Reusable Industrial Control Systems

Andreas Speck

Abstract—Industrial control hardware may be reused for several purposes. The same industrial PC type may control the drives of a portal system, act as a programmable logic controller, or control any other device. Moreover the same piece of hardware may control different device types at the same time in concurrency.

In this paper, we discuss four successive software engineering approaches to exploit the possibilities provided by the control hardware: an object-oriented architecture, an architectural pattern, a control framework, and an architectural model for component-based frameworks.

Each of these approaches provides means of different levels to structure a system and to reuse concepts, patterns, or real code. All presented approaches are used to build control systems controlling different types of robot arms and other devices. Additionally, in the component-based system a new component has been introduced to enable a network connection of the control system.

Index Terms—Factory automation, industrial control systems, software reuse concepts.

I. INTRODUCTION AND MOTIVATION

In industrial automation, the importance of software is constantly increasing. Tasks formerly performed by hardware are now executed by software systems. The main reasons for this trend are twofold.

Firstly, both the production costs for software are smaller and the flexibility of software is higher compared to hardware [6], [7], [10]. In contrast to hardware components, the incurring production costs for software are mainly limited to the development phase. Once developed, software can be copied multiple times at comparatively low expenditure. Therefore, in our work we introduce approaches supporting system development in this domain.

Secondly, the increasing percentage of software is due to the development on the hardware sector. The number of standard systems (off-the-shelf hardware) in circulation is growing [15]. Manufacturer-specific expensive hardware is being replaced by low-cost standard hardware like PCs, workstations, or their industrial counterparts industrial PC’s (IPCs) and industrial-grade workstations, respectively. Some examples for this development are the two robot controls Kuka KR C1 and Bosch rho 4 or the porting of the Programmable Logic Controller system (PLC) S7 of Siemens of the industrial PC line Scomp. Besides the fact that standard hardware is less expensive, it has the advantage that it automatically takes part in the rapid advance of hardware development. In parallel to this trend toward standard hardware, the need for software is growing. Software is necessary to adapt those standard hardware components to specific applications which was formerly done by application specific hardware. The missing application-specific hardware is mostly substituted by software.

This advance and growing importance of software requires more advanced technologies for the software development process than is currently available and applied in industrial controls. Some software systems are still implemented in Assembler or C with little application of software engineering methodologies. Modern approaches like the object-oriented paradigm or the application of patterns are not very widespread yet.

Our work contributes to close this gap between importance of software and support or methodology, respectively, in the domain of industrial automation. Four successive approaches are introduced which support reuse on a different level suitable to the specific problems to be solved by the control system. The level of abstraction provided by each approach increases in parallel to their order.

1) Object-Oriented Architecture: This architecture has been built for a new generation of open universal control systems which run on standard hardware. The architecture has been realized on Sun SPARC workstations in the project HIGHROBOT [10]. The HIGHROBOT system proved that the object-oriented approach can be successfully applied in the area of industrial control systems.

2) Architectural Pattern: The architectural pattern is a further step toward improved reusability. While the object-oriented system may be regarded as a novelty in the field of control systems, this approach is also an improvement in the domain of object oriented software engineering. The pattern provides a guideline how the object oriented system may be reused efficiently.

3) Control Framework: This approach is based on the motivation that the control system has to be ported to new platforms such as Windows NT, Linux, or proprietary industrial PCs [23]. The object oriented-control framework may be applied to systems with POSIX-compliant operating systems like Windows NT or Linux. The framework allows minor adjustments of the operating system calls. (Of course, the potential control application is also a flexible element in the framework.)

Both the pattern and the framework support not only the control of a robot arm, but also other machine tools. The architectural pattern is driven by the experiences acquired from the object oriented architecture. As opposed to the framework, this pattern may also be used for platforms with non-POSIX-compliant operating systems (e.g., the real-time system RMOS for Siemens IPCs).

4) Architectural Model for Component-Based Frameworks: In collaboration with D. Parsons, A. Rashid, and A. Telea,
a component-based approach for frameworks design has been developed [13]. The architectural model is much more flexible than object-oriented frameworks since it covers a higher level of abstraction than patterns and frameworks do.

The model has been evolved and subsequently applied to component-based control frameworks in our work. Automation technology applications in industrial automation have the advantage that the controlled hardware itself consists of different hardware components. Therefore, the component boundaries may be easily found by reflecting the hardware components (and their composition) in the software.

II. STATE OF THE ART AND RELATED WORK

A. Today’s Control Systems

Today’s industrial control systems market is dominated by proprietary solutions. The control systems such as robot controls (RCs), PLCs, and numeric controls (NCs) are incompatible and have only limited communication interfaces [10], [17]. These traditional controllers consist of special hardware and vendor-specific control software (usually offering a more or less standardized application programming interface). The resulting problems are high development costs and long terms of innovation cycles, e.g., the Bosch robot control system rho3 has been sold for almost ten years without any significant modification.

B. Alternative Approaches

Different projects have been initiated in order to develop concepts and reference architectures for open, inter-operable control systems with respect to object orientation. The most important examples are Open System Architecture for Controls within Automation Systems (OSACA) [1], [2], and Open Modular Architecture Controllers (OMAC) [11]. Both provide high-level non-object-oriented reference architectures and sample implementations. They focus on the modular arrangement of abstract blocks of control functionality from the perspective of electrical or mechanical engineering. Besides a very high level overview, they provide no more concrete descriptions of their concepts in any object-oriented notation—neither architectural nor design diagrams.

Another project is the Open Control Architecture for Windows NT [12]. This concept is based on OLE and COM. However, this approach suffers from the limited real-time capabilities of Windows NT. All these approaches neither focus nor apply (object oriented) software engineering principles consistently.

Now, there are successful first steps indicating that the application of software engineering means makes the reuse more effective in the domain of industrial control systems. One example is the application of object oriented concepts for the development and design of control programs which has been demonstrated in various projects, e.g., [30] or [29]. An object-oriented development process for control programs has been presented in [5]. This process is mainly based on Unified Modeling Language (UML). The application of design pattern as one means to express the reuse of design details has been presented in [4], [22], and [28].

Fig. 1. Integrated control systems.

III. REQUIREMENTS

In the beginning of our work, the requirements of the next generation’s control systems have been determined in cooperation with other control experts from different end-users like Daimler-Chrysler (formerly Mercedes-Benz AG) and control manufacturers like Robert Bosch AG, Siemens AG, and MOOG Ltd., as well as with research groups at the Universities of Karlsruhe and Munich, Germany.

The most important requirements are as follows.

• Integrated Control Functionality: The control system has to be universally applicable. Therefore, different traditionally separated control types have to be integrated in one universal control system. Such a control may include robot controls, numeric controls, transfer, and drive controls as well as PLCs (see Fig. 1). Additionally, virtual devices may be simulated by software modules.

• Multitasking Control: In order to control many devices a universal control system has to provide multitasking. The individual devices or groups of devices are controlled by corresponding application tasks which may run in parallel. Communication between these application tasks is possible via the operating system’s interprocess communication mechanisms [7].

Some traditional control systems like the robot control systems Bosch rho3 or Kuka KR C1 already support a limited number of parallel application tasks (with a maximum of five tasks). However, such a small number of tasks is not enough to control an entire production cell. A system based on a POSIX-compliant operating system supports a considerably larger number of concurrent application tasks.


**Platform Independence:** The software architecture of a new control system has to support different platforms (see Fig. 2).

The architecture must be independent from the control hardware. The control may be implemented on workstations as well as PCs or industrial standard hardware such as IPCs. Moreover, the universal control architecture must be operating system independent. The only demand is that the operating system should be POSIX compliant. The devices may be connected directly or by a fieldbus to the control hardware. Interface cards which are plugged in the control hardware or integrated interfaces which are provided by some industrial control systems realize the connection to the devices.

---

**IV. SOLUTIONS**

**A. Universal Object-Oriented Robot Control**

This first system has been designed specifically for SPARC workstations and it was intended to control a limited number of robot arm types in conjunction with some peripheral devices which are currently in wide circulation [10]. The programming language is C++.

1) **Architectural Overview:** The architecture is structured in hierarchical layers (see Fig. 3) which is a typical approach in object-oriented systems [3], [19]. The boxes represent collections of classes or subsystems (class categories according to G. Booch 1994 [3]). Interfaces which represent the layer boundaries define the relationships between the subsystems and allow to exchange them. The main class categories are as follows.

- **Control Subsystems:** These subsystems provide the control functionality, e.g., robot control procedures. The user-written Control Application Programs (e.g., robot programs which are executed on the control system) have access to the Control Subsystems via the Device Control Interface.

- **Communication and Coordination:** This subsystem embraces the system communication and control of the dynamic behavior.

  Interprocess Communication contains mechanisms for the data exchange between the different tasks of the system and their synchronization. It provides a virtual Device Connection Interface for the Control Subsystems. Thus, all Control Subsystems seem to have individual access to the controlled devices.

  The Coordination controls the dynamic behavior of the system (see Section IV-A.2). Therefore, it initiates the internal data exchange and the communication with the devices.

  The concrete device communication is handled by the Device Connection which may be adapted to the specific hardware interface.

- **User Interface:** The user interacts with the control system via this subsystem. Currently, two types of User Interfaces are implemented: a primitive ASCII menu and a Java GUI.

2) **System Tasks and Their Dynamic Behavior:** The control system contains three types of tasks: a single Coordination Task (which instantiates the Communication and Coordination), Application Tasks (embracing the Control Subsystems, the Control
Application Programs and the Interprocess Communication), and finally the User Interface Tasks (with User Interface and Interprocess Communication). These User Interface Tasks may be regarded as special Application Tasks where the user manipulates the control system directly.

The system performs the typical behavior of control systems: it runs in a cyclic loop reading the current state of the devices, executing the control application and transmitting the result to the devices [17]. The sequence chart in Fig. 4 depicts the major interactions between an Application Task (e.g., a robot control application) and the Coordination Task.

B. Architectural Pattern

The architectural pattern is derived from the object-oriented robot control described above. However, this pattern is not limited to robot controls but is also applicable for the development of other automation control systems such as PLCs, numerical controls, or special control systems (even on nonstandard platforms like industrial PCs with proprietary real-time operating systems where code reuse is difficult).

The pattern-specific features are: layered design, multitasking support, a virtual device interface and a unique Communication and Coordination subsystem which controls the dynamic behavior.

The architectural pattern embraces several design patterns described in [4], [6], and [8]: Proxy, Observer, Master Slave, Forwarder Receiver, Facade, and Singleton. Since these design patterns are integrated in an overall structure this architectural pattern may be also regarded as pattern language for the development of automation control systems.

C. Control Framework

In contrast to the pattern our framework provides premanufactured code fragments. Thus, its deployment is restricted to UNIX- and Windows NT-based platforms. Although this is a limitation the usage of the framework shortens the (design and) implementation phase and therefore justifies the existence of such a framework.

The framework itself has been derived in an iterative process (according to [18]) from the universal robot control system introduced in Section IV-A. In this iterative process the flexible elements (called “hot spots” as opposed to the nonflexible “frozen spots”) have been identified and implemented. These “hot spots” are itemized in Fig. 5 (marked with gray ellipses).

D. Architectural Model for Component-Based Frameworks

The architectural model for component-based frameworks is a meta-model in the sense that it describes how a component-based framework is structured efficiently. This meta-model has been elaborated based on the experiences of framework designers in various domains [13] and captures the essential commonalities.

The model provides guidelines for the flexible composition of component-based frameworks. It is a blueprint or pattern...
for the combination of the framework’s components and categorizes these components with their relations in-between. All these framework’s components may be developed by different companies or persons playing different roles in the development process [13]. The only requirement is that the interfaces correspond to a common standard or agreement and thus fit together. Like frameworks (see Section IV-C) the architectural model defines flexible “hot spots” expressed in the Customizes relationship (see customization in [14] and [18]). However, in contrast to a framework the composition of the framework’s subsystems may be adjusted, too (symbolized by the Plug-in relationship). Thus, the flexibility is not limited to a small number of pre-defined “hot spots” but is extended to the entire structure of a framework.

We have identified the following types of components [13]: The Backbone Component (see Fig. 6, dark-gray box) controls the behavior of the system and the interactions between the components. This Backbone Component is surrounded by the Basic Components (light-gray boxes) which provide the basic services used by the Backbone Component. Together, both component types may be regarded as the core of the framework. The Additional Components (white boxes) refine and extend the system.

Fig. 6 demonstrates how the architectural model may be applied in the domain of industrial control systems [22]. The central component is the Communication and Coordination. In conjunction with the Basic Components it provides the most important functionality of the framework as a base for the execution of the control algorithms. Some of these Basic Components (Drive Connection, Data Exchange Interface and POSIX Interface) are customized to the specific needs of a certain control application. Due to the fact that the control algorithms have to be adjusted to the user’s requirements the control functionality cannot be part of the Basic Components. Therefore, the components containing the control functionality are arranged as Additional Components.

This model may be extended by the versioning approach proposed in [25]. The versioning approach may helpful when a larger number of components has to be considered.

The industrial automation domain has a long tradition of “reusing” hardware components. Almost all electrical and mechanical devices and the processes performed by them are well defined and standardized. Standardized interfaces of the hardware may be reflected in the software controlling these hardware components. Thus, hardware standards are an ideal support for standard software components and vice versa.

1) Networking and Simulation Component: The component-based framework has been extended by simulation, networking, and monitoring components. In detail, the simulation component concept makes it possible to simulate a specific device on the control hardware in parallel to the control tasks.
As a concrete realization we implemented a SCARA robot simulation system which accepts similar commands send to a real robot arm. It simulates the responds to these commands and returns them to the control system. The simulation consists of an Additional Component realizing the simulation routines of the virtual device. Additionally, a new Device Connection component has to be build (Basic Component) which provided the connection between the simulation component and the Data Exchange interface component.

The second component is the networking component providing a TCP/IP communication. One prerequisite is that the operating system of the control provides a TCP/IP communication interface (which operating systems like Solaris, Windows NT, Linux, and RMOS do). If the operating system supports a loopback communication the system could be connected with the monitoring component and, if the control system contains network interface hardware as well, the system could communicate with a Java monitoring component on a different system. The networking component is part of the Basic Components (it is possible to extend them by Additional Components).

In contrast to the other components the graphical monitoring component is realized in Java instead of C++. Hence, it may be executed on a large number of platforms. There exist different monitoring components displaying a Bosch SR 60 SCARA robot arm. This robot may be directly manipulated by the user or robot control tasks (Application Programs) can be activated. Although due to the different programming language not being totally integrated in the component framework, the monitoring component may be considered as an Additional Component.

These extra components may be used to complement the rest of the control system. We called the superordinate concept of this combination RoboSiM (robot simulation and monitoring). However, they may be arbitrary combined and used for specific services, e.g., simulation only without graphical monitoring or simulation with graphical monitoring. Further details may be found in [20], [21], and [24].

V. CONCLUSION AND FUTURE WORK

In our work we have built an object-oriented industrial control system proving that the object-oriented paradigm is suitable in the domain of industrial automation (i.e., the constraints—e.g., real-time capabilities—are preserved). Moreover, it has been shown that object-oriented software engineering opens new opportunities and, thus, allows us to build the next generation’s control systems.

On the basis of the object-oriented control system, we have introduced three further approaches with different levels of reuse. Each of these approaches has advantages in specific application scenarios.

The architectural pattern which is derived from the object-oriented control system is particularly suitable where the developer has to deal with the challenge of strongly differing platforms. While differences may prohibit the reuse of code (or may allow the reuse of only some code fragments) the reuse of architecture and design details is possible. We have applied this pattern when porting the control system to a Siemens industrial PC (Sicomp IMC05) with the operating system RMOS.

The object-oriented framework has its advantages when a product line of comparatively similar systems with only minor differences has to be developed. We have used the framework when porting the UNIX control system to Windows NT (both with and without GNU Win 32). An implementation on the latest Microsoft operating systems will follow in the future.

The architectural model provides meta-level support for component-based frameworks. A framework which corresponds to such an architectural model combines both advantages, the flexibility of the architectural pattern and the reusability of code. From this architectural model we have derived a component-based framework for industrial control systems. This framework has proven to be an efficient base in the development of new systems in this domain—both on alternative platforms (e.g., Linux) and with extended functionality. Such additional functionality,
for instance, has been realized by the robot simulation and monitoring system (RoboSiM) [24]. It uses a robot control system derived from the architectural model. The simulation and monitoring elements are realized as Basic Components and Additional Components.

Besides the ongoing collaboration with Siemens AG (modular control systems on new platforms), we are currently working on new frameworks in other domains (e.g., monitoring and three-dimensional visualization in medical science). These frameworks will be derived from the architectural model.

In the software engineering sector, in particular, the architectural model may be used as a blueprint for system generators as introduced in [16]. Such a generator will be focused on in a future work. New approaches in object orientation (e.g., the separation of concerns principle) will play an important role. Work has already been carried out to apply Aspect-Oriented Programming (AOP) [9] in the field of both control systems [26] and generators development [16]. These generators use AOP to combine the components. In this context, we will particularly consider highly customized software [27]. Software development processes with respect to the generative capabilities are a further issue.

REFERENCES


tion, Milan, Italy, 1996.


[17] U. Rembold, B. Nnaji, and A. Storr, CIM: Computer Integrated Manufac-


[19] B. Selic, G. Gullekson, and P. Ward, Real-Time Object-Oriented Mod-

[20] A. Speck, “Robot Simulation and Monitoring on real controllers (Ro-
boSiM),” in Proc. ESS’98, 10th European Simulation Symp. and Exhi-


Andreas Speck received the Doctoral degree in computer science from the University of Tübingen, Tübingen, Germany. In his thesis, he developed new object-oriented and component-based software engineering approaches for industrial control systems. Currently, he is heading the research division of Intershop, Jena, Germany, an eCommerce company building medium- to large-scale Internet marketplaces. The main tasks of Intershop Research are the development of new software engineering concepts in order to increase productivity and to improve the quality of software. Additionally, the integration of new systems such as factory automation devices in networks and virtual information systems is within the focus of his work.