Increasing Portability and Reuseability of Distributed Control Programs by I/O Access Abstraction

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

Abstract

Current component-based approaches for distributed control systems enable the reuse of mechatronic components, for example specific grippers or conveyor belts, but do not allow full application reuse as the actual hardware configuration is an implicit part of the software solution. In contrast to the current object-oriented or component-based engineering models for automation engineering, this paper will introduce a new Model-Driven Architecture based application and hardware modeling approach. Through separation of the logical application domain and the specific hardware domain the logical application is completely independent of the concrete physical plant configuration. Therefore development time for automation applications can be vastly reduced and software quality can be improved.

1. Introduction

Emerging low-labor-cost countries are urging their way into the markets and are increasing the pressure of competition, especially in the low-cost mass market segment, and so high-labor-cost countries have to evolve and strive for the high-quality mass customization market segment. Therefore it is necessary to optimize the production process towards fast process reconfiguration, but also fast plant reconfiguration. However the prevailing control application programming techniques are not suited for fast plant or process reconfiguration.

The problem is that control applications are implicitly hardware dependent, furthermore there are also configuration dependent, as the traditional control application programming techniques tend to mix logical functionality with hardware access methods (see Figure 1). Due to the heavy usage of hardware interfaces this is not surprising, but that is what renders fast reconfigurability and control application reuseability impossible [8]. Another problem arises if we consider typical plant life cycles. As

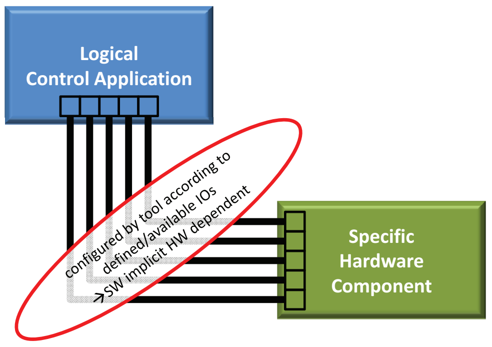

Figure 1. Current Control Application Programming Method

the typical life span of a plant exceeds 20 years it is very likely that some machine parts will not be available over the whole plant life cycle. If such a scenario should arise then the part of the control application concerning this machine part has to be rewritten and retested, which can lead to additional plant downtimes and therefore costs.

The goal of this paper is to show a new modeling approach based on an adapted Model-Driven Architecture approach, which separates the logical control flow from the hardware plant model. Therefore the logical control application will be reuseable on several hardware configurations as long as each hardware configuration is able to execute the logical control application.

2. Model-Driven Architecture

For full application reuse and therefore an abstraction of the hardware configuration of the software solution we suppose an approach from the software development domain to provide a proper solution. Within the software development domain Model-Driven Software Development (MDSD) promises to provide more efficient software development by concentration on the problem domain. MDSD achieves this by the use of Domain Specific modeling Languages (DSL) and transformations of the defined models into executable code. DSLs thereby describe the behavior and the structure of the concerning domain. The scope of the desired domain has to be delimited sufficiently to enable DSL transformations and also an adequate Meta-Model (MM) has to be defined. MMs always describe a specific model type and therefore models can be understood as instances of a specific MM. According to [1] Model-Driven Architecture (MDA) refines this approach by defining a system through models on different layers of abstraction and therefore multi-layered model transformations to different implementation technologies. MDA separates the system behavior specification from the way the platform capabilities are used by that system.

The central steps performed within MDA are according to [6] and [1], which are shown in Figure 2. The numbers in the figure are according to the steps presented here:

1. A system is specified independently from the supported platform by a Platform Independent Model (PIM). The PIM has to be independent of a specific technology as well as of a vendor dependent platform.

2. Possible system platforms are described by a Platform Description Model (PDM). The PDM includes the services provided by the platform as well as the platform's structure.

3. A Platform Specific Model (PSM) is generated by proper transformation rules. The PSM describes how the system uses the particular platform type as well as the technologies and services provided by this platform.

4. Implementation Specific Model (ISM) is generated out of the PSM and includes implementation and runtime details. Therefore the ISM can be understood as the actual code.

Fig. 2 illustrates these steps of MDA. The central model transformation within MDA is the PIM to PSM transformation.

Within MDA portability, interoperability, and reusability are achieved by the architectural separation. Therefore MDA sustains platform independence of software components and facilitates our needs of greater hardware abstraction.

3. Proposed Approach

The existing approaches of [7], [5], [3], and [2] all tried to create an application model suitable for reuse of automation applications and devices. However all have the major drawback that the needed logical application model is enriched with hardware-specific information needed to create the actual code for the automation devices. The Model-Driven Architecture (MDA) approach forces the

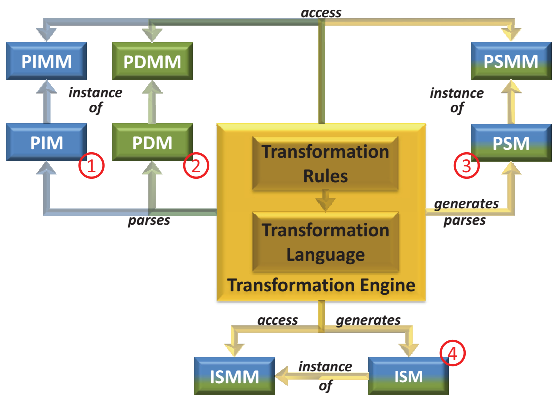

Figure 2. The different Models and steps of the Model-Driven Architecture approach: Platform Independent Model (PIM), Platform Description Model (PDM), Platform Specific Model (PSM), Implementation Specific Model (ISM), Meta-Model (MM)

strict separation of the logical application and hardware-specific details. The field of embedded device engineering already successfully applied the MDA approach and as the problems in the field of automation engineering are similar an MDA based approach should be successful as well.

Within our approach the PIM represents the logical hardware-independent control application, the PDM is an exact model of the concrete plant (from now on referred to as Plant Model) with all its devices, actuators, and sensors. The PSM is the unparameterized but already hardware-specific execution code for specific automation devices, from now on referred to as Hardware-Specific Control Application, where unparameterized means, that the needed functions are fully implemented, but the specific I/O ports are still generic names. The parameterized execution code is equivalent to the ISM.

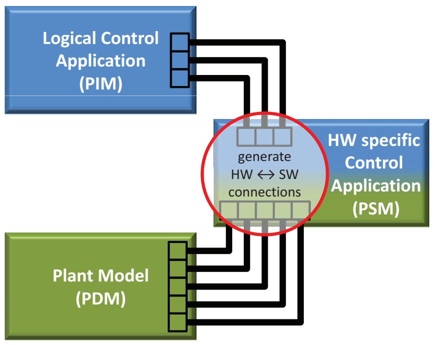
Logical Control Application (PIM)
generate HW ↔ SW connections
HW specific Control Application (PSM)
Plant Model (PDM)

Figure 3. Mapping-based Control Application Generation

This approach enables the automation engineer to use the logical application specification (usually only used as an implementation guide) immediately as a usable model for code generation. The linking between the logical application model and the hardware-specific plant model are the cornerstones of code generation (see Figure 3).

Furthermore it is possible to describe each problem in a domain-specific language, specialized for the corresponding problem domain and therefore best-practice languages can be used for problem description. This "best of breed" approach increases development performance, which is of paramount importance to decrease development costs. This approach enables the ability of fast plant and application reconfiguration as an additional gain.

3.1. Logical Control Application

The logical control application is the pure control logic, comparable to a working schedule, which describes the production process, for example a refinement process or a assembly process. The logical control application divides the task into subtasks relevant for the process (e.g., drilling, milling, assembly, varnishing). Those subtasks are enriched with the necessary parameters needed for the production process (as object coordinates for drilling or varnishing colors and areas), and process-specific parameters (drying times, min/max process temperatures, humidity, etc.).

IEC 61499 is suited as logical control application meta-model, as the standard defines hardware-independent data types which are exchanged by an hardware-independent interface. Processes and steps of procedures can be modeled as function blocks. The event-driven architecture can be used to model the correct execution sequence.

3.2. Plant Model

The Plant Model describes the exact configuration of the actual plant to be programmed. It has to include the exact machine types and their capabilities, transport routes and mechanisms (conveyor belt, etc.), the distances between the machines, the communication parameters of the devices (e.g., latency times, real-time and non-real-time capable channels) and, if applicable, the wiring plans between the control elements and the machines. That means the Plant Model is a detailed CAD-like plan of the plant, where it is possible to identify and extract the spatial distances, machine functions, communication methods, etc. from it.

The meta-model defined in the mechatronic approach of [7] is used as a sub model of the plant model. Its purpose is to describe the mechanical and electrical attributes of each automation devices in the plant.

Ref. [9] has shown that the Field Device Configuration Markup Language (FDCML) is also suitable as a basis for a Plant Meta-Model, since the FDCML semantic is able to describe the complete specification of automation devices. It allows the description of the supported communication interface and protocols, I/O numbers and voltages, CPU, or memory. Possible missing properties can be added to the model in terms of specific properties.

3.3. Hardware-Specific Control Application

The Hardware-Specific Control Application is generated by mapping the logical tasks of the logical application model to suitable machines of the Plant Model. Usually this is a manual mapping, but semi-automatic mapping would be feasible. Apart from the application-specific parameters the implementation of these tasks is precoded in a hardware-independent language, like C/C++, Java, or IEC 61499. In the case of IEC 61499 it is possible to directly link the logical application blocks to the specific hardware through the use of hardware-specific adapters for the task function blocks. Such adapters introduce the polymorphism and interface inheritance mechanism [4]. Based on the application task and the specific-machine type the unparameterized code is generated.

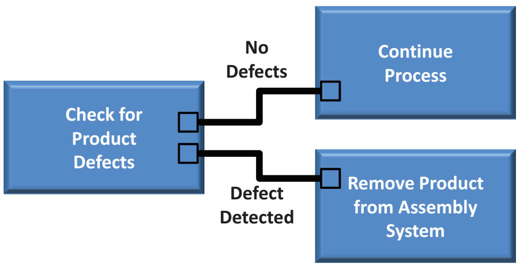
Check for Product Defects
No Defects
Continue Process
Defect Detected
Remove Product from Assembly System

Figure 4. Generic Sample Application Model

4. Example of Use

Our approach is demonstrated by an 2-axis handling unit. The first task is to define the logical control application, which is in this simple case the task to remove a defect product from the system. A simple logical control application block model of this task is shown in Figure 4, where a measurement system (the logical application block named *Check for Product Defects*) checks the product for defects. If no defect is found the production process continues, otherwise the product is removed from the production process.

The second task is to define a mechatronic model of the device for usage in the Plant Model. For this the component-based mechatronic model based on the model of [7] is used.

The main component of the simplified model is the "Handling Unit" node, which represents any handling unit imaginable. Different construction variations of handling

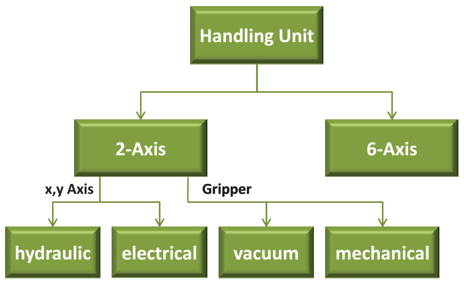

Figure 5. Mechatronic-Component Model of Handling Units Showing Different Realization Variations

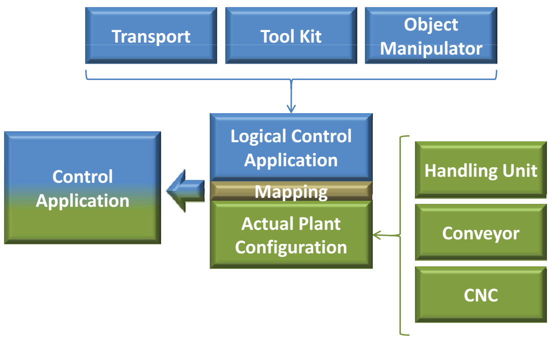

Figure 6. Model Mapping

units exist which have to be taken into account in the model, as one, two or more axes constructions are possible. As seen in Figure 5, this is realized by subcomponents which specialize the model of a handling unit in respect to the number of axes. Furthermore different types of axes can be used in such a handling unit, therefore the 2-axis component has two connection points to define the type of each axis. Possible axis types in this model are the hydraulic and the electrical axis (other types such as pneumatic would also be possible). Another port of the subcomponent defines the mounting tool (vacuum or some sort of mechanical gripper). This component-based approach allows the construction of different handling uni types by combining sub-components to a component. The example handling unit consists of two pneumatic axes and a vacuum gripper. This special device is saved in a mechatronic device library for further use in this or following projects. Therefore the reuse of mechatronic components is ensured.

Through mapping of the logical control application on the specific device the unparameterized code is generated, as shown in Figure 6. As the concrete handling unit is known at this point, and therefore also the imposed constraints, as well as the algorithms needed for this special device, automatic code generation is possible. The last step is to specify which device shall control the handling unit. Based on this information the parameterized control application, with all I/O's, communication interfaces and other needed parameters are determined and set, so that the deployable execution code is generated.

5. Conclusion and Further Work

Industrial automation is striving for handling the complexity imposed by flexible adaptive production systems. One of the biggest problems is the control software development process. Currently applied technologies are partly 20 years behind modern software engineering practices. Therefore the research in the last years has focused on how to improve the control software development process. Techniques applied are object orientation or component orientation in order to improve the structuring and foster reuse of control application parts. However these approaches have the common problem that the software elements are tightly interwoven with the I/Os they need for the performance of their control actions. This reduces reusability and increases the configuration and change effort when the system structure changes.

In order to overcome these limitations we proposed an MDA based approach. There we separate between the logical control application and the plant model. The logical control application models the intended behavior of the control application in a target independent way. The plant model defines the control devices, their abilities, and their interconnections (e.g., communication system). By mapping both models together the control code executed in the control devices with their hardware specific parameters can automatically be generated. Our concept shows that this greatly reduces the dependability of control applications on the physical hardware.

Acknowledgment

The work leading to this invention is supported by the Austrian Federal Ministry for Transport, Innovation and Technology and its FIT-IT (Research, Innovation, Technology Information Technology) research program. Further information is available at: www.frontics.org

References

[1] S. Beydeda, M. Book, and V. Gruhn. *Model-Driven Software Development*. Springer-Verlag Berlin Heidelberg, 2005.
[2] G. Cengic, O. Ljungkrantz, and K. Akesson. A Framework for Component Based Distributed Control Software Development Using IEC 61499. In *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on Digital Object Identifie*, pages 782–789. IEEE, 2006.
[3] S.-M. Lee, R. Harrison, and A. A. West. A Component-based Distributed Control System for Assembly Automation. In *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference*, 2004.
[4] R. Lewis. *Modelling control systems using IEC 61499, Applying function blocks to distributed systems.* IEE - The institution of Electrical Engineers, London, United Kingdom, 2001.
[5] C. Maffezzoni, L. Ferranini, and E. Carpanzano. Object-Oriented models for advanced automation engineering. *Control Engineering Practice*, 7:957–968, 1999.
[6] Object Management Group, Inc. MDA Guide Version 1.0.1. Website, 2003.
[7] A. Storr, J. Lewek, and L. R. Modeling and Reuse of Object-Oriented Machine Software. In *Proceedings of the European Conference on Integration in Manufacturing*, pages 475 – 484, 1997.
[8] K. C. Thramboulidis. Model Integrated Mechatronics: An Architecture for the Model Driven Development of Manufacturing Systems. In *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference*, 2004.
[9] M. Wenger, A. Zoitl, R. Froschauer, M. Rooker, G. Ebenhofer, , and T. Strasser. Model-driven Engineering of Networked Industrial Automation Systems. In *Accepted at: 8th IEEE International Conference on Industrial Informatics (INDIN)*, 2010.