Automation Component Architecture for the Efficient Development of Industrial Automation Systems

Reinhard Hametner, Alois Zoitl
Automation and Control Institute,
Vienna University of Technology
Gußhausstr. 27-29/E376
A-1040 Vienna, Austria
{Hametner, Zoitl}@acin.tuwien.ac.at

Maria Semo
logi.cals Austria, kirchner SOFT GmbH
Mailüfterlweg 1
A-3124 Oberwölbling, Austria
Mario.Semo@logicals.com

Abstract—The increasing complexity of automation applications needs a new framework architecture to develop automation control systems. This paper shows a workflow for engineering and maintenance applications in the automation control system domain. A new component architecture is specified which supports component-oriented design, reusability, and encapsulation of functional parts. The common component includes an automation sub-component, diagnosis sub-component, an optional condition monitoring sub-component, and a test sub-component for an efficient development process.

I. INTRODUCTION

Fast changing consumer demands are a great challenge for production companies. To be competitive in the area of automation control engineering, existing applications will be adapted and reused for future requirements and it is a possibility to cope with these challenges. Furthermore, there is a need for new applications to have encapsulated components and reusability of the application parts. These properties are an important factor in an efficient development process, because changes of the mechanical system also infer changes of the electrical part of the control application. In current control software applications the software code and testing code is often intertwined in the code, which hinders efficient and systematic testing [1]. Therefore a test process within the development phase is a significant benefit which reduces valuable development time. For this reason an encapsulated component-based development approach with diagnosis and testing functionality is required for the automation control software design. We believe devoutly that this component-based architecture approach is of key importance: Where the components are easy to reuse for other control applications, the components have been tested, and these components are easy to test after modification. These requirements need special architecture, design, and implementation rules.

In practice, the terms “architecture”, “design”, and “implementation” can be abstracted full detailed (implementation), few details (design), and the highest form of abstraction (architecture). But the target alone is insufficient to characterize the differences, simply because architecture and design documents contain information which is not in the implementation document (e.g., standards, design constrains). There is a need to separate the information flow in a clear manner. What information goes into architecture documents and what goes into design documents? This basic information should be clear before the implementation is started. The distinction between architecture and other forms of design artifacts has not always been clear. The Software Engineering Institute (SEI) [2] and Eden et al. [3] proposed the Intension/Locality Thesis which provides a foundation for defining the meaning of the terms architecture, design, and implementation. Furthermore, an explanation with two separated interpretation approaches (i.e., Intension/Extensional and Local/Non-Local) for the relationship between architecture, design, and implementation are presented.

- Intensional specifications are abstract rules over an unbounded domain (e.g., rule for identifying prime numbers).
- Extensional specifications are concrete information’s (e.g., EU Member States).
- Non-local specifications are generally applicable.
- Local specifications are only in partial parts of the system valid (e.g., design pattern specifications).

These interpretations is the basis for the component-based automation architecture approach and are practical to the distinction of architecture, design, and implementation.

The remainder of this paper is structured as follows: Section II presents some related work in the area of automation and software architecture. Further explanations on condition monitoring for the control systems domain, which supports significant benefits in industrial automation systems are given. Section III presents requirements for new automation architecture. The automation component model shows Section IV, which is used for a prototype application in Section V. Section VI concludes and presents future work.

II. RELATED WORK

This section summarizes related work on structuring automation component architecture for industrial control applications. Furthermore, an introductory explanation of the condition monitoring issue is presented.

A. Potential Automation Architectures

Conventional PLC-based architectures are the standard for control engineering and applications. This architecture approach causes high costs for developing control applications and maintenance of these applications. Sünder et al. [4] differ...
between a functional and a mechanical view. They used the global view of the mechanical component and structure the software components according to them. Therefore, the control modeling structure based on automation components and the hierarchical software structures are used in distributed control systems. To test and verify the automation components Hegny et al. [5] explained a component-based simulation framework for production systems which is based on IEC 61499. Several simulation scenarios are presented like fully simulated plant, integration of external simulation tool, hybrid simulation, and fully operational approaches.

Čengić et al. [6] present a framework for component-based distributed control software. They use automation components as software components for implementing control software applications. Hence, their architecture is built with as many sub-components as necessary. Therefore, a hierarchical architecture is formed, where higher hierarchical levels are connected by their backend to the frontend of automation components at lower hierarchical levels.

B. Condition Monitoring for Industrial Processes

Condition Monitoring (CM) is a management technique, which uses the regular evaluation of the actual operating condition of plant equipment, production systems, and plant management function for optimizing the total plant operation. Successful maintenance is based upon good management, a practical structure, and the right methods or technique in any given situation, which are presented in [7], [8], and [9]. The main subject of machine condition monitoring is charged with developing new technologies to diagnose the problems of industrial processes. Different methods of fault identification are developed and used effectively to detect the machines faults. CM is used to increase the availability and performances, reducing consequential damage, increasing machine life, reducing spare parts inventories, and reducing breakdown maintenance of the machinery. The information is monitored in the form of primary data and through the use of modern signal processing and analysis techniques. The possibility is given to diagnose the information over the devices before it catastrophically fails. The diagnosis process for automation applications are used intelligent diagnosis algorithm to detect faults of the included mechanical components.

III. OVERALL APPLICATION ARCHITECTURE

Today’s state of the practice for control application development in the field of industrial automation is characterized by an approach where control applications are individually developed for each application. This results in a high development effort per plant as there is only little reuse of software components between different applications. The testing of the application is typically performed rather late in the development cycle together with the plant in an ad-hoc manner. Therefore, the final application often contains many late changes and fixes which may break the overall design. This leads to code clutter and couplings between application parts and therefore reduces the reusability of the application components. A consequence of this is that the development time increases and finally leads to high control application development costs. Therefore means are necessary to reduce the development effort as well as the development uncertainty of industrial control applications.

An important step towards an improved development process is an architecture which defines constraints on the control application structure. At first the architecture needs to define how an application can be decomposed in independent parts. As reuse of application parts is a key feature for reducing the development effort, the architecture needs not only define rules allowing the decomposition of application parts (i.e., top down development), put also rules for aggregating existing application parts (i.e., bottom up development). The independent application parts will need to interact with each other in a defined way in order to clearly decouple the application parts. The decoupling of the application parts is a key requirement for reusing application parts in different applications. The application parts have to behave the same independent of the context they are used in. This allows that once tested and validated components can be reused without additional testing effort. Szyperski [10] coined for software modules with such properties the term software components. According to his definition we will use the term Automation Component (AC) for our reusable control application parts.

With the AC in hand we can now define how applications may be composed out of them and what ACs will represent. Previous investigations have shown that a strong hierarchical approach is deemed to be a necessity for industrial automation (see for example [4]). For defining the hierarchy we will look at the functional or mechanical view of production facilities. There one will notice a strong modularization and compositional approach. Plants can for example be structured in cells, which themselves contain several machines. The machines themselves are also built from mechanical assemblies and these are built from devices such as a cylinder or a pump. The main advantage is that the components on each level can be reused in many applications, because they are independent from their usage. The second point is that sub-components can easily be replaced as long as the new ones meet the mechanical and functional specification. By mapping the mechanical structure of a plant also to the structure of the control architecture the programming of large automation systems can be greatly simplified.

Therefore we propose to structure the control application hierarchy according to the mechanical or functional hierarchy of the plant and each AC represents a mechanical functional unit (i.e., process engineering objects like pumps or motors) on the specific level (see Fig. 1). Each of the ACs specifies different software-specific aspects for the implementation and execution of these units. This means that a lower level AC such as a pneumatic rotating cylinder provides the functionality to rotate to the left or right. On a higher level the AC provides the functionality of moving a part from one conveyor belt to a second. Higher level ACs are composed from such ACs and aggregate the functionality of their contained ACs. They use will use the provided interface
of the ACs one level below, add their own (e.g., coordinating) functionality, and provide it with their interface to the ACs that will use it. To utilize the full advantages of this approach several rules have to be adhered to:

- **Single access point**: Automation components may interact only by means of the provided interface. This implies that another AC serving the same kind of interface can replace an AC and the user of the AC can be left untouched. There may be established AC profiles for typical components in the similar way to existing device profiles for field busses. This especially will make replacement of similar ACs easier.

- **Strict hierarchy**: An AC will not know who is using its interface and can not make assumptions on the provider of an interface it is using. An AC is aware of the ACs’ interface and the functionality of only one level below. This is all the information it can use to provide its own functionality. Therefore it is not allowed to use an AC’s interface that is located in an AC lower than one level below itself.

- **Decoupling of ACs**: ACs located on the same logical level in the plant may not directly interact and may not interact through their interface. If this functionality is needed then it points to a design error because such a direct interaction is clearly a coordinating function between the two ACs and, as already stated, this is the task of an upper level AC.

IV. AUTOMATION COMPONENT MODEL

Based on the process engineering objects that engineers deal with (e.g., a pump, a tank) we define an Automation Component as a collection of hardware, software, and properties describing the interaction between the different parts and specifying different software-specific aspects for the implementation and execution of these objects. These properties include the expected functionality of the process engineering object (to be implemented as control software), errors introduced while implementing the control software, and faults expected to occur at runtime (e.g., due to erosion). The automation component describes these properties, the importance of the properties and methods describing how to achieve the required functionality. For the specification, domain and tool specific methods and standards can be used, (e.g., ISA88, P&ID). A schematical overview of the automation component illustrates Fig. 2.

A. Subcomponents

For a clear and better organization of the properties of an Automation Component, we suggest using 4 Sub-Components, one for automation aspects (logic, behavior, and implementation), one for test aspects (unit tests, integration tests, and factory acceptance tests), one for runtime fault analysis aspects (diagnosis), and finally one for runtime fault prediction aspects (condition monitoring and data analysis).

1) **Automation Sub-Component**: The Automation ("Logic") Sub-Component contains all information needed to realize the desired behavior, as specified by the process engineer. This Sub-Component must not depend on other Sub-Components and implements all requested functionality by itself. The existence or functionality of the other Sub-Components is not necessary for the Automation Sub-Component to fulfill its requirements. But other Sub-Components may influence the behavior of the Automation Sub-Component. For example, the Diagnosis Component may change the execution flow and internal states in case of errors or the Test-Sub-Component may change the state and data values due to external (test framework) demand. The Automation Sub-Component contains:

- Mechatronical components (sensors, actuators) to control the procedural units
- Specification documents (requirements, design, ...)
- Electrical plan for wiring and connection to control system
- Function plan (software) for implementation of control logic (e.g., in programming languages as defined in IEC 61131, IEC 61499)
- The main (public) signal interface description

All other Sub Components can be removed or deactivated without changing the functionality of the Automation Component.

2) **Diagnosis Sub-Component**: The diagnosis part of an Automation Component contains all elements responsible to assert correctness of the executed program and validity of data processed to avoid damage. Diagnosis code handles already occurred errors in such a way, that no subsequent errors will occur. Based on this view the Diagnosis Sub-Component is a supporting Component to make the Automation Sub-Component more robust against software errors and hardware faults. Parts of the Diagnosis Sub-Components is additional source code which performs runtime data validation (“assertions”, “pre-conditions”, “post-conditions”) and influences to Automation Sub-Component based on the results of such analysis.

3) **Condition Monitoring Sub-Component**: This Sub-Component is responsible for monitoring the condition of the automation system (hardware) and to predict the future condition (based on historical data). This Sub-Component has no influence on the primary functionality of the Automation Component but reduces down-times of the system due to unscheduled maintenance work. Therefore, the following is additionally required:

- Specification documents, such as Failure Mode and
Effects Analysis (FMEA) or Hazard and Operability (HAZOP)

- Hardware (sensors) for monitoring the condition of system elements and to display the results of the condition analysis to users
- Wiring of the new sensors in electrical plans and mapping the inputs to new software signals
- Software for processing, recording, and visualization of these new signals and of suggested maintenance work

4) Test Sub-Component: The function of the Test Sub-Component is to support the test of the behavior and functionality of an AC (Automation and Diagnostic Sub-Components) and of the system itself in the context of a test framework to find systematic errors from the development process. A sample for such a test framework is presented in [1]. Therefore, this Sub-Component contains:

- Code inspections, reviews, automatic code analysis ("LINT"\(^1\) for Automation), code verification (against specification)
- Test scripts for automatic execution and operation of tests
- Additional software and interfaces to support and communicate with the test frameworks (test drivers, mocks, specification and implementation of simulation components)
- Requirements concerning the enhancement of functionality of the test framework (e.g., interfaces, commands)
- Enhancements of diagnosis and control logic, such as specific dynamic activatable Forcemarker\(^2\) (to force signals/data to a specific external given value or to reach nearly unreachable system states, e.g., special error states).

B. Interfaces

An Automation Component has a three dimensional interface.

1) Inter-Component Interface: This interface is between ACs for aggregation of components. The down interface of higher level ACs are connected with the up interface of lower level ACs which the higher level ACs are composed of. The Inter-Component interface of ACs are defined by the up/down Interface of the Sub-Components where the main (must have) interface is the one from the Automation-Sub-Component. The interfaces from the Test and from the Diagnosis Sub-Component are additionally used to control the Test Sub-Component and read internal states via the Diagnosis interface. The implementation of the interfaces may result in hidden interfaces (e.g., direct access from the external test framework into a specific component).  

\(^1\)LINT can be used to check C code for errors that may cause a compilation failure or unexpected results at runtime. In many cases, LINT warns about incorrect, error-prone, or nonstandard code that the compiler does not necessarily flag. It also issues warnings about potential bugs and portability problems. Many messages issued by LINT can assist in improving program’s effectiveness, including reducing its size and required memory. http://docs.sun.com/source/806-3567/lint.html, Feb. 2010

\(^2\)Forcemarker is a specific feature of the logi.CAD Software Tool to set or read values of used variables

2) Intra Component - Inter Sub-Component Interface: That means the interfaces between Sub-Components of an Automation Component. This interface describes which Sub-Component is allowed to read/write (modify) data/states in other Sub-Components.

3) Inter-Tool Interface: The interfaces between different tools used to describe the different aspects of an AC. This interface is orthogonal to the previous two interfaces.

V. BOTTLE SORTING MANUFACTURING IMPLEMENTATION

In order to validate the presented approach a pilot application was implemented within the Odo Struger Laboratory at the Vienna University of Technology. A picture and the schematic overview of the laboratory configuration are shown in Fig. 3. The pilot application represents a bottle sorting machine with the task to sort the bottle of the appropriate color into the correct box. The implementation is realized in IEC 61131 with the development environment tool logi.CAD [11].

A. Overview and Specification

The configuration of the system consists of three major parts: a) Feeder, b) High-Speed Pick and Place (HSPP), c) Sorter. First of all a bottle has to be put on the conveyor belt after switching on the system by the on/off switch. Then the conveyor transports the bottles towards the HSPP. The first station is the bottle stopper entity with the feed separator which controls the bottle flow on the conveyor. If a bottle is detected by the presence sensor 1, the feed separator puts out a gate for stopping the bottle. Afterwards the bottle is transported further and is detected by the presence sensor 2 which informs the HSPP to grasp the bottle and put it to the other conveyor belt. There the color detection sensor identifies the color of the bottle and informs the turnouts for the ejection of the bottles into the correct box.

B. Resulting Component Hierarchy

The bottle sort application is a presentable example to show a hierarchical structure with encapsulated and reused components. Therefore these used components can be reused for other applications in our laboratory. For example: The actuator AC is connected with the valve AC. We need
for the bottle sorter application pneumatic actuators. If the pneumatic actuators should be exchanged by electric actuators, only the valve AC has to be exchanged by a new electrical component. Another case for exchanging an AC is when a special sensor type is broken and the device is not available anymore. A new AC can be developed or the existing AC can be adapted and exchanged by the existing component.

The component architecture can be chosen by their functionality and/or by their mechanical/physical structure. Fig. 4 shows the hierarchical structure of the bottle sorting production plant. All components can be represented by the automation component model which is explained in Section III and Section IV.

C. “Turner” - Automation Component of the Bottle Sort

One of the automation components is explained in detail. The “Turner” component from the application plant is chosen which is shown in Fig. 5. The mission of the Turner is to turn the bottle from one conveyor belt to the second. The automation component “turner” includes two end position sensors and one actuator which turns the turner between the two end positions. In regular operation only one “end position sensor” must respond.

Based on the automation component architecture model a diagnosis sub-component and a test sub-component are included beside the automation sub-component which includes the main functionality of the turner. The diagnosis sub-component and the test sub-component are independent of the functional part. Therefore the behavior cannot be disturbed by the diagnosis and test sub-component. Furthermore the diagnosis and test sub-component can be exchanged and reused if necessary.

The diagnosis sub-component has the job to monitor the working operation system for their correct procedure of the control commands. In this case two diagnosis terms are implemented: a) The Turner has to turn after a control command from one end position to the other in 1s (time terms) (e.g., faults: no/low pressurized air, pivot arm lock), b) The two end position sensors must not indicate active values at the same time (e.g., fault: one/both sensors broken). If one of the terms is not achieved a diagnosis message is sent to the user to react on it.

Opposite to the diagnosis sub-component, the test sub-component has functional access to the automation sub-component. It is necessary to define the test cases before the implementation process started. Therefore the most failures can be avoided. Two kinds of test strategies are possible: a) Unit Test; b) System Integration Test. In case of the unit test procedure, all other ACs are disabled. Hence, the behavior of the component under test is limited. For system integration tests all ACs are activated to test their interaction. A detail explanation about testing is shown in [1]. When the bottle sort application is in unit test mode the pivot arm must turn once in both ways whereas the user have to confirm each movements to test the sensors and the actuator for their accuracy. Finally a status code of the actual state is displayed.

The test case implementations are realized in two ways. The first way was the development by Sequential Functional
Chart (SFC), the other and more comfortable and efficient way is the Structured Text (ST) implementation. Complex functions are easier and more clear to solve with ST.

A condition monitoring sub-component was not implemented at this time. This will be done at future work.

D. Resume

The automation component model and the hierarchical structure is a valuable approach for encapsulated and reusable components. Therefore the encapsulation of the automation, diagnosis, condition monitoring, and test sub-component allows other components to monitor the health of the system and test the component after changes. This implementation approach is more time intensive at the first time but the comfort of reusability of tested components excuses that. Also simple applications, time-critical, and/or safety-critical applications are reasonable to use this component-based approach.

VI. Conclusion and Future Works

We presented in this paper a new automation component approach for developing industrial automation control systems. This approach reduces valuable development time because the automation component can be tested with their internal test functionality and supports additional diagnosis functionality with an optional condition monitoring sub-component for watching the health of the automation system. This concept has been evaluated in a bottle sort implementation within the Odo Struger Lab, realized in IEC 61131 with development environment logi.CAD [11].

As a next step the bottle sort laboratory system is used for the same implementation approach for the realization by IEC 61499 standard with the development environment 4DIAC-IDE and our own runtime environment 4DIAC-RTE [12]. Hence, both implementations (IEC 61131 and IEC 61499) will be compared and analyzed. Furthermore the condition monitoring sub-component will be implemented and a test framework for testing the automation component in logi.CAD will be defined.

VII. Acknowledgments

We want to thank our partners from academia and industry in the logi.DIAG project for their valuable discussions and feedback. Furthermore, we want thank Ingo Hegny for the fruitful discussion, Alexander Prostejovsky and Andreas Pfeffer for support the implementation of the bottle sort. Parts of this work were funded by the Austrian Research Funding Agency (FFG) grant logi.DIAG (Bridge7-196929).

REFERENCES


