24

IEC 61499 ARCHITECTURE, ENGINEERING METHODOLOGIES AND SOFTWARE TOOLS

James H. Christensen
Rockwell Automation Advanced Technology
JHChristensen@ra.rockwell.com

The IEC 61499 standard defines an architecture and software tool requirements for the encapsulation, embedding, deployment and integration of intellectual property (IP) in intelligent devices, machines and systems. A reference framework and engineering methodology is presented for the use of IEC 61499 in the design, development, simulation, testing and implementation of distributed control and automation systems employing intelligent mechatronic components.

1. INTRODUCTION

Advances in hardware and software technology have made possible the embedding of unprecedented levels of functionality in end devices (sensors and actuators) for industrial control and automation. In turn, this has generated significant technical and commercial opportunities for system architectures, engineering methodologies and software toolkits capable of supporting the cost-effective development and widespread deployment of intellectual property (IP) in such devices and their composition into scalable, flexible automated (SFA) systems. The principal requirements for such architectures, methodologies and toolkits include:

- *software component* orientation for IP encapsulation, reuse and portability;
- device *interoperability*;
- the ability to *distribute* and *integrate* applications;
- *functional completeness*;
- *scalability*;
- *extendability*; and
- flexible *reconfigurability*.

Over the past ten years, Technical Committee 65 (TC65) of the International Technical Commission (IEC) has been developing a series of architectural standards for the use of *function blocks* to meet these requirements (IEC, 2001, 2002). This paper presents an overview of this architecture and illustrates its use in conjunction with an appropriate engineering methodology and software tools to meet these requirements.

V. Mařík et al. (eds.), *Knowledge and Technology Integration in Production and Services*
© Springer Science+Business Media New York 2002

2. IEC 61499 ARCHITECTURE

The fundamental unit of software encapsulation and reuse in IEC 61499 is the *function block,* considered to be an *instance* of a function block *type.* As illustrated in Figure 1, an IEC 61499 function block type includes *event* inputs and outputs as well as the more traditional *data* inputs and outputs as seen, for example, in the IEC 61131-3 standard (IEC, 2002) for programmable controller languages. In this way IEC 61499 is able to account explicitly for the synchronization between data transfer and control algorithm execution in *distributed* as well as *centralized* systems.

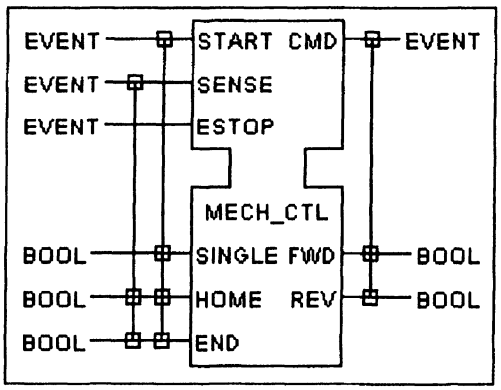

Figure 1 - Example of an IEC 61499 function block type

At the lowest level (the so-called *basic function block type*), intellectual property (IP) is encapsulated in the form of control *algorithms.* Each algorithm expresses a mapping from the current set of values of input, output and internal variables of the function block to a new set of values for its output and internal variables. These algorithms may be expressed in the programming languages of IEC 61131-3, or other procedural languages as appropriate.

Additional IP may be encapsulated in the *Execution Control Chart (ECC)* of a basic function block type. An ECC is an event-driven state machine determining the relationships among current state and input event occurrences, transitions between states, and the algorithm(s) to be executed and output event(s), if any, to be issued upon entering a new state.

IEC 61499-1 also provides *composite function block types* for the encapsulation of new IP developed through the functional composition of existing IP.

An important mechanism for interfacing to *services* provided by the underlying operating environment (called a *resource* in IEC 61499) is defined through the *service interface function block* (SIFB) construct of IEC 61499-1. This provides interfaces to such services as graphic user interface (GUI) components, timing and event handling, communications, and sensor/actuator interfaces. The externally visible behaviors of SIFBs are documented by their provider in the form of *service sequence diagrams* following the format defined by ISO 8509 (ISO, 1987), thus providing for complete protection of the encapsulated IP. The use of SIFBs in conjunction with basic or composite function blocks to provide the local portion of a distributed control application in a resource is illustrated in Figure 2.

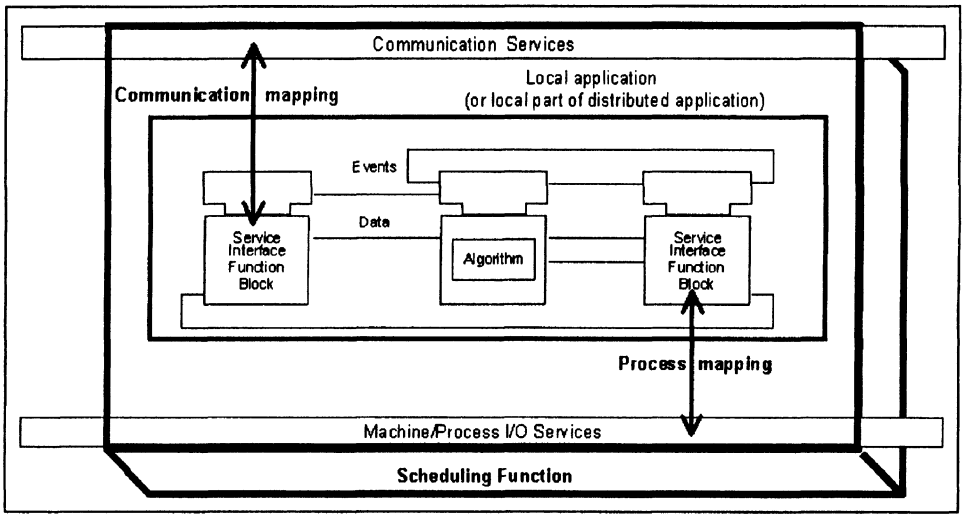

Figure 2 - IEC 61499 resource model (IEC, 2000)

The final element of the IEC 61499 architecture is the *device*, which serves as a container for multiple resources and provides them with *interfaces* to communication networks, sensors and actuators; the services provided by these interfaces are delivered by SIFBs in the resources to support the implementation of distributed applications. The communication networks in turn provide the means for integration of the devices into complete control and automation *systems*.

3. ENGINEERING METHODOLOGY AND TOOLS

IEC 61499-2 (IEC, 2001) defines general requirements for software tools for building elements defined in the IEC 61499-1 architecture (IEC, 2000). These requirements include but are not limited to: (i) reading and writing *library elements* (data types, function block types, resource types, device types, system configurations, etc.) in the standard XML (W3C, 1998) formats defined in IEC 61499-2; (ii) manipulating the *declarations* contained in the library elements; (iii) configuring *devices* and *resources* according the declarations contained in corresponding *system configurations*; and (iv) simulating and validating the operation of various library elements.

Specific software tool requirements will depend on the particular engineering methodologies employed. One such methodology (Christensen, 2000) proposed an extension of the well-known Model/View/Controller (MVC) user interface framework to encompass the development, simulation and deployment of IEC 61499-based systems. In this framework, each of the following elements would be represented as an *instance* of a function block *type*:

- **Model:** A function block that represents the time-dependent logical behavior of the system or device being controlled.
- **View:** A function block that represents the graphical display associated with one or more **Model** types.
- **Controller:** A function block that encapsulates the control functions to be performed on one or more instances of associated **Model** types, and presents

appropriate *event* and *data interfaces* for integration of its functions with those of other **Controller** blocks.

It was further suggested that as part of an associated engineering methodology, **Model** and **View** elements could be encapsulated together in composite **MV** function block types, and that a further encapsulation step could be used to produce **MVC** function block types. However, in practice this methodology has been found to have two major drawbacks:

1. The use of **MV** composite elements leads to *complex resource configurations* because graphic display configuration data is intermixed with model configuration data and interconnections. This complexity is further increased by the use of **MVC** elements.
2. The use of **MVC** composites makes it difficult to separate the **Controller** element from the **MV** element when configuring an actual system by replacing the MV element with appropriate actuator and sensor interfaces.

The layered architecture in Figure 3 overcomes these problems by placing functional elements of each type in a separate layer and explicitly adding a layer for human/machine interface (HMI). Elements within each layer communicate with each other via normal event and data connections. Communications between adjacent layers are implemented using communications service interface function blocks (CSIFBs) as defined in IEC 61499-1, enabling the contents of individual layers to be allocated to different devices as required. When adjacent layers are allocated to the same device, communications are implemented in an optimized way using specialized parameters of the CSIFBs.

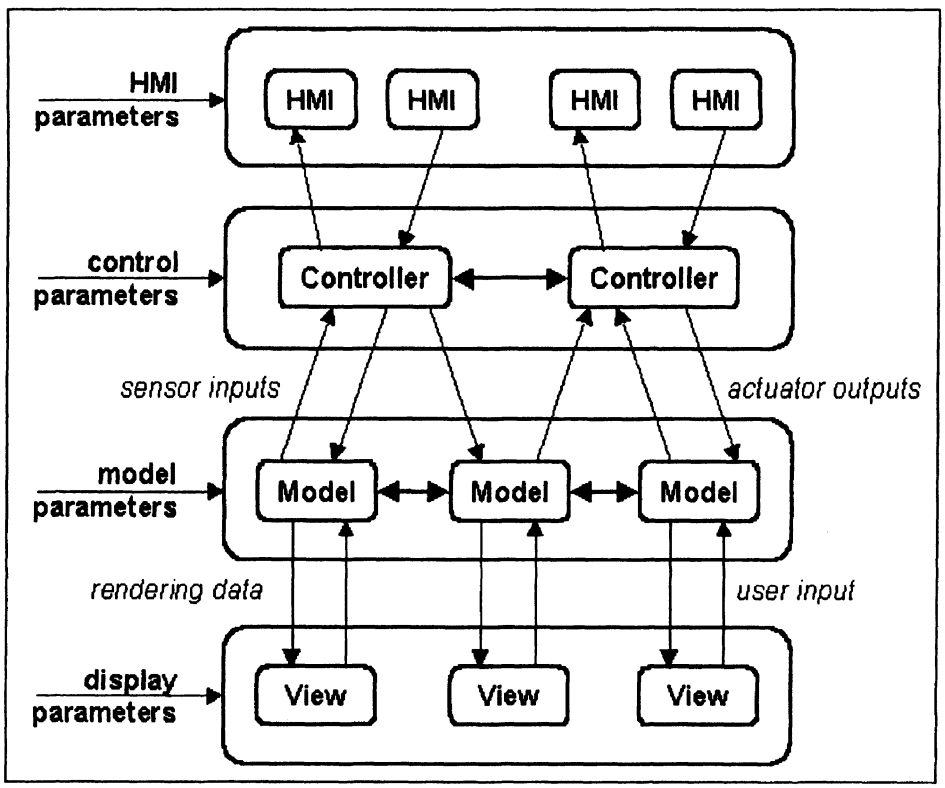

Figure 3 - Revised MVC framework

This framework enables the use of the following simplified version of the engineering methodology described in (Christensen, 2000), with the modified step numbers shown in **bold face**:

1. Start with a sketch of the machine or process to be controlled, along with a verbal description of the desired behavior.
2. From the sketch, develop and test a number of **Views** that present visually the essential information about the states of the controlled devices.
3. Utilizing the View testing mechanism in Figure 4(a), integrate the views into a static animation of the system to be controlled, and utilize the animation to develop descriptions of the desired operational sequences of the system under both normal and abnormal conditions.
4. For each view, develop and test one or more **Models** capable of simulating the dynamic behavior of the associated machine or process equipment in response to external stimuli and commanding the associated View to display the corresponding equipment states.

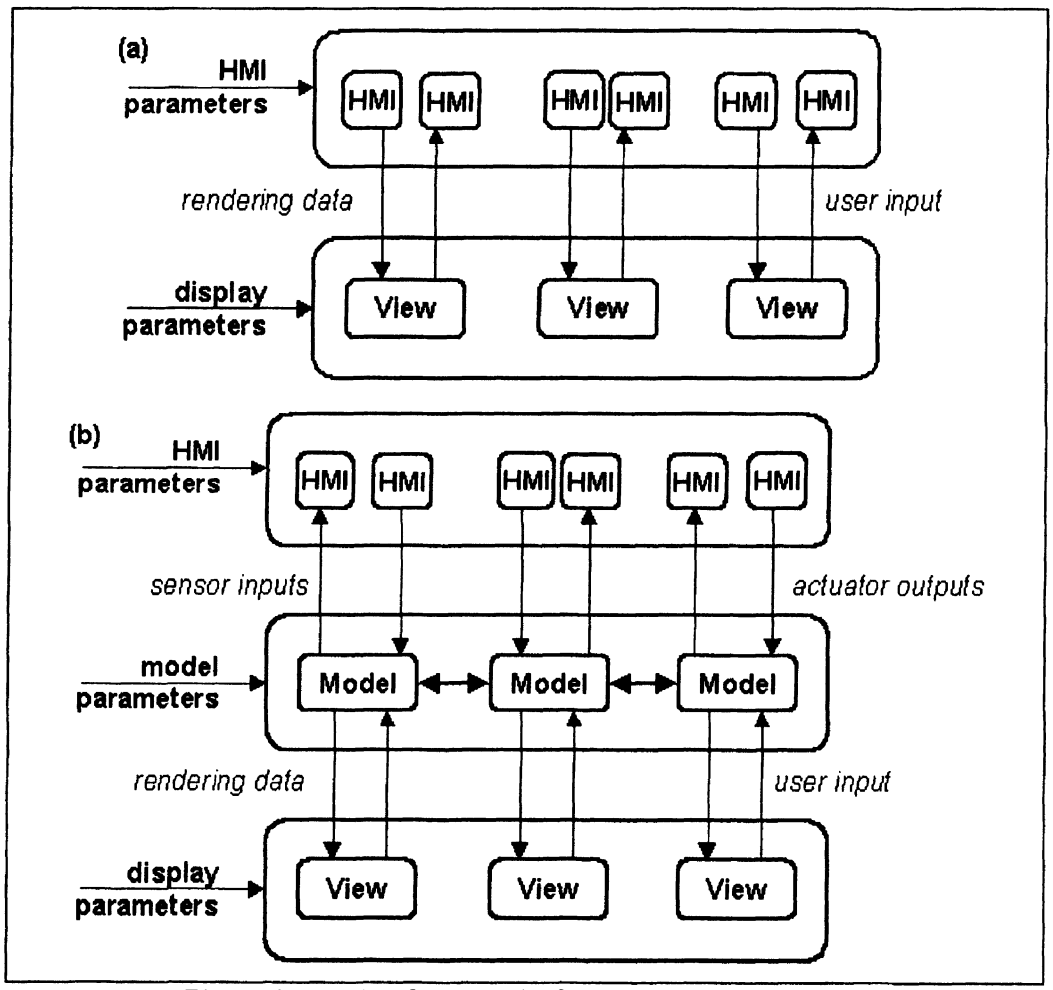

Figure 4 - Testing frameworks for: (a) Views, (b) Models

5. Use the Model testing mechanism shown in Figure 4(b) in conjunction with the previously tested Views to verify that the Models provide the correct behaviors in response to actuator inputs.
6. Develop **Controller** blocks as necessary to achieve required functions, e.g., sequencing of the simulated equipment, event and data interfaces for integration with other controller blocks. Test the Controller blocks, in conjunction with the previously developed Models and Vies, in the overall framework of Figure 4.
7. Implement the physical system as shown in Figure 5 by replacing the Model and View layers with the corresponding actual physical devices. Configure these devices to present to the Controller layer logical interfaces that are identical to the interfaces previously presented by the Model layer.

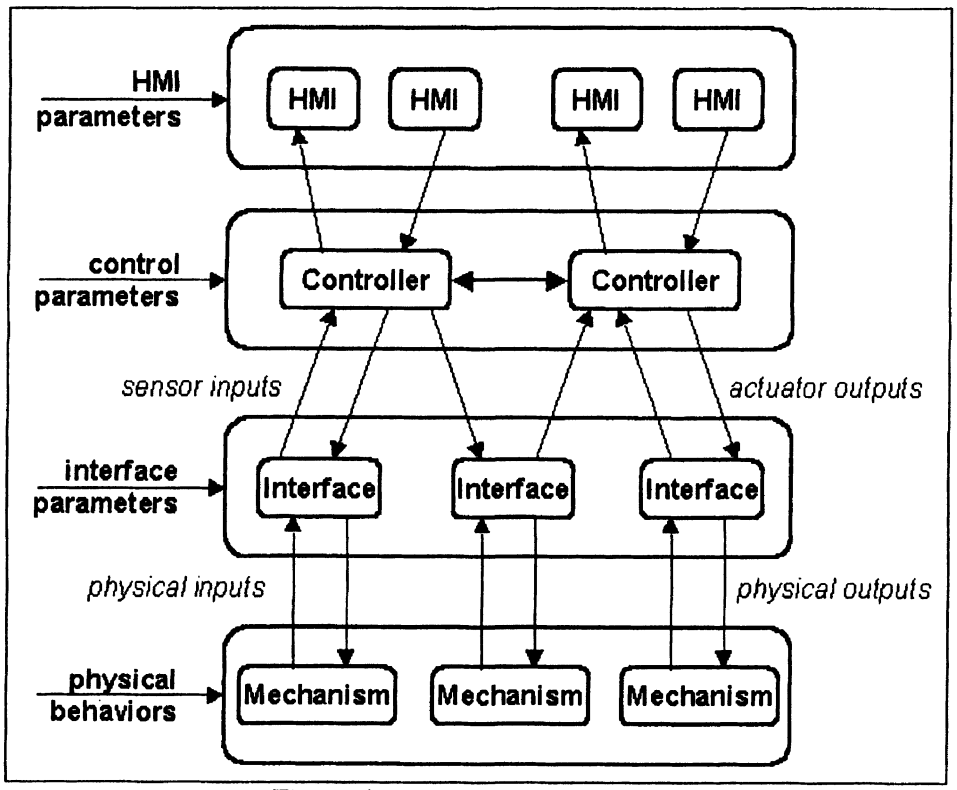

Figure 5 - Deployment framework

8. When possible, factor the Controller functions into **Low Level Control (LLC)** function blocks representing the control functions specific to physical devices and their interactions with other devices, and **High Level Control (HLC)** function blocks representing those functions which require interfaces with multiple devices, and which may utilize more complex technologies such as software agents. By allocating the LLC functionality to the physical devices, an architecture for intelligent "mechatronic" devices is created as illustrated in Figure 6, where the device boundaries are indicated by the dotted lines.
9. When possible, generalize the LLC function blocks and make them available, along with appropriate service interfaces, for reuse in libraries of intelligent mechatronic devices.

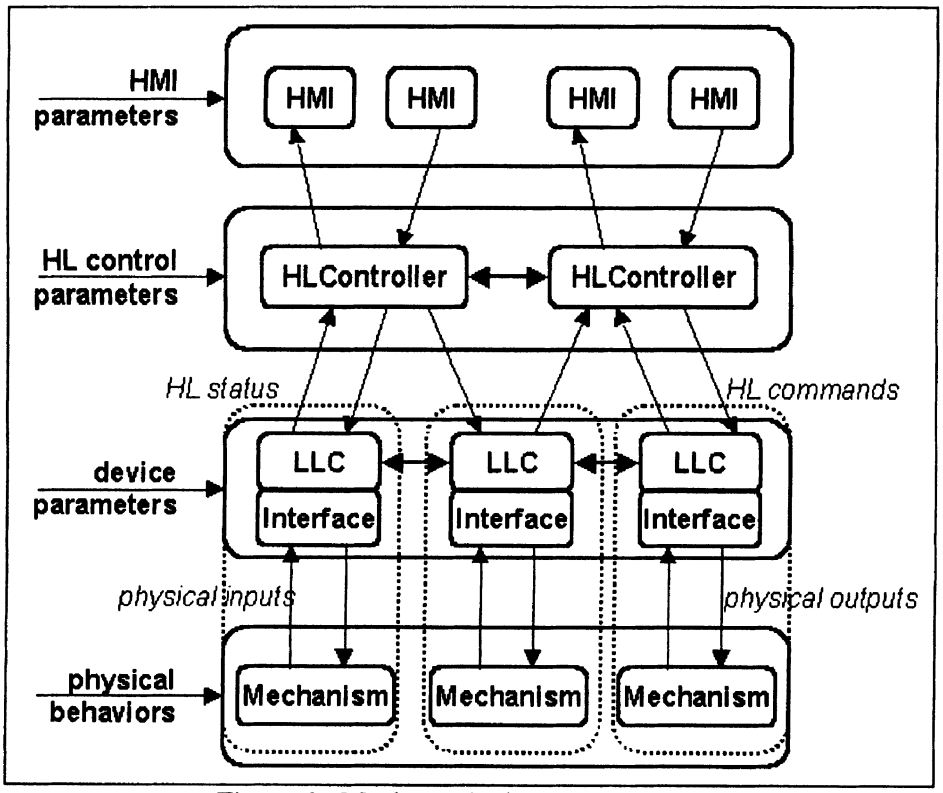

Figure 6 - Mechatronic device architecture

5. CONCLUSIONS

IEC 61499, with appropriate software tools and engineering methodologies, can be an effective way to meet the requirements for future scaleable, flexible automation. The original (Christensen, 2000) framework and engineering methodology have been successfully applied in a number of simulated and physical testbeds. A formal validation methodology and toolkit have also been applied to the original framework (Vyatkin, 2000). In addition, an electromechanical design workbench has been developed around this framework and methodology (Jain, 2002). The adaptation of this previous work to the improved framework and methodology presented in this paper, and the migration of the associated toolkits to an updated (Java 2) platform, are currently in progress.

6. ACKNOWLEDGMENTS

The author is indebted to Mr. Franz Auinger and Mr. Werner Rumpl of Profactor GmbH for pointing out the weaknesses of the original MVC framework and demonstrating the required refactoring of functionality. An immeasurable debt of gratitude is also owed to the late Dr. Odo Struger of Allen-Bradley and Rockwell Automation for his unflagging support and encouragement.

7. REFERENCES

1. Christensen, J. "Design patterns for systems engineering with IEC 61499." In Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, Ch. Döschner, ed. Magdeburg, Germany: Otto-von-Guericke-Universität, 2000.
2. IEC (International Electrotechnical Commission) 61499-1, Function blocks - Part 1, Architecture, Geneva, 2000.
3. IEC (International Electrotechnical Commission) 61499-2, Function blocks - Part 1, Software tool requirements, Geneva, 2001.
4. IEC (International Electrotechnical Commission) 61131-3, Programmable controllers - Part 3, Programming languages, Geneva, 2002.
5. ISO (International Organization for Standardization) TR 8509, Information processing systems - Open Systems Interconnection - Service conventions, Geneva, 1987.
6. W3C (W3 Consortium), eXtended Markup Language (XML) Specification, available at http://www.w3c.org/TR/1998/REC-xml-19980210, 1998.
7. Vyatkin, V., H.M. Hanisch, P. Starke, and S. Roch, "Formalisms for verification of discrete control applications on example of IEC 61499 function blocks." In Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, Ch. Döschner, ed. Magdeburg, Germany: Otto-von-Guericke-Universität, 2000.
8. Jain, S., C. Yuan and P. Ferreira, "EMBench: A Rapid Prototyping Environment for Numerical Control Systems," accepted for presentation, ASME IMECE, New Orleans, November 2002.