Maximize the Success of Your Control System Implementation with Standards-Based Object-Oriented Design

The successful implementation of control systems for continuous or batch processes starts with good design. While there are many possible ways to approach the design effort, the best result should blend appropriate quantities of tried-and-true methods with value-adding emerging technologies. While object-oriented design may no longer be considered an emerging technology, it is nonetheless treated as a newcomer in the process control field. The purpose of this paper is to describe the many benefits of utilizing object-oriented design principals in the design of a modern process control solution.

Object-Oriented Design Helps Realize Project Goals

In simple terms, object-orientation is simply the technique of bundling code into individual software objects. Numerous benefits result from this methodology; and in turn, those benefits streamline the effort of creating a well-targeted, feature-rich and rock-solid control system solution. These benefits are prevalent in so many successful software implementations, that their legitimacy is well established. Some of the primary benefits are:

Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object is easily duplicated throughout the system, enhancing consistency and simplifying understanding.

Abstraction: By interacting only with an object's public interface (i.e. methods and attributes), the details of its inner workings remain hidden from the outside world. After all, once an object is coded and thoroughly tested, what does it matter how it works? It just does.

Re-use: If an object already exists which performs a needed function (perhaps created by someone else), you can use it in your system. This idea takes advantage of the availability of specialists to implement/test/debug complex, task-specific objects, which can then be reliably deployed in any project as needed. If a library of these objects is developed and maintained, chances are good that most of the functionality required for a new control system could simply be constructed from existing building blocks. How much effort could that eliminate?

Pluggability: If a particular object turns out to be problematic, you can simply remove it from your system and plug in a different object as its replacement. So long as the interfaces are consistent, the replacement should work without modification to the rest of the system. This is analogous to fixing mechanical problems in the real world. If a part breaks, you replace *the part*, not the entire machine. Furthermore, you would expect that replacement of a pluggable component doesn't require the disassembly of the entire machine.

Tools for designing and implementing object-oriented software are ubiquitous. In fact, numerous languages rely on object-orientation at their core. But what about process control platforms? Until recently, implementing an object-oriented design was tedious and difficult – with very few tools to assist with the effort. Thankfully, that's all changing. Hardware vendors are now offering highly functional software development tools which are designed with object-orientation in mind. I'm not sure if suppliers or users are to be given credit for causing this recent advancement, but it doesn't really matter. Let's just be glad that this wonderful technology, which has been available to mainstream programmers for so long, is finally available to the rest of us!

Successful Results Start With Worthy Goals

Object-orientation by itself is a powerful tool, but when combined with time-tested project implementation methods and solid goals, successful results can and should be expected. A logical starting place is with goal-setting.

Build the right thing the first time. Many endeavors only produce satisfactory results after combining the efforts of several partial attempts. This results from reasoning that human and financial resource limitations require that a project's scope be limited only to the need at hand. While this may make sense in the short term, it often consumes more money and time overall because of rework required to later assimilate the cumulative results from multiple iterations. By contrast, when a solution's architecture takes future needs into account, a proper foundation can be created which accommodates those needs without costly rework. Thus, a little extra work at the outset can pay sizeable dividends down the road - and getting it right the first time is a worthy goal of any project.

Worthy project goals include:

- Build the right thing the first time (architect a complete solution)
- Achieve 'best practices' result by following established standards
- Employ an agile methodology to speed development, hasten iterations
- Maximize simplicity and maintainability
- Minimize engineering effort through modularity and re-use
- Optimize the 'whole' of the system rather

Aim for a 'best practices' solution. Once the 'what' is determined and the right architecture is chosen, focus should be shifted to the 'how' aspects of design. When searching for examples of best practices, a good place to start is within well-established standards. For control system design, there are several applicable standards which embody the collective knowledge of hundreds, if not thousands, of highly skilled engineers.

Of the pertinent standards, the most applicable are those specifically targeted toward control system design, namely ISA-88 (US) and IEC-61499 (Europe). Both of these standards are intended to be guidelines for applying object-oriented principals to process control, and provide some limited specific details for implementation of the resulting design. When properly applied, these guidelines help to produce an appropriately modular and flexible organization of process equipment and functionality. The resulting system model then becomes a roadmap for the detailed design and execution stages of the project.

Of somewhat lesser relevance, but still very important, are standards related to safety functionality and business integration. Safety standards, such as ISA-84 (US) and IEC 61508 (Europe), focus on the proper performance and management of the safety system life cycle. Business integration standards, such as ISA-95 (US) and IEC 62264 (Europe), focus on the vertical integration of process control with a plant's operating and business systems. In large projects, one or both of these extensions to the typical control system project scope may be required elements.

Strive for an 'Agile' methodology. Customers know that the best solutions are executed quickly and readily adapt to changing needs and requirements. Cumbersome development methods, driven by the desire for complete code documentation and rigid change management procedures, produce well-managed software projects – but often at the customer's expense, by limiting adaptability for incorporating new or changing needs. While conventional methods are useful to programmers, they typically limit customer participation to only the beginning and end of the project life cycle. A better approach would include customer input throughout the development cycle, by incrementally reviewing and approving frequently delivered, working portions of software. The frequent feedback from the customer aids in refinement of requirement details and helps the developer understand what functionality provides the best solution for the customer.

The term agility implies a lightweight process that can change direction quickly. This weight analogy has direct application to staffing of the development team. The typical project team of highly specialized individuals is often large in size and slow to understand details that fall outside individual specialties. A better approach to staffing would incorporate fewer, cross-trained individuals with broad experience in the many facets of process control design and implementation. This small and highly capable core of people is more capable of understanding the entire scope of project details, and by virtue of small size can quickly adapt to fluid requirements. This 'skunk-works' approach to project development has far more chance of pleasing the customer than the mainstream alternative – and will do so in a more timely and cost-efficient manner.

Maximize simplicity and maintainability. Few will argue the virtues of the KISS principle, and with good reason. Simplicity of design is often at the heart of a long-lasting success story. Simplicity reduces the 'clutter' and focuses on the most important aspects of a product or solution, making sure these most-satisfying attributes are solid and on target. Proper simplification of design minimizes the quantity and/or variation of details without sacrificing functionality. By its nature, this reduction of unique extraneous details enhances maintainability. After all, the easiest system to maintain is the one that is simple to understand.

Minimize engineering effort through re-use. The concept is simple. When components of a system are designed to be re-usable, the fruits of expensive engineering efforts are maximized. Furthermore, when a component will be used over and over, committing resources for adequate testing and documentation is rarely an issue. The result is a solid, well documented product that can be quickly re-deployed elsewhere with low effort and risk. As a primary tenet of object-oriented design, re-use is a solid expectation of any properly executed project which utilizes its methods.

Optimize the whole rather than the parts. It seems logical that by fully optimizing each part of a system, the system is in turn optimized. This can be true to some extent, but individual optimization ignores the needs of the system as a whole and by definition puts a lower value on overall system functionality. A better approach is to first optimize the system, and then optimize the resulting components – but only to the extent that the effort doesn't detract from any prior system optimization. This explains why use of generic objects produces a better overall solution, even when the generic object may contain common denominator functionality that isn't required by each unique function to which it's applied.

Provide support for frequent software verification. If a system commissioning is part of your recent memory, you don't need to be told about the need to sometimes simulate code functionality without the aid of real process conditions. This is especially true when testing abnormal situations, since creating the abnormal condition could be detrimental to equipment or personnel safety. Simulation functionality, if designed into the code, permits testing the expected response to a condition without having to create the actual condition.

Adding temporary code for the purpose of testing is also an option, since the temporary part can be removed after commissioning. But what if testing is required more often? Safety standards dictate that any change made to a system which could possibly affect safety functionality necessitates re-testing of all safety functions. Object-oriented components simplify this requirement, since adding simulation capabilities to each object in a class library is an easily accomplished task; and since that functionality is hidden behind the object interface, it doesn't add to the visible code that a technician or engineer must search through while troubleshooting.

Support safety integration. In addition to re-testing requirements, safety standards impose the need for reliable mechanisms for manipulating process or equipment actions when adverse conditions are detected. Reliability is key, and a simple, consistent and thoroughly tested means of intiating pre-determined actions is critical to the integration of safety systems into the process control system. As before, objects easily facilitate this capability when designed with a configurable override response feature and a simple, consistent and reliable mechanism for triggering it.

Integrate project documentation. Everyone knows how much information is collected during the execution of a process control project. A mound of documentation containing details about equipment and software functionality is collected or generated during design and is used extensively during commissioning, but what happens to it afterward? It is probably shoved into a cabinet and forgotten until some problem arises. When it's needed again, it's likely that important details have faded from memory and the once fresh documentation can be less than the efficient reference it was – assuming that it's even still accurate. If design documentation is integrated into the system in the form of an online reference, chances are it will be maintained better and will be accessed with greater ease.

When systems are built from objects, documentation becomes more granular and organized. Creation of a thorough description of object functionality is not only easy, but requires less total effort because it's only created once – when the object class is first created. Likewise, it requires little effort to integrate that documentation into online help, which can be included as part of the HMI template used for each instance of the object. The end result is documentation that doesn't have a separate identity from the system itself. It's part of the solution, and because it's easily accessed it is frequently used and understood.

One further enhancement relating to documentation is to extend the concept of object-oriented design beyond the typical realm of software development. If the organizational benefits of this methodology are so useful to the controls engineer, why wouldn't they be useful to other disciplines as well? Process control projects are typically a subset of a larger scale plant design project. When this is true, the amount of information gathered and generated can go up by an order of magnitude. How is all that information organized and managed? Defining object classes for physical components (e.g. a pump) with their associated methods and attributes (e.g. mounting, piping interface, pump curves, etc.) is a means to organize information and descriptive documents. Once an organization strategy is established, it's not much of a step further to implement a storage/retrieval mechanism – like a relational database - to manage the mountain of information.

Managing project design information using this type of structure opens up the possibility of design re-use between projects. Just think how much time could be saved if physical design elements could be easily shared between multiple design efforts. The key enabler of this time saving capability is the consistent organization of design information according to object-oriented principles, coupled with an appropriately designed relational database schema. That schema could then be applied to a SQL database of the user's choosing, followed by a simple user interface to assist with the storage and retrieval of project information. How useful would that be?

Standards-Based Implementation of Object-Oriented Design Maximizes Success and Sustainability

The successful implementation of a control system may start with good design, but it's the details of the execution phase that will ultimately determine the timeless viability of a solution based on that design. Hopefully, the previous discussion has made the case for object-orientation as the basis for quality design. While there are many possible ways to realize an object-oriented design, the most sustainable solution relies on a thoroughly detailed, standardized approach.

Standardization, when done well, reduces variability of solutions and maximizes the use of best practices. As mentioned above, two dominant standards exist for the design of process control solutions – ISA-88 (US) and IEC 61499 (Europe). These internationally-accepted standards are the culmination of many years of effort and the collective input of a horde of experienced engineers. As a result, the guidelines promoted in these standards represent the best knowledge of process control experts relating to nearly all aspects of design and implementation.

Good Organization and Proper Orchestration are Key to Sustainability

The standards mentioned above differ somewhat in their implementation details, but nonetheless have many core similarities and share a common purpose of promoting the use of object oriented techniques in process control systems. The common elements of these standard methodologies form a solid foundation on which to build an efficient, adaptable, long-lasting control solution. Some desirable core attributes of these solutions are itemized in the inset. These attributes result from careful planning and the use of well-formed, holistic design strategies like those described below.

Preferred system attributes include:

- Clear separation of procedural and equipment-related code
- State-driven execution
- Hierarchy of code objects
- Flexible inter-object communications
- Proxy mechanism for communicating with external components
- Well-developed mode control with propagation
- Well-managed alarm annunciation

Keep Procedural Logic Separate. Much of the code running in legacy systems these days freely mingles direct control (the *how*) with sequential or procedural logic (the *when* and *why*). This can seem like a more efficient style of coding, since all of the relative aspects of a particular function are kept together in the same place. However, a nasty side effect of this approach is a close inter-dependency of all the code (aka spaghetti code). As a result, it's difficult to modify any part of the code without somehow affecting the rest of it. Moreover, this often happens in a less-than-obvious way; resulting in buggy code segments that act like hidden time-bombs waiting for the right condition to blow up - perhaps weeks or months later, and after programming resources have been assigned elsewhere.

A better approach is to separate the part of the code that typically doesn't change from that which might. Code which mimics and supports the attributes and actions of real devices (i.e. direct control) can be debugged during commissioning and then shouldn't need touched again until the associated device is changed. The same cannot be said of the 'automatic' portion of control. This form of code dictates when and under what process conditions the device needs to operate. As this aspect of control can often experience flux, it makes sense to separate it from the more rigid, unchanging direct control. The result of this strategy is bullet-proof direct control that is indirectly manipulated by separate automatic logic that is more concise, easier to understand, and is simpler to modify and re-validate.

Use state-driven execution. In object-oriented terms, *methods* are the means by which objects carry out their prescribed actions. The various methods (i.e. actions and algorithms) pertaining to process and equipment objects are best coded using state-based, event-driven logic. This technique simplifies the initiating, executing, concluding, and exception-handling aspects of method accomplishment, and as a by-product creates a convenient, highly organized framework for differentiating how code should respond to varying circumstances.

Organize objects into a hierarchy. Complex control systems are rarely flat. Direct control is generally at the bottom, managed by group supervisory code. That supervisory code is in turn supervised by more sophisticated control or sequential logic that is in turn manipulated by even more sophisticated procedural code, and on it goes. Diagrammatically, the hierarchy mimics the roots of a tree, with the various root hairs (field I/O) converging over and over again to form devices, groups, process functions, operations, procedures, etc., until everything combines into a common trunk (the operator). This natural hierarchy is dictated by the process and/or operating requirements and is generally not difficult for a process control engineer to uncover. Once known, it makes sense to structure the code objects in a similar hierarchy, with each object capable of performing its desired function while collaborating with other objects to carry out group actions.

ISA-88, being initially focused on batch processes, provides a thorough set of guidelines for properly sorting out the desired process/equipment hierarchy. The standard also defines different object types used to model the hierarchy, namely procedural and control. Procedural objects deal with the execution of pre-defined, recipe-driven operations that produce a product with the highest quality or in the greatest quantity. Ordering and sequencing of process steps are the principal focus of procedural objects.

By contrast, control objects provide coordinating and basic control functionality, but sometimes include equipment-specific procedural logic as well. *Equipment Modules* (EM) and *Control Modules* (CM), which are the common variants of the control object, translate process-oriented functions into equipment actions, and form the all-important interface between the control system and the physical plant. An appropriate collection of these workhorse objects can fully realize all of the operator-directed functionality of a process which does not require automation of its operating procedures. For this reason, a lot of care is taken to describe the details of their creation.

The corresponding workhorse elements of IEC 61499 are called basic and composite *function blocks*, but their purpose and intent generally matches that of EMs and CMs. One distinction of IEC 61499 is in how it further breaks down the function block into two separate parts, namely *algorithms* and *equipment control charts* (ECC). Algorithms manipulate process data and ECCs process event information. While EMs and CMs contain the same inherent functionality, ISA-88 does not specify that a clear separation exist between the two halves.

Utilize flexible mechanisms for communications between objects. A hierarchical control structure only works when objects can communicate effectively with each other. The communication mechanisms supporting the supervision of one object by another need to be simple and reliable, but also need to be flexible enough to permit more complex cases - where the conditional selection of which object is in charge at any given moment may be required (commonly needed for batch recipe execution). In this way, the hierarchy can be modified as needed at runtime, adapting to the varying needs of the process or the operating personnel. Reliable, but temporary connections can be established between objects and then replaced with equally reliable connections with other objects when necessary. This is analogous to joining objects with Velcro rather than welding the pieces together.

Another aspect of inter-object communication is that of entity location and identification. In truly distributed systems, it is often a requirement that the communications between objects include information about the ID (e.g. DNS name) and sometimes location (e.g. network segment) of the message originator or recipient (or both). This is especially true of IEC 61499, where a stated goal is to achieve distributed control that permits the flexible assignment of execution responsibility for the various logic components across a widely distributed physical architecture. This is similar to the scheme employed by applications constructed from software-as-a-service (SaaS) components. This service-oriented architecture (SOA) permits any component of the system to reside anywhere in the network space without impeding the operation of the system as a whole. The execution flexibility gained by this approach is valuable for obvious reasons, but requires a sophisticated and well-structured messaging scheme that can be too much effort for many projects to justify. Therefore, since the degree of required flexibility – and resulting message complexity – varies for each project, choosing an appropriate messaging scheme to meet project goals should be left to the discretion of the solution designer.

Establish a proxy for external interfaces. When communication must cross boundaries of systems which are created or managed by different resources, it is a good idea to have a proxy object to facilitate data mapping and execution control (blocking or re-directing messages). These features can greatly enhance the flexibility of development and start-up efforts, not to mention ad-hoc troubleshooting of problems that arise later. Quite often, the interface is provided by a middleware product which already possesses these desired features; but when a sufficiently functional middleware solution doesn't already exist, a proxy object should be created to provide adequate user control of the communication link.

Establish well-defined and well-managed modes. Modes are an important aspect of control system design, and are a key mechanism used in the management of process constraints. Without modes, process constraints must be defined in a one-size-fits-all manner, which is obviously not the ideal case.

With modes, however, situational awareness can be integrated into the operation of the process – invoking certain constraints when they are relevant and ignoring them when they are not.

To maximize the adaptability aspect of using modes, it is important to have a clear understanding of inherent process challenges together with well-defined operating requirements. Once these important details are known, appropriate modes can be defined that simplify the controls design process and maximize operating flexibility – without sacrificing personnel safety or equipment protection. Once these modes are incorporated into the control scheme, an operator can knowingly choose the mode which best matches their functional needs and current process conditions, and be confident that the equipment will behave in a safe and reliable manner.

It is also helpful when different categories of modes are considered. Unit control modes, for example, apply to the entire production unit and can distinguish between normal production and maintenance, or between stages of production (like setting-up, producing product, and clearing the machine after a failed operation). Similarly, equipment control modes apply to individual devices or subsystems and can dictate one control strategy under normal conditions and another for exceptions to the norm (e.g. maintenance, override, etc.). In yet another vein, supervisory modes can be used to dictate which resources (operator vs. program) have permissibility of control at any given moment.

After modes have been defined and properly integrated into the control scheme, it is important that a mode management strategy also be employed. Certain modes are the result of others, or may overlap or interfere with them. For this reason, a mode selection scheme is needed to both prevent undesired interference and facilitate the desired propagation of a mode through the system. The result is prevention of unnecessary or unwanted mode combinations, and added operating convenience.

Design HMIs according to ASM guidelines. The benefits of a graphical HMI are numerous. Of paramount importance is the ability to effectively convey status information to the operator. In doing so, however, it is important to consider the aspect of annunciating abnormal conditions (e.g. warnings and faults). Timely reaction to these conditions can be critical to personnel safety or equipment health, and great care must be given to designing HMI screens which maximize an operator's ability to recognize abnormal situations. In recent years, research has been published by the Abnormal Situation Management Consortium (ASM) that describes guidelines to best address this important aspect of design.

The consortium was initially formed by Honeywell in the mid '90s to address a growing concern over the increasing frequency and cost of incidents like unplanned shutdowns, environmental excursions, fires and explosions. The consortium's mission was to investigate causes and identify problems that occur during abnormal process situations in hopes of finding ways to mitigate their effects and improve early detection and response. When research showed that human factors had the largest correlation to incident occurrence, R&D efforts were directed toward understanding the human response aspects of graphically presented data. As a result of this research, guidelines have been published which summarize learnings and offer conceptual solutions for maximizing the effectiveness of operator displays, procedural practices, and alarm management.

In the simplest terms, the ASM guidelines promote the notion that less is more. Significant improvement in cognitive response has a direct correlation to the reduction of unnecessary information - particularly graphical content. Subsequently, when graphics are simplified, both in color and form, abnormal conditions are easier to recognize and understand. Color is used sparingly; and when present, it has a clearly-defined meaning – namely, to indicate whether a condition is normal or aberrant. Content organization and screen navigation are also important. Overview displays should be prevalent and present a complete picture of a process area's status without having to click through multiple views. Overview content should by mainly qualitative in nature, and include only essential quantitative information. Details should be moved to highly-focused views which can be retrieved with a minimum of keystrokes.

Manage alarms to prevent information overload. An important feature of an HMI is to annunciate alarms. Most modern control systems provide a plethora of monitored alarm conditions, with the intention of providing as much diagnostic information as possible about process and equipment behavior. Unfortunately, this feature can present a large quantity of information so quickly that an operator can be easily overwhelmed and lose the ability to distinguish important events from trivial or irrelevant ones. Rapid alarm bursts of numerous conditions and frequent repetition of the same condition often cause an operator to ignore the alarms entirely – far from the intended purpose of this potentially valuable feature.

A better approach to alarming includes the active management and filtering of abnormal conditions. Integrated alarm management uses known relationships between events to suppress alarms which are the direct result of another. This eliminates meaningless alarm bursts, and tends to focus attention at the root cause of an event. Similarly, keeping statistics about alarm generation makes data analysis possible – leading to the quick identification of problem areas which can then be resolved. Similar analysis can produce metrics related to the frequency of all alarms, which subsequently can be used to indicate the relative performance and merit of the alarming function as a whole.

Support the integration of safety functions. The sacred cow of safety is rarely questioned, but nonetheless is often overlooked while designing the core components of a control system. Safety countermeasures are often programmed at a later time into the process logic as an afterthought – with too little consistency of implementation. With the growing emphasis on the application of safety standards such as ISA-84 (US) and IEC 61508 (Europe), safety instrumented functions (SIF) are showing up more frequently in process control systems. Whether they are contained in a stand-alone safety instrumented system (SIS) or integrated directly into the control software, consistency and reliability of function are necessary outcomes.

An important metric used to determine the safety integrity level (SIL) of a safety system is the probability of failure on demand (PFD) of the system. PFD is nothing more than the likelihood that a safety function will work when called upon, and it depends on the reliability of the components from which is it is constructed. Without getting too deep into the details of all the specific terms and methods pertinent to the application of these standards, it should be easy to make the connection between the need for reliability of function during an emergency and the benefit of simple, consistent and effective means for the software to carry out the required emergency action.

Many processes are not so dangerous as to require a completely separate system for implementing safety functions. Instead, safety countermeasures are integrated directly into the basic control functionality. The best way to accomplish this is through simple, uniform mechanisms to override the current state or action of an object, along with verification of the desired result. Each object, therefore, should provide a configurable override capability that both standardizes and fully supports the means to deploy whatever safety countermeasures result from the application of a safety standard. In this way, safety functions can be realized with little effort during initial programming and readily accommodated later if safety requirements change.

Support S88-based recipe execution. This optional feature is sometimes relevant when the process being controlled is a batch process using multiple recipes. That in itself does not require the sophisticated recipe execution capabilities ascribed to S88, but certainly warrants the consideration. When it is determined that the process can benefit from this functionality, then some additional logic elements are needed. First and foremost is a routine which sequences the process operations contained in the recipe and translates them into specific actions. This routine also works with the recipe manager to collect associated parameters from the recipe and pass them along with the action information.

Another important feature is intelligent command and parameter processing. This capability provides filtering and re-direction of commands and parameters that result from the actions being executed by the current recipe operation(s). Since multiple units may be capable of performing the necessary action, the pertinent data must be routed to the resource which has been selected by the recipe execution manager (after being filtered by mode), and subsequent status information must be routed back.

Mentioned previously, resource allocation is another important capability that permits a recipe operation to temporarily take exclusive ownership of the object(s) which are needed to carry out its associated actions. This can be accomplished through object-level functionality or by means of a system-level utility, but is nonetheless a required component of a well-designed S88 batch execution system.

Recipe execution management is an operator-driven function which provides storage and retrieval of recipes, the means to define the logic and data associated with a recipe, and also provides capability to execute a recipe. This functionality can be custom built, but many turn-key products already exist and would most likely provide a higher level of functionality at a lower cost.

Standards Are the Key Which Unlock the Treasure of Sustainability

For all of the reasons described above, the application of standard methodologies is a wise choice for the implementation of a process control system. Given the time and effort committed to these standards, it’s very unlikely that many needs, details, or aspects of integration have been overlooked. Quite the contrary, published standards are a storehouse of useful and holistic information about the numerous details involved in developing a quality process control solution. They contain a thorough explanation and breakdown of techniques which are considered the best practices in the industry.

Often considered an over-used buzzword, sustainability simply refers to the capacity of something to endure. Many things affect this capacity, like continued relevance, adaptability to change, ease of management, and simplicity of use. When standards form the basis of the approach used for the design and implementation of a control system, sustainability is a logical and expected outcome. It may stretch the willingness of project participants to consider new ideas, and may even impose an uncomfortable and unpopular learning curve on the time and budget constraints of a project, but pushing through these hurdles will pay dividends for years to come – and if done well, perhaps decades.

About the Author

Rick Slaugenhaupt runs TechMinder Consulting, Inc., based in western PA, and specializes in projects relating to process control and plant-to-business integration. He has been working as an electrical engineer since 1983, having spent most of that time focused on process control design, implementation and support for large manufacturing companies.