5 PackML Standard

Chapter Topics:

- State machine methodology
- Alarm/error handling
- Operator interface
- Implementation

OBJECTIVES

Upon completion of this chapter, you will be able to understand:

- ANSI/ISA-88.00.01 concepts that apply to motion control
- The PackML structured approach
- PackML implementations for Rockwell and Siemens processors

Scenario: Packaging machine problem.

A particular packaging machine using Rockwell Kinetix drives was being tested by the system integrator before shipping the system to the customer. As part of the testing, the power to the machine was shut off. When power was restored, and the system restarted, the coordinate system for the robot had motion instruction errors. A fault reset would not clear the error. In order to restart the robot, the program needed to be downloaded again. You are called to help diagnose and fix the problem.

Solution: The particular program used the PackML approach to programming the machine, so the program was easy to navigate and equipment modules could be disabled in order to narrow the problem diagnosis to the one robot equipment module exhibiting the problem.

A small test system with the same type of processor and drives was quickly built to reproduce the problem. However, the motors were not loaded. The particular alarm scheme was studied, and the alarms that prevented testing were disabled. When the HMI "start" button was toggled, the motors did the expected motion with no problem. Power was cycled and the robot equipment module restarted without any motion instruction errors.

After talking to the system integrator again, you discover one important piece of missing information – the power to the machine was shut off during motion. With this information, you are able to reproduce the error on your test system.

You see that the axes have a fault just before the processor powers down. By disabling various parts of the logic related the axis fault handling, you are able to narrow the problem down to a motion instruction executed as part of axis fault handling. By blocking that motion instruction in this scenario, the machine can be power cycled without problems.

5.1 INTRODUCTION

PackML (Packaging Machine Language) defines a common approach, or machine language, for automated machines. The primary goals are to encourage a common "look and feel" across a plant floor and to enable and encourage industry innovation. For this text, PackML primarily provides a structured approach to the control program. In addition, PackML pack tags are standardized variable structures for communications between machine controllers and to higher-level HMI, MES or Enterprise systems.

PackML was originally developed by OMAC (Organization for Machine Automation and Control) in early 2000's. It was adopted as part of the ANSI/ISA S88 (Batch Control) industry standard in 2008 and later revised (ISA, 2015 and 2022). Most PLC manufacturers that cover motion control applications have PackML implementations.

Since PackML is based on concepts in the ANSI/ISA-88.00.01 standard, that is described first, followed by the PackML state machine description of machine operation. Alarm and error handling are next covered and followed by HMI concepts. The chapter concludes with an implementation of PackML for the Rockwell and Siemens PLCs for a two-axis machine. The chapter appendix describes a simplified implementation suitable for small systems.

5.2 ISA-88.01 MODELS

Many of the concepts used in PackML and in the implementation of motion control have their roots in the ANSI/ISA-88.00.01-2010 standard (shortened to ISA-88.01 in this chapter) and so a brief explanation is needed. The ISA-88.01 standard was originally developed for batch control systems, but also applies to continuous process control and discrete parts manufacturing. For PackML some model aspects are simplified. While most machines that use PLC motion control are producing discrete parts, PLC motion control is not limited to discrete parts manufacturing. PackML provides a common framework in either case.

The ISA-88.01 standard (ISA, 2010) defines three models: a process model, a physical model, and a procedure control model. All of these models are organized hierarchically.

5.2.1 Process Model

The process model is organized as shown in Figure 5.1 and describes the processing actions required to convert the raw materials into finished product. Starting from the top, the overall process is divided into stages, which are major processing activities needed to produce the finished goods. Stages are divided into operations, which are groupings of the minor processing actions.

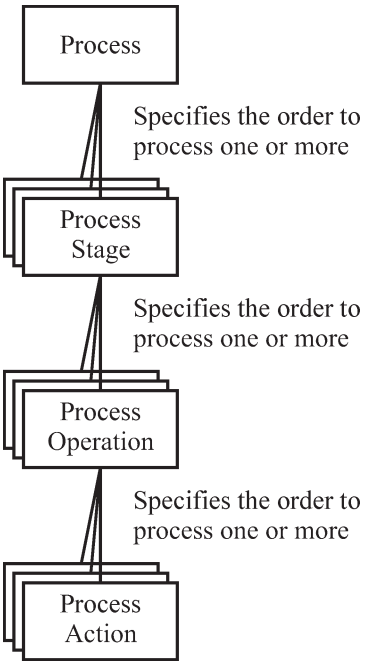

Figure 5.1. ISA-88.01 process model. (Copyright © International Society of Automation. Used with permission of ISA.)

According to ISA-88.01, a process consists of one or more process stages (Figure 5.1, which can be serial, parallel, or both: A "process stage [is] a part of a process that usually operates independently from other process stages that usually results in a planned sequence of chemical or physical changes in the material being processed." (ISA, 2010, p. 23) The key concept for a process is that it transforms a material flow. For discrete-parts manufacturing, example stages are:

Machine raw stock
Subassembly
Final assembly
Package

Each process stage consists of one or more process operations: A "Process operation [is] a major processing task that usually results in a chemical or physical change in the material being processed and that is defined without consideration of the actual target equipment configuration" (ISA, 2010, p. 23). Example operations for a *Package* stage are:

Form carton
Fill
Seal

Each process operation is further subdivided into one or more process actions. According to ISA-88.01, A "process action [is] a minor processing task that may be combined with other minor processing activities to make up a process operation" (ISA, 2010, p. 23). Example actions for the *Form carton* operation are:

Crimp
Fold
Heat-seal

5.2.2 Physical Model

The physical assets of an enterprise are organized in a hierarchical manner, as shown in Figure 5.2. The physical model describes the physical assets of an enterprise in terms of enterprises, sites, areas, process cells, units, equipment modules, and control modules. The physical assets of an enterprise are usually grouped on a geographical basis, rather than in a strict hierarchical manner. Practically, geographical divisions apply at the higher levels

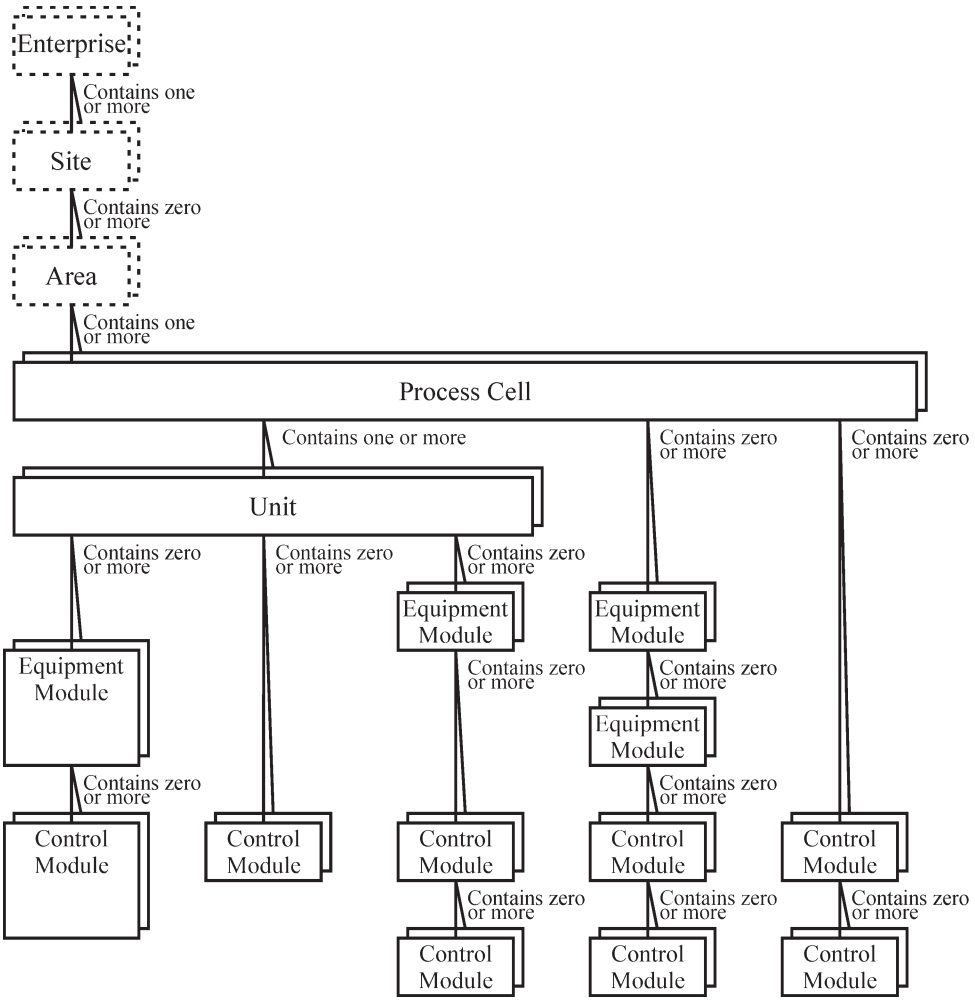

Figure 5.2. ISA-88.01 physical model. (Copyright © International Society of Automation. Used with permission of ISA.)

(enterprise, site, and area). Groupings at the lower levels (cell, unit, equipment module, and control module) are based on functionality.

An *enterprise* coordinates the operation of one or more sites. It is responsible for determining what products will be manufactured and in general how they will be manufactured. An enterprise can be confined to one country or may be global in scope.

A *site* is a geographical grouping determined by the enterprise. All physical equipment in a site share a common geographical location. A site is typically called a "plant", although enterprises sometimes divide a site into multiple "plants". Because of the confusion over this terminology, the word "plant" by itself is avoided in this model. A site consists of plant areas.

An *area* is a geographical grouping smaller than a site. An area is generally a process or a utility, but it could be something else, for example, a warehouse. An area is a physical, logical grouping of process equipment that performs the desired cell operations (mechanical, thermodynamic, chemical, biological) to make a product. Plant areas are often set by enterprise policy such as operator jurisdiction, product, or other criteria. The plant areas are sometimes called plant sections. A plant area is composed of process cells.

A *process cell* is a set of cooperating units. Typically, it is a logical grouping of equipment required to process one stream or manufacture one product or group of products. In a typical discrete-parts manufacturing process, a cell makes one product or processes one stream. Sometimes, an area is composed of two or more cells that operate in parallel and concurrently in making the same or different products. In this case, each cell is referred to as a process train or a process line.

A *unit* is a set of equipment modules and control modules; from an operating viewpoint, it is the smallest subdivision of the process plant equipment in an area or cell. A unit is usually centered on a major piece of equipment, such as an assembly machine or packager. For the purposes of motion control, a unit is the machine. Units often operate relatively independently of each other. Units may also be shared by more than one cell.

A unit consists of equipment modules and/or control modules. An *equipment module* (EM) is a subdivision of a cell, unit, or another equipment module. It is a functional part required to perform one or multiple process actions, such as,

Infeed section
Forming section
Filling section
Sealing section
Outfeed section

A *control module* (CM) is a subdivision of a cell, unit, equipment module, or another control module. Example control modules,

Individual sensor or actuator
Servo drive
Servo control function, such as cam or electronic gear
Safety light curtain

An equipment module may be part of a unit or a stand-alone equipment grouping within a cell. Equipment and control modules may be shared among cells or units. A stand-alone equipment/control module can be an exclusive-use resource or a shared-use resource. The equipment module contains all necessary physical processing and control

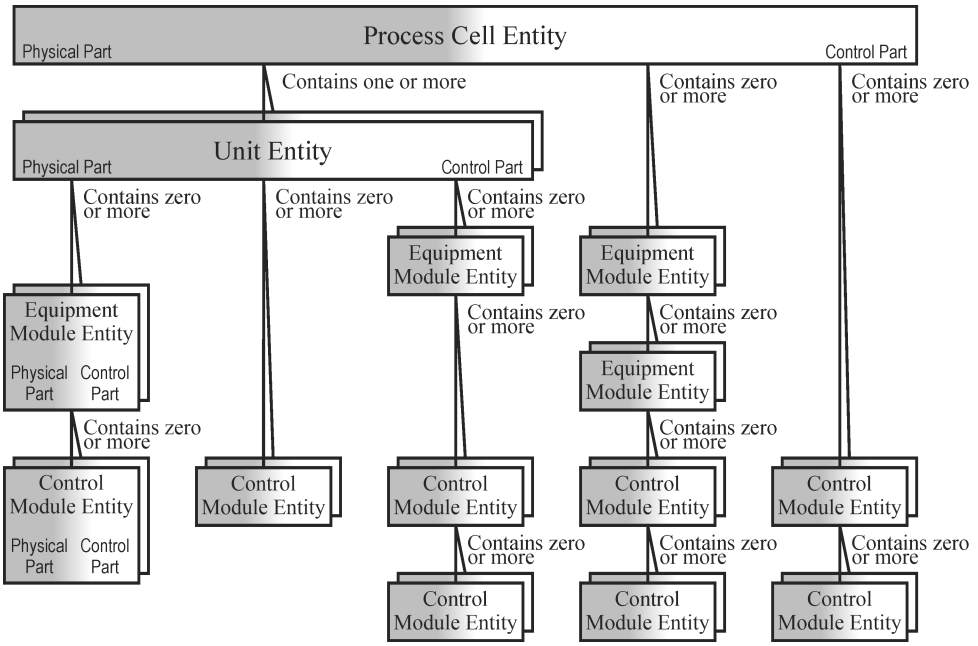

Figure 5.3. ISA-88.01 equipment entity model. (Copyright © International Society of Automation. Used with permission of ISA.)

equipment required to perform those activities. The scope of the equipment module is defined by the finite tasks it is designed to perform.

As can be seen in Figure 5.2, it is often hard to follow a strict hierarchy at the cell level and below. For example, equipment modules can be controlled from the cell level or the unit level.

Recognizing that it is often difficult to separate the physical equipment from its control, ISA-88.01 defines an *equipment entity model* for the cell layer and below, shown in Figure 5.3. Equipment entities are the combination of equipment control and physical equipment and the shading between the physical part and the control part indicates the boundary is frequently indistinct. For example, a servo loop includes the processor, drive module, and motor. Parts of the control algorithm are in the processor and in the drive module, and the particular organization depends on the PLC manufacturer. Conceptually, the servo loop is treated as one CM.

5.2.3 Control Model

The ISA-88.01 standard defines three types of control: basic control, procedural control, and coordination control. Basic control establishes and maintains a specific state of the equipment or process. It includes regulatory control, discrete control, sequential control, interlocking, monitoring, and exception handling. At the cell and unit levels, basic control is generally performed by equipment modules and control modules. In an equipment module, basic control is generally done by control modules. However, equipment modules may directly execute basic control, though the interaction with the physical equipment is through

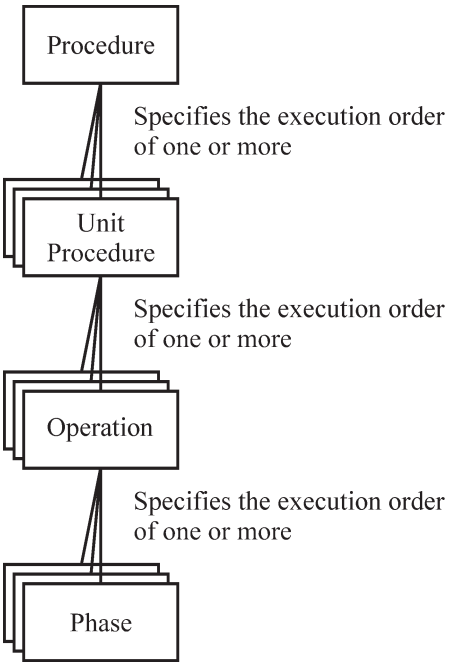

Figure 5.4. ISA-88.01 procedural control model. (Copyright © International Society of Automation. Used with permission of ISA.)

control modules. In ISA-88.01, the procedural control model is hierarchical, as shown in Figure 5.4, and is definitely biased toward batch control. Coordination control initiates, directs, and/or modifies the execution of procedural control.

In the procedural control model, the *procedure* occurs at the cell level and specifies the execution of one or more unit procedures. A *unit procedure* specifies a production sequence that occurs in a single unit and specifies the execution order of one or more operations. An *operation* is a major processing sequence, specifying the execution order of one or more phases. A *phase* is the lowest level procedural control and is a set of steps associated with specific equipment.

For a batch process, an example procedure and associated operations and phases is as follows

Procedure: Make *x* product

Operations: Bring raw ingredients from their tanks and mix together
Cook by heating to certain temperatures for certain lengths of time
Send product to next machine

Phases of cooking operation: Preheat to 150°
Cooking phase 1 at 150° for 10 min
Cooking phase 2 at 175° for 5 min
Cool product to 90° in 30 min

The relationship between the lower layers of the process model, physical model, and the procedural control model is shown in Figure 5.5. Note that there is not a strict

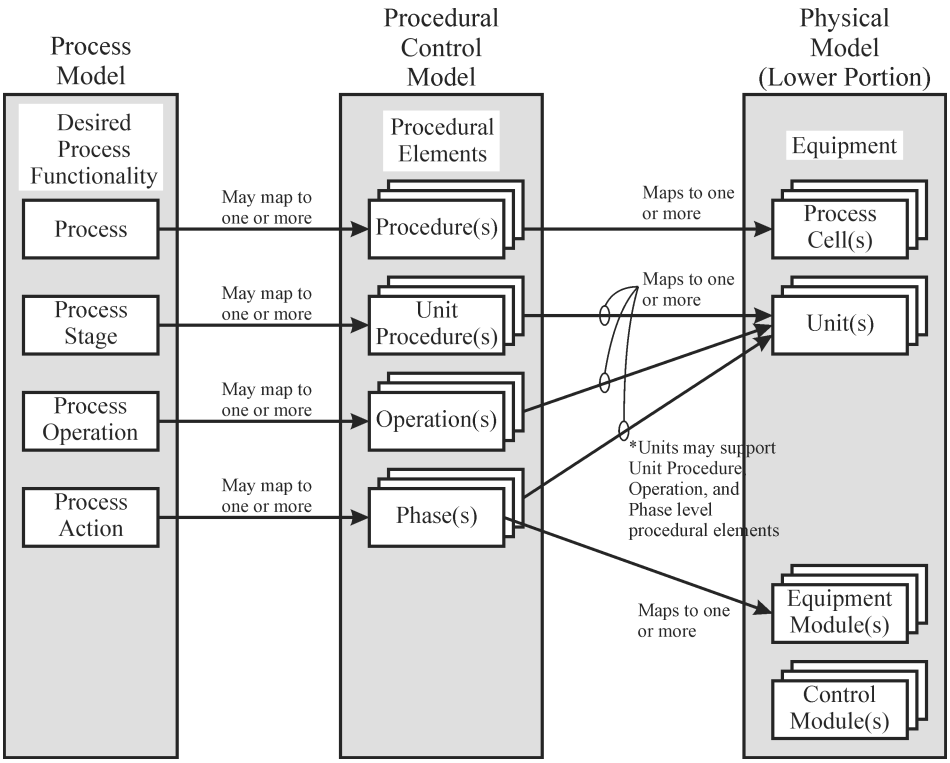

Figure 5.5. Relationship between ISA-88.01 models. (Copyright © International Society of Automation. Used with permission of ISA.)

hierarchical relationship. Phases can be associated with units and/or equipment modules. Units can have phases and operations in addition to unit procedures.

In the context of machines in discrete manufacturing, the focus is on the unit as the highest-level layer in the model, as that layer represents the machine. Furthermore, the phase level of the procedural control model is combined into the operation level. This "collapsing" of the procedural control model is allowed in ISA-88.01 since the operation layer assumes the functions of the phase layer. So, the control model used in this text is shown in Figure 5.6 and shows the separation between procedural control and basic control. All of these functions are assumed to reside in the PLC processor. The highest level of procedural control, procedure, is associated with cell control and is therefore assumed to be external to the PLC. The unit procedure handles the mode selection logic, fault handling, and local HMI interface. The operation layer implements the state logic associated with each of the machine modes. Each mode has an associated operation. Each operation then interacts with the appropriate equipment modules and control modules.

The state logic in the operation layer is described by the PackML state machine, the subject of the next section.

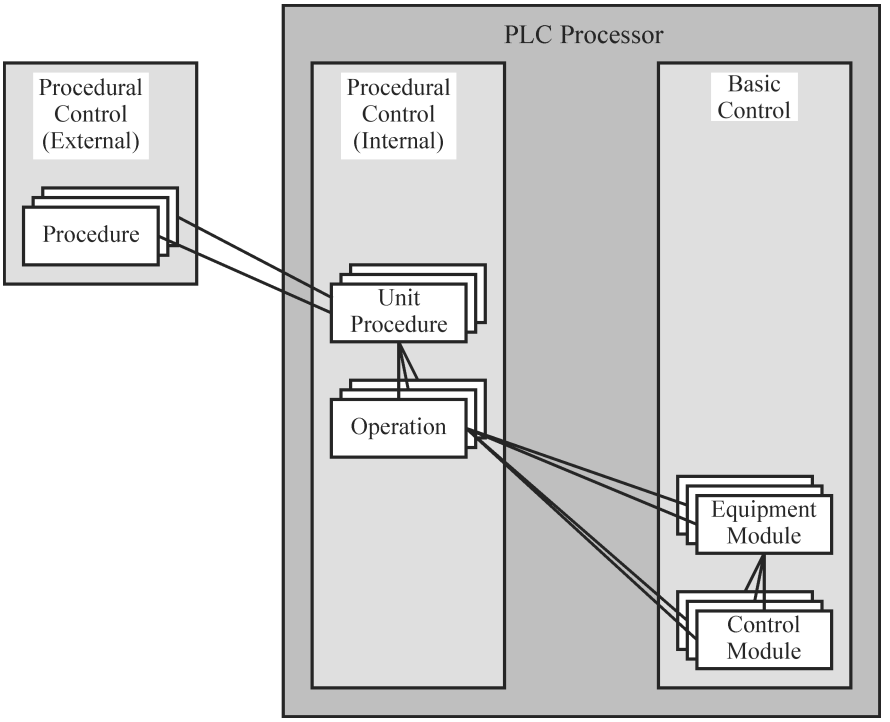

Figure 5.6. Machine control model.

5.3 PACKML STATE MACHINE

In PackML, the machine state specifies the current condition of the machine and defines how the machine will operate and how it will respond to commands. The concept of *state* can be also apply to equipment modules and control modules (Erickson and Hedrick, 1999), but PackML basically only considers the machine level. In addition, a machine can be in only one state at any one instant. Example machine states include Running, Idle, Holding, Paused, Stopped, and Aborted.

The PackML model defines two types of machine states:

Acting state – A state which represents a processing activity. Steps within the activity are processed in a certain order until a specific condition is reached. An acting state ends in "ing." In ANSI/ISA-88, an acting state is called a *transient state*.

Wait state – A state which identifies that the machine has achieved defined conditions. In a wait state, the machine is maintaining a status until transitioning to an acting state. In ANSI/ISA-88, a wait state is called a *final state* or *quiescent state*.

The PackML machine states are defined in ANSI/ISA-88 as *procedural states*, that is, the states of a procedural element. A *procedural element* is defined as a phase, operation, unit procedure, or procedure, the "building blocks" for batch control. In PackML there is only one major processing activity and so all of these elements lumped into a machine. In addition, PackML defines some machine states that are not ANSI/ISA-88 procedural states, and PackML does not include some of the ANSI/ISA-88 procedural states.

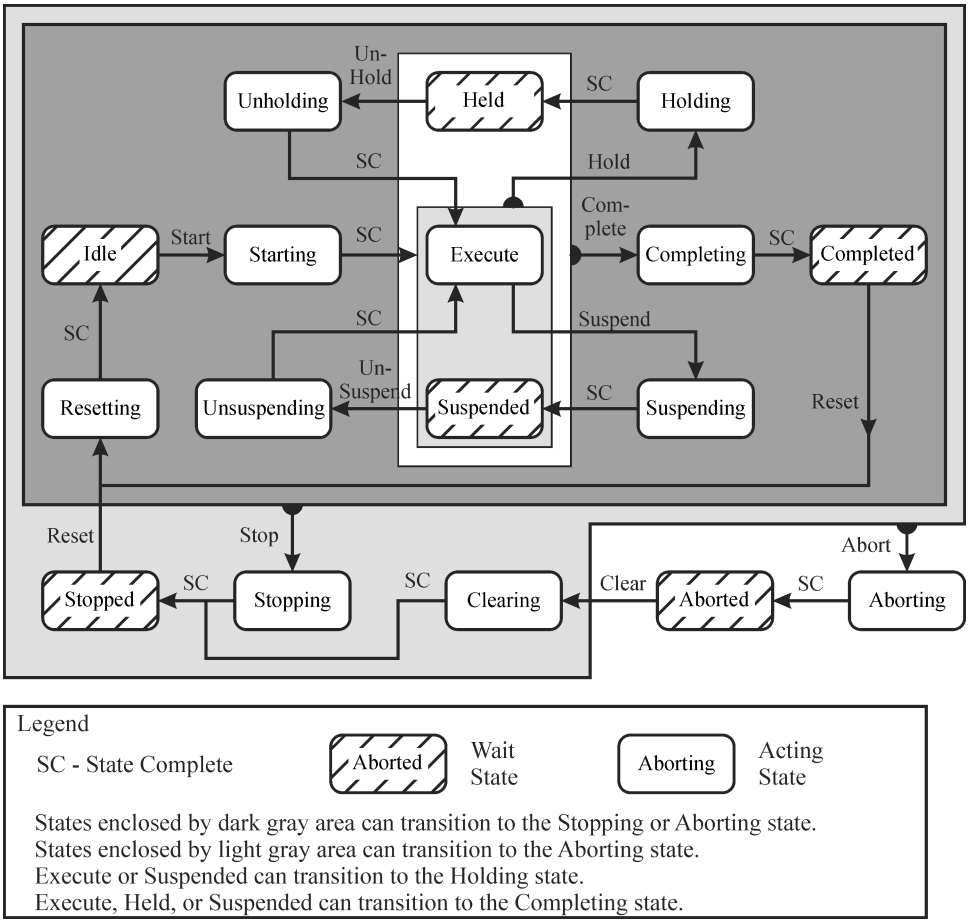

Figure 5.7. PackML complete machine state model.

Figure 5.7 shows the complete PackML state model with transitions between the states. The area boundaries group states together to simplify the depiction of the transitions. Any of the states in the dark gray area can transition to the STOPPING or ABORTING state and any of the states in the light gray area can transition to the ABORTING state. A transition to the HOLDING state is allowed from the EXECUTE or SUSPENDED states. A transition to the COMPLETING state is allowed from the HELD, EXECUTE, or SUSPENDED states. The hatched states are wait states. The "SC" transition designates "State Complete." The machine normally starts in the STOPPED state.

The machine states are described as:

STOPPED - The machine has completed its STOPPING or CLEARING logic and is waiting for a RESET command to transition to RESETTING.

RESETTING - The machine has received a RESET command while in the STOPPED or COMPLETE states. Anything that causes a stop is reset, and safety devices are energized. After these actions, the machine transitions immediately to the IDLE.

IDLE - The machine is waiting for a START command that will cause a transition to the STARTING state.

STARTING - The machine executes startup logic. Once complete, the machine automatically transitions to the EXECUTE state.

EXECUTE - Normal operation. Normal machine operations are executing. Generally, the machine remains in this state until all materials have been processed or until forced into another state by the operator or an abnormal condition.

COMPLETING – Normal processing of materials has run to completion and any special logic needed to bring the process to a shutdown state is executed. Once finished, the machine automatically transitions to the COMPLETED state.

COMPLETED - Once the machine has finished any special completion logic, the state changes to COMPLETED. The machine is waiting for a RESET command that will cause a transition to RESETTING.

HOLDING - The internal (inside this machine and not from another machine on the production line) conditions are such that the EXECUTE state must be exited. The HOLD command could come from an operator. Typical conditions include needing to refill a glue dispenser or carton magazine. The machine is typically brought to a controlled stop and then transitions immediately to the HELD state.

HELD - Once the machine has completed its HOLDING logic and is at the known state, the state changes to HELD. This state is normally used for a medium-term stop. A change in the machine condition or UNHOLD command causes a transition out of this state.

UNHOLDING - The machine has received a UNHOLD command while in the HELD state. If no sequencing is required, the machine transitions immediately to the EXECUTE state; otherwise restart logic is executed.

SUSPENDING - The external (outside this machine and usually from another machine on the production line) conditions are such that the EXECUTE state must be exited. The SUSPEND command could come from an operator. Typical conditions include upstream or downstream production line conditions. The machine is typically brought to a controlled stop and then transitions immediately to the SUSPEND state.

SUSPEND - Once the machine has completed its SUSPENDING logic and is at the known state, the state changes to SUSPEND. This state is normally used for a medium-term stop. A change in the machine condition or UNSUSPEND command causes a transition out of this state.

UNSUSPENDING - The machine has received a UNSUSPEND command while in the SUSPEND state. If no sequencing is required, the machine transitions immediately to the EXECUTE state; otherwise restart logic is executed.

STOPPING - The machine has received a STOP command and is executing its STOPPING logic, which facilitates a controlled shutdown. If no sequencing is required, then the machine transitions immediately to the STOPPED state.

ABORTING - The machine has received an ABORT command and is executing its ABORTING logic, which facilitates a quicker emergency shutdown. It is usually controlled and even a coordinated stop but one does not finish any processes. Generally, little sequencing is required, and the machine or equipment entity transitions immediately to the ABORTED state.

ABORTED - The machine has completed its ABORTING logic and is waiting for a CLEAR command to transition to CLEARING.

CLEARING - The machine or equipment entity has received a CLEAR command while in the ABORTED state. This state typically clears faults that occurred while ABORTING. Once faults are cleared, the state automatically transitions to STOPPED.

In an actual process, the EXECUTE state may be broken down further. For example, a processing machine may have Feeding, Stamping, and Ejecting states.

A machine does not need to have all of these states. In order for a machine to be considered an appropriate PackML implementation, it must have at least the STOPPED, IDLE, EXECUTE, and ABORTED states. All other states are optional.

5.4 PACKML MODE

PackML machine control mode "determines the subset of states, state commands, and state transitions that determine the strategy for carrying out a unit/machine's process." (ISA, 2015). Typical machine control modes include production, manual, semi-auto, and maintenance. A particular control mode defines the set of states, state commands, and state transitions.

ANSI/ISA-TR88.00.02-2022 (ISA, 2022) defines three control modes:

Production mode – Machine is operating in normal production. Commands may be from an operator or supervisory controller.

Maintenance mode – Machine is being operated independently from the production line. Authorized personnel are generally troubleshooting the machine, testing changes to the machine, or initially commissioning the machine.

Manual mode – Individual machine modules are being directly controlled. For example, this mode allows for individual drive testing and commissioning.

OMAC later (Nokleby, 2016) defined four additional modes:

Change Over – For machine recipe changeover.

Clean – For routine machine cleaning.

Set Up – For mechanical adjustments and testing.

Empty Out – For emptying machine at the end of a shift or for emptying out residual product within the machine that could be sent for processing by subsequent machines rather than sent to product rework.

A Semi-Auto mode is defined by Aleksa (2010). Other modes are possible and left to the discretion of the implementer.

Each control mode has its own state model. The change from one mode to another mode must be appropriately managed, as it is generally possible to only change modes when in certain states. Typical transitions between modes occur only at wait states. Generally, the mode changes only when the machine is in the Stopped state. Though, when in the Aborted state, the mode can change to Manual. Also, the PackML state does not change when the mode changes. For example, if a machine is in the Production mode and in the Stopped state, a transition to the Maintenance mode does not change the state. The machine is still in the Stopped state.

Example state models for the Maintenance and Manual modes are shown in Figures 5.8 and 5.9. In the Maintenance mode, the Suspended state is not needed, as they should be no external conditions to force this state. Also, note that the operations/functions of the

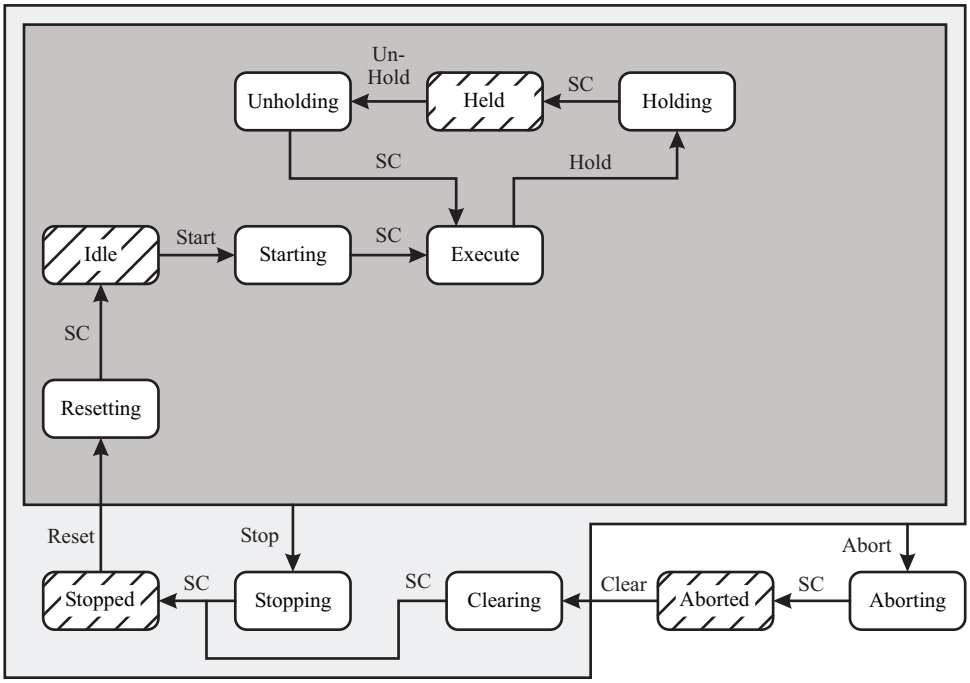

Figure 5.8. Maintenance mode reduced PackML state model.

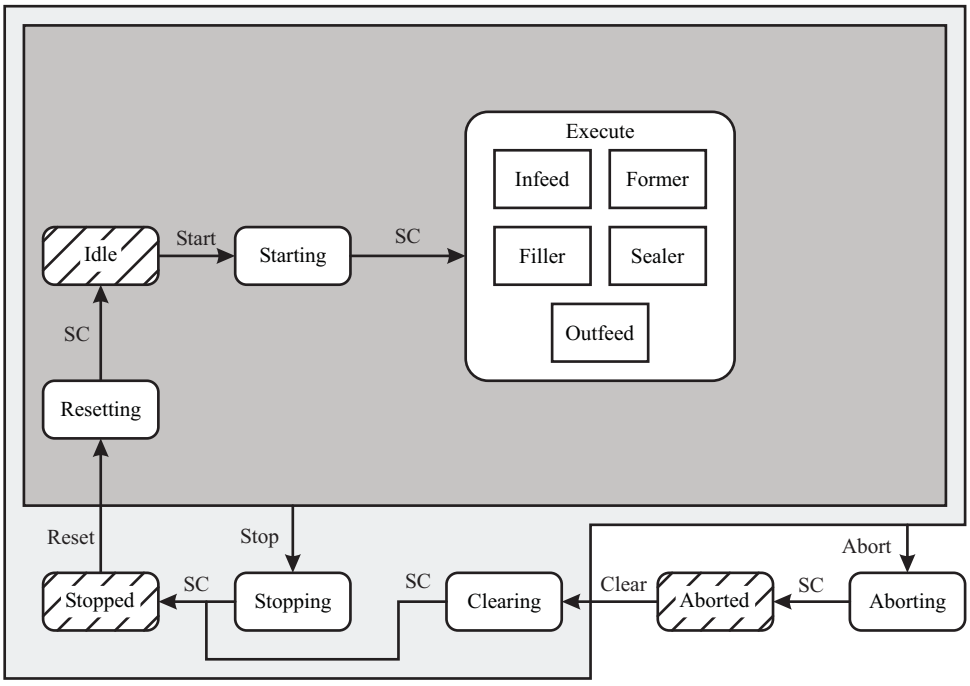

Figure 5.9. Manual mode reduced PackML state model.

Execute state are different than that for Production mode. In the Manual mode, both Suspended and Hold states are not needed and the Execute state is expanded to show the individual equipment modules that may be individually controlled.

5.5 PACKML TAGS

PackML also defines a naming convention for the data elements used within the state model, called *PackTags*. PackTags are useful for data exchange between machines and between the machine and the human-machine interface and higher-level systems, such as enterprise information systems. The syntax, contents, and structure of the tag (variable) lists for signals which are significant for the control system and for line visualization are defined in ISA (2022).

PackTags are divided into three groups:

Command – data consumed by the machine to control the operation of the machine. These tags include commands to control the unit state and unit mode transitions as well as parameters and process variables.

Status – data produced by the machine describing the machine operation. These tags include information about the machine states and modes as well as information about the machine parameters and process variables.

Administration – data collected by higher-level systems for machine performance. These tags include alarm information.

The PackTags are contained in one data structure, each section indicated by the appropriate prefix:

Command – command tags
Status – status tags
Admin – administration tags

Table 5.1. Minimum Set of PackML Tags

Tagname	Data Type
UnitName.Command.UnitMode	Int (32 bit)
UnitName.Command.UnitModeChangeRequest	Bool
UnitName.Command.MachSpeed	Real
UnitName.Command.CntrlCmd	Int (32 bit)
UnitName.Command.CmdChangeRequest	Bool
UnitName.Status.UnitModeCurrent	Int (32 bit)
UnitName.Status.StateCurrent	Int (32 bit)
UnitName.Status.MachSpeed	Real
UnitName.Status.CurMachSpeed	Real
UnitName.Status.EquipmentInterlock.Blocked	Bool
UnitName.Status.EquipmentInterlock.Starved	Bool
UnitName.Admin.ProdProcessedCount[#].Count	Int (32 bit)
UnitName.Admin.ProdDefectiveCount[#].Count	Int (32 bit)
UnitName.Admin.StopReason.ID	Int (32 bit)

ANSI/ISA-TR88.00.02 (ISA, 2022) defines over 200 tags; the minimum set of tags is shown in Table 5.1

5.6 ALARM/FAULT HANDLING

In every application, unexpected behavior needs to be detected and the appropriate action taken in order to protect operators and the machine from harm or to prevent unacceptable product. The alarm condition might originate either from:

The physical equipment, for example, an axis is physically blocked so it cannot reach the target position,

The application program, for example, a motion block that has an error,

The operator, for example, pressing the e-stop button.

In addition other conditions such as low packaging material or low product level are detected and reported so that the operator has sufficient time to take appropriate action before the machine can no longer continue to produce product.

The basic alarm categories are

Major fault, which causes the State Machine to go the Aborting state

Minor fault, which causes the State Machine to go to the Stopping or Holding state

Warning, which will alert the operator but causes no action

Depending on the application, alarm categories may be further subdivided according specific stopping / aborting actions that are appropriate for that machine. For example:

Emergency stop – Aborting state and all axes and equipment are stopped individually as fast as is acceptable for the mechanics. Note that for an emergency stop other safety measures are usually in place which are responsible for removing the energy sources after a defined time delay.

Fast stop – Aborting state and the axes remain synchronized and stop together quickly without regard to the final position of the machine.

Normal stop – Stopping state or Holding state and the axes remain synchronized and stop at a defined position or place in the production sequence. This is the same action that is taken if a stop command comes from the operator or other source.

Run empty – A certain number of machine cycles are executed to finish product already in the machine. Then a normal stop is executed.

Each module (EM or CM) generates its own alarms and assigns each individual alarm to the appropriate category. Often alarms set individual bits from a fault integer for that category and a general fault bit for that category is active. In order to easily find the alarms, they are often placed in a Fault routine or at the bottom of the other routines.

The next higher level of module collects all the fault information from the module underneath it. For example an EM collects the fault information from each of its CM's and the Unit collects the fault information from each of the EM's. Then the Unit will issue the Aborting command or Stopping command for the State machine and set any other status bits that are necessary for the EM and CM to execute the appropriate action.

In addition all alarms should be displayed on the operator interface, for example on an alarm display and single line banner at the top of each display. This will inform the operator the reason why the machine has aborted or stopped and in case of a warning, of action which should be taken. Fault handling is included in the implementation example in section 5.8.

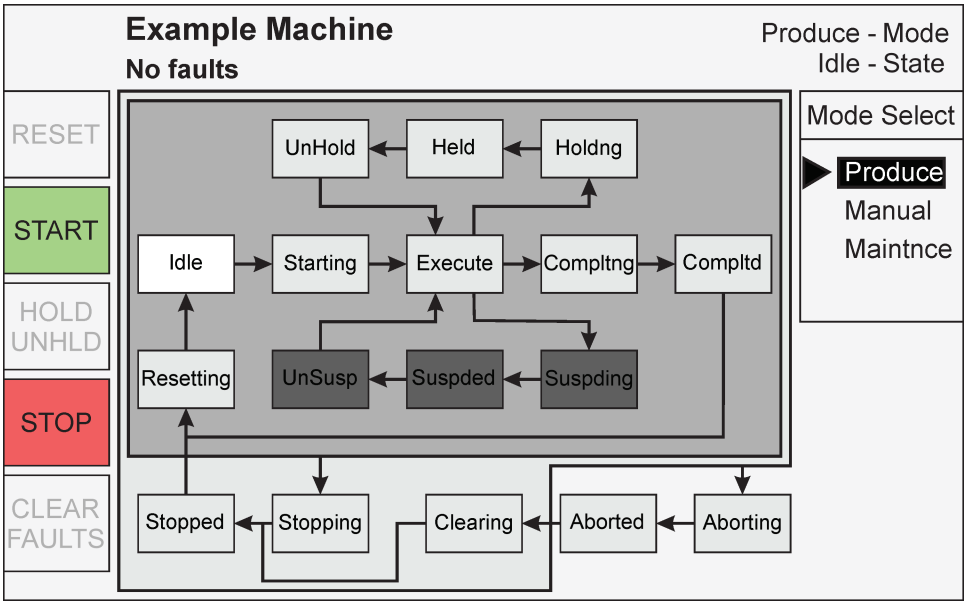

Figure 5.10. Example PackML state model operation display.

5.7 OPERATOR INTERFACE

Rather than show a complete operator interface, only a display showing the PackML mode/state status and commands is shown in Figure 5.10. This display is representative of a touch-screen display. The main part of the screen shows the status of each state. The current state is shown as a white rectangle. Disabled states are shown as dark gray. Selection of the mode is shown on the right side. Buttons for state commands are shown on the left side. Only valid state commands for the particular state are shown in color. Invalid state commands are shown as muted gray. For the display in Figure 5.10, the current state is Idle and the Start and Stop commands are valid. The top part of the display is a banner that appears on other screens and shows the mode and state as text in the upper right of the display.

5.8 EXAMPLE APPLICATION

A simple two-axis application, shown in Figure 5.11, is used to demonstrate an implementation of the PackML approach to machine control. The machine basically moves a part from one position to another. The horizontal movement is the X-axis (positive direction to the right) and the vertical movement is the Z-axis (positive direction up). Both of these axes are shown as linear lead screw drives, but they could be linear belt drives. The "home," or starting, position is on the left and down. During the execute state, the sequence is as follows:

1. Wait for a "cycle start" signal, indicating part has been placed in holder.
2. Move horizontally right to desired X-axis position.
3. Move vertically up to desired Z-axis position.
4. Wait for a "cycle finish" signal, indicating part has been removed.

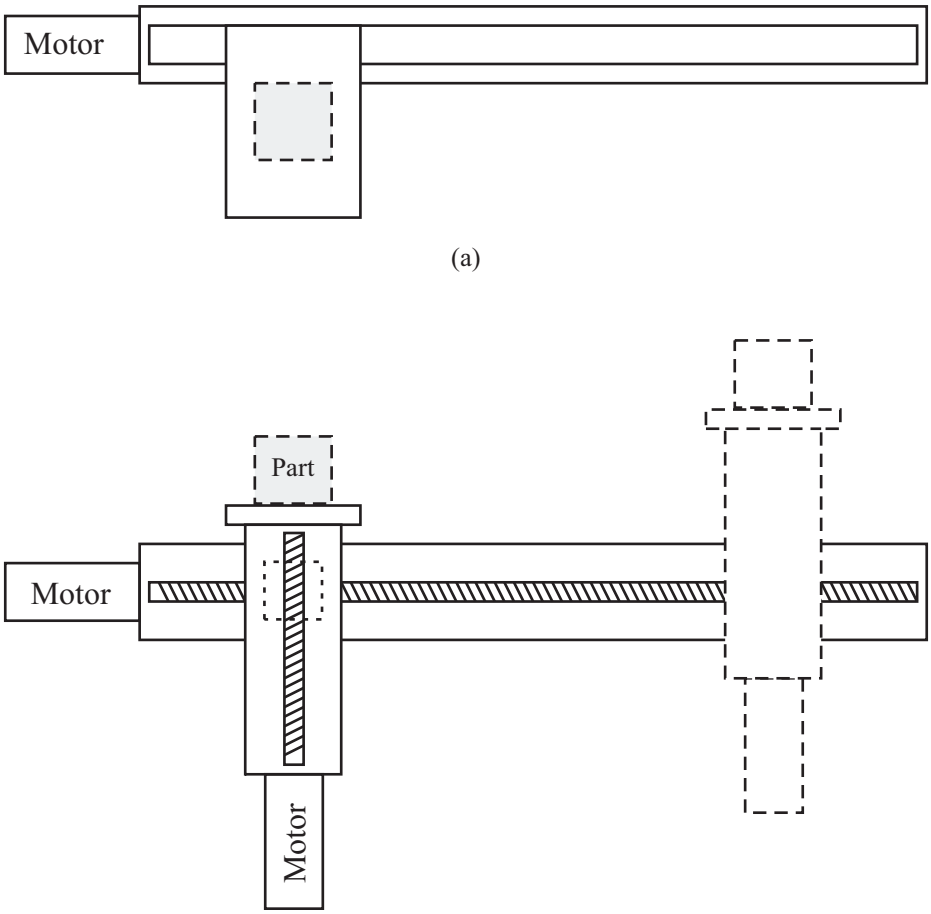

Figure 5.11. Two-axis system: *(a)* top view; *(b)* side view, showing intermediate position.

5. Move vertically down to Z-axis home position.
6. Move horizontally left to home position.

The sequence then repeats. This particular machine is simpler than a real application, but will be used to illustrate the basic concepts of a PackML implementation. The machine has one equipment module (EM) and the EM consists of two axes, each of them a control module (CM).

5.9 ROCKWELL IMPLEMENTATION

The ControlLogix implementation is based on the Power Programming (PP) V4.2 example program (Rockwell Automation, 2020). The implementation generally follows the "Basic" template that has a simplified program structure and error handling. The complete project is included in the text supplementary materials.

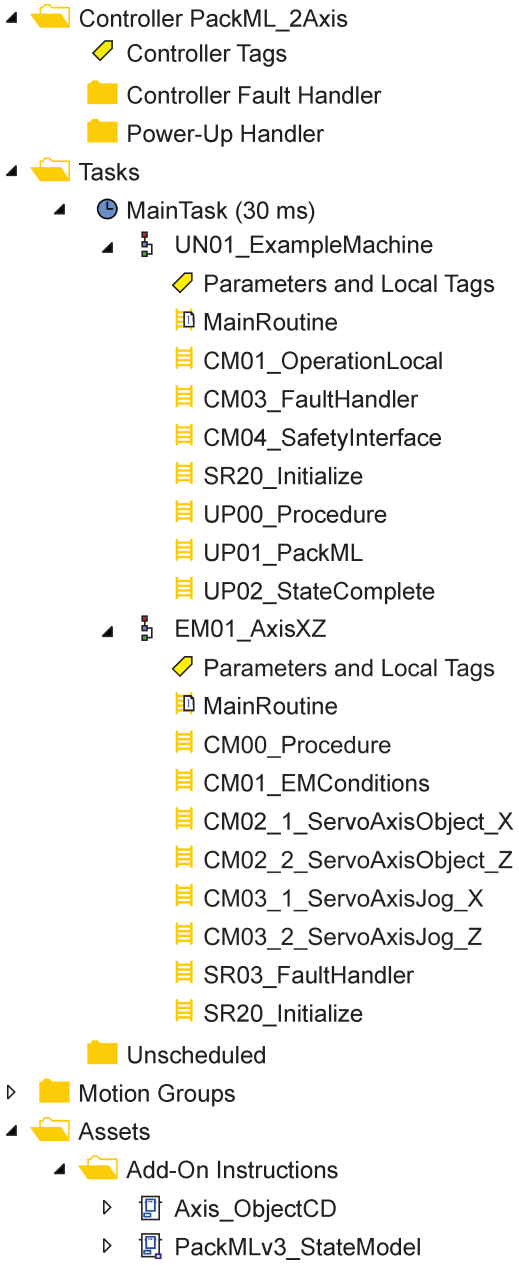

Figure 5.12. Overall structure of ControlLogix project.

The overall structure of the ControlLogix program is shown in Figure 5.12. The motion groups, data types, and I/O configuration are not shown. The main task is divided into two programs:

UN01_ExampleMachine – handles unit-related functions, including the overall state machine, operator interface, and main fault handling.

EM01_Axis_XZ – handles EM-related and CM-related functions, including the sequence during the execute state, the individual axis control, and fault handling.

This approach deviates from the PP examples. In the PP examples, each axis is treated as a single EM and thus each axis is contained in a different program. While that approach is valid and allows for easy replication of axes, in most applications it is more logical to group related axes together. In this particular simple example, if each axis is treated as an EM, then the sequence becomes a unit procedure, probably an exaggeration. This approach of combining multiple axes into an EM is also used in the examples in later chapters.

The routine name prefixes are explained as follows:

CMxx – control module

EMxx – equipment module

UNxx – unit

UPxx – unit procedure

SRxx – subroutine (general logic, such as initialization and fault handling)

The routines in the UN01_ExampleMachine program are described as follows:

MainRoutine – Basically just "calls" the other routines in the program.

CM01_OperationLocal – Handles the commands for the PackML state machine. Any operator-related commands are combined with machine-related information such as faults, upstream/downstream conditions, execution completion.

CM03_FaultHandler – Handles the machine faults, including the unit fault summary and alarm annunciation for the HMI.

CM04_SafetyInterface – Connects the safety function block outputs, which are in safety tags to the machine program non-safety tags.

SR20_Initialize – Initializes unit-related tags.

UP00_Procedure – Any unit sequential procedures are in this routine. Since this application has no unit procedure, it is essentially "empty."

UP01_PackML – Interface to the AOI that implements the PackML state machine.

UP02_StateComplete – Generates the State Complete input to the state machine from the individual EM State Complete statuses.

The routines in the EM01_Axis_XZ program are described as follows:

MainRoutine – Initializes constants that identify the EM number and "calls" the other routines in the program.

CM00_Procedure – The EM sequential procedures that handles the operation in the execute state is in this routine.

CM01_EMConditions – Generates EM State Complete status for the acting states.

CM02_1_ServoAxisObject_X – Interface to the Axis_ObjectCD AOI for the X axis. The "1" in the name refers to the machine axis number.

CM02_2_ServoAxisObject_Z – Interface to the Axis_ObjectCD AOI for the Z axis. The "2" in the name refers to the machine axis number.

CM03_1_ServoAxisJog_X – Handles X axis jog when in the manual mode.

CM03_2_ServoAxisJog_Z – Handles Z axis jog when in the manual mode.

SR03_FaultHandler – Handles EM faults. Summarizes axis and procedure faults.

SR20_Initialize – Initializes EM-related tags.

There are two add-on instructions (AOIs):

Axis_ObjectCD – Performs the enable, disable, fault reset, home, stop, abort, diagnostics, and status functions for a servo axis. This particular AOI works with Ethernet/IP drives.

PackMLv3_StateModel – Handles the PackML state machine transitions between modes and states.

The PackMLv3_StateModel AOI follows the 2015 version of the PackML standard (ISA, 2015) which had fewer transition possibilities. The AOI has the following changes for Figure 5.7:

The COMPLETED state is named COMPLETE.

The transition from SUSPENDED to HOLDING is not allowed.

Transitions from HELD or SUSPENDED to COMPLETING are not allowed.

Two controller tags need further explanation, as they are important for communication between the unit program (UN01_ExampleMachine) and the EM's. The "UN01_MachineStateModel" tag holds information about the state of the unit. The more important parts of this structure are:

Name	**Type**	**Description**
.Cmd_Mode	Dint	Commanded mode
.Cmd_Reset	Bool	Command to go to Reset state
.Cmd_Start	Bool	Command to go to Start state
.Cmd_Stop	Bool	Command to go to Stop state
.Cmd_xxx	Bool	Command to go to "xxx" state
.Cmd_StateComplete	Bool	"State Complete" command
.Sts_StateCurrent	Dint	Current state number (1-17)
.Sts_ModeCurrent	Dint	Current mode number (1-31)
.Sts_Clearing	Bool	Clearing state is active
.Sts_Stopped	Bool	Stopped state is active
.Sts_xxx	Bool	"xxx" state is active

Many of these parameters can be seen in rung 11 of the UP01_PackML routine in Figure 5.14. There is also a set of EM_<State>_Done tags for the acting states, each of them a DINT and each bit represents the "state complete" bit for an EM. Bit 0 is for EM #0, bit 1 is for EM #1, and so on. The list of these tags is:

Name	**EM "state complete" for**
EM_Aborting_Done	Aborting
EM_Clearing_Done	Clearing
EM_Completing_Done	Completing
EM_Execute_Done	Execute
EM_Holding_Done	Holding
EM_Resetting_Done	Resetting
EM_Starting_Done	Starting
EM_Stopping_Done	Stopping
EM_Suspending_Done	Suspending
EM_UnHolding_Done	Unholding
EM_UnSuspending_Done	Unsuspending

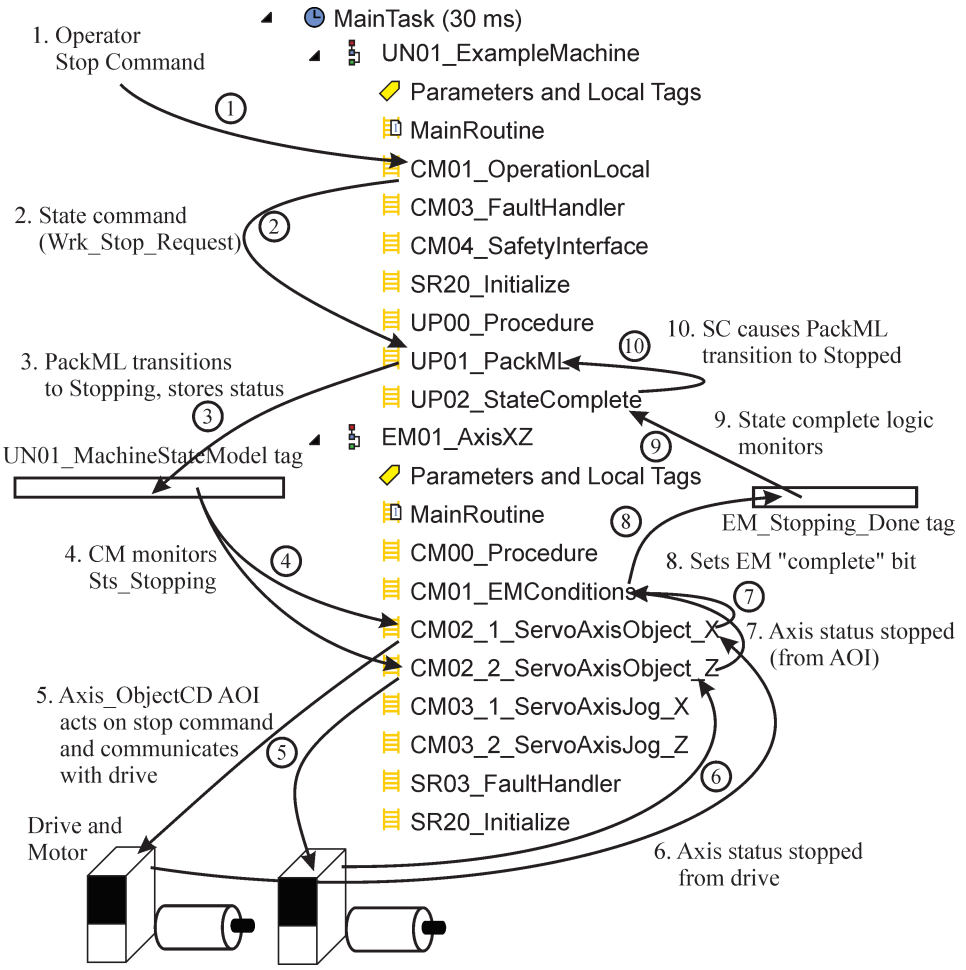

Figure 5.13. Command and status signal flows for stop command.

The signal flow between the various routines in response to a command is shown in Figure 5.13 for a stop command from the operator. The ladder logic code for the routines in the UN01_ExampleMachine program are in Figure 5.14 and Figure 5.15 shows the ladder logic for the routines in the EM01_Axis_XZ. The ladder logic for the Axis_ObjectCD AOI is shown in Figure 5.16. Note that not all of the ladder logic is shown.

The steps in the process shown in Figure 5.13 are as follows:

1. Logic in CM01_OperationLocal senses operator stop command, either from a hardwired switch or HMI. This routine turns on the Wrk_Stop_Request tag (CM01_OperationLocal rung 4, Figure 5.14).
2. Logic in UP01_PackML senses Wrk_Stop_Request and generates a command to the state machine AOI to change to the Stopping state (rung 8, Figure 5.14).
3. The PackMLv3_StateModel AOI executes, transitioning to the Stopping state. Status is stored in UN01_MachineStateModel.Sts_Stopping (UP01_PackML rung 11, Figure 5.14).

4. The logic in both CM02_1_ServoAxisObject_X and CM02_1_ ServoAxisObject_Z sense the Sts_Stopping bit. Each routine generates a command to the Axis_ObjectCD AOI (one for each axis) to stop the axis (rungs 3 and 5 in each routine, Figure 5.15).
5. Both instances of the Axis_ObjectCD execute motion instructions to stop the axis. The instructions act on the two drives to stop the axes.
6. Each drive generates an "axis stopped" status when the axis is stopped.
7. The stopped status of each axis is sensed by the respective Axis_ObjectCD instance and passed to a Sts_StopDone bit, one bit for each axis (rung 5 in both CM02_1_ServoAxisObject_X and CM02_2_ServoAxisObject_Z routines, Figure 5.15).
8. The logic in CM01_EMConditions rung 8 (Figure 5.15) uses the Sts_StopDone for each axis, and the Sts_Stopping from the state machine, to set the EM_Stopping_Done bit cor,esponding to the EM (EM #1).
9. The logic in UP02_StateComplete rung 3 (Figure 5.14) senses that all configured EM's (only one in this case) have their corresponding bit set in EM_Stopping_Done and then sets the Cmd_StateComplete bit.
10. The PackMLv3_StateModel AOI executes (UP01_PackML rung 11, Figure 5.14), transitioning to the Stopped state. Status is stored in UN01_MachineStateModel.Sts_Stopped.

The previous example illustrated the general flow of logic in response to an external command. The logic in Figures 5.14 through 5.16 is largely self-explanatory. A few aspects need further explanation. For the EM01_Axis_XZ program, a few program tags are aliased to controller tags:

EM01_Axis_XZ program tag	**Aliased to controller tag**
Inp_MachineStateModel	UN01_MachineStateModel
Ref_ServoAxis_1	Axis_1_X
Ref_ServoAxis_2	Axis_2_Z

The tag prefixes are explained as follows:

Inp_xx – input connection to real input/output point or to another block
Cmd_xx – command input from the operator or from the program
Cfg_xx – configuration value, typically seldom changed
Par_xx – parameter, typically changed regularly
Wrk_xx – working tag used internally by program or AOI
Sts_xx – status output of AOI

The safety aspects of the example, rung 4 of the SR20_Initialize routine, have been disabled and the CM03_FaultHandler routine is empty. Example 11.5 in section 11.8 shows the modifications to the program to implement an E-stop and light curtain.

The sequence to move the two axes is contained in the CM00_Procedure routine. The motion instructions to do the moves are in this routine. The step number is contained in the Wrk_EM_ProcedureStep tag. Note that the sequence is not started unless the Execute state is active in the Produce mode and that the state must remain in Execute in order for the sequence to proceed. If a hold is initiated while a step is active, any move command is completed, but the sequence remains in this step until an unhold is initiated and the Execute state resumes. The sequence repeats until the machine is stopped or a fault causes an abort.