4. HMS/FB Architecture and its Implementation

James H. Christensen

Rockwell Automation, 1 Allen-Bradley Drive,
Mayfield Heights, Ohio 44121 USA
Email: jhchristensen@ra.rockwell.com

Abstract. Properly architected holonic systems can enhance the ability of each set of players in the industrial automation and control market to deliver added value by encapsulating, reusing and deploying their specialized intellectual property at succeedingly higher levels of integration. Such an architecture expands on the HMS concept of *cooperation domains* to include both low-level control (LLC) and high-level control (HLC) domains. LLC refers to normal, non-holonic control and automation functions, while HLC refers to the integration of these functions into holons through the use of software agent technology. *Function blocks,* as defined in the International Electrotechnical Commission (IEC) 61499 series of standards, can be used for encapsulation, reuse, distribution and integration of both LLC and HLC functions, while HLC functionality can be standardized as defined by the Foundation for Intelligent Physical Agents (FIPA).

4.1 Introduction

This chapter outlines an open, standards-based architecture for holonic manufacturing systems (HMS) which is capable of fulfilling the economic and technical requirements for global adoption, deployment and support. Issues arising in the implementation of this architecture, and means for their resolution, are presented. Throughout this chapter, the following definitions apply:

- *Architecture* The structure and relationship among *functional units* in a *system*, including their mutual *interfaces*. The architecture may also include the system's interfaces with its environment.
- *System* A set of interrelated elements considered in a defined context as a whole and clearly delineated from its environment.
- *Functional unit* An entity of hardware or software, or both, capable of accomplishing a specified purpose, and which may be composed of other *functional units*.

- *Interface* A shared boundary between two *functional units*, defined by functional characteristics, common physical interconnection characteristics, signal characteristics and other characteristics, as appropriate.
- *Holon* A *functional unit* capable of both *autonomy* and *cooperation*.
- *Autonomy* The extent to which an entity can create, control and monitor the execution of its own plans and/or strategies, and can take suitable corrective actions against its own malfunctions.
- *Cooperation* A process whereby a set of entities negotiate and execute mutually acceptable plans and take mutual actions against malfunctions.
- *Holonic system* A *system*, some of whose *functional units* are *holons*.
- *Holonic manufacturing system* A *holonic system* intended for application in the domain of manufacturing.

4.2 Architectural Requirements

This section describes the economic and technical requirements for a practical HMS architecture which is capable of global adoption, deployment and support.

4.2.1 Economic Requirements

Holonic systems will only be succesful in the industrial automation and control market if they enhance the ability of each set of players to deliver added value by encapsulating, reusing and deploying their specialized intellectual property (IP) at succeedingly higher levels of integration, as illustrated in Fig. 4.1.

This market exhibits the characteristics of a *network economy* as described by Shapiro and Varian [4.1]. These characteristics include *large network externalities* (user economies of scale), where the value of a technology increases exponentially with the number of users, and *positive feedback,* where successful application of the technology encourages additional users and vendors to enter the market.

A key ingredient of success in markets of this type is the rapid development of a whole range of *complementary products and services* which facilitate the deployment of the technology. In Fig. 4.1, these are identified as *runtime platforms, engineering methodologies* and *software tools*. Hence a major prerequisite for the success of HMS in this market is that it be *open* and *standards-based* to encourage early entry of suppliers of such products and services. The technological feature of *openness* will be defined in the next section of this chapter.

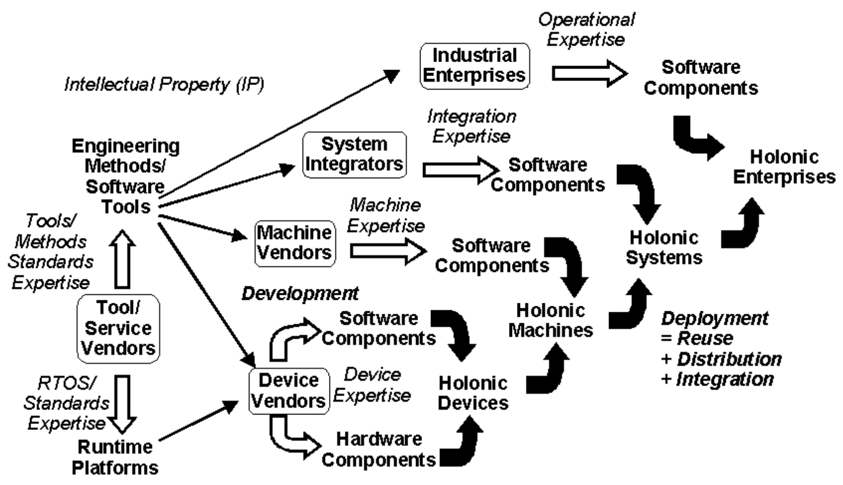


Fig. 4.1. Holonic value-add chain

4.2.2 Technical Requirements

In order to meet the economic requirements expressed above, an HMS architecture must be *component-based* to support the encapsulation and protection of intellectual property (IP). Furthermore, the components developed within this architecture must be *portable* among both software tools and runtime platforms, both in order to increase the value of the encapsulated IP and in order to encourage the emergence of a broad spectrum of software tools for component development and deployment.

HMS applications are strongly differentiated from traditional information technology (IT) applications in that they are directly involved in the control of *physical devices* and *machines*, and are highly *distributed* in nature. Hence, a successful HMS architecture must also be *inherently distributed*, and must provide a straightforward mapping from *distributed applications* to physical devices. This increases the value of the embedded IP for the device vendor, and improves the deployability and hence the value of IP components.

In order to achieve the desired end user economies of scale, the HMS architecture must be *functionally complete*, that is, capable of encapsulation and deployment of IP into components addressing all the technological requirements of holonic systems for industrial process measurement, control and automation, including:

- control and automation components,
- machine and process interface components,

- communication interface components,
- human/machine interface (HMI) components, and
- software agent components.

Industrial automation and control systems have a huge installed base with physical equipment which may have a lifetime of 10 years or longer. Hence, an important requirement for timely adoption of an HMS architecture will be its provisions for *retrofit* of existing systems and their *migration* to the HMS architecture over time.

As discussed in the preceding section, an HMS architecture must be *open*; that is, it must exhibit the following characteristics, as illustrated in Fig. 4.2:

- *Portability*
  Software tools and agents can accept and correctly interpret library elements (software components and system configurations) produced by other software tools.
- *Interoperability*
  Devices can operate together to perform the *autonomous* and/or *cooperative* functions specified by one or more distributed applications.
- *Configurability*
  Devices and their *software components* can be dynamically *configured* (selected, assigned locations, interconnected and parameterized) by multiple software *tools* and/or software *agents*.

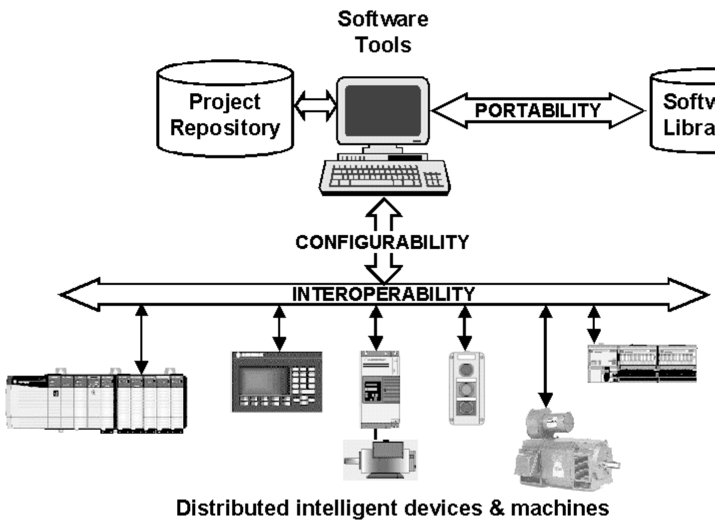


Fig. 4.2. Attributes of open industrial systems

4.3 Architecture Overview

It has been proposed [4.5] that holonic systems be viewed as consisting of one or more *cooperation domains*, that is, logical spaces in which holons may locate, contact, communicate and interact with each other. To meet the requirements expressed in section 4.2, it is proposed that this concept be expanded to include both low-level control (LLC) and high-level control (HLC) domains. LLC refers to normal, non-holonic control and automation functions, while HLC refers to the integration of these functions into holons through the use of software agent technology.

Table 4.1 summarizes the major characteristics of these two domains. To meet the critical requirements for openness, it is proposed that these domains be characterized principally in terms of two sets of standards:

- The IEC (International Electrotechnical Commission) 61499 series of standards [4.2, 4.3, 4.9] for the use of *function blocks* in distributed industrial automation and control systems.[1]
- The FIPA (Foundation for Intelligent Physical Agents) architectural standard for software agent systems [4.4].

The following sections describe in detail the proposed LLC and HLC architectures and their integration.

Table 4.1. LLC (low-level control) and HLC (high-level control = cooperation) domains

Domain	Major functions	Standards	Entities	Ontology	Response times	Message size
LLC	Control/ automation	IEC 61499 [4.2, 4.3]	Function blocks	Physical operations	10 μs to 100 ms	1 bit to 100 bytes
HLC	Negotiation/ coordination	FIPA [4.4]	Agents	Manufacturing tasks	100 ms to 10 sec	100 to 4K bytes

4.4 Low-Level Control Architecture

As shown in Fig. 4.3, the LLC architecture addresses the functions associated with the domain of real-time control, including:

- the control and automation of physical equipment;
- real-time communications among controllers;
- input/output (I/O) between controllers and the controlled machines; and

[1] See http://www.holobloc.com for software tools, runtime platforms and up-to-date information on the IEC 61499 standard and its application to HMS.

- interface among the controlled systems and their human operators, designers, installers and maintainers.

The remainder of this section describes the major architectural elements of IEC 61499 and their application to the implementation of these functions.

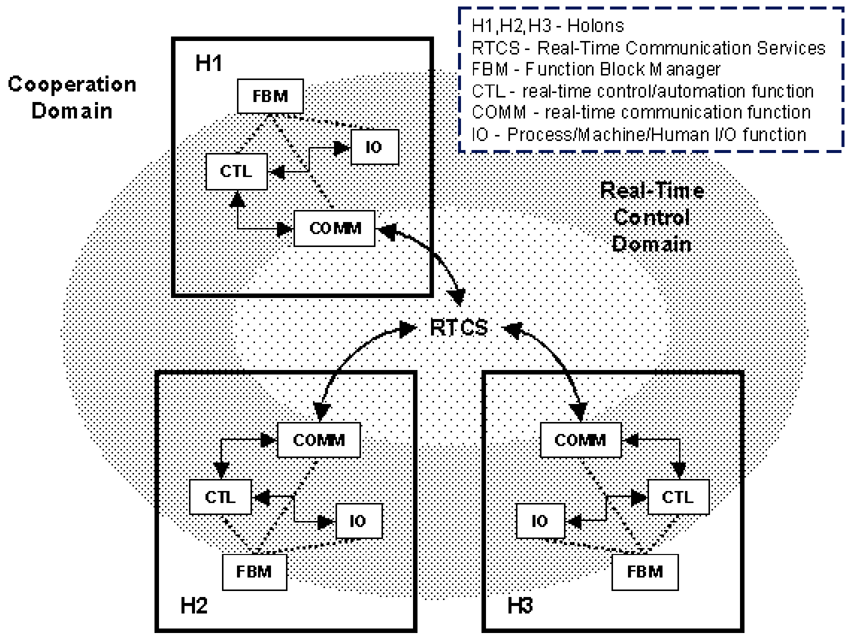

H1,H2,H3 - Holons
RTCS - Real-Time Communication Services
FBM - Function Block Manager
CTL - real-time control/automation function
COMM - real-time communication function
IO - Process/Machine/Human I/O function
Cooperation Domain
Real-Time Control Domain
H1
FBM
IO
CTL
COMM
RTCS
H2
H3

Fig. 4.3. Low-level control architecture

4.4.1 Basic Architectural Elements of IEC 61499

As shown in Fig. 4.4, the IEC 61499 model of a *system* consists of a number of *devices* (physically contained *functional units*). These devices can:

- communicate with each other over communication networks;
- interface to physical processes, equipment or machines; and
- serve as platforms for the execution of distributed applications.

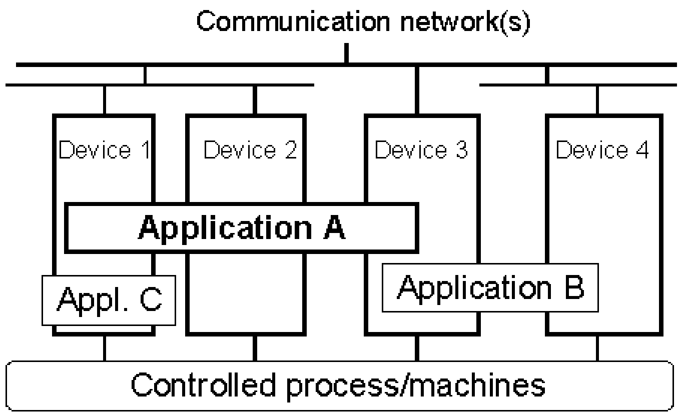


Fig. 4.4. IEC 61499 system model[4.2]

As shown in Fig. 4.5, the IEC 61499 model of an application comprises a network of function blocks interconnected by flows of events and data over event connections and data connections, respectively. As shown in Fig. 4.4, the elements of these applications are in principle distributable among multiple devices. The function blocks in turn are considered to be instances of function block types, which are specified in formal declarations using the means defined in IEC 61499-1 [4.2].

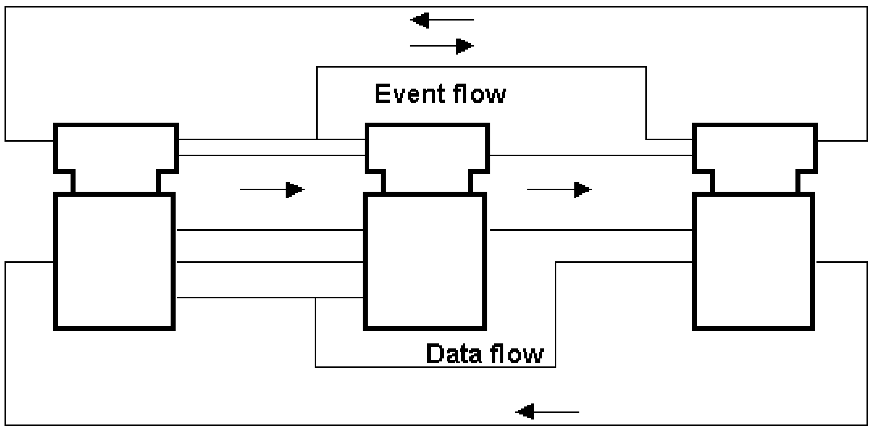


Fig. 4.5. IEC 61499 application model [4.2]

In IEC 61499 a device may consist of multiple resources, as shown in Fig. 4.6. These resources may share communication and process interfaces. Each resource may contain local applications, or the local parts of distributed applications, as shown in Fig. 4.7. In addition, the resource provides a platform for scheduling the execution of algorithms in function blocks, and for mapping underlying operating system services such as communications and machine/process I/O into service interface function blocks.

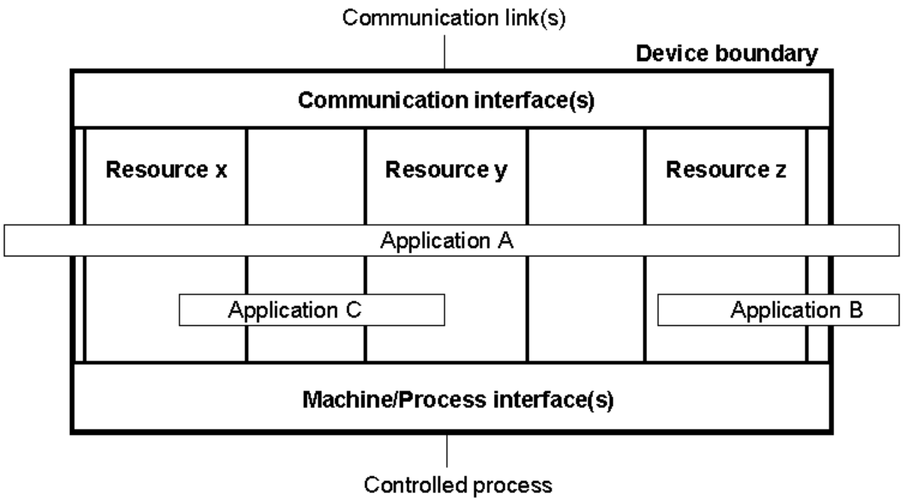


Fig. 4.6. IEC 61499 device model [4.2]

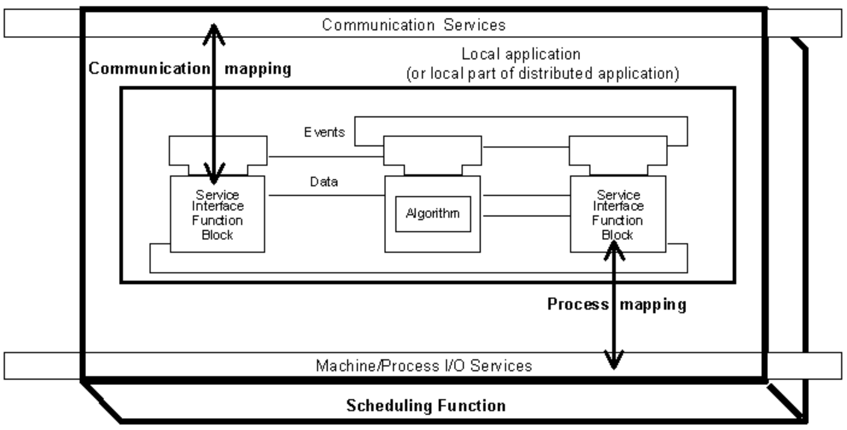

Communication Services
Communication mapping
Local application
(or local part of distributed application)
Events
Data
Service Interface Function Block
Algorithm
Service Interface Function Block
Process mapping
Machine/Process I/O Services
Scheduling Function

Fig. 4.7. IEC 61499 resource model [4.2]

4.4.2 Reference Example

The example shown in Fig. 4.8 will be used throughout this section to illustrate the application of the elements of IEC 61499 to meet the requirements of the LLC architecture. In this example, an actuator can move a workpiece along a slide in the "forward" direction at a velocity *VF* and in the "reverse" direction at a velocity *VR*. These velocities are characteristic of the particular physical actuator. In some mechanisms these velocities may even be time-varying according to a predetermined "velocity profile". The workpiece itself may be be another mechanism, for instance, an end effector such as a gripper.

Associated with the mechanism are two sensors: a HOME sensor, which is activated when the workpiece has moved to the end of the slide in the reverse direction, and an END sensor, which is activated when the workpiece has moved to the end of the slide in the forward direction. These sensors may exhibit hysteresis as a function of the sensor characteristics and their interaction with the mechanism.

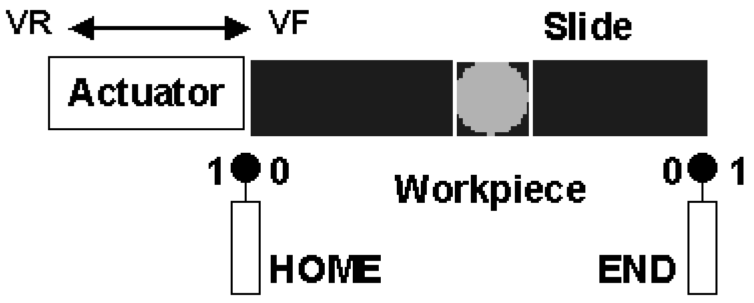


Fig. 4.8. A bidirectional mechanism

4.4.3 Interface Abstraction

IEC 61499-1 [4.2] defines the *adapter interface* construct to permit the abstraction of interfaces between function blocks. This has three major benefits, as will be seen later:

1. It permits the representation of multiple data and event connections between function blocks with a single line, thus helping to eliminate diagram clutter.
2. It enables the abstract representation of patterns of interaction between function blocks through the use of **service sequence** notation.
3. It facilitates the reuse of the interaction patterns for differing functional implementations , for instance in converting from a simulated to an actual physical system.

For the example of the bidirectional mechanism, this pattern is represented as shown in Fig. 4.9. Here a CMD event is used to convey Boolean values FWD and REV from a control algorithm, indicating that motion is commanded in the forward or reverse direction, respectively. Similarly, an IND event is used to convey changes in the boolean values of the FWD and REV sensors back to the control algorithm. Typical sequences of operation are shown in Figures 4.9b-d, and the corresponding values conveyed are shown in Table 4.2 in the informal notation of IEC 61499.

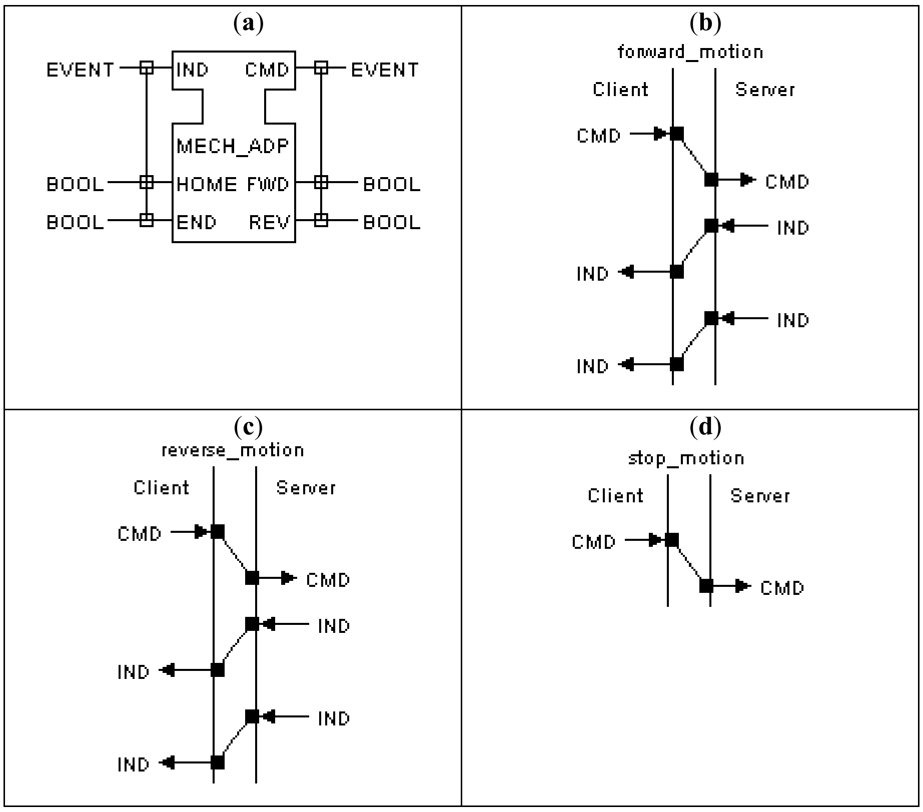


Fig. 4.9. Example of IEC 61499 adapter interface. (**a**) Interface. (**b**) Forward motion sequence. (**c**) Reverse motion sequence. (**d**) Stop motion sequence

Table 4.2. Operational sequences of mechanism interface

```
SERVICE Client/Server
SEQUENCE forward_motion
    Client.CMD(FWD=1,REV=0) -> Server.CMD(FWD,REV);
    Server.IND(HOME=0) -> Client.IND(HOME);
    Server.IND(END=1) -> Client.IND(END);
END_SEQUENCE

SEQUENCE reverse_motion
    Client.CMD(FWD=0,REV=1) -> Server.CMD(FWD,REV);
    Server.IND(END=0) -> Client.IND(END);
    Server.IND(HOME=1) -> Client.IND(HOME);
END_SEQUENCE

SEQUENCE stop_motion
    Client.CMD(FWD=0,REV=0) -> Server.CMD(FWD,REV);
END_SEQUENCE
END_SERVICE
```

4.4.4 Control and Automation Functions

In the IEC 61499 LLC architecture, combinational and sequential control functions are typically performed by *basic* function blocks, i.e., *instances* of basic function block *types*. In these types, the execution of control algorithms is under the control of event-driven state machines represented as *Execution Control Charts* (ECCs). As shown in Fig. 4.10, each *state* in the ECC is associated with one or more *actions*. Each action consists of zero or one *algorithm* to be executed and zero or one *output event* to be issued upon the completion of algorithm execution (or immediately if no algorithm is associated with the state). The algorithms are executed and output events issued as soon as possible after activation of the associated execution control state (EC state). Transition conditions between states are expressed as Boolean combinations of event inputs and Boolean expressions involving input, output and internal variables of the function block. A transition is *cleared*, that is, its predecessor state is deactivated and its successor state is activated when triggered by an associated event when its transition condition is true.[2]

[2] Subclause 2.2.2.2 of IEC 61499 [4.2] provides a rather complex model for the operation of ECCs. A simpler model based on Harel state charts [4.6] is under consideration. In either case, transition conditions consisting of a single event, a pure Boolean condition not involving an event, or an AND combination of both will function as expected.

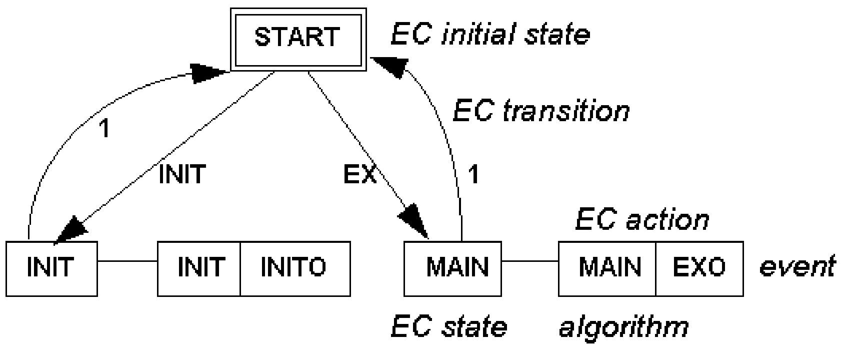


Fig. 4.10. Execution control chart (ECC) elements [4.2]

Figure 4.11 shows the usage of an ECC in a basic function block which provides cyclic operation of the bidirectional mechanism shown in Fig. 4.8. Operation for a single cycle only, or continuous cycling is provided.

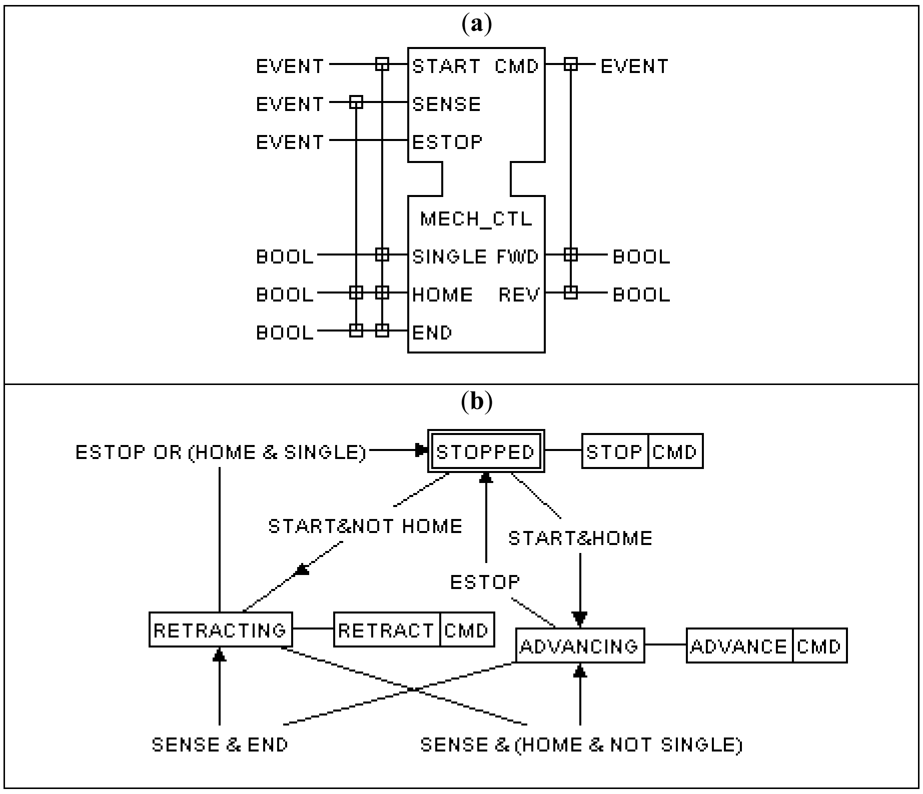


Fig. 4.11. A basic function block for low-level control. (**a**) Interface. (**b**) Execution control chart

Algorithms in basic function blocks are typically expressed in one of the IEC 61131-3 programming languages [4.7, 4.8], specifying a mapping from the values of the input, output and internal variables to new values for the internal and output values. Table 4.3b shows the simple algorithms associated with the MECH_CTL function block type shown in Fig. 4.11, expressed in the IEC 61131-3 Structured Text (ST) language [4.7]. Table 4.3a shows the informal semantics of the inputs and outputs of this function block type expressed in the textual syntax defined in IEC 61499-1 [4.2].

Table 4.3. Partial textual declarations for the MECH_CTL function block type. (**a**) Interface declarations. (**b**) Algorithms

(a)

```
EVENT_INPUT
    START WITH SINGLE, HOME, END; (* Start Cycle *)
    SENSE WITH HOME, END; (* Sensor Change Notification *)
    ESTOP; (* Emergency (Stop Immediately) *)
END_EVENT

EVENT_OUTPUT
    CMD WITH FWD, REV; (* FWD/REV Command *)
END_EVENT

VAR_INPUT
    SINGLE : BOOL; (* 1=Single Cycle, 0=Auto-Repeat *)
    HOME : BOOL; (* Tool at HOME position *)
    END : BOOL; (* Tool at END position *)
END_VAR

VAR_OUTPUT
    FWD : BOOL; (* Forward actuation *)
    REV : BOOL; (* Reverse actuation *)
END_VAR
```

(b)

```
ALGORITHM ADVANCE IN ST :
    FWD:=TRUE;
    REV:=FALSE;
END_ALGORITHM
ALGORITHM RETRACT IN ST :
    FWD:=FALSE;
    REV:=TRUE;
END_ALGORITHM
ALGORITHM STOP IN ST :
    FWD:=FALSE;
    REV:=FALSE;
END_ALGORITHM
```

4.4.5 Diagnostic Functions

Figure 4.12 shows a function block type that may be used for model-based diagnostics of the operation of the bidirectional mechanism shown in Fig. 4.8. It is intended for use in conjunction with the operational control block MECH_CTL. shown in Fig. 4.11 and Table 4.3, and an external timer of the IEC 61499-1 E_DELAY type [4.2]. The function block's algorithms and the informal semantics of its inputs and outputs are shown in Table 4.4.

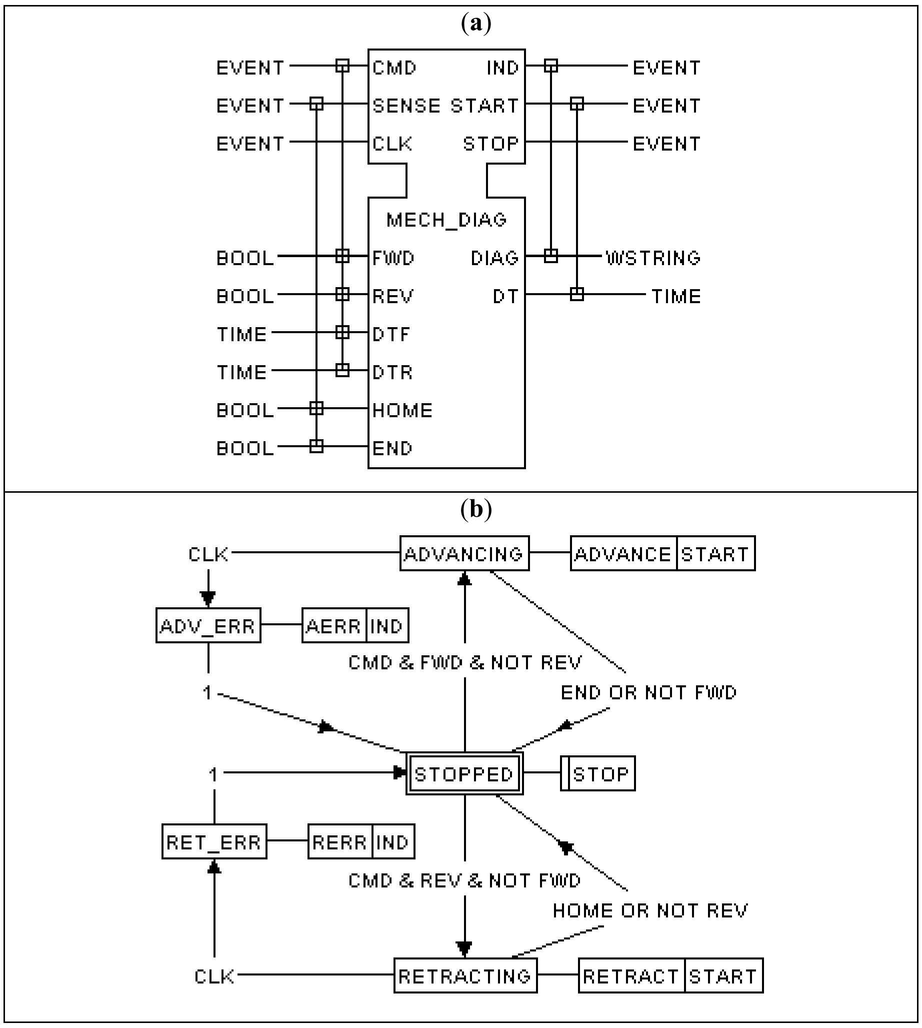


Fig. 4.12. A basic function block for diagnostics. (**a**) Interface. (**b**) Execution control chart

Table 4.4. Partial textual declarations for the MECH_DIAG function block type. (**a**) Interface declarations. (**b**) Algorithms

(a)

```
EVENT_INPUT
    CMD WITH FWD, REV, DTF, DTR; (* Motion Command *)
    SENSE WITH HOME, END; (* Sensor Change Notification *)
    CLK; (* External Timer Expired *)
END_EVENT

EVENT_OUTPUT
    IND WITH DIAG; (* Diagnostic Indication *)
    START WITH DT; (* Start External Timer *)
    STOP; (* Stop External Timer *)
END_EVENT

VAR_INPUT
    FWD : BOOL; (* Move in HOME -> END direction *)
    REV : BOOL; (* Move in END -> HOME direction *)
    DTF : TIME := t#5s; (* Maximum HOME -> END delay *)
    DTR : TIME := t#3s; (* Maximum END-> HOME delay *)
    HOME : BOOL; (* Tool at HOME position *)
    END : BOOL; (* Tool at END position *)
END_VAR

VAR_OUTPUT
    DIAG : WSTRING; (* Diagnostic String *)
    DT : TIME; (* Delay Period for External Timer *)
END_VAR
```

(b)

```
ALGORITHM ADVANCE IN ST :
DT:=DTF;
END_ALGORITHM

ALGORITHM RETRACT IN ST :
DT:=DTR;
END_ALGORITHM

ALGORITHM AERR IN ST :
DIAG:="TIMEOUT_ADVANCING";
END_ALGORITHM

ALGORITHM RERR IN ST :
DIAG:="TIMEOUT_RETRACTING";
END_ALGORITHM
```

4.4.6 Functional Composition

IEC 61499 provides for the construction of IP through the reuse and functional composition of lower-level encapsulated IP. This is done through the construction of *composite* function block types, whose bodies consist of networks of interconnected lower-level *component* function blocks. The execution control of the composite function blocks in an instance of this type is performed through the propagation of input events via the event connections of these networks. Therefore, a function block of this type does not directly utilize an ECC for execution control.

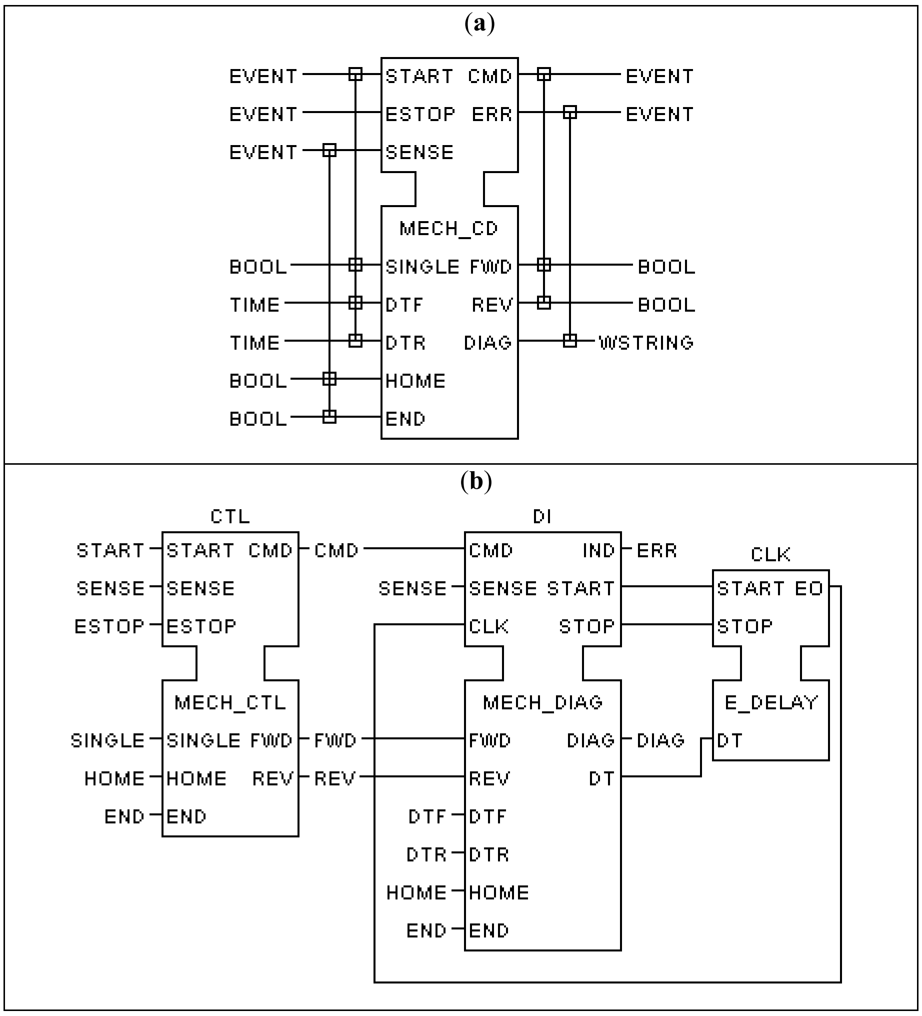


Fig. 4.13. A composite function block. (**a**) Interface. (**b**) Body

As an example of such encapsulation, consider the fact that the control and diagnostic functions for the bidirectional mechanism described earlier are closely related. Hence, it adds value for such functions to be "packaged" together in a composite function block as illustrated in Fig. 4.13. This provides the potential to make the mechanism and its associated control device more "intelligent", i.e. both automatically operating and self-diagnosing.

4.4.7 Human Interface Functions

Human interface services provided by an underlying operating environment may be encapsulated in IEC 61499 *service interface* function blocks. These blocks in turn may be combined into composite function blocks to deliver an appropriate set of human interface elements for a particular application. For instance, Fig. 4.14 shows the encapsulation of an appropriate set of human interface elements for the cyclic operation of the bidirectional mechanism described above.

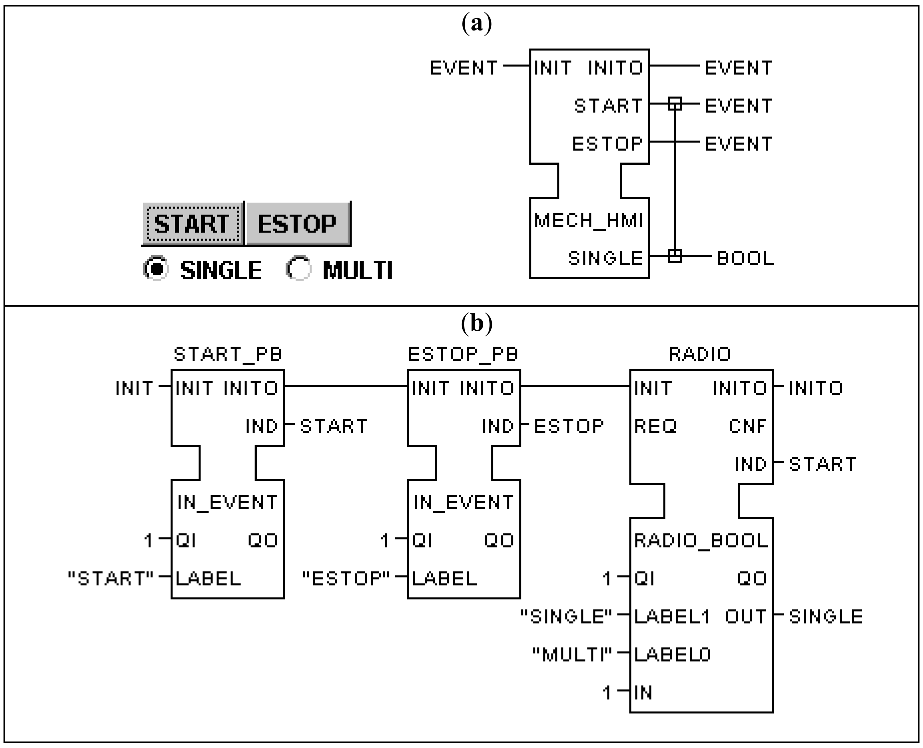


Fig. 4.14. A composite human-interface function block. (**a**) Display and functional interface. (**b**) Body

Modeling, Simulation and Animation Functions

Modifications to the classical Model/View/Controller (MVC) design pattern [4.6] have been proposed [4.10] to permit the use of IEC 61499 for the purposes of modeling and simulation of intelligent devices and systems, and animated display of the results of such simulations. Inclusion of Diagnostic (D) and Adapter (A) elements leads to a proposed MVCDA design pattern.[3]

More recent experience[4] has indicated that a functional partitioning into Model/View/Adapter (MVA) and Control/Diagnostic/Adapter (CDA) groupings is more advantageous in the development and debugging of LLC systems. This modified design pattern is accordingly denoted MVA/CDA. The advantages of this partitioning are seen particularly in managing the transition from simulated to actual physical systems.

Figure 4.15 shows a composite function block type capable of simulating the operation of the bidirectional mechanism shown in Fig. 4.8. This utilizes standard component function blocks defined as specified in Annex D of IEC 61499-1 [4.2], in addition to a specialized service interface function block type (DRIVE) for simple motion simulation.[5] Table 4.5 provides documentation of the inputs and outputs of this composite function block type.

---

[3] See http://www.holobloc.com/doc/despats/ for more extensive descriptions of the use of design patterns for IEC 61499.

[4] The author is indebted to Mr. Franz Auinger and Mr. Werner Rumpl of Profactor GmbH, Steyr, AT for their demonstration of the benefits of the MVA/CDA design pattern.

[5] See http://www.holobloc.com/doc/despats/mvc/DRIVE.htm for documentation of the DRIVE function block type.

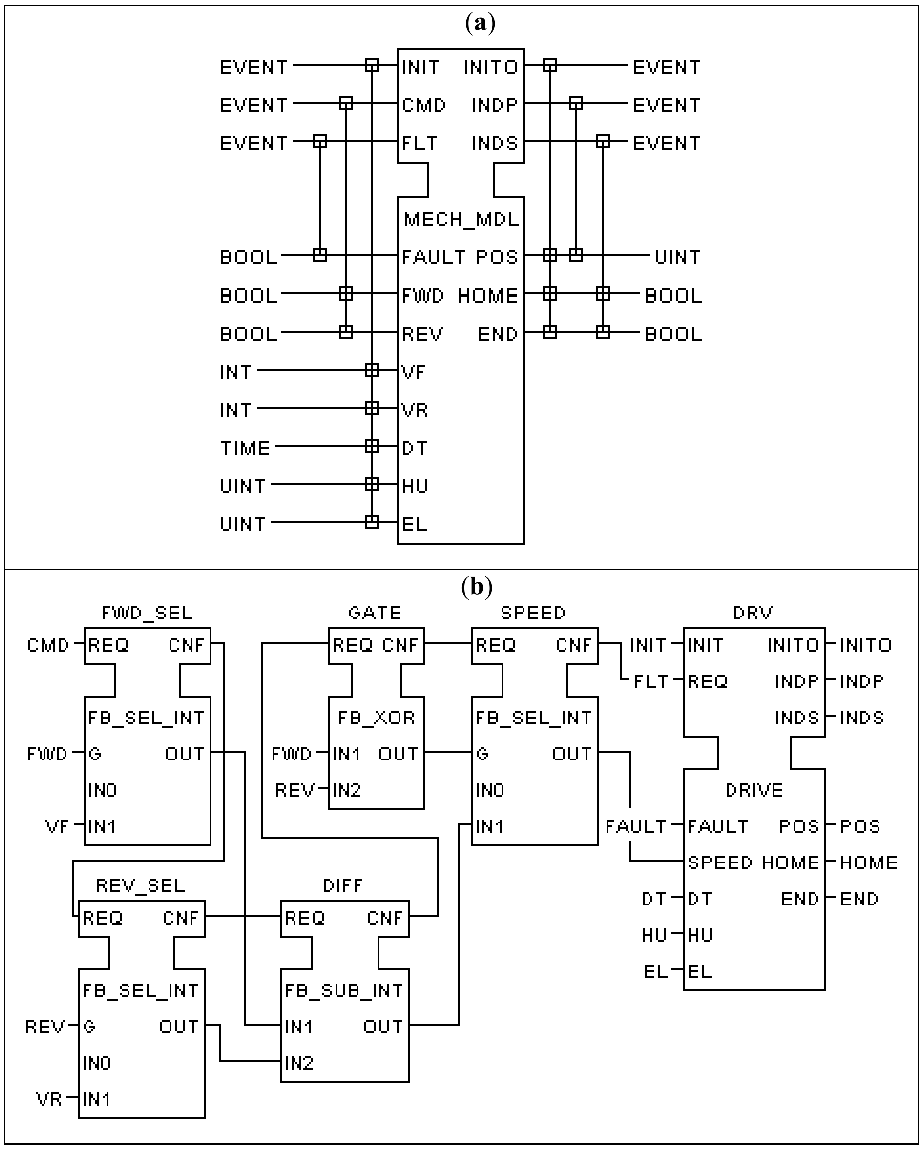


Fig. 4.15. A simulation model of the bidirectional mechanism. (**a**) Interface. (**b**) Body

Table 4.5. Interface declarations for the MECH_MDL function block type

```
EVENT_INPUT
    INIT WITH VF, VR, DT, HU, EL;
    CMD WITH FWD, REV; (* Motion command *)
    FLT WITH FAULT; (* Change in FAULT status *)
END_EVENT
EVENT_OUTPUT
    INITO WITH POS, HOME, END;
    INDP WITH POS; (* Position change *)
    INDS WITH HOME, END; (* Sensor change *)
END_EVENT
VAR_INPUT
    FAULT : BOOL; (* 0=Enable,1=Simulate Fault(pause) *)
    FWD : BOOL; (* Move Forward at VF *)
    REV : BOOL; (* Move Back at VR *)
    VF : INT := 20; (* Forward speed in percent/sec *)
    VR : INT := 40; (* Reverse speed in percent/sec *)
    DT : TIME := t#250ms; (* Simulation clock interval *)
    HU : UINT := 10; (* Upper(advancing)limit of HOME LS *)
    EL : UINT := 90; (* Lower(retracting)limit of END LS *)
END_VAR
VAR_OUTPUT
    POS : UINT; (* Position, 0->100% *)
    HOME : BOOL := true; (* HOME position, 0->HU *)
    END : BOOL; (* END position, EL<-100 *)
END_VAR
```

Figure 4.16 illustrates the combination of the MECH_MDL function block described above with an XBAR_VIEW service interface function block displaying the progress of the simulation. Table 4.6 provides a description of the XBAR_VIEW inputs. A typical appearance of the XBAR_VIEW element is shown in Fig. 4.17. Note the small colored circle at the upper left of the display. The user can click on this sub-element to toggle the FAULT output; this variable is used by the MECH_MDL function block to stop motion, thus simulating a simple fault. The colored circle changes from red when FAULT is true to green when FAULT is false, giving the user some feedback on the simulated FAULT state.

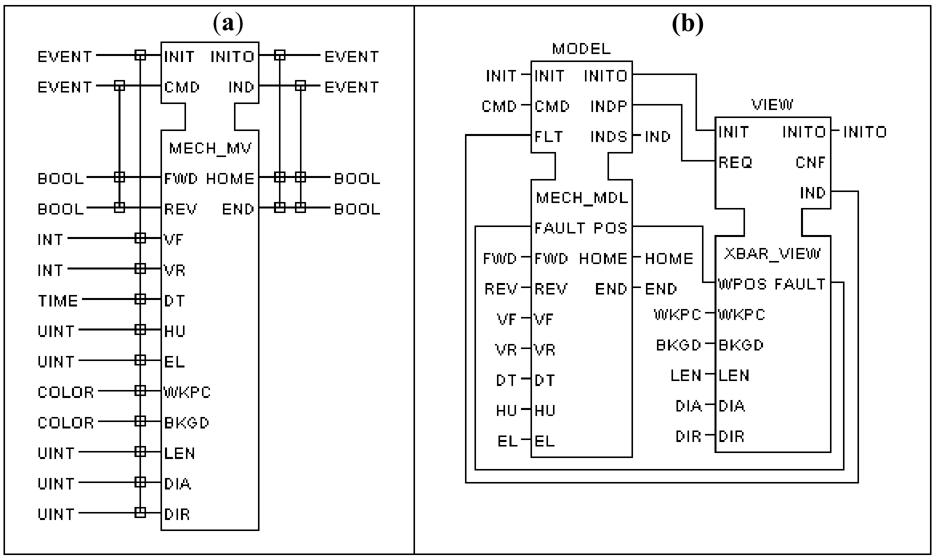


Fig. 4. 16. Composite function block for simulation and display of the bidirectional mechanism. (a) Interface. (b) Body

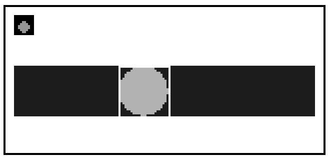


Fig. 4. 17. Appearance of the XBAR_VIEW element.

Table 4.6. Partial interface declarations for the MECH_MV function block type

```
WKPC : COLOR := cyan; (* Workpiece Color *)
BKGD : COLOR := blue; (* Transfer Bar Color *)
LEN : UINT; (* Bar Length in Diameters *)
DIA : UINT; (* Workpiece diameter *)
DIR : UINT; (* Orientation:0=L/R,1=T/B,2=R/L,3=B/T *)
```

Adapter interfaces can be used to simplify the interconnection of control and diagnostic elements with simulation and display elements, and to facilitate the conversion from simulated to physical devices. The convention is that addition of adapters to a Model/View (MV) element converts it to a Model/View/Adapter (MVA) element, and addition of adapters to a Control/Diagnostic (CD) element converts it to a Control/Diagnostic/Adapter (CDA) element. Figure 4.18 illustrates the application of this principle to the previously developed MECH_MV and MECH_CD function block types.

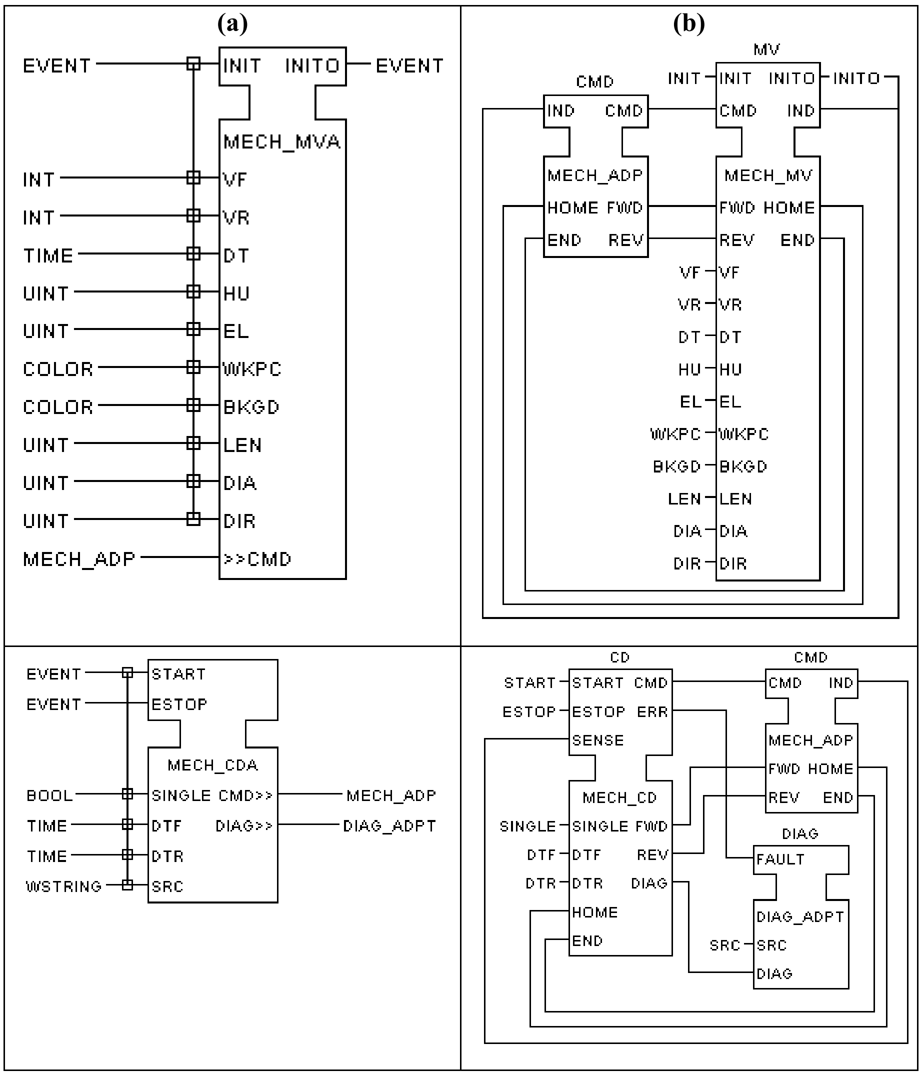


Fig. 4. 18. Addition of adapter interfaces. (**a**) External interface. (**b**) Body

4.4.9 System Configurations

A simple system configuration for testing the combined MECH_MVA and MECH_CDA functionality is shown in Fig. 4.19. This system contains a single device of type FRAME_DEVICE, which in turn contains a single resource of type PANEL_RESOURCE. These elements encapsulate the functionality of the classes Frame and Panel, respectively, in the Java AWT user interface [4.11]. The resource is populated with instances of MECH_CDA, MECH_MVA and associated user interface elements. An instance of the IEC 61499 standard type E_RESTART is used for initialization.

The appearance of the resulting user interface is shown in Fig. 4.20, and the composite user interface function block DIAG_HMI is shown in Fig. 4.21.

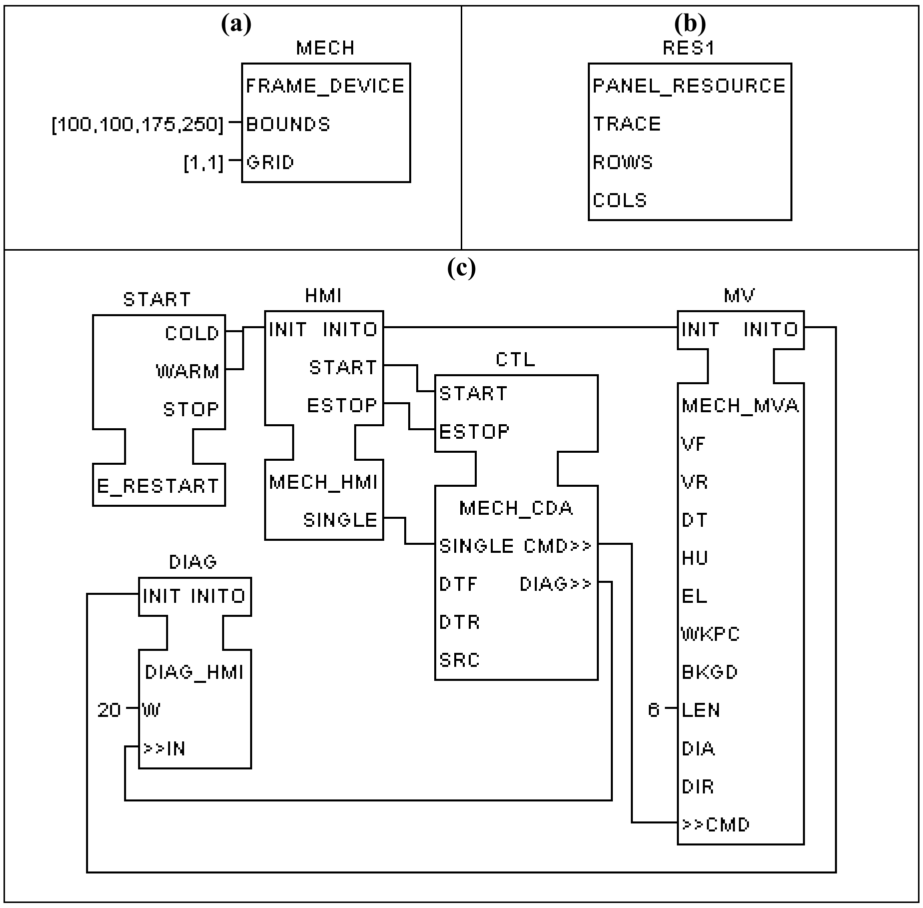
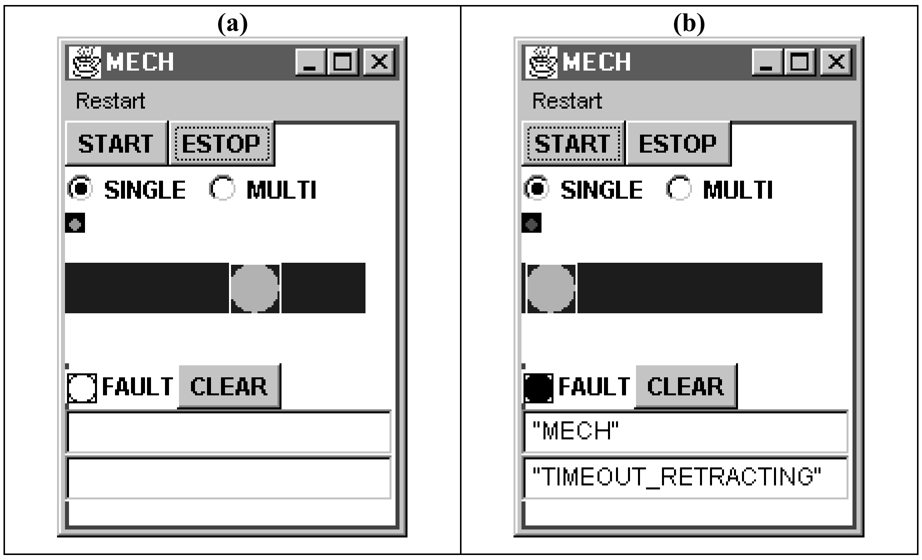


Fig. 4. 19. A simple system configuration. (**a**) System view. (**b**) Device view. (**c**) Resource view



"MECH"

"TIMEOUT_RETRACTING"

Fig. 4.20. The user interface to the simulation. (**a**) Normal operation. (**b**) Faulted operation

4.4.10 Communication Functions

Annex F of IEC 61499-1 [4.2] provides informative definitions for generic service interface function blocks to the most commonly used communication services in LLC, namely broadcast PUBLISH/SUBSCRIBE and one-to-one CLIENT/SERVER connections. In addition, IEC 61499-1 Annex F defines ASN.1 abstract syntax [4.12], and basic [4.13] and compact encoding rules for the IEC 61131-3 [4.7] elementary and derived data types used in IEC 61499.

Since this information is not normative, additional technical agreements, e.g. [4.15], must be employed to ensure interoperability of devices using these service interfaces.

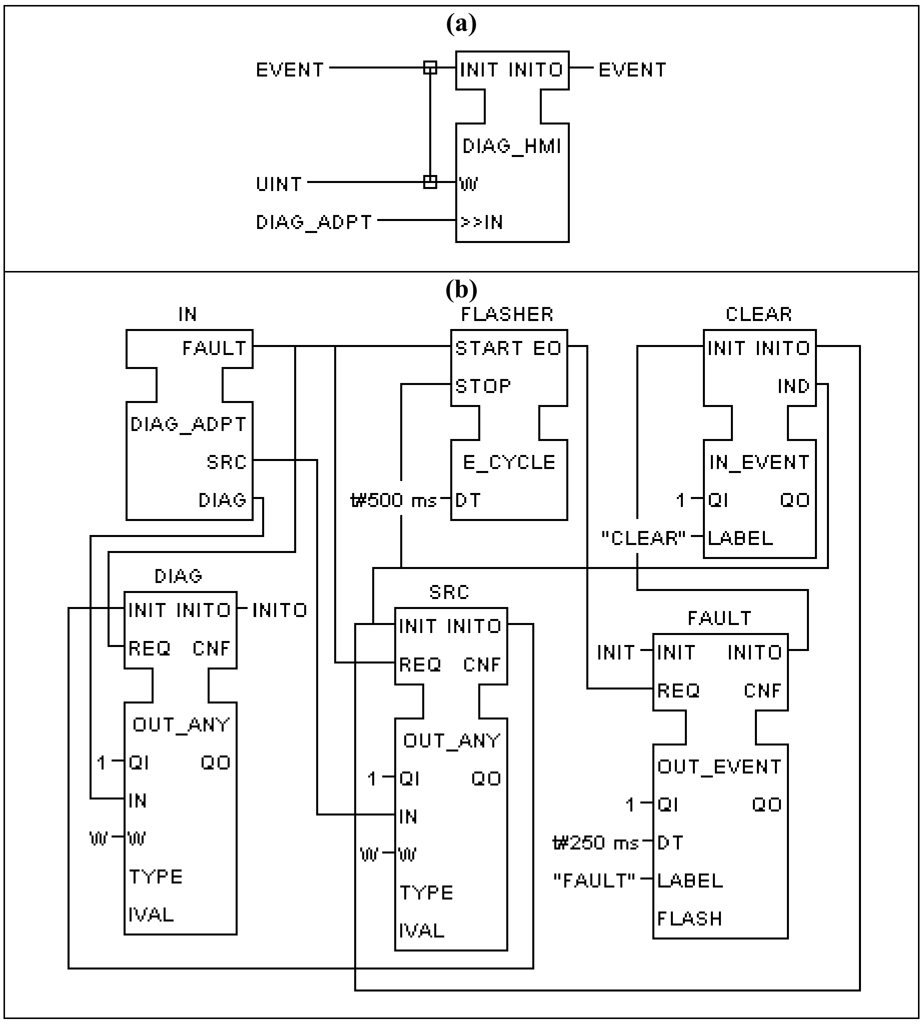


Fig. 4.21. A human interface for diagnostic messages. (**a**) Functional interface. (**b**) Body

4.4.11 Resource and Device Type Specifications

IEC 61499 provides for the definition of *device types* and *resource types*, for instance the previously illustrated FRAME_DEVICE and PANEL_RESOURCE types, respectively. Resources may be considered as providing "plug-in module" functionality within devices. To illustrate this point, consider the MECH_PNL resource type shown in Fig. 4.22. Note that the only interface defined for such types is a set of input variables that may be used for configuring instances of the type.

Note also the provision of communication service interface function blocks to establish connectivity to the “outside world”:

- An instance of the type SUBSCRIBE_1 receives the START event and value of the SINGLE variable for the MECH_CDA instance.
- An instance of the type SUBSCRIBE_0 receives the ESTOP event for the MECH_CDA instance.
- An instance of the composite DIAG_PUB type defined in Fig. 4.23 is used to publish the diagnostic information from the MECH_CDA instance. This publiched information is received by instances of the DIAG_SUB type also shown in Fig. 4.23.

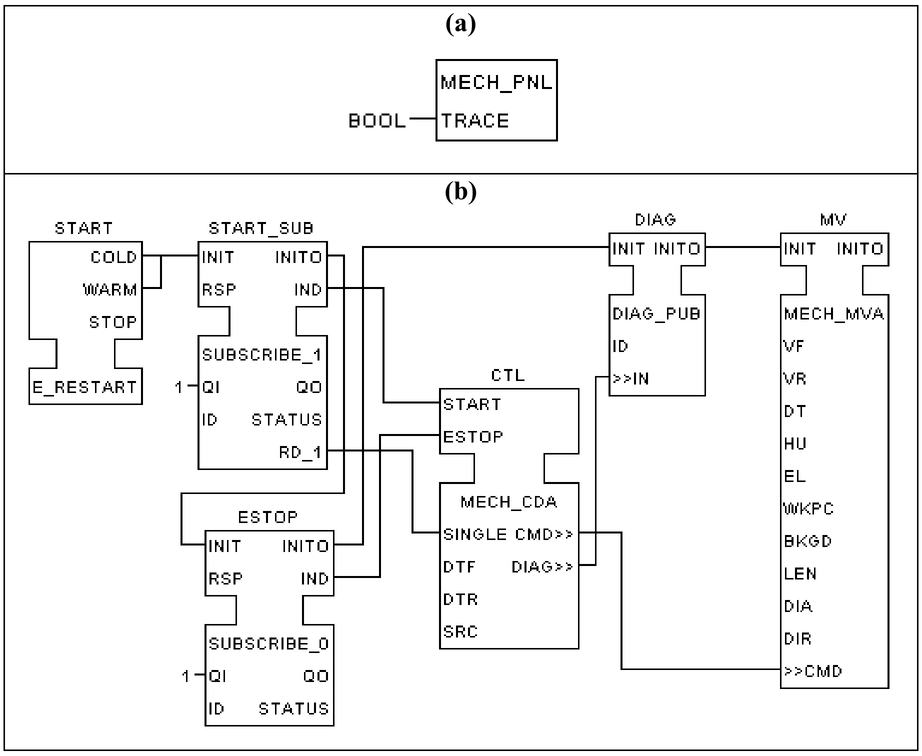


Fig. 4.22. A resource type definition. (**a**) Configuration interface. (**b**) Body

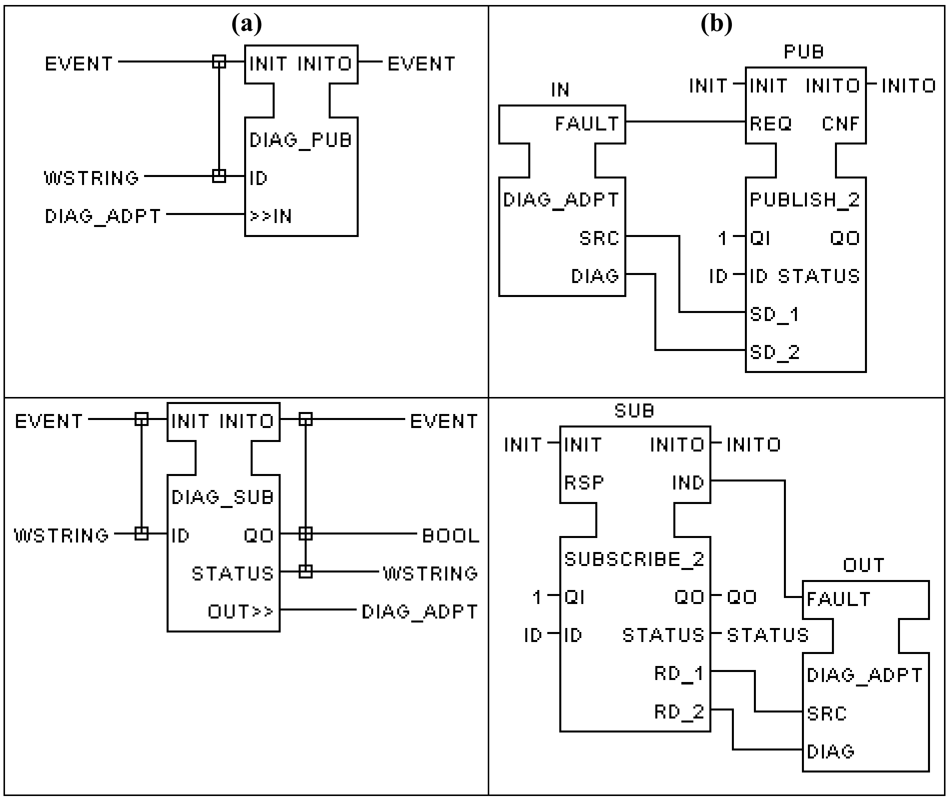


Fig. 4.23. Composite communication service interface types. (**a**) Functional interfaces. (**b**) Bodies

The system configuration shown in Fig. 4.24 presents a simple example of the integration of multiple distributed resources. The use of a grid layout illustrates the use of a device as a “rack” or “manifold” for multiple intelligent electomechanical devices, each of which is represented as an IEC 61499 resource within the manifold.

In this distributed system configuration, the operator interface is represented as a separate IEC 61499 device. Since the PUBLISH and SUBSCRIBE service interface function blocks in this implementation utilize multicast communications, a single operator interface can be used to control the operation of and receive diagnostic messages from multiple intelligent electromechanical devices of the same IEC 61499 resource type (in this example, the instances MECH1 and MECH2 of the MECH_PNL resource type in the MANIFOLD device). The OPERATOR.DISPLAY resource of this system configuration is shown in Fig. 4.24d.

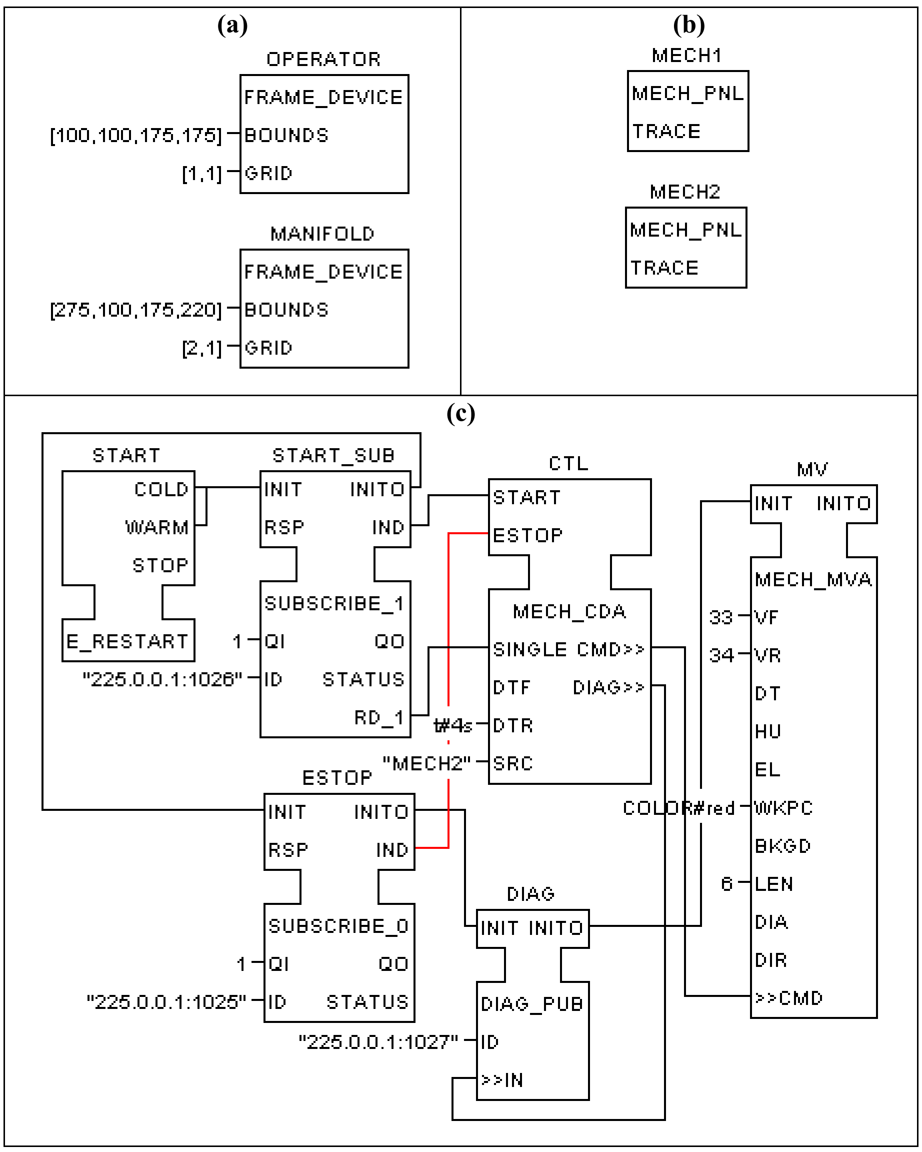


Fig. 4.24. A distributed system configuration. (**a**) System view. (**b**) MANIFOLD device view. (**c**) MANIFOLD.MECH2 resource view. (**d**) OPERATOR.DISPLAY resource view

Figure 4.25 shows the appearance of the operator interface, and of the manifold with two simulated electromechanical devices in a 2-row by 1-column grid, for the system configuration given in Fig. 4.24. Figures 4.25b,c show that the operator interface can receive error messages from either of the simulated electromechanical devices. Physical or simulated devices can easily be added to this system and be compatible with the same operator interface, as long as such devices receive their operative commands and send their diagnostic messages through the same UDP multicast socket addresses (for instance, they may receive the ESTOP command via a SUBSCRIBE_0 block with UDP socket address 225.0.0.1: 1025).

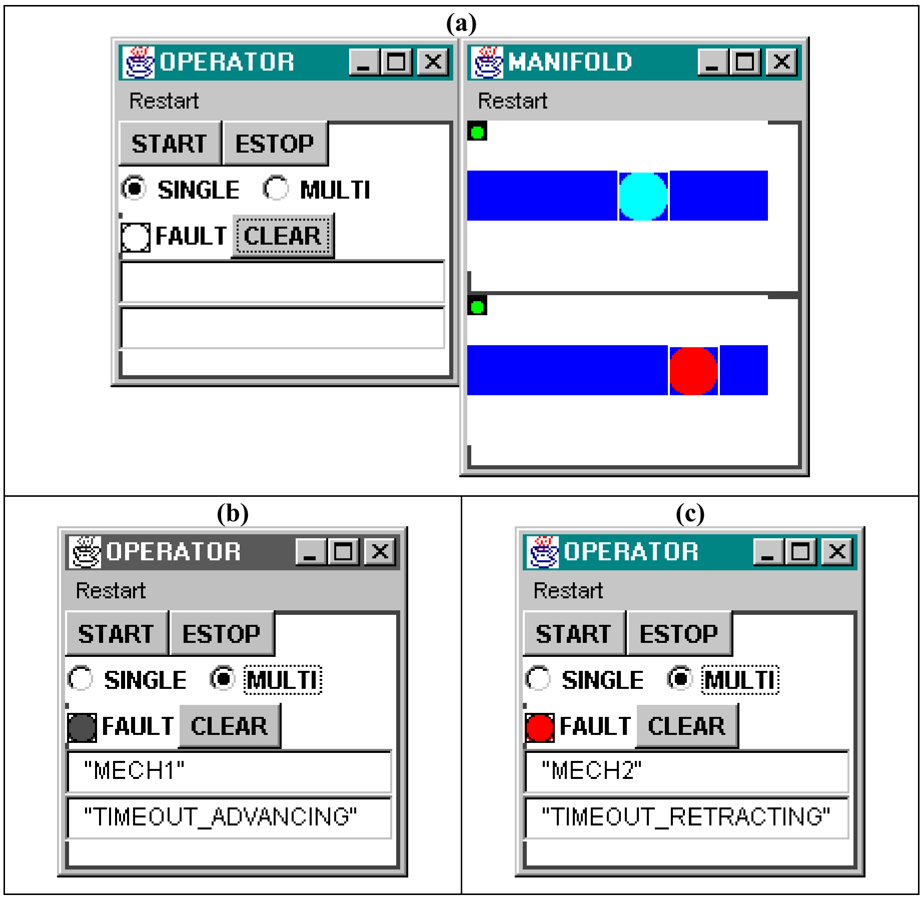
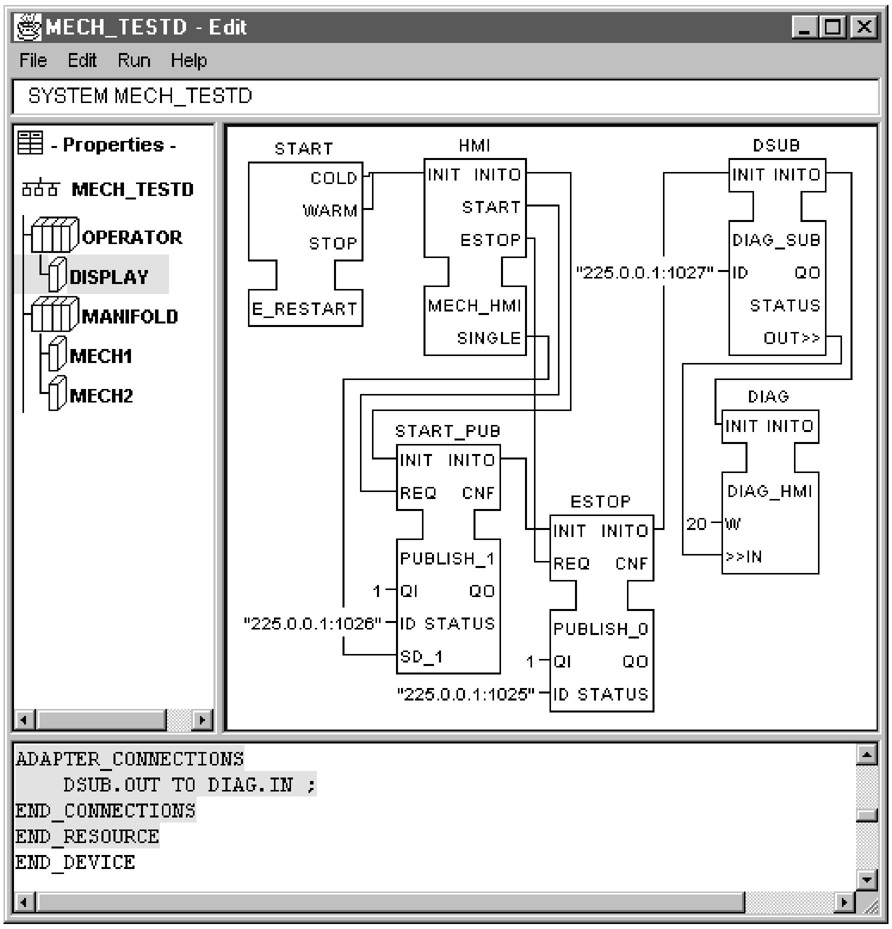


Fig. 4.25. The user interface to the simulation. (**a**) Normal operation. (**b**) MECH1 fault. (**c**) MECH2 fault

4.4.12 Software Portability

Portability of data types and other library elements (adapter types, function block types, resource types, device types and system configurations) is achieved in IEC 61499 via the use of XML [4.14] documents for exchange of library elements among software tools. The XML Document Type Definitions (DTDs) are given in IEC 61499-2 [4.3].



```
ADAPTER_CONNECTIONS
    DSUB.OUT TO DIAG.IN ;
END_CONNECTIONS
END_RESOURCE
END_DEVICE
```

Fig. 4.26. An IEC 61499-2 compliant software tool.

The user interface of a prototype software tool[6] complying with the provisions of IEC 61499-2 is shown in Fig. 4.26. A human-readable textual representation of the library element being edited (in this case the system configuration previously shown in Fig. 4.24) is shown in the lower pane of the tool, in the syntax defined in IEC 61499-1. When a system configuration is being edited, the navigation tree in the upper left panel of this tool reflects the system/device/resource hierarchy of the IEC 61499 architecture, which is also reflected in the tree structure of the corresponding XML document. In this example, the OPERATOR.DISPLAY resource of the system configuration shown in Fig. 4.24 has been selected.

[6] Available at http://www.holobloc.com.

4.4.13 Device Configurability

To facilitate the configurability of devices and resources, subclause 4.3 of IEC 61499-1 [4.2] defines a management service interface function block type with standardized management commands for the dynamic creation and deletion of managed objects such as function block instances and event and data connections. However, the encoding of management commands and responses as well as the implementation of messaging services for remote configuration is given in non-normative Annexes of IEC 61499-1. Hence, as in the case of communication services, additional technical agreements must be employed to ensure configurability.

On the basis of practical experience, the previously mentioned technical agreement for feasibility demonstrations [4.15] proposes the use of an XML [4.14] DTD for encoding of management commands and responses. This agreement also specifies the combination of the normal IEC 61499 CLIENT/SERVER communication with the management service interface function block, to achieve configurability by any remote software tool or agent capable of encoding and decoding the appropriate XML commands and responses over a TCP/IP socket connection. This management service architecture is encapsulated in the device management kernel function block type, illustrated in Fig. 4.27.

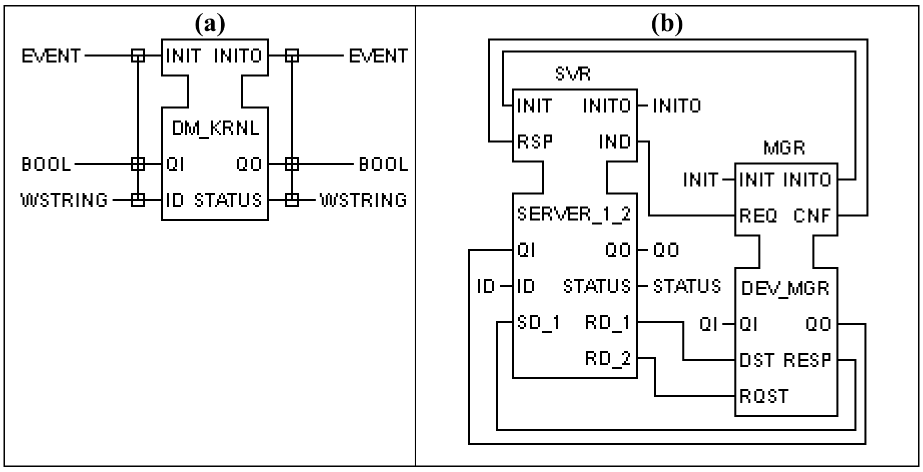


Fig. 4.27. Device management kernel. (**a**) Interface. (**b**) Body

4.4.14 LLC Architecture: Conclusion

Table 4.7 shows the partitioning of the major holonic system functions of *autonomy* and *cooperation* between the architectural levels of low-level control (LLC) and high-level control (HLC). From the preceding material in this section it can be seen that the IEC 61499 architecture addresses all of the LLC requirements as defined in Section 4.2. The extent to which this architecture can be used to address the remaining HLC requirements is addressed in the following section.

Table 4.7. Partitioning of holon functionality

Function	Architectural Level
Autonomous:	
Task planning	HLC
Task execution	LLC
Fault detection	LLC
Fault recovery	LLC/HLC
Cooperative:	
Task scheduling	HLC
Task coordination	LLC/HLC
Fault detection	HLC
Fault recovery	HLC

4.5 High-Level Control Architecture

As shown in Table 4.7, the HLC architecture addresses the functions associated with the domain of inter-holon *cooperation*, including negotiation and coordination of mutually agreed tasks and mutual action to recover from operational faults. Also included in this level of the HMS architecture are those intra-holon aspects of *autonomous* task planning and sequencing that contribute to the cooperative accomplishment of manufacturing tasks beyond the scope of LLC functionality.

4.5.1 HLC Structure

The overall structure of the HLC architecture is shown schematically in Fig. 4.28. HLC functions are performed within one or more *cooperation domains* which coexist with and are interfaced to the LLC functions of the real-time control domain. These HLC functions are built on an underlying set of *cooperation communication services*. The partitioning of functions within a cooperation domain is shown in Table 4.8.

Other chapters of this book provide further details on the requirements and implementation of HLC functions.

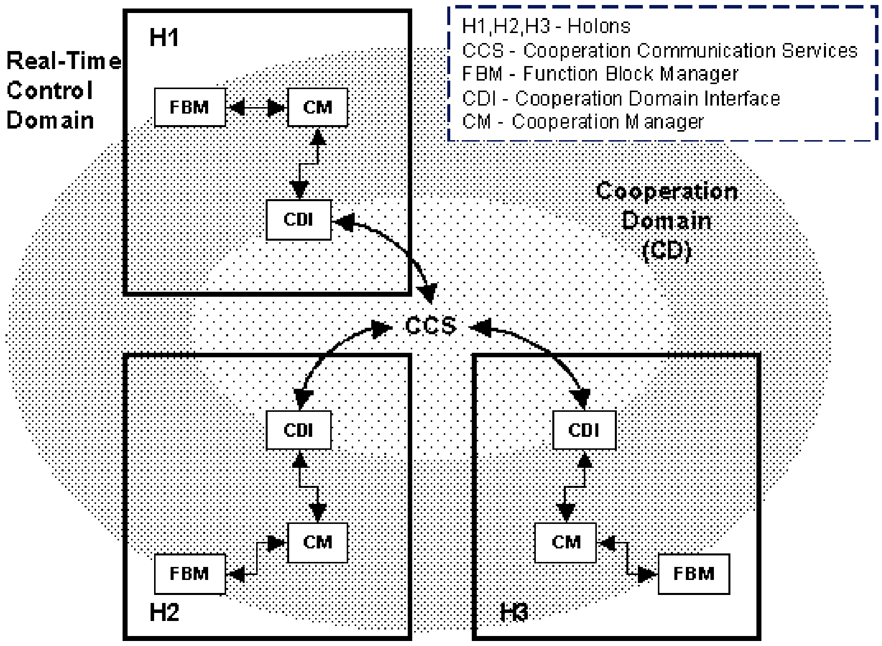


Fig. 4.28. High-level control architecture

Table 4.8. Partitioning of HLC functions

Module	Functions
CDI (Cooperation Domain Interface)	Locate, join, leave CDs Offer and find services ("Yellow Pages") Communicate with other agents
CM (Cooperation Manager)	Negotiate ontologies Maintain knowledge bases Negotiate tasks and schedules Generate/update LLC configurations Coordinate task performance
FBM (Function Block Manager)	Manage LLC configuration

4.5.2 LLC/HLC Integration

In accordance with the Open Systems Interconnection (OSI) model [4.16], the layered "onion skin" architectural models shown in Figs. 4.3 and 4.28 lead to the appearance of “service stacks” within individual holonic devices. Each layer of the stack provides value-added services to the layer above by utilizing the services of the layer below to interact with its distributed peers within the same layer. As shown in Fig. 4.29, the LLC and HLC stacks coexist within the same device, with the FBM (function block management) services providing the interface between the LLC and HLC stacks.

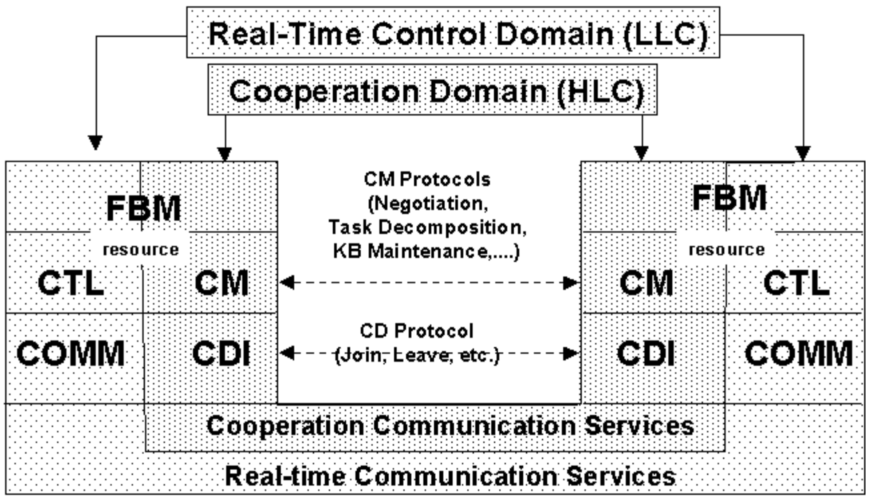

Real-Time Control Domain (LLC)
Cooperation Domain (HLC)
FBM
resource
CTL
CM
COMM
CDI
CM Protocols (Negotiation, Task Decomposition, KB Maintenance,....)
CD Protocol (Join, Leave, etc.)
FBM
resource
CM
CTL
CDI
COMM
Cooperation Communication Services
Real-time Communication Services

Fig. 4.29. LLC/HLC integration

4.5.3 Application of IEC 61499 to LLC/HLC integration

Since the proposed HLC architecture utilizes a layered service model, IEC 61499 *service interface* function block types can be used to model the encapsulation and integration of the services described in Table 4.8. Thus, an instance of the HMS_KERNEL type shown in Fig. 4.30 can be used in place of the FB_KERNEL type shown in Fig. 4.27 to transform an IEC 61499-compliant LLC device into a holonic device, provided that the device has sufficient resources to support the additional functionality. It is expected that the DEV_MGR functionality can be reused without modification, while the HMS_CM and HMS_CDI types will provided interfaces to the cooperation management (CM) and cooperation domain interface (CDI) services described above.

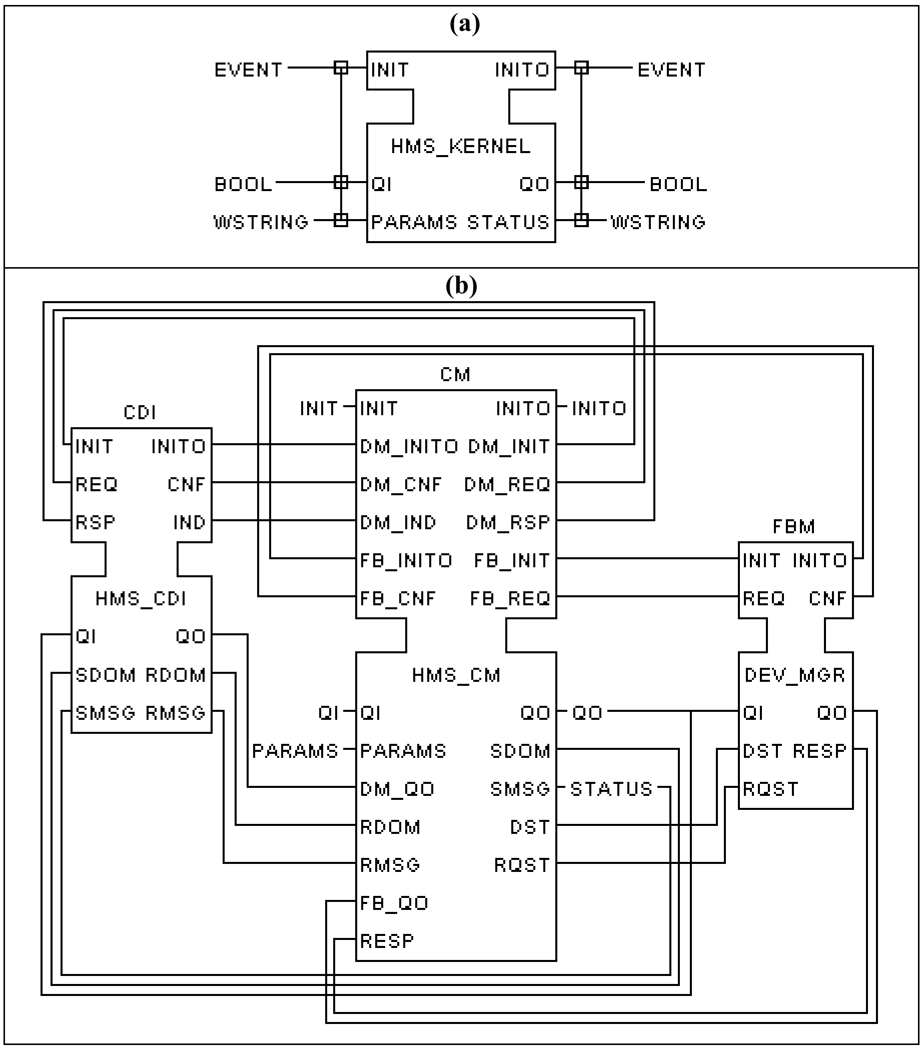


Fig. 4.30. The holonic kernel. (**a**) Interface. (**b**) Body

4.6 Conclusion

It has been shown that devices and software tools compliant with the IEC 61499 series of specifications [4.1, 4.2] and associated techical agreements are capable of meeting all the requirements for low-level control and its integration with high-level control functions in a holonic systems architecture. The LLC architecture has been extensively demonstrated in practical examples. The implementation of HLC functions within the IEC 61499 framework and their integration with LLC functions in practical physical equipment remain a topic of ongoing research.

References

[4.1] H. Shapiro and C. Varian:: *Information Rules: A Strategic Guide to the Network Economy,* Harvard Business School Press, Boston (1999).

[4.2] International Electrotechnical Commission: *Function Blocks, Part1 – Architecture*, IEC PAS 61499-1, Geneva (2000).

[4.3] International Electrotechnical Commission: *Function Blocks, Part 2 – Software tool requirements,* IEC PAS 61499-2, Geneva (2001).

[4.4] Foundation for Intelligent Physical Agents: *FIPA Abstract Architecture Specification*, http://www.fipa.org/specs/fipa00001/ (2001).

[4.5] M.Fletcher, E. Garcia-Herreros, J.H. Christensen, S.M. Deen and R. Mittmann: "An Open Architecture for Holonic Cooperation and Autonomy", in *11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, IEEE Computer Society, New York (2000).

[4.6] B.P.Douglass: *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison Wesley Longman, New York (1997).

[4.7] International Electrotechnical Commission: *Programmable controllers, Part 3 – Programming languages,* IEC 61131-3, Geneva (1993).

[4.8] R.W. Lewis: *Modelling control systems using IEC 61499,* Institution of Electrical Engineers, Stevenage (2001).

[4.9] R.W. Lewis,: *Programming industrial control systems using IEC 61131-3*. Institution of Electrical Engineers, Stevenage (1995).

[4.10] J.H. Christensen: "Design patterns for systems engineering with IEC 61499", in *Fachtagung Verteilte Automatisierung – Modelle un Methoden fuer Entwurf, Verifikation, Engineering und Instrumentierun,* ed. Ch. Doeschner, Otto-von-Guericke-Universitaet, Magdeburg (2000).

[4.11] M. Campione and K. Walrath: *The Java Tutorial,* 2nd Edition, Addison-Wesley, New York (1998).

[4.12] International Electrotechnical Commission: *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1),* ISO/IEC 8824, Geneva (1990).

[4.13] International Electrotechnical Commission,: *Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1),* ISO/IEC 8824, Geneva (1990).

[4.14] W3 Consortium: *eXtended Markup Language (XML) Specification*, http://www.w3c.org/TR/1998/REC-xml-19980210 (1998).

[4.15] *Technical Agreement for IEC 61499 Feasibility Demonstrations*, http://www.holobloc.com/doc/ita/index.htm (2002).

[4.16] International Organization for Standardization: *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model,* ISO/IEC 7498-1, Geneva (1994).