBAs we don’t need to think about these different cycle times, because a continuous time model is considered. This is a great advantage of FBAs.

4.3 How are the functions Delay, WaitFor, SendAsync, and SendSync implemented?

Delay
For time delays a special function block called TON is provided in [2]. (Within the S7-SCL this function block is called S_ODT)

WaitFor
The implementation of WaitFor is a combination of R_TRIG and TON. The timer TON is used to generate the timeout.

SendAsync and SendSync
Line (4) of Figure 12 is an example implementation of SendAsync with the use of the Rational Rose C++ Realtime Library. For a synchronous message the C++ function Port::invoke() is provided.

5. Summary and Future Work

Within this paper we have shown that with Function Block Adapters the integration of systems designed in UML-RT into an existing PLC environment can be easily specified. The specification of a Function Block Adapter is completely hardware-independent. It describes only the "What" should be done for the integration and not the "How". This aspect is very important because the "How" is highly hardware-dependent.

An approach related to our FBA-Language is proposed in the Statemate Approach [3]. In Statemate reactive mini-specs are used to specify data-driven activities. Data-driven activities are continuously
(cyclic) executed, which is expressed with \textit{TICKs} in a \textit{mini-spec}. Conditions are evaluated in \textit{IF THEN ELSE} statements. In our approach FBA-operations are only executed on associated signal events, which is a different semantic than data-driven activities have. For this reason we introduced the notion of a FB-Signal. Conditions on data-values are evaluated with the \textit{WaitFor} function. The decision if conditions are computed continuously or interrupt-driven is left to the implementation. Whereas data-driven activities are suitable for raw sensor data the FBA-Language is easier to use with IEC 61131-3 Function Blocks. We assume that raw sensor data is computed within a Function Block.

With a specification given in the FBA-Language a developer has an unambiguous description of the requirements for connecting the UML-RT system to the PLC. Because of the simplicity of the FBA-Language both UML-RT developers and IEC 61131-3 developers can understand and validate the specification.

Currently, we are interested in the development of an implementation framework for \textit{Function Block Adapters}. This framework contains

- an integration process,
- class and Function Block libraries,
- design patterns,
- a FBA-Language parser and compiler,
- a simulation environment for validation purposes.

Furthermore we plan to adapt \textit{Function Block Adapters} to IEC 61499. Function Blocks defined in IEC 61499 distinguish between event input and output signals and data input and output signals. These separation would ease our definition of FB-Signals.

\textbf{References}


