Abstract

This paper presents an object-oriented approach for interfacing industrial automation software with technical plants. The paper focuses on strategies for structuring the software in objects in such a way that the structure may be applied even when using different communication systems. A case study, the automation of a modular production system based on an industrial fieldbus is presented.

1 Introduction

Applications in the field of industrial automation are characterized by a high coupling between a computer-based system and a technical process, viz. a chemical plant or a manufacturing process (see fig. 1). Such systems cannot only be described in terms of static relationships between their input and output signals, since the correctness of those systems depends on the time point at which the outputs are produced or at which input signals are read.

Due to this temporal dependency, industrial automation systems belong to the class of the real-time systems [1], [2].

Automation systems are nowadays characterized by a highly decentralized control architecture, being distributed along a wide physical area. In this context, the purpose of a fieldbus is to interconnect the sensors and actuators of the industrial automation system with the controlling computers. They are usually serial busses, oriented to short to medium messages and speed optimized. Most of modern sensors and actuators can support some processing by themselves (because of this they are called intelligent or smart sensors and actuators), enforcing a tendency to encapsulate some processing capabilities within them. That leads to a 'distribution of intelligence' in the automation structure, improving the parallelism and enhancing the flexibility. The approach presented in this paper, which is based on the concept of active objects, is intended for development of such distributed automation systems.

A research project which investigates the application of object-oriented techniques to all stages of the development process of industrial automation systems has been carried on over the last five years in a cooperation between universities in Brazil, Germany and Uruguay. Experience gained in this project has shown that, in order to be robust when faced with changing system requirements, the system structure should be based on elements that are barely presumed to change. In automation projects, this can be achieved if the structuring process takes into account the physical structure of the technical plant under control. Such 'component-based' structuring approach tends to be more stable than one based on system functionality (this topic is discussed elsewhere [3]). This paper focuses on aspects related to the task of interfacing object-oriented automation software with technical plants. It presents

---

Pablo Darscht

darscht@ias.uni-stuttgart.de

Institute for Automation and Software Engineering
University of Stuttgart, Germany

Alceu Heinke Frigeri

alceu@ieee.ufrgs.br

Universidade Federal do Rio Grande do Sul - (DELET-UFRGS)
Porto Alegre, Brazil

Carlos Eduardo Pereira

cpereira@ieee.ufrgs.br

Universidade Federal do Rio Grande do Sul - (DELET-UFRGS)
Porto Alegre, Brazil

---

* on leave from the National University of Uruguay

Fig. 1 Block diagram of the Automation System
strategies for structuring the automation software in objects in such a way that the structure may be applied even when using different communication systems, viz. different fieldbusses. In order to enhance the paper’s comprehension, all the issues will be discussed using examples of one of the case studies we have performed.

The paper is divided as follows: the next section introduces the application used as case study. Section three briefly presents the adopted object-oriented methodology, while section 4 and 5 summarize the results of applying it to develop the automation software and the communication software respectively. Section 6 describes briefly the architectural domain, discussing some implementation decisions concerning the software design policies adopted for this case study. Finally, in section 7, lessons learned from this experience are drawn and directions of our future work are identified.

2 Case Study

In the case study discussed in this paper, the general block diagram presented in Fig. 1 is specialized as follows:

- The technical process to be automated is the so called Modular Production System (MPS). It is an prototype of an industrial system, that encompasses several activities usually present in the practice.
- The computer based system is built of one or more PC-compatible computers, running a POSIX-compliant real-time operating system. The real-time automation software has been implemented in a real-time extension of C++ (described in [4]).
- The Communication System is based on a standardized industrial fieldbus system, the Controller Area Network (CAN) [5,6].

2.1 The Modular Production System

The Modular Production System is a model of an industrial system (produced by FESTO DIDATIC GmbH, Germany) consisting of five manufacturing cells, which are physically connected (see Fig. 2).

Each one of these manufacturing cells processes workpieces (small cylindrical elements made from different raw material and having different properties as weight, color, etc.). Each station stresses a typical functionality usually present in integrated manufacturing systems, such as transport, measurement, drilling, quality control and storage of workpieces.

2.2 The Communication System.

The fieldbus selected in our case study, CAN (ISO 11898 and 115981-1), was originally developed to communicate a set of sensors and actuators in an automotive system by the German company Bosch. Due to its simplicity, performance and robustness, it has gained importance in the field of industrial automation and has been used by several companies as a fieldbus for factory automation.

The standards mentioned above refer to the physical and data link layer of the OSI/ISO reference model. Besides, CiA (‘CAN-in-Automation e.V. international users and manufacturers group) has proposed an application layer protocol called CAL (CAN application layer) [7]. Other proposals for higher level protocols on CAN are DeviceNet from Allen-Bradley and SDS (Smart Distributed Sensors) from Honeywell. However, his high level protocols were not used in this case study. Some of the main characteristics of CAN are:

- carries messages with up to 8 bytes,
- speed optimized: up to 1Mbit/s (distances <= 40m)
- supports linear or star connections

---

**Fig. 2 The technical plant to be automated: modular production system (MPS)**
3 Object-Oriented Automation System Development.

In order to maximize the benefits of adopting an object-oriented approach, it is important to apply a development process that supports the transition from one development stage to the next one in a sound and seamless way. Recursive Design [8] is such an approach. Starting from an informal client 'manifestation of desire' stating the intended system behavior, a set of domains are identified.

A domain is defined as 'an independent real or abstract world populated by separate sets of conceptual entities or objects'. Recursive Design uses four kinds of domains: application (the specific subject matter that concerns the user), server (provide generic functions needed throughout the application), architecture (concerned with system-wide policies for managing data and control) and implementation (programming language and operating system).

A domain may use certain functionality provided by other domains: this client-server relationship is called a bridge between domains. Fig. 3 shows a domain chart, depicting domains and bridges for the modular production system.

For instance the bridge Br3 in Fig. 3 expresses a client-server relationship between the application and the communication domain: the first one assumes that there exists a way to communicate with the technical plant to be controlled, and the second one must provide facilities to allow this communication.

After having separated the problem to be solved into domains, the domains themselves have to be specified. This was carried out using the notation proposed in [9]. The following two sections are concerned with a more detailed description of the application and communication domains.

4 Application Domain: Modular Production System.

4.1 Analysis

The main goal of the analysis process is to give a deeper insight on the problem to be solved. In the case of industrial automation systems, it is necessary to understand the structure of the technical plant being automated as well the behavior of their components. Hence, we start the analysis process with the 'mapping' of the existing (or projected) technical plant to an object-oriented representation. Fig. 4 depicts the information model of an identification station, where one can easily observe a direct correspondence between 'real world objects' and the 'analysis conceptual objects'.

Fig. 5 shows the information model of the modular production system at a very high level of abstraction. An abstract class, station, summarizes common characteristics of all specialized stations that inherit from it (relationship R8).

Each child of station corresponds to a manufacturing cell in the technical plant. Relationship R10 states that at a certain point in time a workpiece is located at a specific station (and at the same time, that each station can contain many workpieces).

Relationship R4 specifies the (physical) connection between stations. It is conditional on both sides, to allow the existence of a first and a last station, i.e. stations that have only one neighbor station. Relationships R2 and R3 indicate that a station may have moving parts and containers to store workpieces. The information about
Fig. 4 Identification station and its object-oriented static model

Fig. 5 Information model of the Modular Production System (a partial view).
moving parts and storage capabilities (if any) is registered in each specialized station, and is described in a separate information model.

4.2 Design

In order to design the software for the application domain, the design policies and rules established in the architectural domain (described in section 6) must be applied to it. The choice of which classes will be active plays a key role during design. The fact that a class becomes active means that it knows its state and has the ability to handle both in normal cases as well in presence of exceptions (errors or fails). It also coordinates its subparts.

In this case study, those classes that resemble physical modules in the technical plant, i.e. the five types of station, have been chosen as active ones.

The workpiece class was selected to take the role of coordinating all the stations of the manufacturing system. This major architectural decision leads to consider the workpiece also as an active object. In fact, it is the 'most intelligent' object, as it knows what kind of processing should be performed by each station in order to be processed. For instance, when a workpiece arrives in the drilling station, it communicates with it, ordering a service and passing drilling parameters. The fact that the sequence and kind of processing to be performed is stored in the workpiece itself allows a very flexible manufacturing system to be built, since in case of failure in a manufacturing cell, for instance, the automation system could be easily reconfigured.


5.1 Conceptual framework

Modern industrial automation systems exhibit a multi-layer, hierarchical control structure [10, 11]. In order to depict the hierarchy and the properties of the communication among the different layers, a pyramid is frequently used (see Fig. 6): lower levels handle the largest volume of information, operate under the tightest time constraints and deal with less sophisticated data structures.

Fieldbusses are communication systems specially designed to meet these requirements. They are usually serial busses, oriented to short or medium messages and speed optimized. As depicted in Fig. 6, fieldbusses are used at field and automation level.

At the field level, device controllers (internetworked through the fieldbus) are attached to each automated device. The controller implements low level automation functionality and build the interface of the computer system to the technical process.

Within an object-oriented framework, the responsibility for the functionality at the automation layer is assumed by the objects found during the analysis of the application domain, that model the technical process (the objects depicted in Fig. 4 and Fig. 5 are examples to this). In the case study presented here; these objects are implemented on distributed hardware, which is also connected to the fieldbus.

The higher layers, not covered here, are usually implemented using conventional local area networks.

Two kinds of interactions take place on the fieldbus. On the one hand, the communication among device controllers is characterized by a high rate of short messages with severe real-time constraints (latencies in the order of one millisecond). Thus, a message-oriented communication system, that supports on-demand messaging without need of opening and closing links for each information exchange is required. On the other hand, each of the object modeling the technical plant at the automation layer needs to communicate 'vertically' with the device controller corresponding to the equipment which the object is modeling. Although the same fieldbus is used, the nature of this interaction is different: there are pairs of 'communication partners', that remain unchanged during program execution. The messages are of higher level but with lower rate and less exigent concerning timing constraints. Consequently, a virtual point-to-point link (on the basis of the fieldbus) between the two communication partners is to be used here.

Fig. 7 depicts the structure of the automation system, focusing on the conceptual framework for the communication domain that has just been proposed. Dark rectangles represent hardware, both at the field as well at the automation layer. A CAN fieldbus interconnects the field devices and one of the computers in the automation

---

1 The object will also exchange messages „horizontally“ with its peer objects (which can be implemented on the same computer or not)
layer (a PC under the real-time operating system QNX). Objects that implement non-time critical features of the automation system run on other computers internetworked by a LAN.

5.2 Analysis

The communication domain aims to hide low level details of the communication between application objects with their counterparts in the technical plant. Hence, a 'bridge' between the application and the communication domain is established, so that objects in the application domain are able to act on the technical plant. One of the goals is to make this communication independent of the kind of protocol and of the communication medium adopted, enhancing the portability of the automation system to various software/hardware architectures.

The objects that populate the communication domain were partitioned into three layers (see object communication diagram in Fig. 8), according to their level of abstraction and offered services:

1. The lowest level supplies a minimal functionality to properly initialize the communication hardware and make usage of it. The hardware drive provides a transparent access to the communication hardware, hiding details such as I/O address, low level initialization, etc. It provides the basic functionality to access the fieldbus (read/write operations).

2. The second layer isolates the hardware from the rest of the system. It is hardware independent, but specific for a particular fieldbus protocol. The main object in this layer is the communication drive, which assures that the hardware drive would be properly accessed and completely hides the fieldbus protocol. Hence it presents an interface which is independent of the low level protocol used, and can thus be reused in a wide range of applications.

3. The third and last layer introduces the concept of communication channels, in order to fulfill the requirement for 'vertical communication' discussed before. It issues to offer a seamless communication facility between application objects and their counterparts in the technical plant. In this way, application objects have to manage neither fieldbus dependent message identifiers (this is a responsibility of the communication channel) nor addressing issues (which are resolved at channel creation time). Moreover, each channel knows a specific set of messages. That way, after the initialization of the communication channel, the application software disposes of a vocabulary to communicate with the
technical plant which has the application domain level of abstraction rather than the (much lower) communication domain level of abstraction.

Fieldbus protocols usually have to cope with tight timing requirements. Because of this, they are usually restricted to layers 1 (physical), 2 (data link) and, in some cases, 3 (network) of the OSI/ISO reference model. This resembles the MAP/TOP protocol schema, where the layers 3-6 are empty (for node interconnections). Actually, a consequence of this model is that much of the work ‘normally’ performed by the layers 3-7 has to be provided by the application software itself. The hardware and communication drives are the ones that provide the functionality of the missing layers:

- The packages coming from the fieldbus are classified and delivered to the proper application objects, in correspondence to the network layer (no. 3).
- The definition of ‘allowed messages’ contains functionality that corresponds to the presentation layer (no. 6). In effect, this layer provides a common representation for application information while it is in transit between peer application processes over an OSI/ISO ‘data communication channel’, i.e., a session connection.
- A connection-oriented communication model is being used here, and the communication channel in Fig. 8 manages aspects such as open and release of a channel, which correspond to the session layer (no. 5) in OSI/ISO model. This layer has no concern with syntax or semantics of the information, or with its encoding for transfer.

Fig. 9 shows a simplified information model of the communication system (for the sake of clarity, the station object is also shown, although it is part of the application domain), where the three object layers described before can be easily identified.

As stated before, each instance of a communication channel knows a set of pre-defined messages. Relationship R13, together with the associative object known message, formalizes this concept. Thus, an instance of the class ‘channel’ represents an uniquely defined logical channel between a specific physical unit and its logical counterpart in the automation software (e.g., between the software object ‘rotating arm’ and the real manipulator). If its set of known messages is adequately defined, it ensures that only valid messages will be sent to the recipient devices. That guarantees a consistent communication\(^2\)

In some cases, it is necessary to create specialized communication channels, derived from the basic one, in order to support a more differentiated set of messages (Types yy and xx in Fig. 9). For instance, a broadcast communication channel may be created, whose goal is to send one message to many recipients at the same time. A typical use of such channel is to ease the sending of emergency stop commands to the technical plant.

5.3 Design and implementation

Standard off-the-shelf communication hardware was used for the implementation of this case study. It was delivered with software equivalent to the hardware drive object in Fig. 8. The next object layer, represented by a communication drive object in Fig. 8 depends heavily on the operational system used. Hence, in this case study, it takes the form of a POSIX device driver for the real-time operating system QNX. The interface to the CAN bus is realized by a PC-CAN board on the automation level side and by I/O cards with integrated CAN interface (the field devices in Fig. 7) on the field side.

6 Architectural Domain

The architectural domain fills the gap between the analysis results (a specification based on real-world entities, such as stations, workpieces and motors) and the abstract concepts used in software implementation, like tasks, lists or memory addresses. For other architectural

\(^2\) Note that usually in distributed application it is entirely the responsibility of the programmer to guarantee that the data format sent by one client process in one node matches the format expected by the server process in another one.
domain examples, see [9] or [12].

Due to the recursive nature of the development process used, some of the main characteristics of the architecture domain were already mentioned, during the description of the application and communication domains. This section summarizes those characteristics.

An object-oriented architecture suited for real-time systems has been adopted, which can be characterized as follows:

- A reduced set of active objects are identified in the application domain. Usually they correspond to significant physical devices in the technical plant. However, abstract objects with interesting behavior are candidates as well.

- Each active object possesses its own thread of control, and thus the events to or from an active object are mapped as asynchronous communications between threads. Accesses to object intern data are mapped as synchronous communications. (For a more detailed description of the active objects implementation see [4])

- There is an initial thread, independent from the active objects’ threads, which is responsible for the creation of all other active objects (This corresponds to the main program in a C++ implementation or the root object in Eiffel).

- The remaining objects are passive. They exist within either the initial thread or in the thread of an active object.

The design concludes when the strategy and mechanisms defined in the architectural design are applied to translate the models obtained during analysis in source code based on the selected programming language.

7 Conclusions.

The development of modern industrial automation systems is undoubtedly a very complex task. The challenge of developing these systems cost-effectively, on time, fulfilling prescribed quality criteria, can only be met by applying apt system engineering approaches. In this paper we presented an object-oriented approach to achieve these goals.

The approach follows the core idea: when developing an industrial automation system, before starting to think of what hardware components, bus systems, etc. will be used, one has to fully understand the problem to be solved. Hence, a deep analysis of the automation problem to be solved is a must. Based on this first analysis, the software and hardware structure must be defined. Special
care must be taken when choosing/defining this software and hardware structure, in order to ensure that it will be less subject to change than functional requirements are.

The main advantage of the approach presented in this paper is its seamlessness: application classes identified in the initial analysis phase, evolve (in a gradual fashion) through each development phase to a stable representation form. The existence of server domains, such as the communication domain described in this paper, which supports the communication between the application objects, and the use of the same paradigm along all project phases, allows a smooth transition from one phase to another, easing the traceability of the system developed.

By layering the objects in the communication domain according to their level of abstraction, a structure similar to the OSI/ISO reference model is achieved. Consequently, the solution has the following advantages:

- All application objects are connected to the technical plant in a similar way, using one or more 'communication channels'.
- The application software is independent from the operating system, communication protocol and hardware (at least concerning the communication with the technical plant). Problem-oriented decisions are separated from implementation-driven ones, enhancing the understandability of the application software.
- Host computer can interwork with distributed intelligent controllers. The message passing communication among objects used at the automation layer - characteristic of object-oriented software structures - is harmonically integrated with the need for efficient communication with field devices, by introducing the concept of 'communication channels'.

The communication domain, its structure and implementation, were created to provide the objects in the application domain with an appropriate mechanism to communicate with field devices. Therefore, the approach focuses much more on obtaining a high degree of integration between object-oriented automation software and communication with field devices than on issues such as using standardized high level protocols such as CAN.

The implementation of the ideas presented in this paper is encouraging, because it shows that the aimed high integration can be achieved. However, two main issues require to be investigated in more detail, and are subject of current research:

- Real-time issues: although in the case study presented here no significant performance degradation due to the additional software layers was observed, it must be taken into account and quantified. A mechanism to by-pass the introduced networking software for the case of high priority messages should be considered, attempting to preserve a neat structure. Giving application objects the possibility of specifying the communication latencies they can accept during the creation of a communication channel, in order to 'negotiate' an adequate priority with the communication system is another direction of research.
- Standards: The possibility to use de-facto standards such as MMS (Manufacturing Message Specification, layer 7 of MAP) at a certain level within the communication domain should be seriously considered.

References.