Utilizing IEC 61499 in an MDA Control Application Development Approach

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

***Abstract*— Traditional control application engineering techniques tend to mix logical functionality with hardware access methods. This greatly impedes reusability. Through separation of the logical control application domain and the specific hardware domain the MDA (Model-Driven Architecture) proposes a solution to this problem. In the domain of embedded systems development this approach brought a great advantage in reducing the complexity of the development process. In this paper we investigate if and how IEC 61499 can be utilized as a DSL (Domain Specific Language) for control application development in the domain of industrial automation systems. We will show the potentials of our work by developing a sample control application using the suggested methods.**

I. Introduction

Manufacturing industries in high-labor-cost countries are under an increasing pressure, especially in the low-cost mass market segment, by emerging low-labor-cost countries which are urging their way into the markets. In order to stay profitable manufacturing industries in high-labor-cost countries have to evolve and strive for the high-quality mass customization market segment. For this market segment it is necessary to align their production process towards fast and cheap process reconfiguration as well as fast and cheap plant reconfiguration. However the commonly used control application programming techniques are not suited for fast plant, process reconfiguration, or even initial setup.

One of the biggest issues in control application development is the intrinsic hardware dependability of the developed control applications. The reason for this is that traditional control application programming techniques tend to mix logical functionality with hardware access methods. The heavy usage of hardware interfaces renders fast reconfigurability and control application reuseability impossible [1]. Another problem arises if we consider typical life cycles of plants. As the typical life span of a plant exceeds 20 years it is very likely that some machine parts or control devices will not be available during the whole plant life cycle. If such a scenario arises the part of the control application concerning this machine part has to be rewritten and retested [2]. This can lead to additional plant downtimes and therefore costs.

To solve this problems software-reuse paradigms from software engineering disciplines were used in the field of automation engineering. The two most famous software-reuse paradigms are the object-oriented paradigm [3] and the component-based paradigm [4]. Both paradigms support software-reuse in software engineering applications, but fail to unravel their full potential in control applications, as they do not take the heavy usage of hardware-interfaces and the arbitrary allocation from I/O points to devices into account [5]. For instance the common way to implement a digital output interface in the object-oriented paradigm would be to model the digital output as an object. The object variables would hold the information about the interface configuration, whereas the methods would describe the necessary steps to set or reset the digital output. This hardware-specific digital output object would be reusable in other control applications if the same kind of hardware is used. However the main focus should not be to reuse singular elements of a control application. In fact the main focus should be the reuse of the whole (or at least large parts of the) control application for a specific automation process. The object-oriented approach does not allow this kind of reuse, because the relationship between objects form the application and the reuse is additionally complicated by the fact that most objects are hardware dependent and configured for a special use case [6]. The component-based software engineering approach tries to develop use case-independent deployable software components [7]. This approach was inspired by other engineering disciplines, e.g., the mechanical engineering discipline, where such components, like standardized bolts, have already been in use for a long time. In turn the component-based software design approach inspired the mechatronic component modeling approach in control engineering. The goal of this approach is to combine the mechanical, electrical and software parts of a component into one universal mechatronic model [1], [8]. But this approach has also the weakness that each component is inherently hardware-dependent.

The goal of this paper is to present a design method based on the IEC 61499 standard which allows the definition of logical control applications reuseable on different hardware setups with minimal manual configuration effort. Therefore we will at first introduce an example we want to use for

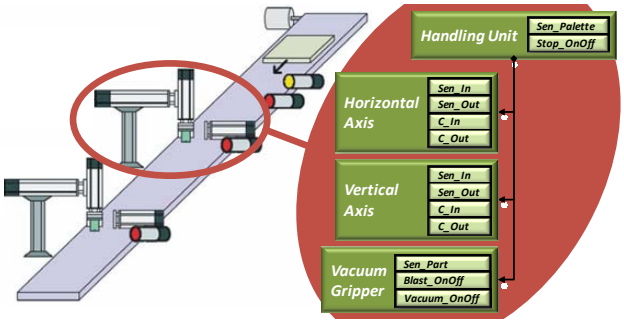

Fig. 1. Sorting Machine Example with the Logical Structure of one Handling Unit

demonstration issues. Afterwards we explain the current hardware access methods supported by the two most known concepts named IEC 61131 and IEC 61499. We will also highlight the problems of the current design methods. And last but not least we propose a new modeling approach based on a concept from the software development domain allowing hardware abstraction.

II. Automation System Example

For supporting our investigation we use a pick and place component of our sorting machine as illustrated in Fig. 1. The parts transported on the conveyor belt are identified by an index station and afterwards sorted by one of the two Handling Units. Each Handling Unit is activated by a component of the parent level according to the part type transported on an incoming palette. Every palette is detected by a sensor *Sen_Palette* and if necessary stopped by a pneumatic stopper *Stop_OnOff*.

The Handling Unit itself consists of a horizontal and a vertical axis as well as a Vacuum Gripper. All parts of the Handling Unit are pneumatically operated. Both axes have two final position sensors *Sen_In* and *Sen_Out* as well as two control valves *C_In* and *C_Out* for moving in and out. Every part taken by the Vacuum Gripper is picked up by generating a vacuum (*Vacuum_OnOff*). An established vacuum is detect by a sensor *Sen_Part*. The vacuum has to be switched off and a short blast (*Blast_OnOff*) is generated to effectively drop the part.

Therefore every component within this two layered hierarchy of the Handling Unit needs hardware access.

III. Control Application Modeling in IEC 61131-3

The IEC 61131-3 standard is the most used approach in today's automation systems. In the following we explain and demonstrate the modeling approach supported by this standard.

A. Hardware Access in IEC 61131-3

In current IEC 61131-3 based control applications there exist several ways for hardware access. Within IEC 61131 PROGRAMs have direct hardware access whereas FUNCTION_BLOCKs are only supposed to have indirect hardware access due to the use of global variables. FUNCTIONs are not supposed to have any hardware access. [9, p. 45,47]

Within IEC 61131 we identified three different ways

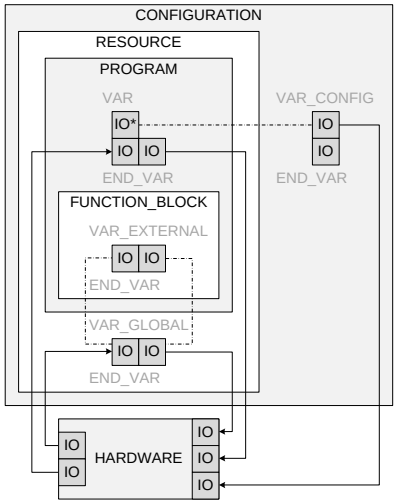

Fig. 2. Hardware access methods within current IEC 61131-3 control applications

to use hardware access for PROGRAMS and FUNCTION_BLOCKs. Fig. 2 illustrates those methods according to the IEC 61131-3 standard.

The first one is the use of global variables and assigning them proper hardware addresses. The standard introduces VAR_EXTERNAL [9, p. 43] definitions within a PROGRAM or a FUNCTION_BLOCK for accessing global variables defined in a RESOURCE or PROGRAM. Such a definition has to conform to the definition of the corresponding global variable. The concept of external variables is not always considered within IEC 61131-3 tools since global variables can be accessed directly by their symbolic names within PROGRAMs, FUNCTION_BLOCKs, and FUNCTIONs. Therefore this method can also provide hardware access for FUNCTIONs in currently available IEC 61131-3 tools.

This approach can lead to quite long lists of global variables. As longer such list as harder the maintenance will be, especially when hardware changes lead to different input and output numbers or port sizes. If the used IEC 61131-3 tool is implemented according to the standard hardware, access is provided for different Program Organization Units (POUs) through a repeated definition in terms of VAR_EXTERNAL. Due to such rather distributed and respectively repeated hardware definitions every change of a global variable results in searching through the whole control application and updating the corresponding external variables. Such an approach also causes duplicate information within IEC 61131-3 models.

today's IEC 61131-3 tools still do not support an automatic adaption of external variables dependent on global variable changes. A name based matching is supported by the tool at [10]. This matching is not automatic but based on user request.

An alternative to the use of VAR_EXTERNAL is provided by interfaces. Therefore all global variables and respectively hardware access needed in a lower level of the hierarchy have to be passed through the whole tree down to the desired POU. A hardware change might lead to a change of all POUs

contained in the branch to the desired POU. This seems to be the most used applied method according to the IEC 61131-3 standard.
The last possibility is the use of VAR_CONFIG [9, p. 39]. Those variables are only allowed within CONFIGURATIONS and can only be used for variables marked with an asterisk in a POU. A hardware change might lead to a change within the configuration variable definitions as well as the corresponding POU declarations using the asterisk notation. VAR_CONFIGs are not used very often in industrial applications.

B. Hardware Access Example in IEC 61131-3

Applying the IEC 61131-3 modeling approaches the Handling Unit example introduced within Section II might lead to a control application consisting of the following components:

- PROGRAM *handlingUnit*: representing the Handling Unit and managing its subcomponents according to its sensor signal which is polled every 10ms and indicates if a new palette has arrived. Since PROGRAMs have direct hardware access we can define proper internal variables.
- FUNCTION_BLOCK *verticalAxis* and respectively *horizontalAxis*: are instances of the same FUNCTION_BLOCK type and parameterized according to the specific hardware addresses specified in terms of global variables of the RESOURCE or PROGRAM. Therefore each axis is able to move to one of two end positions when its valve is opened or closed.
- FUNCTION_BLOCK *vacuumGripper*: in case of asterisk use all variables have to be defined in the CONFIGURATION section in terms of VAR_CONFIGs.

In general the concept of global variables and the hardware access support is not consistent within the IEC 61131-3 standard. There is a lot of freedom according to the implementation of IEC 61131-3 compliant control applications, which further depends on the supported means provided by the engineering tool.

IV. Control Application Modeling in IEC 61499

The IEC 61499 standard is the successor standard of IEC 61131-3 but still mainly used within research facilities. In the following we explain and demonstrate the modeling approach supported by this standard.

A. Hardware Access in IEC 61499

IEC 61499 proposes hardware independent modeling of control applications but in fact the situation is similar to IEC 61131-3.
Control applications according to IEC 61499 are modeled in three steps. At the beginning the platform independent application model is created also referred to *Application Model* [11, pp. 21ff]. Besides the logical behavior of the system implemented in terms of Composite Function Blocks (CFBs) and Basic Function Blocks (BFBs) this model also contains parts for hardware access. Hardware access is established by Service Interface Function Blocks (SIFBs) [11, pp. 43-54] allowing the isolation of hardware dependencies and keeping the logical functionality separated from physical access. As long as those hardware accessing SIFBs are instantiated at the highest level they can be parameterized as desired. Usually there exits a deeper hierarchy and the hardware accessing SIFBs are instantiated within CFBs. Since CFBs represent types instantiated within the application parameterizing of internal Function Block (FB) input values is not possible. Therefore IEC 61499 makes use of an inside-out approach [12] where all hardware specific parameters have to be passed to the interface of a higher hierarchy.
Within the second step the *System Model* is designed. This model contains the structure of the system in terms of devices and a proper communication infrastructure represented by *Segments* and *Links*.[11, pp. 18f]
The last step is the creation of the *Distributed Model* [11, p. 26] by mapping of the application on the desired control devices. This distribution process may cause a gap between application parts running on different devices. Such *Device Models* [11, pp. 19f] allow device dependent parameter specification since the mapped application parts are not logically linked but a real copy of the mapped application part. The gap between application parts introduced in the mapping has to be closed by communication connections. The communication connections are realized by inserting proper communication SIFBs [11, pp. 47f]. Currently there is only one IEC 61499 tool which automates this insertion, therefore this has to be done manually in most cases. Those communication SIFBs also cause a separation of a consistent communication infrastructure.

B. Hardware Access Example in IEC 61499

Fig. 3 shows the IEC 61499 implementation of the picking sequence for our handling unit example introduced in Fig. 1. The platform independent application model at the top of Fig. 3 implements the control for the gripper, the vertical axis, and the horizontal axis. The control algorithm of the vertical axis depends on the actual position of the horizontal axis.
The mapping is represented by the dashed arrows. In our example the horizontal axis is mapped to *Device 1*, and the vertical axis as well as the gripper are mapped to *Device 2*. The sensors and actuators used within the platform independent application model have to be configured according to the specific device as well as the communication between the mapped application parts.
The horizontal axis is designed as CFB containing SIFBs for hardware access. In our example the parameters of the *P_Cylinder* FBs and those of the *V_Gripper* have to be specified. The hardware specific parameters of these three FBs and the corresponding SIFBs therefore provide the *Process Interface* for interaction with the controlled process. The *Network Interface* is provided by the *SEND* and *RECEIVE* SIFBs, added to the device specific applications.
The modeling within IEC 61499 is quite straight forward, but the more complex the system the more parameters have to be specified. Usually more complex systems also have

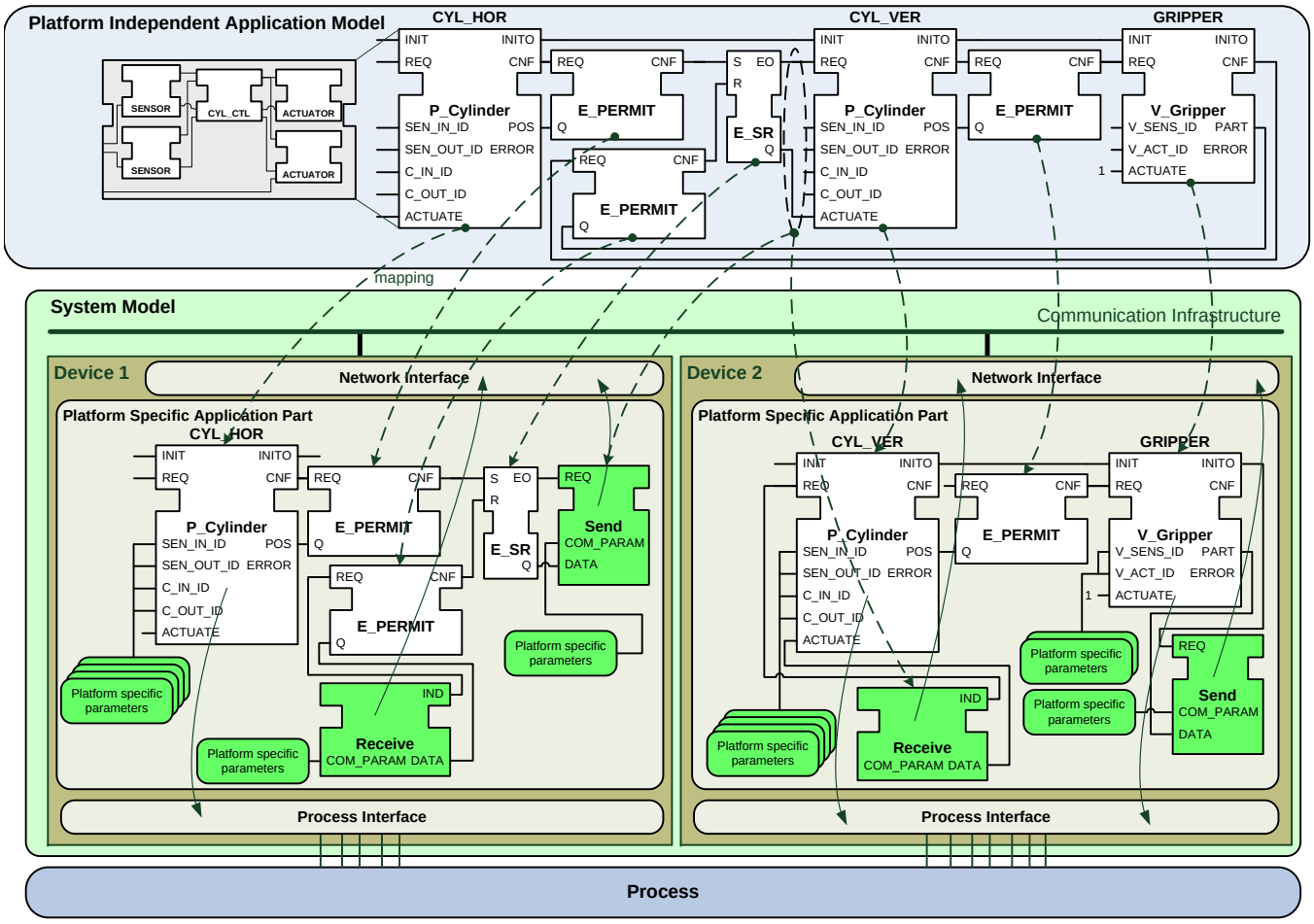

Fig. 3. Overview on the platform independent application model of IEC 61499 and its relation to the control devices in a distributed control system

several hierarchy levels causing more passing efforts for the parameters and also more effort when the hardware is changed.

V. Control Application Modeling in the Software Development Domain

The application and hardware interweaving of today's IEC 61131-3 tools makes PLC software reuse hard. The approach provided by the IEC 61499 standard also lacks in a clearly defined point of hardware intersection.

The software development domain provides a promising method for hardware abstraction we also want to introduce as a basis for improvement of today's modeling approaches. The concept of Model-Driven Architecture (MDA) describes a system through models on different layers of abstraction. Therefore the different aspects of a system are described by different models. The platform capabilities used by a system are therefore separated from the system behavior specification. The designed models are subsequently transformed into executable code for different implementation technologies. The different models specified by the MDA concept according to [13] and [14] are:

- PIM: The Platform Independent Model describes the system independently from the supported platform and therefore contains no specific technology or vendor dependent platform. The PIM defines the logical functionality of a control application.
- PDM: The Platform Description Model describes the platforms available for the desired system and therefore contains services provided by the platform as well as the platform's structure. The PDM describes the hardware supposed to be operated by the control application.
- PSM: The Platform Specific Model is generated out of the PIM and the PDM. It therefore describes how the system uses the particular platform type as well as the technologies and services provided by this platform.
- ISM: The Implementation Specific Model is generated out of the PSM. It therefore contains implementation and runtime details and equals actual code.

Fig. 4 illustrates the different models of MDA and respectively their generation flow. MDA achieves reusability by architectural separation and platform independence of software components allowing an easy implementation on a variety of targets. This feature greatly improved flexibility and efficiency of the development process for embedded systems [15].

VI. Control Application Development Approach

The IEC 61131-3 standard is currently the preferred Domain Specific Language (DSL) within the automation domain

Fig. 4. The different Models of the Model-Driven Architecture approach: Platform Independent Model (PIM), Platform Description Model (PDM), Platform Specific Model (PSM), Implementation Specific Model (ISM)

but it only provides an interweaved concept of hardware modeling which makes reusability difficult to achieve. Also testing and simulation are hindered by hardware dependent control code.

Due to the very conservative automation domain a new concept should use a similar design language but provide application separated representation of hardware. It should also support complex systems since reusability problems are in proportion to complexity. The IEC 61499 standard provides a more suitable environment and hardware independent way of design therefore we want to apply the concept of MDA to the IEC 61499 standard.

The usage of the MDA approach clarifies the explicit split of the application modeling from the hardware also provided by the IEC 61499 standard. Since there is still better hardware abstraction needed we identified additional concepts supposed to simplify hardware configuration of complex systems.

A. Mapping MDA to IEC 61499

System design in IEC 61499 can be divided into four steps, the functionality modeling, the hardware configuration, the mapping as well as the code generation and deployment. This design method is quite similar to those of MDA [16] used for the development of complex software and embedded systems.

The first design step of IEC 61499 is the functionality design of the whole system. In IEC 61499 the functionality model is called *Application Model* and corresponds to the PIM of MDA. The *Application Model* is fully executable and therefore contains a detailed specification of its functions and compositions of its sub-functions. Just as the PIM an *Application Model* should not contain any hardware specific information neither for devices nor for communication protocols.

The second design step of IEC 61499 is the hardware configuration. The details of a particular hardware configuration including devices and the communication infrastructure are taken into account in terms of the called *System Model*. The *System Model* corresponds to the PDM of MDA.

The third design step of IEC 61499 is the mapping. During this step the functionality of the *Application Model* or PIM is mapped onto the hardware *System Model* or PDM, resulting into the *Distributed Model* or the PSM of MDA.

The fourth design step of IEC 61499 is the code generation and deployment. During this step the platform specific code is generated and deployed onto the specific hardware. The platform specific code corresponds to the ISM of the MDA concept. The applied MDA concept will be illustrated based on the example described in Fig. 1.

4DIAC-IDE [17], an engineering environment for distributed automation systems based on the IEC 61499 standard, already allows the mapping of application parts on different devices. But communication blocks for the specific device have to be added manually. The descriptions of devices and segments do also not provide specific parameters for communication facilities or I/Os.

B. Hardware Abstraction for IEC 61499

As mentioned before hardware access in IEC 61499 is encapsulated by SIFBs. As these blocks are connected to the remaining control application via event and data connections which renders the whole application hardware dependent. Hardware changes lead to exchange of the needed hardware-access SIFBs. To circumvent this problem the following approach is used:

The SIFBs connected with the control application do not represent specific hardware access methods anymore. Instead they represent certain logical services the hardware provides to the control program. For instance the movement of an axis, gripping and releasing of an object, or the reading and writing of I/Os. Such blocks will be named Logical Service Function Blocks (LSFB). Each type of service specifies a minimal logical interface needed for the control program flow, hardware-dependent parameters are fully avoided. Furthermore such an LSFB does not implement any functionality. It defines an InterFace Contract (IFC) for hardware-specific implementation FBs (HSIFB). The IFC also defines the adapter interface between LSFB and HSIFB. The HSIFBs encapsulate the needed value transformations to transform the HW-specific data representation into the IFC compliant format and vice versa, and the hardware access algorithms which implement the appropriate service. The HSIFBs also extend the interface defined by the IFC with needed HW-specific parameters. At the deployment phase the appropriate HSIFB is selected and connected to the LSFB via the IEC 61499 adapter mechanism.

This approach has the advantageous side-effect that the control application does not have to be changed for testing, simulation, or online-diagnosis, as these functionality can also be covered by special HSIFB versions.

VII. Applied Control Application Development Approach

The stated hardware abstraction approach is now applied to the control application presented in Fig. 3. Fig 5 shows the new control application design. The application coordination algorithm, which represents the control logic needed for the sorting task, is encapsulated in the *Sorting_Application* FB to enhance readability. The hardware access methods are covered by the various LSFBs. Each LSFB only implements the minimal needed interface to perform its task.

Incoming palettes are announced via the *Sensor_Indication_LSFB* which represents an indicating type of hardware sensor. This means the LSFB sends an *IND* event every time the sensor value changes enough, according to some criteria (e.g., switching thresholds,

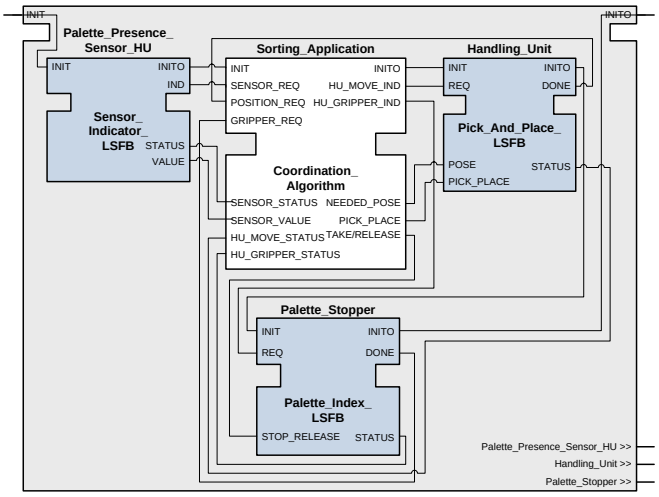

Fig. 5. Application Design According to Proposed Approach

anti-noise thresholds), to represent a significant event. In our example this significant event is the arrival and the departure of an palette. Arrival and departure can be distinguished through the *VALUE* output of the LSFB. The *STATUS* output indicates the present operational status of the sensor (e.g., if a hardware error has occurred, or if the measurement failed for some reason).

Triggered by the *IND* of the sensor LSFB the *Coordination_Algorithm* FB processes the next needed pose and if a picking or releasing process shall be initiated, based on the current state of its data inputs. Then the corresponding events for the pick and place unit and/or the indexing station are sent. The different poses of the application are represented by symbolic names (e.g., Postion1_Conveyor1, Position1_Tray1). Taking or releasing of a part is represented by symbolic names (e.g., PICK, PLACE).

During the deployment process the LSFBs are connected to the appropriate HSIFBs. The HSIFBs implement the hardware access algorithms, translate symbolic names if necessary, and provide additional HW-specific parameter inputs. Fig. 6 shows the *Pick_And_Place_LSFB* and two possible hardware-specific HSIFBs.

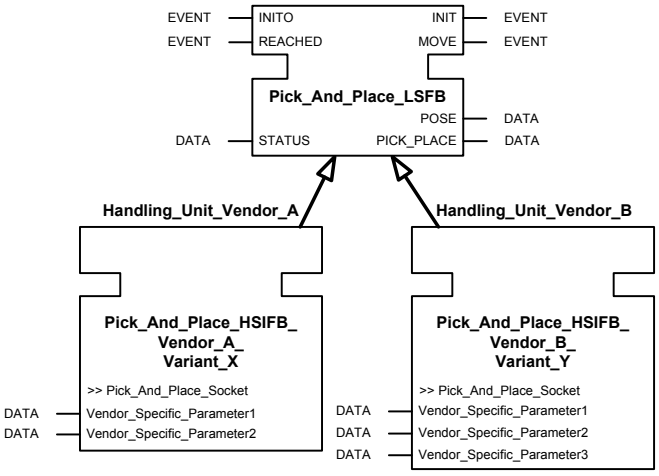

Fig. 6. The Pick_And_Place_LSFB and its possible HSIFBs

VIII. Conclusion and Future Works

In order to overcome the very interweaved hardware access modeling methods of today's control application development tools we provided a proper modeling approach based on MDA and IEC 61499. The separation of logical services and their hardware-specific implementations allows the reuse of the control application on different kinds of hardware. Hardware developers are now able to generate additional unique selling point based on their intellectual property contained in the HSIFBs. Our approach offers a much clearer, more flexible and also more reusable way of control application development, than state of the art development approaches. The next steps are the identification and standardization of LSFBs and their minimal interfaces. Also the addition of device meta data and service sequence contracts for the LSFBs and their usage for semi-automatic mapping of HSIFBs must be evaluated. Also the support for the human operator to solve ambiguous mapping situations will be investigated.

References

[1] K. C. Thramboulidis, "Model Integrated Mechatronics: An Architecture for the Model Driven Development of Manufacturing Systems," 2004.

[2] H. A. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, Oct. 2005.

[3] 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*, 1997, pp. 475 – 484.

[4] S.-M. Lee, R. Harrison, and A. A. West, "A Component-based Distributed Control System for Assembly Automation," 2004.

[5] K. C. Thramboulidis, "Challenges in the Development of Mechatronic Systems: The Mechatronic Component," 2008.

[6] C. Maffezzoni, L. Ferranini, and E. Carpanzano, "Object-Oriented models for advanced automation engineering," *Control Engineering Practice*, vol. 7, pp. 957–968, 1999.

[7] C. Szyperski, *Component Software Beyond Object-Oriented Programming*, 2nd ed. Addison-Wesley, 2002.

[8] G. Cengic, O. Ljungkrantz, and K. Akesson, "A Framework for Component Based Distributed Control Software Development Using IEC 61499," in *ETFA*. IEEE, 2006, pp. 782–789.

[9] IEC 61131-3, "Programmable controllers – Part 3: Programming languages," International Electrotechnical Commission, Geneva, Tech. Rep., 1993.

[10] logi.cals by kirchner SOFT GmbH, "logi.CAD and logi.DOC," 2008. [Online]. Available: http://www.logicals.com/products/

[11] IEC 61499-1, *Function blocks – Part 1: Architecture*. Geneva: International Electrotechnical Commission, 2005.

[12] R. Hametner, A. Zoitl, and M. Semo, "Component architecture for the efficient development of industrial automation systems," in *6th IEEE Conference on Automation Science and Engineering Proceedings*, August 2010.

[13] Object Management Group, Inc., "MDA Guide Version 1.0.1," Website, 2003. [Online]. Available: http://www.omg.org

[14] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*. Springer-Verlag Berlin Heidelberg, 2005.

[15] B. Huber, R. Obermaisser, and P. Peti, "MDA-based development in the DECOS integrated architecture - modeling the hardware platform," in *Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on*, April 2006.

[16] Object Management Group, "Model Driven Architecture," Jun. 2009. [Online]. Available: http://www.omg.org/mda/faq_mda.htm

[17] 4DIAC Consortium, "4DIAC - Open Source for Distributed Industrial Automation," Mai 2008. [Online]. Available: http://www.fordiac.org/