Abstract

The application of distributed control systems is one of the main trends in current automation. Here, besides the programming of the control application code and its functional allocation to specific devices, an important point is the implementation of communication related code to exploit the underlying communication platform and realise the collaboration of the distributed control application code.

Due to the real-timeliness and even time-criticality of control applications, the correct handling of real-time constraints has to be considered. This paper presents two approaches for integrating communication related code into distributed control application code.

One integration method is based on the application of AspectJ, an extension for aspect-oriented programming with Java, and the other method is based on ordinary object orientation applying standard design patterns.

This work is part of the research project TORERO, which was funded by the European Commission under the IST program.

1. Introduction

The current situation in automation systems is characterised by a strong trend towards distributed systems based on distributed intelligence and intelligent devices [1]. From the software point of view, a major milestone for this is the international standard IEC 61499 [2,3] which has been finalised in 2005 after a long period of development.

Within industry this control specification and programming standard has enforced new control programming strategies as for example integrated in the PROFinet CBA based iMAP [4] and the TransparentFactory [5] approach to mention only the two main approaches. The intention of this work is to simplify the development of distributed control systems by mainly covering the setup of appropriate communication systems and device interaction strategies.

Within the academic world the IEC 61499 has been adopted as a basis for the consideration of distributed control systems with respect to specification, implementation, and validation. With the Function Block Development Kit (FBDK) [6], the tools around the CORFU approach [7,8], and the Function Block System Designer (FBSD) [9] first control design architectures enabling the application of distributed control systems based on distributed intelligence and distributed decision making are available.

All these approaches have the common modelling paradigm of programming distributed applications by utilizing and interconnecting elementary function blocks residing on a system of devices via a “system editor”. However, a wide variety of solutions to problems such as the specific ways to design distributed control applications independent from the underlying resources (e.g. communication protocol or device hardware characteristics like processing power and memory), to perform optimum allocation, to implement communication related code, to deploy the control application code, to integrate the Internet in the automation system (e.g. for maintenance purposes), and thus the support of the total life cycle of the distributed control system are still under development.

In this respect, the research project TORERO (Total life cycle web-integrated control) funded within the IST initiative of the European Commission has aimed at specifying both an Integrated Development Environment (TORERO IDE) and an architectural platform on specific devices (TORERO Devices) which together allow for the distribution of control applications based on event driven Functions Blocks (FBs) related to IEC 61499 standard [10]. The FBs can be accessed either locally or remotely over the network and the appropriate automation protocol. The development of the control application code (FBs) is independent of the afterwards distribution of this code.

Some of the aspects of distributed systems include component interaction, synchronisation, remote
invocation, parameter transfer strategies and failure handling. Some aspects follow structures which naturally crosscut generalised procedures such as control flow or data flow. For example, synchronisation, real-time constraints, and object interaction follow control flow and parameter-transfer strategies in distributed systems. The work presented in this paper focuses on the usage of aspect-oriented languages and design pattern for integrating communication related code into the distributed application control code.

Within the TORERO approach control application code of a Distributed Control System (DCS) is programmed in Java regarding the Real-Time Specification for Java [11,12] with support by the TORERO IDE. The distribution of control code and thereby of the control application itself will be realised in two steps. Step 1 is associated with the semi-automated allocation of the control application building blocks to single devices, while step 2 covers the implementation of the communication related code necessary for the proper functioning of the distributed control application. Once the control application code is allocated to the single devices, the implementation of communication related code into the allocated control application code will be done automatically by the TORERO IDE by means of either aspect-oriented language AspectJ [13,14] or by applying the Strategy Design Pattern [15] and “ordinary” object oriented design methodologies.

Both approaches will be described in the following sections.

2. Function Blocks based on IEC 61499

The IEC 61499 standard is based on a fundamental module, the Function Block (FB), which represents a functional unit of control software, associated to a hardware resource of a control system. A FB instance is characterized by its type name and instance name, a set of event inputs/outputs, a set of data inputs/outputs, an execution control chart (ECC), internal data, and internal algorithms.

The type and instance name is used to uniquely identify a FB, the event and data inputs and outputs are required for the interconnection of different FB to a FBS, and the ECC, the internal data, and the internal algorithms will describe the internal behavior of the FB.

For the overall behavior of a FBS the ECC and its connection to events, data, and algorithms is of most importance. An ECC consists of states, transitions and actions, which invokes the execution of algorithms, which are associated to the ECC states, in response to event inputs. An incoming event will be read by the ECC and can enforce the invoking of an algorithm in dependence of the active state of the ECC, the possible state changes enforced by the incoming event, and internal bordering conditions of state changes described by transition guards. When the execution of an algorithm is invoked, the needed input and internal data values are read and new values for output and internal data may be calculated. Furthermore, upon completion of execution of an algorithm, the execution control part generates zero or more event outputs as appropriate which can be transmitted to other FB.

By properly connecting more than one FB, a FBS generating a distributed application can be defined. The event flow between ECCs of FBs determines the scheduling and execution of the algorithms and thereby the behavior of the complete FBS based control application.

An application can be distributed among several devices. A device uses the causal relationship specified by the application FBs to determine the appropriate responses to events, which may include communication and process events, utilising the different resources associated to the devices.

3. Aspect Orientation of Control Code

3.1. Introduction

Aspect Oriented Programming (AOP) is a software development paradigm that aims at attaining a higher level of separation of concerns in both functional and non-functional matters by introducing a new modular unit, called aspect. The idea of AOP is to separate the component code from the aspect code. Thus, it will be possible to develop the functional program at first (control application code), and then add specific aspects (e.g. communication related code covering an automation protocol such as Modbus/TCP or EtherNet/IP) at a later point in the development phase of the project. This leads to a resource (hardware) independent programming of the control application code and to an increasing possibility of reusing software components.
The aspect code can consist of several aspect programs, each of which implements a specific aspect (e.g., different automation protocols used in the system). Crosscutting lies at the heart of aspects. Modular units of decomposition are organized into clear hierarchies, whereas aspects crosscut such hierarchies. Join points act as the location where the aspect code is inserted into the component code, where the crosscut is. In general, we can distinguish between three types of join points between aspects: methods and attributes, qualified “by name” references, and references to patterns (see [16] for more information).

The Aspect Weaver is the core component of AOP, which takes the aspect code and the component code, finds join points and weaves them all together to form a single entity, approach presented in Figure 2.

According to the time the code is combined there are two different options for an aspect weave. Static, where the code is combined during compilation time and may not be changed at runtime, and dynamic, where the code is combined at runtime. This is the most flexible variant of AOP since it uses a minimal coupling between aspect code and component code.

To implement AOP in Java, AspectJ was created, consisting of a special aspect language and a static code weaver. This project is supported by different tools e.g. by plug-ins for the Eclipse development tool [13].

3.2. Weaving of communication related code in IEC 61499 FBs

Providing support for an aspect involves two things: implementing abstractions for expressing the aspect and implementing weaving for composing the aspect code with the primary code and the code for other aspects.

There exist three possible ways to implement the aspects: encode the aspect support as a conventional library, design a separate language for the aspect, or design a language extension for the aspect.

Among the three possibilities for capturing aspects, modular language extensions have a number of advantages and thus are preferable than a fixed set of separate languages. Language extensions are more scalable. Plug and unplug is possible, a feature particularly useful, when, during system development, more aspects need to be addressed. Moreover, language extensions allow the reuse of compiler infrastructure and language implementation. Also, one extension can work with many others.

Using AOP to implement the communication related code into the control application code gives the following benefits:

• Generic approach for the communication interface,
• Possibility to use different communication protocols without any changes in the control application code,
• Integrating of local and remote access without adaptation of the control application code,
• Open for extensions, e.g. the integration of a new communication protocol,
• Increase in the level of interoperability and interworkability of a DCS with reference to the communication infrastructure.

As mentioned in the introduction of this paper FBs that establish a control application may interact either local or distributed over the network. In Figure 3, these two different options are depicted. On the first hand, the function block FB 1 resides on Device 1 and triggers locally FB 2 and FB 3 also residing on the same device. On the other hand, the function block FB 1 residing on Device 1 triggers remotely FB 2 and FB 3 residing on Device 2, by means of the communication infrastructure, which could for instance be a network like Ethernet combined with an appropriate automation protocol, e.g. Modbus/TCP, EtherNet/IP, or something else.

As a result, the communication between FBs may vary significantly ranging from local procedure calls to...
the utilisation of services of the industrial communication protocol infrastructure. In the former case the FB has to call the appropriate receive method of the local FB while in the latter case the FB has to initialise the automation protocol such as Modbus/TCP or EtherNet/IP. Thus, it is evident that the communication itself does not present an elementary part of the FB, it is rather “something that can be said about it”, e.g. “the FB communicates over the network” or “the FB calls local methods” illustrating the fact that communication is actually an aspect of the FB.

Applying the methodology of AOP described in the section before, the following steps have to be done to implement the approach for FBs:
- Writing a class body: Implement a class according to the runtime where FBs are executed
- Writing a pointcut for communication initialization: The pointcut has to specify code location to the initialization of FBs (e.g. within constructor) to establish communication links (e.g. TCP connections) if necessary.
- Implement the aspect code for establishing communication link e.g. open sockets for TCP communication
- Writing a pointcut for sending/receiving event/data: The pointcut has to intercept the event sending/receiving method that have to be redirected
- Implement the aspect code for sending/receiving events and data.

3.3. Application example

As an example, a simple network of FBs containing Drive1 and Drive2 as shown in Figure 4 will be considered. Both drives need to be synchronized in a certain way by the application (The intended real-time conditions of the synchronization are not relevant within this paper).

![Figure 4. Simple FB network example](image)

The integration of the communication relevant code within the FB code requires the integration of communication system calls within the event and data handling mechanisms by the event and data queues of the FB implementation classes. This is depicted by the following figure.

![Figure 5. Aspect oriented integration of communication system related code](image)

To describe the required communication parameters such as bandwidth or latency, within Torero the Hard Real-Time Classification (HRC) a-b-g according to the IAONA Real-time classes [20] are used for a qualified description of the event connections within a FB network. The classification is expressed by the term HRC a-b-g whereas a determines the required maximum latency by distinguishing six classes A-G, b determines the required synchronicity by distinguishing five classes A-E and X for no requirements, and g determines the required bandwidth by distinguishing four classes A-E and X for no requirements. As an example the classification HRC A-X-B describes a maximum latency between 100ms and 1s (A), no synchronicity (X), and a bandwidth between 10MB/s and 100MB/s (B).

4. Object Orientation of Control Code

4.1. Introduction

The current implementation of AspectJ has shown a problem regarding the application within Torero. Since it comes with own code and libraries that are added to the project, which is only JDK SE compliant, it was not applicable to the CLDC runtime on embedded devices used with TORERO.

Thus, it became necessary to implement a different approach for such devices which makes use only of common function blocks.

This was the main reason to consider the application of design pattern and the “ordinary” object orientation. With the Strategy Pattern [15] an appropriate design pattern for integrating communication system related control code within a distributed control application has been found.

The Strategy Pattern intends to allow the use of different algorithms depending on the context in which they occur. This is realized by separating the communication system access from the initiation of communication by FBs. For the communication system access integration the different algorithms are the different possible useable communication systems. The
access to the different communication systems is implemented independent from the FBs within an additional class which will be integrated depending on the FB communication system call context.

4.2. Integration of communication related code by IEC 61499 Communication FBs

To enable this integration a feature of the IEC 61499 is applied, the Communication Function Block (CFB). CFBs are used within IEC 61499 to describe explicitly the access to communication systems. If two function blocks have to communicate with each other crossing device borders it is necessary to integrate on each device an additional CFB.

Lets assume that the two function blocks of Figure 4 are allocated to two different devices (TD1 and TD2).

Further on, both devices provide a common communication protocol, say Modbus/TCP. As stated above, CFBs are generated for connecting the two FBs. For every event/data connection that is "disconnect" due to distribution to different devices of the source and the target FB, an event/data input is added to the source CFB and an event/data output is added to the target CFB.

Regarding the above introduced example, Figure 6 shows the connections of the FB after the generation of the CFB.

The former direct connections between Drive1 and Drive2

- `eoi → eil (event)`
- `xout1 → xin2 (data)`

are now replaced by

- `eoi → eoi_Drive1_eil_Drive2 (TD1)` and `eoi_Drive1_eil_Drive2 → eil (TD2)` for event connection and
- `xout1 → xout1_Drive1_xin1_Drive2 (TD1)` and `xout1_Drive1_xin1_Drive2 → xin1 (TD2)` for data connection.

As can be seen in the Figure, Drive1 and Drive2 are no more connected directly in the sense of IEC61499...
connections. Instead, the CFBs act as end points for those connections and handle over the event information (source FB, target FB, data etc.) to a protocol implementation, that translates those information to the underlying protocol (e.g. Modbus/TCP). The information is then decoded on the opposite side of the connection and translated by the target CFB back to IEC61499 connections. This is depicted in Figure 7.

The concept described in this section is applicable to different communication mechanisms such as time-triggered or event triggered. This topic is out of the scope of this paper, a more detailed description can be found in [21].

5. Comparison

The two different approaches described have different benefits and drawbacks.

The AspectJ based version works without additional automatically generated function blocks realizing the necessary communication system access. This is not compliant to the IEC 61499.

But this version is most flexible. It is able to integrate each communication system independent from its specialties of the communication path establishing and use. On the other hand this flexibility requires more implementer skill for the application on the side of the communication system access code provider.

The main drawback of the AspectJ based version is its compliance to the Java Standard Edition only. The within industrial automation on limited devices relevant CLDC compliance is not given. Hence, the application of AspectJ based code is mainly not possible on control devices.

The CFB based version is compliant to the IEC 61499. Its is more intuitive understandable and requires less implementer skills. It is more easy to handle also for small and medium size device vendors which have to provide the necessary communication system access code and it is compliant to CLDC. But in contrast to the AspectJ version it limits the applicability of different communication systems.

CFBs provide a fix interface for the establishing and use of communication paths. Communication systems which can not be handled using this interface can not be integrated in the CFB based approach.

6. Application in the TORERO approach

6.1. Basics

Based on the comparison described above within the TORERO project the decision was taken to apply the CFB based integration of communication code.

As mentioned in the introduction the TORERO project has aimed on developing an integrated methodology for life cycle support of distributed control systems integration the necessary design and maintenance activities within one approach. To support this methodology the TORERO Integrated Development Environment (TIDE), which supports the user during set-up, runtime, maintenance and termination of a distributed control system [17, 18] has been developed.

The TIDE is based on ECLIPSE and consists of a set of ECLIPSE plug-ins allowing an easy exchangeability and extendibility of the components. There are primary plug-ins providing basic functionalities (XML Importer, RDBMS, Plug and Play, and Device Model) and secondary plug-ins supporting all steps of the methodology (Function Block System Developer, Compiler, Java Development Tools, Allocation, Web Services, Weaving, BootP, FTP, HTTP, Security). For an in-depth description of the TORERO IDE please refer to [19].

Using the Function Block System Developer a user is able to design a function block system (FBS) and to distribute this system among a device system. Based on this device system the Weaving building block of the TIDE integrates the necessary TORERO communication function blocks (TCB) using the Strategy Pattern and the context sensitive integration of the right communication system access implementation. The necessary context description is integrated in this process by the allocation of FBs to different devices providing only a limited variety of communication system access possibilities.

6.2. TORERO communication function block implementation

The final implementation of TCB is based on the class CommunicationBlock. This class is responsible for handle incoming and outgoing connections in the sense of IEC61499 and therefore acts as a connector between the event oriented connections of the function block runtime environment and the underlying protocols of the TCB.

Every TCB consists of an automatically generated class derived from CommunicationBlock in collaboration with an implementation of the actual protocol (e.g. Modbus/TCP) based on an interface IProtocolImplementation which has to be provided by the device supplier or system integrator implementing the communication system access on the individual device.

The interface provides three methods that have to be implemented for each protocol. After the creation of the CommunicationBlock, the protocol implementation gets informed about his owner by using the method setOwner. Additionally, the method init of IProtocolImplementation is invoked, which contains all possible targets and sources as parameters. In case of
Modbus/TCP, this method might be used to open TCP connections to all targets to guarantee a faster communication.

The implementation structure within the TIDE is given in Figure 8.

To send an event over the network, CommunicationBlock invokes the method send of IProtocolImplementation. As a parameter, the target is given as an instance of RemoteEvent. In case of sending events the instance contains:
- the name of the target FB
- the address of the target device
- the name of the target event

For the example given in Figure 7, in case an event is sent over the network from Drive1 to Drive2 using TCB_MODBUS on Device1 and TCB_MODBUS on Device2, the RemoteEvent instance passed to the IProtocolImplementation would contain:
- name of the target FB (attribute FB): TCB_MODBUS
- address of the target device (attribute address): 192.168.22.2
- name of the target event (attribute event): eo1_Drive1_ei1_Drive2

As a second parameter for the method send, an array of Data instances will be given containing the name of the data output of the target FB (in the example above xout1_Drive1_xin1_Drive2) and the value as an instance of type ANY.

The encoding of the given parameters is left to the protocol implementation.

A IProtocolImplementation can inform its owner (the CommunicationBlock) about a received event by invoking the owners method receive. Again, as parameters an instance of RemoteEvent and an array of Data instances are given to the method. For the example shown in Figure 7, the FB TCB_MODBUS on Device2 would get the RemoteEvent from its protocol implementation with the following values:
- name of the target FB (attribute FB): TCB_MODBUS
- address of the target device (attribute address): 192.168.22.2
- name of the target event (attribute event): eo1_Drive1_ei1_Drive2

7. Conclusions and outlook

Within this paper two approaches for integration of communication system access within distributed control applications have been described and compared.

It was shown that an integration communication system access code based on aspect oriented technologies as well as an integration based on IEC 61499 compliant communication function blocks is usable. Both version have its own drawbacks and benefits.

It can be stated that the application of aspect oriented technologies is preferable for systems mainly PC and Java Standard Edition based systems since it provide a larger flexibility. For systems based on limited devices the communication function block based version is preferable.

Within the TORERO project the later version has been implemented prototypically. The reached results...
within first test implementations of control systems are satisfactory.

Within further approaches dealing with distributed control systems based in IEC 61499 function block systems the reached implementation will be applied further. Here, especially the combination of IEC 61499 function block systems and mobile agent technology will be considered. In this case the context sensitivity of communication code is not static anymore but becomes dynamic by the agent migration among within a device system.

Acknowledgement

The work presented within this paper was partially supported by the European Commission under the 5th Framework Program within the TORERO project (IST 2001-37573).

References


