Designing your first PackML implementation for machine control

Three key design decisions to get you started

Douglas Meyer, Motion Application Engineer
Yaskawa America, Inc.

Table of Contents

Designing your first PackML implementation for machine control

There can be no doubt that a majority of controls engineers in the packaging industry have at least heard of the term “PackML” by now. For the past two years, Dr. Bryan Griffen, co-chair of the OMAC Packaging Workgroup and Electrical and Automation Group Manager at Nestlé, has led a renewed effort to communicate the benefits of PackML at nearly every industry trade show and conference (Campbell, 2011). But understanding how PackML is actually implemented on a real machine, and having experience in designing and coding such an implementation is likely a different story. For many engineers in the packaging industry, and for nearly all engineers outside the industry, PackML remains an amorphous concept, not completely understood and missing a clear best practice for getting started. This Tactical Brief will set aside the discussions of ‘Why PackML?’ and instead focus on ‘How PackML’ so that controls engineers in any industry can better understand the design and implementation process.

PackML - not just for the Packaging Industry

It is important to say up front that, despite its name and origins, PackML can be applied to *any* automated machine, regardless of industry. Technically, PackML is known as ISA-TR88.00.02 and was born out of the need to improve production efficiencies at packaging end-user locations and to reduce the amount of time and expense involved with integrating machines from different suppliers into a cohesive production line. The intention was to build a global standard for automated machine code architecture that could be adopted by packaging OEMs to ease the line integration and support processes. What emerged is a code architecture that applies to machinery beyond the packaging industry.

The heart of the current PackML v3.0 standard is the machine state diagram. Starting in the early 2000’s, the OMAC (The Organization for Machine Automation and Control) Packaging Workgroup (OPW) spent a good deal of time breaking down the operational sequencing of automated machinery and creating a standard model of logical states and transitions. The resulting state diagram for ISA_TR88.00.02 was released in 2008 and is shown in Figure 1. (ISA, 2008). Note that this model contains nothing specific about the machinery used in the packaging industry. Instead, the model uses a simple flow diagram to define a generic, standard nomenclature for how machines operate. All machines will have a state where the machine is running the intended process (Execute) and not running (Stopped or Idle). Similarly, all machines must apply some sort of fault detection and recovery sequence (Aborting, Aborted, Clearing).

The additional Held and Suspended branches were added by the OPW to further identify those times when the machine may be capable of producing something, but other conditions are impeding production. The Suspended branch is for use if the machine is waiting (starved) for material from an

upstream process or blocked by a downstream process. The Held branch is intended for operator-induced production holds such as a Pause function.

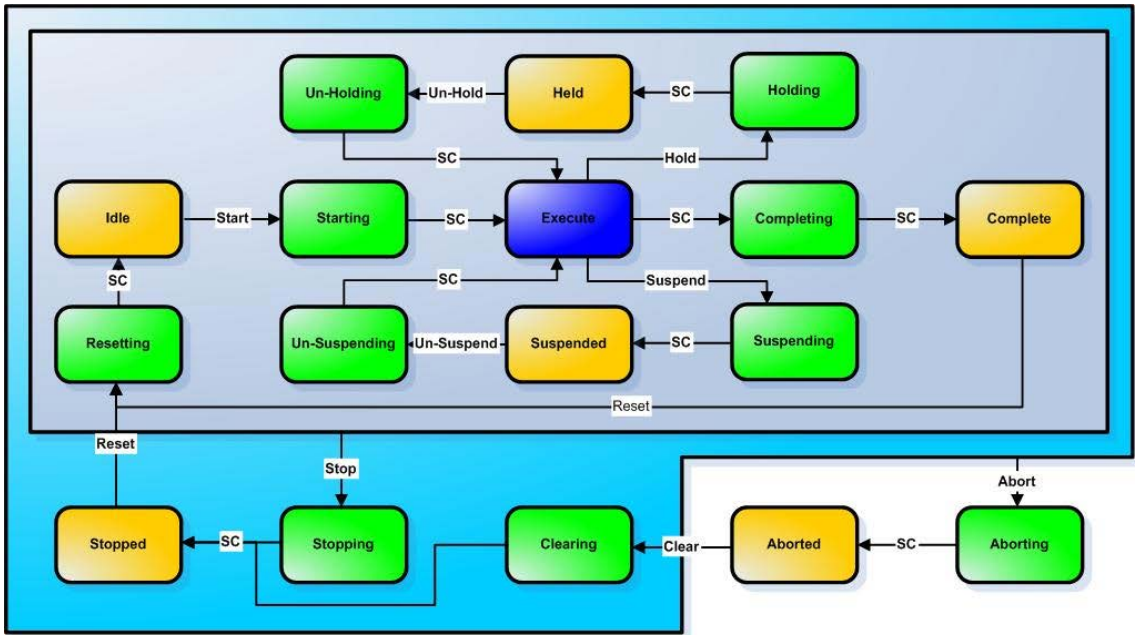

Figure 1: PackML State Model.

The transitional flow of the model is important. Observe that the model forces states to be activated in a sequential manner with exceptions only for Stop and Abort. Therefore the model is useful not only for monitoring the current state, but also for driving to the next state, and thus can serve as the high-level master for machine sequencing.

In addition to the standard state model, PackML provides for user-defined *modes of operation* that may allow or restrict access to certain states or branches. Think of modes as a third dimension of the PackML State Model that gives the model layers of depth, like floors in a building. Like a building, not all the rooms (states) may be accessible, and movement between floors is only possible at stairway locations (transition states). It is with the definition of these operational modes that the implementation process begins.

Your Implementation – Key Design Decisions

Define the PackML Modes to be used

The first major decision to make in your new PackML-based application is ***which modes are included and what they are named***. Although PackML technically allows for an unlimited number of machine modes, most applications have some version of these three: Automatic, Manual, Maintenance. The names are user-definable and the programmer can also define which states are accessible when each mode is active.

Typically, *Automatic Mode* is used for normal production and includes full access to all the PackML states in the model. *Maintenance Mode* is often used to run all sections or individual sections of the machine in a 'dry cycle' manner for setup, debug, or testing. Certain branches such as the Held or Suspended branches are usually disabled while in Maintenance Mode. In many implementations, Maintenance Mode is entry-protected with a password that limits access to only authorized users. *Manual Mode* is used for manual operation of individual mechanisms on the machine, most often for setup, commissioning or debug by an authorized user. It may be best to disable virtually all of the PackML States while in Manual Mode.

To complete the mode definition, the user will specify the states at which the mode can be changed. These transitional states are generally set to 'quiet' states such as Aborted, Stopped and Idle where the machine is not producing anything.

Assign functionality to PackML Modes and States

While the PackML standard does a great job at defining the state names and transitions, it leaves the decision of what happens in each of those states up to the user. Therefore, the second major decision is to ***clearly define what machine functions happen in each state***. For example, if there are servo axes on the machine, when are they enabled – in the Stopped State?, during Resetting State? Where are servo axes disabled – during Stopping? Only during Aborting? When Aborted is reached? Another good example is in which state is a Homing function employed – Resetting? Starting? How about a CycleStop function? Should CycleStop code reside in Stopping or Completing?

All these decisions and more should be clearly identified in a document so that all project engineers can achieve a uniform understanding of what-happens-when during the overall machine sequence. Using the PackML nomenclature and state model allows engineers to speak about machine operations in a common, high-level manner that improves communication and speeds development. Bryan Griffen of Nestlé comments that this step is crucial to realizing the benefits of PackML because it allows Nestlé to clarify the whole task of interpreting and implementing the state model. A clear specification can then be delivered to OEMs that is process-specific instead of hardware-specific. This allows OEMs to choose best-fit hardware so long as the machine controls can integrate horizontally and vertically into the Nestlé line. (Reynolds, 2011).

Modularize the machine code

The PackML State Model works best when implemented in a modular way. By that, it is meant that PackML provides the supervisory commands and status for high-level overall machine sequencing that can be passed down to functional code modules. The modules, in turn, send back completion status that the state model uses to move to the next state. By performing the next step to ***separate the code into logical modules that match the physical machine construction***, the foundation is laid for code that is more organized, more reliable, easier and faster to debug, and more easily reusable in other applications. For PackML, such a modular code model is further defined in S88:Make2Pack.

The ISA88 physical hierarchy for code modules contains six levels that range from the entire global company level down to each individual function. For this discussion, we will focus on the bottom three that relate to an individual machine:

- **Machine (also known as Unit, or UN):** a collection of related modules (mechanical and electrical assemblies) that carry out one or more processing activities
- **Equipment Module (EM):** a functional group of modules that carries out a finite number of activities
- **Control Module (CM):** the lowest level of control where a single function is executed. (ISA, 2008)

High-level PackML commands originate at the Unit Machine level and flow down through all the Equipment Modules to the Control Modules as shown in Figure 5. In turn, the completion status of each module is reported and transmitted back up the chain.

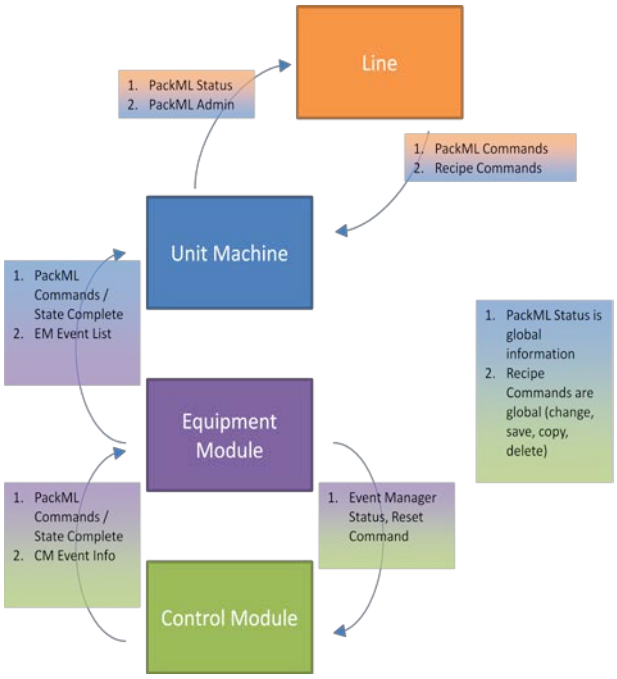

Figure 2: Data Flow in a PackML application (OMAC, 2009)

For most applications, the code deals with only one single machine or unit. A machine is made up of one or more Equipment Modules, each of which contains one or more Control Modules. The real decision is how to define the EMs and CMs for logical and efficient operation. One could easily get carried away and define a Control Module for every device on the machine. However, given that a machine of medium complexity may have a hundred or so devices, this approach might be a bit impractical. Instead, it is better to focus on slightly larger groups of functionality where the *reporting of PackML State completion* is needed.

Try to avoid using a single CM in an EM. If that is the case, perhaps that piece of equipment performing a single function should become a CM in a larger EM. Likewise, if a CM has no individual reporting impact on the completion of a PackML State, then that CM functionality should be included elsewhere in a different CM.

Sometimes, it may make sense to include multiple devices in a control module if they are linked in some way. For instance, if one servo is a master to another servo in a camming relationship, these two devices should perhaps be contained in a single CM since they are so closely tied and their *combined* status is what is most important to the high-level control.

Example: An automated case packing machine for snack bags has 9 servo axes and 7 key functions.

- Case Feeding
    - Sheet Lifter servo
- Case Erecting
    - Case Opener servo
    - Case Transfer servo
- Bag Sorting
    - Sort 1 servo
    - Sort 2 servo
    - Sort 3 servo
- Bag Loading
    - Bag Pusher servo
- Case Loading
    - Case Elevator servo
- Case Closing
    - Case Closer servo
- Case Ejecting

One could set up 7 Equipment Modules in the machine. However, if that were done, only two would have more than one Control Module. Therefore it may be better to first divide the machine into larger functional groups such as 'Bag Handling' and 'Case Handling'. If this were done, the resulting configuration might look like that in Figure 3.

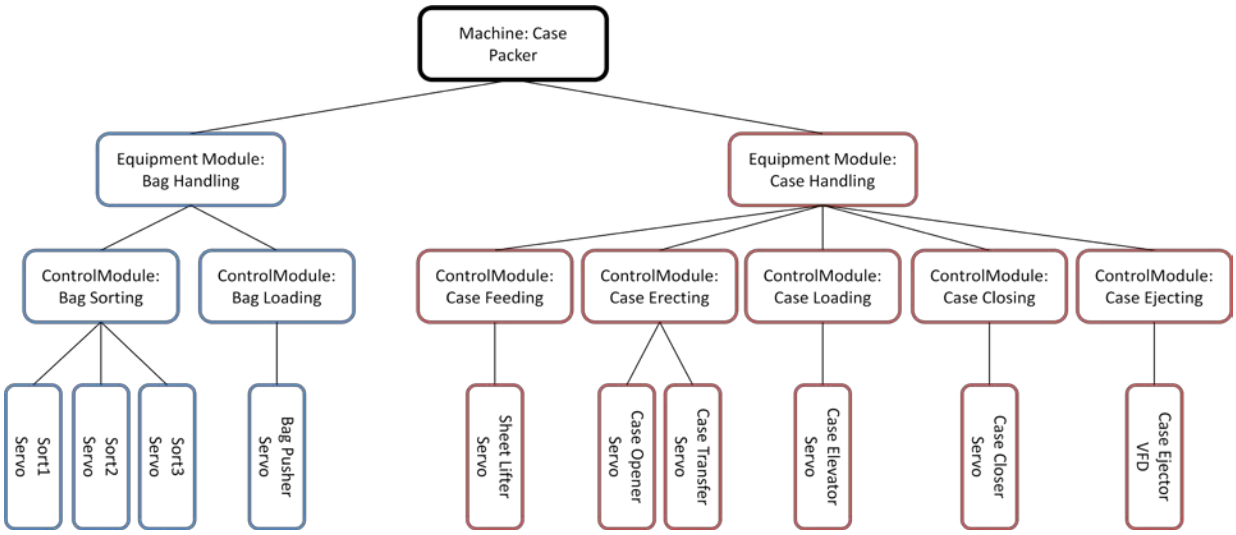

Figure 3: Module Architecture

This configuration splits the machine into two logical equipment groups which align with the two main inputs to the machine – bags and cases. All EMs contain more than one CM. Two of the CMs contain multiple devices that have high degrees of combined interaction and the number of EMs and CMs is reasonable for the project size.

Customize vendor-supplied templates

After making the three big design decisions just described it is time to write some code. This is an exciting, yet unnerving part of the process for one may wonder where to begin! Fortunately, several machine controller vendors have already created a starting point in the form of a PackML project template. These templates lay the foundation for the project by providing the PackML State Model code, a means to configure modes and states, a means for user definition of EMs and CMs, and example code for getting started. Depending on the vendor, these templates may be written in either pure ladder-based code or in the global standard IEC61131-3-based code.

Figure 4 shows a project tree from an IEC61131-3 based template that comes with two pre-defined Equipment Modules, each containing three Control Modules. Users can rename, add or subtract modules as necessary to fit the needs of the application. Key predefined code worksheets include PackML_Initialize, UN_PackML_StateControl, UN_Control_Inputs, UN_ModuleControl, EMxx_ModuleControl, and Ex_CMxx_Control_Outputs. (Yaskawa, 2012).

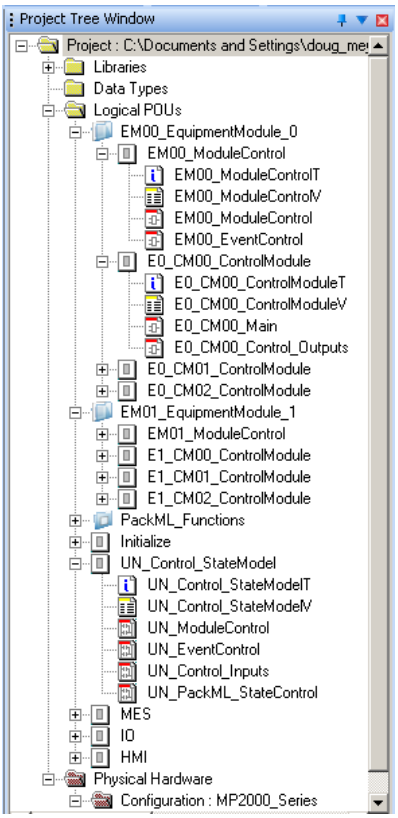

Figure 4: IEC61131-3 Project Tree for vendor-provided PackML template

IEC-based templates can take advantage of the multiple languages available in IEC61131-3 to create a more understandable and readable version of code. The function block for UN_PackML_StateControl is written in Sequential Function Chart (SFC), a direct graphical representation of the state-transition block diagram. Control output worksheets used for rolling up state completion status are written in Ladder Diagram (LD) since it is easiest to set and debug status coils in ladder-based code. Finally, configuration and command management functions are written in Structured Text (ST) since it is easiest to perform array manipulation and initialization using text commands.

In the template shown, the UN_PackML_StateControl worksheet contains the core code for the PackML State Model. The function block shown in Figure 5 accepts the high-level PackML Commands as inputs (the transitions), and sets the PackML States as outputs (the actual states). The block also monitors the completion status of each transitional state and moves the model to the next state. The function does not allow invalid transitions or invalid changes of mode.

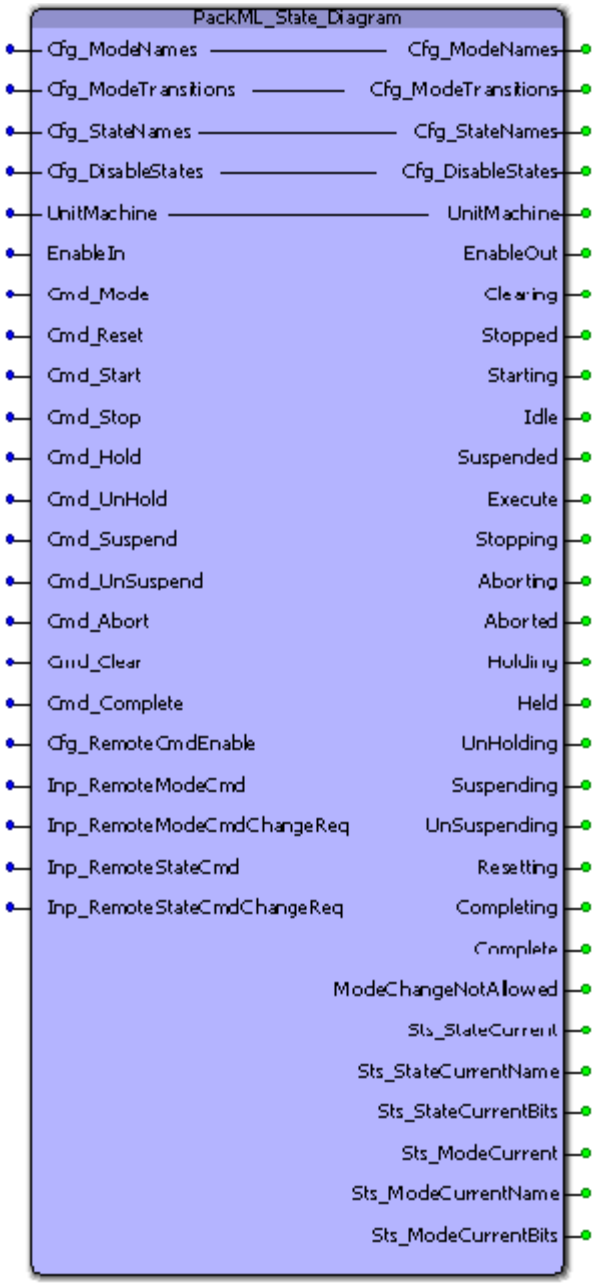

Figure 5: PackML State Model function block

Users configure the system by editing the PackML_Initialize worksheet. Mode names can be customized and defined starting around line 32, as in Figure 6.

```
32  (* Initialize elements being used*)
33  (* It is up to the User to determine how many valid modes are defined in the program and what
34  functionality each mode performs *)
35  PackML_ModeNames[1]             := 'Automatic';
36  PackML_ModeNames[2]             := 'Maintenance';
37  PackML_ModeNames[3]             := 'Manual';
```

Figure 6: Defining Modes in the PackML_Initialize worksheet

Likewise, users configure how many Equipment Modules and Control Modules are to be enabled starting around line 91, as in Figure 7.

```
91   (********** Initializing Equipment Modules and Control Modules to be used in the Unit Machine **********
92   (* Each UNxx defined can have up to 16 EMs, each with up to 16 CMs *)
93   (* Enabled EMs and CMs must be consecutive and will be numbered starting from 0 up to 15 *)
94   (* Example: If 3 Equipment Modules are enabled, they will be numbered 0, 1 and 2 *)
95   (* The enabled EMs and CMs can be individually deactivated in the UN_ModuleControl and EM_ModuleControl
96       worksheets *)
97
98   UN00_Modules.EnabledEMs             :=  INT#2;
99   UN00_Modules.EM[0].EnabledCMs       :=  INT#3;
100  UN00_Modules.EM[1].EnabledCMs       :=  INT#3;
```

Figure 7: Defining the enabled EMs and CMs in the PackML_Initialize worksheet

Finally, users are able to programmatically determine when states are considered complete by editing the code in Ex_CMxx_Control_Outputs. As displayed in Figure 8, each Control Module has an output worksheet that sets a coil for each state complete bit that is fed back up to the main UN_PackML_StateControl. Users need simply add contacts to the ladder if there are other machine conditions that impact the process for each particular state.

Figure 8: Ex_CMxx_Control_Outputs State Complete bits

Using PackML to Drive the Machine Operation

PackML is best used in a full implementation. By full implementation, it is meant that PackML is not just overlaid on top of an existing application for monitoring purposes only. Instead it is implemented from the foundation so that the organizational benefits of the modular approach can be fully realized. For this reason, inputs that bring the machine to an emergency stop condition should not just trigger actions independently, those inputs should instead trigger the PackML Command 'Abort', which will be automatically passed down to all active equipment modules. In the same manner, a Cycle Start input, if in the correct mode, should trigger the automatic sequence via the PackML Command 'Start'. All active equipment modules can then be programmed to respond either to the local PackML command 'Start' or the global PackML State 'Starting'.

Hooks for setting PackML commands

Most templates will have provision for users to patch into the PackML controls. For the template described, this is done in the UN_Control_Inputs worksheet. The rung shown in Figure 9 allows the command 'Start' to be issued if the user conditions are right and the operator has pressed a button on the HMI. The command 'Start' latches in until state 'Execute' is reached, or 'Abort' or 'Stop' is issued.

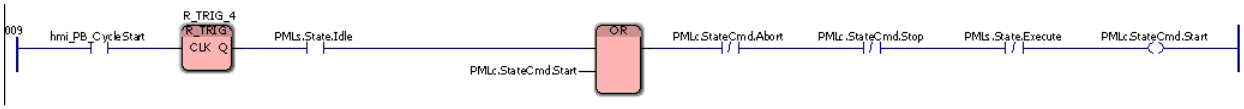

Figure 9: Start Command Logic

Extracting useful data for production statistics

PackML-based applications make it easy to extract production statistics since the PackML Mode and State provide most of the necessary information. If the mode and state are 'Automatic' and 'Execute' respectively, then it directly follows that the machine is producing its intended output. If the state is 'Aborted', then the machine is faulted in some way. Many users construct their own statistical gathering functions, but some templates already have certain functions built-in. Figure 10 shows an example found in the IEC61131-3 template shown earlier, where a vendor-provided function block called PackMLModeStateTimes continuously stores time information into the structured array of modes and states.

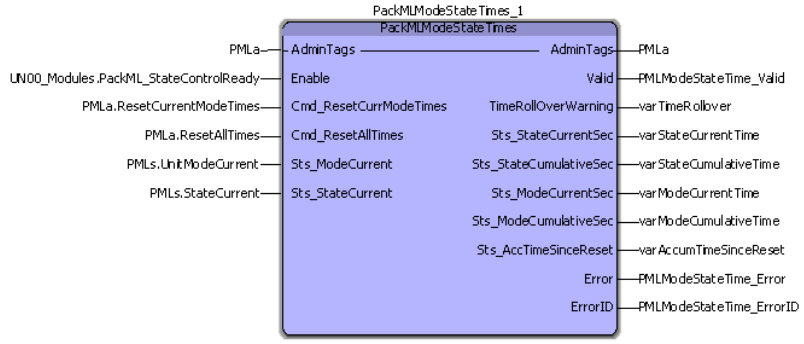

Figure 10: PackML Time collector for modes and states.

By comparing the current times in each state of Automatic Mode to the total elapsed time, the production manager can immediately view useful machine efficiency data.

Interfacing to upstream and downstream equipment

Although many machines can be considered 'stand-alone', a great many of them actually get installed into a production line among other machines. This is particularly true for the packaging industry, but often applies to other industries just the same. One of the promises of PackML is the ability to easily communicate status upstream and downstream to other equipment, even if that equipment was made

by a different vendor and uses a different machine controller. A forthcoming specification from the PackConnect subcommittee of OPW will outline such a communication protocol standard over an industrial network. Until then, the basic PackML foundation provides for easy interfacing through digital I/O triggered by the current PackML Mode and State.

Conclusion

Although the mission to establish a standard for machine control code architecture was at first entirely based on the needs of the packaging industry, the PackML ISA-TR88.00.02 and ISA88:Make2Pack standards contain methodology that extends well beyond packaging to ***all*** automated machinery. There is a learning curve for proper PackML implementation. However, getting your first PackML project up and running is made easier by focusing on three critical design decisions:

1) which modes are included and what they are named,
2) which machine functions happen in which state, and
3) how can Equipment Module and Control Modules be organized into a configuration that logically matches the physical machine.

Beginning a project with pre-built vendor templates also reduces development time. By following the model of the PackML Standard, your application can be more organized, easier to commission and debug, more modular and transportable to other applications, and easier to add functions for data collection and production statistics.

References

Campbell, Keith. (2012, April 3). OMAC Packaging – Is the Connect-And-Pack Message being heard?. Packaging World. Retrieved September 16, 2012 from http://www.packworld.com/omac-packaging-workgroup-connect-and-pack-message-being-heard

ISA. (2008). ISA-TR88.00.02 Machine and Unit States: An Implementation Example of ISA88. Retrieved September 16, 2012 from http://www.isa.org/Template.cfm?Section=Standards2&template=/Ecommerce/ProductDisplay.cfm&ProductID=9999

OMAC. (2009). P&G PackML Implementation Guide. Retrieved September 16, 2012 from http://www.omac.org/content/packml

Reynolds, Pat. (2011, April 9). Nestle makes packaging a strategic priority. Packaging World. Retrieved September 16, 2012 from http://www.packworld.com/controls/strategy/nestl%C3%A9-makes-packaging-strategic-priority

Yaskawa America, Inc. (2011). PackML Template. Retrieved September 16, 2012 from http://www.yaskawa.com/site/dmcontrol.nsf/SearchV/86256EC30069B634862579B20057C4EA?OpenDocument&Source=SearchResultPage