Inheriting

BY BRUCE JENSEN AND BHAVIN PATEL

What to look out

A legacy system is a process automation system that entered service sometime in the past. Many legacy systems developed before the widespread use of modern software engineering methods.

With standardization efforts in networking through Ethernet and TCP/IP and through operating systems such as Unix and Windows NT and 2000, differences among systems in these areas were reduced.

These legacy systems are now reaching the end of their product life cycles. Without continuing long-term support, the problems of spare parts, people knowledgeable in the systems, and system reliability, their costs have risen to the point that replacement or renovation is necessary to maintain operation.

Also, many companies are investing in current technologies to take advantage of their openness and connectivity to business and other operation systems.

the wind

for when replacing a legacy system.

THROUGH THE GENERATIONS

Computer and software technologies have changed rapidly in the last 30 to 40 years. Early systems, 1970s through '80s, using minicomputers did not have a wide set of configuration tools. These systems had a monolithic structure (practically no structure).

In the evolution of programming languages, often the terms 1GL *(first-generation language),* 2GL, 3GL, 4GL, and 5GL apply to represent major steps or *generations*.

1GL was, and still is, machine language, or the level of instructions and data the processor actually uses and works on. In conventional computers, this is a string of 0s and 1s.

2GL is assembler or assembly language. A typical 2GL instruction looks something like *ADD 12,8*.

3GL is a *high-level* programming language such as Fortran, Basic, PL/I, C, or Java. A Java language statement looks like this:

```
public boolean handleEvent (Event evt) {
    switch (evt.id) {
        case Event.ACTION_EVENT: {
            if ("Try me".equald(evt.arg)) {
```

A compiler converts the statements of a specific high-level programming language into machine language. In the case of Java, the output is byte code, which converts into appropriate machine language (1GL) by means of a Java virtual machine that runs as part of an operating system platform.

In earlier versions of Fortran, the compiler converted the code to assembly language (2GL), which was in turn converted to machine code (1GL).

A 3GL requires a considerable amount of programming knowledge.

4GLs are closer to natural language than a 3GL. Languages for accessing databases are 4GLs. A 4GL statement might look like this:

```
EXTRACT ALL CUSTOMERS WHERE
"PREVIOUS PURCHASES" TOTAL
MORE THAN $1,000
```

5GLs are visual or graphical development interfaces that create a source language that usually compiles using a 3GL or 4GL compiler. Microsoft, Borland, IBM, and other companies make 5GL visual programming products for developing applications in Java, for example.

Visual programming allows easily envisioned object-oriented class hierarchies and drag-and-drop icons to assemble into program components.

BUILDING WITH TRANSISTORS

These same terms sometimes describe the computing platform (hardware) as well. A first-generation computer is a computer based on vacuum tubes and other esoteric technologies. These are the computers designed before the mid-1950s.

Second-generation computers were computers built from transistors, designed between the mid-1950s and mid-1960s. With the advent of second-generation computers, it became necessary to talk about them as computer systems because the number of memory units, processors, I/O devices, and other system components could vary among different installations, even though the same basic computer pushed the arrangement.

Third-generation computers refer to ones built with small-scale integration integrated circuits designed after the mid-1960s. Third-generation computers used semiconductor memories in addition to, and later instead of, ferrite core memory.

Fourth-generation computers were computers built using very large scale integration (VLSI) integrated circuits, especially a microcomputer based on a microprocessor or a parallel processor containing two to thousands of CPUs.

VLSI made it routine to fabricate an entire CPU, main memory, or similar device with a single integrated circuit that industry mass produced at low cost. This resulted in new classes of machines such as high-performance parallel processors that contain thousands of CPUs and the PCs we see deployed in the process industries today.

The computers and languages are basic building blocks needed to create software tools used to build process control applications. These applications are generally custom

designed to perform control and monitoring of process equipment, units, and facilities. They provide information for operations and business management.

We describe process control configuration software tools that build control applications in the same vein that we labeled the computers and languages—by generation: 1GS, 2GS, 3GS, and 4GS.

First-generation applications tools (1GS) used the monolithic structured languages. These are often 3GLs that create an application through pure programming. Functions written as reentrant routines often used structures for I/O processing and control algorithms such as proportional-integral-derivative (PID) controllers.

Second-generation process software (2GS) included *fill-in-the-form* types of tools. A form included the pertinent information in a tabular or comma delimited format. An application compiler would operate on this form to create the control or human-machine interface (HMI).

Often a PID controller was represented by several tags: one tag to represent the input, one to represent the algorithm, one to represent its set point, a tag to represent the alarms, and a tag to represent the output.

With the advent of language standards such as IEC 61131-3, third-generation process application tools (3GS) became popular. IEC 61131-3 describes programming languages of ladder logic, function blocks, sequential function charts, and structured textual languages of the 3GL variety.

Third-generation software systems enabled modularity of the control application such that the control function emulated the equipment it was controlling. That is, a PID function block would perform input processing, the PID algorithm with alarming and output processing within the one definition of a function block and thus labeled as a single tag.

Fourth-generation tools (4GS) are applications built based on object-oriented software structures. Here a graphical object representing a pump contains data attributes for the representation of the object on the graphic, such as color change for different operating modes, as well as control attributes describing its I/O and behavior for state changes. In addition, methods describing its behavior under certain conditions and interfaces function within the object.

MULTIPLE GENERATION SYSTEMS

In replacing a legacy distributed control system (DCS), one first determines the generation of the computer, the languages, and the software tools of the legacy system. The largest differences between legacy and state-of-the-art systems are the data structures: hierarchical and/or relational and the methods of programming that link them.

Hardware differences are also an issue. These differences have more to do with capacity and computing ability than functionality, though the type of system may force a type of functionality on the user.

What is of issue is whether to replicate the existing functionality exactly or redesign much of the functionality to take advantage of the increased power, sophistication, and ability of the newer-generation systems.

In replicating the functionality, the design analysis is to map existing structures representing functionality to new data structures.

The most common mistake assumes the mapping phase can take place in one pass. But there is rarely an expert in the legacy system who can completely pass knowledge about the older system to the designer of the new system.

Typically, project timelines often underestimate the data problems in the migration tasks.

DIFFERENCES IN EQUIVALENCE

Replacing a legacy system based on 3GS data structures with a new 3GS system might seem to be simple.…Not!

Al Qaeda studies cyberattack systems

The instrumentation, systems, and automation industry uses digital DCSs and supervisory control and data acquisition (SCADA) systems as tools of the trade to lower costs, get up on the competition, and operate more safely.

The U.S. federal government, in the form of the Commerce Department's Critical Infrastructure Assurance Office, is looking at DCSs and SCADAs as weak links in the fight to preserve security against those whose aims are to inflict catastrophic harm on the industrialized world.

Working together, the FBI, Lawrence Livermore National Laboratory, and the Defense Department compiled a forensic summary tracing telecommunications routed through Saudi Arabia, Indonesia, and Pakistan that *cased* emergency telephone systems, electrical generation and transmission, water storage and distribution, nuclear power plants, and gas facilities.

The Washington Post reported that some of the probes suggested planning for a conventional attack. But others homed in on *a class of digital devices* that allows remote control of services such as fire dispatch and of equipment such as pipelines. More information about those devices and how to program them turned up on al Qaeda computers seized this year.

Most significantly, perhaps, U.S. investigators found evidence in the logs that mark a browser's path through the Internet that al Qaeda operators spent time on sites that offer software and programming instructions for the digital switches that run power, water, transport, and communications grids.

In some interrogations, al Qaeda prisoners described intentions, in general terms, to use those tools.

Millions of these specialized digital devices operate and direct the brains of American *critical infrastructure*. Federal directive defines this term to mean industrial sectors that are essential to the minimum operations of the economy and government.

The digital devices are DCSs and SCADA systems. The simplest ones collect measurements, throw railway switches, close circuit breakers, or adjust valves in the pipes that carry water, oil, and gas.

More complicated versions sift incoming data, govern multiple devices, and cover a broader area.

What is new and dangerous is that most of these devices are now connecting to the Internet, some of them in ways their owners do not suspect.

Industry designed these digital controls without public access in mind. They typically lack even rudimentary security, having fewer safeguards than the online purchase of flowers.

Much of the required technical information to penetrate these systems is widely discussed in the public forums of the affected industries, and specialists say potential attackers are well aware of these systems' security flaws.

For starters, it's impossible to select a *single simplistic* approach because legacy systems typically are heterogeneous, using various languages, database systems, and hardware platforms. Engineering software tools to implement the process application and view the process are still widely diverse.

Differences in function blocks, structured textual language statements, and other tools differ greatly from *vendor to vendor* and from *decade to decade* even from the same vendor.

When translating from 2GS to 3GS systems, the complexity increases. Inherent system features may replace functionality that required application programming in the legacy system.

Here, lines of sequential code in 2GS systems must be captured and translated into either function blocks, logic charts (truth tables), sequential code, or other types of software implementation tools. Decisions of this type must capture generic functionality and handle exceptions on an individual basis.

The intricacy increases exponentially again when translating 1GS to 3GS or even 4GS systems.

FEEDING THE RAW VALUES

The two main differences for an equivalent functionality in a function block are in the *algorithm* and the *data structures.*

One of the most common algorithms in the industry is also one of the most diverse. The PID algorithm and tuning constants may be standards in their most basic forms, but different vendors implement them in different ways. There are noninteracting and interacting controller types, for instance.

Also, the tuning constants are often different. For example, some systems refer to controller gain, while others summon proportional band. Integral units may have to translate from repeats per minute to seconds per repeat. And derivative gain may express in different time units that may depend on controller gain.

To ensure that the tuning constants of the new system will perform like the loops already in place, a conversion between the existing tuning factors and the new ones is necessary. Even that may not solve the problem completely.

Some systems do I/O scanning asynchronously, feeding the raw values to a buffer that the PID function uses to calculate its output on a periodic basis and which then feeds the result to another data buffer that writes to the output.

Other systems perform the input, calculation, and output in a single dedicated scan cycle. This means the overall timing of the loop is in question, and different loop performance may occur.

Calculation blocks among systems are also quite diverse. Some have limits on the number of inputs and outputs, the number of internal registers for data storage, normalization, and the number of lines of structured text that may be applied.

Often, scripts or use of tools, such as Microsoft Access with its query tools, can aid in parameter mapping. Advantages in writing these scripting tools are that translation can take place for a myriad of components quickly.

Also, reusing scripting substantially reduces time on subsequent projects. The downside is the time needed to create the script, the debugging effort, and other tuning and tweaking necessary to ensure proper conversion.

In late July, Joe Weiss, ISA Fellow and control system cybersecurity expert for KEMA Consulting, appeared before the U.S. House of Representatives to address the topic of the cybersecurity of the critical industry infrastructures. Read his testimony, his assessment of the issue, and his recommendations at www.isa.org/Content/ContentGroups/News/2002/July6/InTech5/Control_system_cyber_securityandNum8212;maintaining_the_reliability_of_the_critical_infrastructure.htm. Or go to isa.org, and search on Weiss.

MAPPING THE ROAD FROM PAST

Differences in data structures are the most challenging. A PID function block in a legacy system, for instance, has several parameters from which the function block operates.

We can assume every PID should have the parameters of set point, measurement, and output as well as proportion gain, integral time, and derivative gain. However, other features have to be verified, such as antireset windup, set point rate of change, output limiting and tracking, and others.

Believe it or not, many systems do not have all these features within the same function block. Other blocks came on board, at some time in the past, to augment missing features. Thus, it may be a *many to one* relationship among function blocks.

Alternatively, the new system may have more powerful tools with which to implement the legacy system functionality. The best tools should be used to achieve the equivalent functionality. This allows for increased efficiency of implementation, display, and maintenance.

Thus, a documented mapping strategy must take place at the design stage. This provides not only the rules and road map to replace existing functionality but also the basis for documentation of functional design, implementation, and testing.

Also, when replacing legacy systems, the differences in programming software execution structure are often overlooked, underestimated, and misunderstood. The legacy system may have software execution similar to a programmable logic controller that uses ladder logic. These systems execute code left to right, top to bottom in a single sweep.

The replacement system may have software execution that is sequential, similar to a DCS. Then one must pay very close attention to the existing application and sort out code that must execute continuously, such as interlocks, from code that must execute sequentially.

A further issue is converting the HMI's look and feel. In theory, the HMI should be the easiest to replace or convert. But the new systems usually offer more sophisticated and flexible HMI tools and operation abilities.

And that's the problem!

You might have a few engineers responsible for the DCS system, yet there is an army of operators running the plant. For most of them, HMI or graphics is an emotional issue.

Some operators want the new system to look and feel the same as before, while some want to use all the *bells and whistles,* incorporating three-dimensional graphics, animation, colors, and the like.

Also, one has to be sensitive to the needs of some operators, such as color blindness.

It is very important to devise consensus *HMI standards* using input from operators to fashion a road map of HMI design. By doing so, the operators are part of the team so they necessarily buy in to the conversion. This avoids the common unpleasantness that management is forcing the system on the operators. IT

Behind the byline
Bruce Jensen and **Bhavin Patel** work for Yokogawa Corporation of America in Newnan, Ga. Jensen is the systems engineering support manager, and Patel is a senior applications engineer.