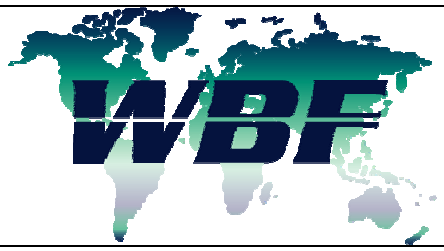
Presented at the
WBF
North American Conference
Atlanta, GA
March 5-8, 2006



195 Wekiva Springs Road, Suite 200
Longwood, FL 32779-2552
+1.407.774.5764
Fax: +1.407.774.6751
E-mail: info@wbf.org
www.wbf.org

Software modularity: a powerful tool for packaging automation

Tom Jensen
Senior Technology Evangelist
ELAU Inc.
165 E. Commerce Drive
Schaumburg, IL 60173
USA
T: +1 847 490 4270
F: +1 847 490 4206
tom.jensen@elau.com

Leif Juergensen
End User Business Development Manager
ELAU AG
Dillberg 12
97828 Marktheidenfeld
Germany
T: +49 93 91 606 0
F: +49 93 91 606 300
leif.juergensen@elau.de

KEY WORDS

Software Modularity, IEC 61131-3, Make2Pack

ABSTRACT

Modular software design benefits packaging automation for the same reason that WBF recommends modularity in process automation. When implemented with ISA-88 (IEC 61512) principles in the IEC 61131-3 international standard automation programming languages, modularity becomes a powerful tool for both machine builders and users.

The cost is upfront development of programming best practices, templates, and software object (IEC Function Block) libraries. The benefits include documented engineering time savings of 70 to 80% by machine builders, better engineering collaboration between OEMs and specifiers, and ongoing reduction in users' Total Cost of Ownership through reusability, consistency, maintainability, mechanical modularity and execution speed. In addition, validation savings of 40% have been documented in regulated environments.

This paper will compare examples of typical control functions written in monolithic and modular formats, as well as best practices for combining the IEC 61131-3 languages, ISA-88 (PackML) constructs, leveraging other emerging technologies, and managing the logical design, process design and deployment & architecture.

This paper will also define the business imperatives and benefits driving software modularity, including faster time-to-market, capturing operational efficiencies, streamlining of Management Execution System and Supply Chain Management implementations, maintenance outsourcing, and more.

Introduction

Modular software design benefits packaging automation for the same reason that WBF recommends modularity in process automation. When implemented with ISA-88 (IEC 61512) principles in the IEC 61131-3 international standard automation programming languages, modularity becomes a powerful tool for both machine builders and users.

The cost is upfront development of programming best practices, templates, and software object (IEC Function Block) libraries. The benefits include documented engineering time savings of 70 to 80% by machine builders, better engineering collaboration between OEMs and specifiers, and ongoing reduction in users' Total Cost of Ownership through reusability, consistency, maintainability, mechanical modularity and execution speed. In addition, validation savings of 40% have been documented in regulated environments.

This paper will compare examples of typical control functions written in monolithic and modular formats, as well as best practices for combining the IEC 61131-3 languages, ISA-88 (PackML) constructs, leveraging other emerging technologies, and managing the logical design, process design and deployment & architecture.

This paper will also define the business imperatives and benefits driving software modularity, including faster time-to-market, capturing operational efficiencies, streamlining of Management Execution System and Supply Chain Management implementations, maintenance outsourcing, and more.

The state of machine programming

Before we can tell the story about software modularity and efficiency, we must review a bit of the past…

For many years and for many reasons, machine programming has been implemented by starting on line one, and ending where it ended many lines later. This fact has several roots. The first reason is straight forward; the first generations of machines were very simple. The control structure consisted of a PLC that was replacing relay logic (if you were lucky). The drafter creating the schematics for the machine builder usually drew the wiring of electrical hardware as a ladder diagram. Ladder diagram was a clear and graphical way to show the relationships between simple physical objects (buttons, relays, lights) as they were found on the machine. It was a logical and user friendly offering for PLC providers to extend ladder programming to a technology wary public. By offering ladder language programming, the PLC providers of the day were changing the hardware, not the rules.

Secondly, as programmers became comfortable with the easy adoption of ladder programming, they didn't feel a need to move to another language. This didn't mean other languages weren't available, it means that ladder programming was more than adequate for Generation 1 machines.

As machines became more complex (especially after the introduction of variable speed motors, programmable limit switches, temperature controls and servos), machine programs grew very large to articulate the new found sophistication of machines. It was no longer enough to "Start a motor" or "Check a switch." Programmers had to find a way to "Home" motors, synchronize them, adjust travel distances, and have them recover lost position. The programs created with ladder programming began to look less and less like the machine being controlled. Instead of the single line of code that showed the interaction between physical hardware objects, (i.e., contact turns on relay), it took large parts of code to describe the actions of a single motor. The end result was machines that were highly complex, with limited flexibility and difficult to modify, integrate, troubleshoot and own. Today, packaging machines on average contain multiple motors, temperature controls, PLS, load cells, and weigh scales. They are also responsible for machine performance data, product data collection, preventive maintenance features and more. They also have to conform to more stringent government regulations.

The language problem

Ladder programming proved itself to be useful for describing massive amounts of Booleans, but more difficult to use for describing complex machines. It was commonly realized that other languages may be helpful in describing these more complex generation 2 machines. There were many languages available from the industry, but they all suffered from one of two conditions. The languages commonly offered on the market were either text based or graphical. Text based languages in general were capable of describing complex objects with much less code, but were not intuitive. This meant a programmer could actually create machine code that worked well and described how that machine should operate, but nobody without programming education would be able to work on the machine. Secondly there were graphical languages. They were intuitive but lacked the depth to completely fulfill the desire to have the "Code replication of the real world."

The IEC 61131-3 proposition

In the mid 1990's, a standard developed in Europe to address the "single language" approach many technology providers had adopted. A suite of 5 languages (really, 4 languages and a structure) had been adopted as a worldwide standard for programming PLC's: IEC 61131-3. The standard provided for the nesting of one language inside another, structures, complex data types, libraries, templates and more. It provides a way for programmers to develop code the way a programmer does, but then to display the code to a machine technician in a graphical way. The 1131 standard promises the ability to modularize machine software, and to deliver truly reusable code that will:

- Increase machine performance
- Reduce delivery time to market
- Improve serviceability
- Speed validation

- Reduce the Total Cost of Ownership

Sounds good, how do I begin?

Generation 3 machines are far different than Gen 1 machines. A Gen 1 machine could be programmed as a monolith -- one big block of code -- because there was not much to them in terms of electrical controls. For example, a Gen 1 tube filler may have had 1 main motor driving a Geneva gearbox, with all motion mechanically cammed to it. On the other hand, Generation 3 machines are electrically complex and should be broken down into smallest possible pieces (granularity). The smaller the pieces, the more reusable the code will be A Gen 3 tube filler consists of servo controlled pumps, tracks, nozzles and sealing bars. Also, there are temperature controls, metal detection and HMI's to consider.

Now that we have drawn a distinction between how things have worked in the past and where we would like to go, we need a plan to get us there. Before pressing a key on your PC, the first consideration should be modeling your machine with granularity in mind. If we are programming a tube filler, we may have created a machine model that consists of the following mechanisms:

- Pump and Manifold
- Tube loading, tube track and orientation
- Filling nozzles and seals
- Gates between the filler and the timing belt to the cartoner
- HMI and data collection
- Temperature control for the end seals

All of these mechanisms are made from:

- Servos
- Variable frequency drives
- Touch screens
- I/O
- Heating elements

We have enough of a map for our machine that we can begin programming. We should begin by making the smallest components first, keeping in mind that in some way operators and maintenance people are going to need access to them for setting up and operating the machine.

Let's start with making a piece of modular code for a servo motor. Once we model the servo in a Function Block (a software object as defined by IEC 61131-3) and store it in a library, we can drop it into any other code module where a servo is needed. At this point we need to decide which language would be the best, and the obvious choice would be the 1131 language "Structured Text" because of the mathematical demands as well as the complicated sequencing needed. Any of the 5 languages will do, but the goal is to reduce code. We will define a number of input and output variables to the Function Block we are writing, and set out. All possible needs (operating modes, inputs, outputs, etc.) should be provided for at this point, no matter how "rarely" the feature may be used. Why? Because if we spend a small amount of time at the front end to take care of all needs, it will preclude us from revisiting and rewriting later (if you make a good list, you will only have to go to the grocery store once!).

Once completed, the Servo Function Block should be tested, documented and locked. It is your property. Whether I have 1, 2 or 100 servos in this program, this is the only FB I need. We follow the same process for the other simple components listed with the servo, and create a locked library as is provided by IEC 61131-3. Even though we may have used different languages for the Function Blocks we've just made, we can call them into the next layer in whatever language we want. This means the servo object we made in Structured Text can be displayed as ladder, creating a graphical representation of the machine that most technicians will be comfortable with.

Modules for machine variants

Now that our simple objects have been saved as a library, we can start programming the mechanisms that comprise our machines. Again, we can pick a language that suits us. But there is one problem; the machine I'm programming is one in a family of machines. The smallest operates on 2 tubes with one color, the largest 8 tubes and 2 colors. The process we used for the simple objects still holds true. We should write the code to accommodate the largest, most difficult machine (you'll have to do it anyway), and make it able to be configured by parameters to do the smallest machine as well. When we do this, we will be creating a library of a few objects that you can place in any tube filler your company makes, and commission the machine by setting parameters.

We have hit a milestone, by adopting a process for developing code (ISA-88, M2P) we have just leveraged a practice that will improve our business. From this point…

- The cost of engineering code goes down
- The cost of developing new machines goes down
- The integration of new features becomes simpler
- The stability of our products becomes better
- Delivery time drops
- Documentation and validation decreases

Programming the top level

Now that we have made simple components and mechanical modules, we need to tie them together in a frame work. The consideration at this point should be to simplify line integration, connectivity to upper

lever systems (MES), and promote service and diagnostics. And of course, the machine should be simple to operate. This is where the unique code that make this machine different from al others will be written. This should make sense because all of the layers below were made of objects we've written for the machines our company makes (tube fillers, cartoners, case packers, etc). But how unique are most machines? They all have the same basic composition. They need…

Error Checking, Inputs, Outputs, Mode Control, Messaging, and tags for integration. If we employ the language best suited to describe a structure, to act as a filing cabinet, we can greatly simplify even the code at this level. We may not be able to put it into a library, but surely make it into a template. Then, any service person, operator or IT engineer will be able to get the information they need. Therefore, we should use Sequential Function Chart (Graphset) in this case, and again use a top layer standard (PackML) to standardize the tags needed for all these features. By using SFC and PackML at the top layer, and standard libraries for the components below, we have created a machine that has the following benefits…

- Reduced installation costs
- Reduced line integration costs
- Better OEE and root cause analysis
- Simpler operation and troubleshooting

Software modularity put into practice

The process of changing monolithic code to modular code has been gaining momentum over the last few years. A number of companies have gone far enough with the process to share results…

Douglas Machine

At the Packaging Machinery Manufacturers' Institute (PMMI) Conference at PACK EXPO 2005, Joe Faust, Electrical Engineering Manager for Douglas Machine, shared his company's experience with the underlying software architecture, the initial cost to build a foundation of modular software using the IEC 61131-3 languages, and the resulting benefits.

That experience was eye opening, revealing the competitive advantage to Douglas as an early adopter. The methodology is simple. Machine builders can reduce engineering costs and increase consistent quality through the use of libraries of tested, reusable software modules that 'plug' into a state model as prescribed jointly by the OMAC Packaging and Make2Pack groups.

Combined program development, program testing and machine testing with modular software took 28 to 34 days, compared to 50 to 65 days for a conventionally programmed or 'monolithic' program.

Programming time was reduced at least by 50% (from 20 to 25 days down to 10), and program testing was reduced by approximately 80% (from 15 to 20 days down to just 3 to 4). The savings potential was clearly derived from better software engineering, as the machine testing component remained unaffected at 15-20 days.

Faust cited shorter development time, standardized program structure, the ability to increase machine features, ease of machine upgrades, expanding the breadth of machine offerings, and protection of intellectual property as benefits to machine builders as well as users.

Faust duly noted the upfront costs as well. First, the modular foundation must be developed, and for the first modular machine that took 100 days of software development and 30 days of software testing in addition to normal machine testing.

Kappa Herzberger

Kappa Herzberger, recently merged with Smurfit-Stone, specializes in high speed packaging machinery for the beverage and food industries. The company recently applied IEC 61131-3 conforming modular programming techniques in a complete redesign of its product line, according to Michael Heise, Manager for Automation Technology.

Through the use of these modular software building blocks, Kappa has consequently reduced engineering time by 50%, while the new machines realized 75% higher cycle rates, 35% less commissioning time, and 60% faster format changeovers.

HAMBA Filtec

HAMBA Filtec has applied no fewer than 19 servo-automated functions for flexibility in plastic bottle, cup and tub filling and sealing machinery. The container station uses servos for container gripping, automatic de-nesting, tub transport, and scissors-operation. The metering unit uses servos for lifting, piston and valve rod actuation. The lidding station uses servos for lid gripping, application and drum movement. At the sealing station, the sealing bridge, lift mechanism, support and closure systems all are servo-operated, as are leak testing, container ejection and chain cycling functions

Benefits of the modular mechatronic design include not only rapid pushbutton format and product changeovers, but convenient operation and maintenance. For example, with two lidding station magazines accessible from outside the machine, one can be reloaded while the other remains in operation. The upper section of the sealing station has been designed to pull out and to the side for easy cleaning, setting and maintenance access. The systematic implementation of a new generation of diagnostics makes fault detection and troubleshooting easier to than ever for operators.

A high end automation solution is essential to control such a flexible machine, which requires up to 34 servo motors, according to Uwe Gerasch, Development Manager for HAMBA. To do so, the company implemented a completely modular control software structure. The control system integrates motion and logic control in the same IEC-conforming program. .

The control system also operates up to 1,000 distributed digital inputs and 1,000 digital outputs, plus 50 to 170 analog I/O points per machine. Instead of discrete hardware controllers, software PID loops control heating circuits and sterilization, while automated software cam adjustment controls numerous cam positioners throughout the machine. In addition to interactive diagnostic guidance from the touch panel, the operator also receives graphical assistance, fault displays and spare parts management.

Hassia Redatron

It's generally a contradiction in terms to equip packaging machines so that they combine maximum output with maximum flexibility, notes Andreas Hollmann, Form/fill/seal Machine Sales Manager for Hassia Redatron. These requirements led the company, a member of the IWKA Group, to the development of a modular approach to designing control software.

Specific objectives included the ability to:

- Develop applications from standardized logic and motion control software modules, assuring consistent, accessible, supportable and reusable software
- Create a consistent control concept across machinery, based on the existing HMI
- Use standardized programming languages and templates that allow programming assignments to be outsourced

Diagnostics can be performed directly from the controller, with all system data and faults displayed on the machine's HMI. The controller also supports remote diagnostics by factory trained personnel.

Hollman says that their experience to date working from an existing, pre-tested modular software platform has shown that it is possible to shorten the time required for program startup by 50%.

Conclusions

Packaging machinery builders implementing the modular principles of ISA-88 Part 5 (IEC 61512) through best practices in the IEC 61131-3 languages have proven that engineering time can be reduced to develop sophisticated, flexible, high performance capabilities with the documentation and diagnostics needed to achieve maintainability.

ARC Advisory Group has validated the benefits of the emerging trend of software modularity within a standards-based, Make2Pack environment, as described in this paper, in no fewer than 5 analyst reports in the past year. In summary:

- **Software Management Strategies for Packaging Machine Builders, March, 2005** – "Relay ladder logic no longer has the qualifications to serve the needs of the machine builder today, as software that leverages modularity and encapsulation is simply not supportable in this environment. It is the IEC 61131-3 programming languages that are the most viable alternative for OEMs…."
- **Will Make2Pack Precipitate a Manufacturing Performance Revolution? June 9 ,2005** – "There is also resistance to change among packaging machinery OEMs….However, some OEMs have realized design and post-installation services cost reductions using this modular approach….Make2Pack is developing a limited set of common control and equipment modules to enable interoperability and reduce the Total Cost of Ownership (TCO) in a multi-vendor environment typical in packaging operations."

- **Make2Pack Helps OEMs and Integrators Respond to Changing User Requirements, August 25, 2005** – "Packaging OEMs and systems integrators in the forefront of the Make2Pack effort have an opportunity to improve their financial performance by more effectively responding to changing user requirements, while those who lag too far behind the technology adoption curve increase risk to both their business and financial performance."
- **Integral Robotics Raises Agility and Flexibility of Packaging Machinery, September 2005** – "Integration of a robotic manipulator further leverages the concept of modularity by encapsulating this element of the machine as a functional subcomponent that is using a common time base of the overall machine control system….The IEC 61131 industrial programming standard, which has become the norm for Generation 3 machines, allows automation suppliers to extend the language for robotics application. Automation suppliers that support a high degree of software modularity are encapsulating robotic functionality as a published library of standardsized Function Blocks."
- **Packaging Machinery Strategies for End Users and Machine Builders, November 2005** – "OEM machine builders must adopt the principles of modular software design based on international standards and guidelines such as the ANSI/ISA-88 batch standard and Make2Pack initiative (S88.05), the OMAC packaging guidelines, PackML state model, IEC 61131-3, PLCopen Motion Control Library, and the evolving PackAL library definitions….The modular approach enables software and documentation reusability as well as making it easier to understand the functions….benefits…include reduced development cost, reduced delivery time, reduced validation and start-up time and cost, and improved post installation service capability and margin."