Putting Knowledge-Based Concepts to Work for Generic Programmable Logic Controller Programming

Ian C. Campbell and Izrail Bukshteyn

This chapter is a case history of the computer-aided logic expert system (CALES), a quality and productivity tool for programmable controller software engineering now being used in an engineer-to-order manufacturing company. Strategic goals for CALES included the removal of a corporate bottleneck in the electrical engineering department and the development of a foundation for knowledge-based engineering in other parts of the company. Our ultimate knowledge-based engineering goals are to make our corporation more responsive to customer needs and to offer greater customer values with less effort, thereby ensuring a more profitable corporation.

CALES helped remove the electric corporate bottleneck and provided a foundation for other knowledge-based engineering tools. As a result of our experiences, we developed the following generalizations: First, we want to develop only those knowledge-based applications that will assist the most strategically important of our company's tasks. Second, human education and commitment are necessary ingredients in the creation of our knowledge-based solutions. Third, the development of a strategic knowledge-based application will be guaranteed a success only if it is conducted from within an operating department and is not delegated to a third-party environment. Development must be under the direct control of a line operating manager. Fourth, cultural and conceptual contributions of the AI community are much more important than using the most advanced AI technologies.

The Original Problem Defined

Control engineers at the Lamb group of companies design hardware and software to control large electromechanical systems handling thousands of rolls of paper each day in a pulp-and-paper assembly line. These control engineers learn to design over 50 different machine types and to customize these designs (if required) to move, weigh, wrap, label, and package rolls of paper or bales of pulp. At the same time, the engineers protect the company's reputation as a supplier of high-quality, unique finishing systems currently handling over 80 percent of the world's newsprint.

These unique systems are complex, expensive and constantly changing. Some systems cost over $10 million, and if they do not operate reliably, a billion dollar paper plant feels the financial consequences. Systems often consist of over 200 individual machines and are configured to meet the customer's constantly changing needs as s/he (or the consulting engineer) thinks more about his(her) problem.

Our company sales strategy, despite difficulties faced by Lamb engineers, remains "the customer is king," thereby invalidating the obvious option of freezing a design early in the design cycle. Compounding their problem, engineers implement these unique systems on one of approximately 12 different programmable logic controllers (PLCs).

PLC can be defined as a solid-state device—a member of the computer family capable of storing instructions to implement control functions such as sequencing, timing, counting, arithmetic, data manipulation, and communication—to control industrial machines and processes. PLC can be considered as an industrial computer that has a specially designed architecture in both its central processing unit (CPU) and its interfacing circuitry to field devices (input-output connections to the real world). These field devices can be limit switches, pressure transducers, push buttons, motor starters, solenoids, and so on. The input-output interfaces provide the connection between the CPU and the information providers (input) and the controllable devices (output).

The first PLCs were more or less just relay replacers. Their primary function was to perform the sequential operations that were previously implemented with relays. PLC has been improved over the years when it comes to speed of operation, types of interfaces, and data processing capabilities; however, its design requirements still hold to the original intention: It is simple to use and maintain.

PLC programming has a standard: the ladder-logic format. *Ladder-logic diagrams* have been the traditional way of representing electric sequences of operations. These diagrams are used to represent the interconnections of field devices in such a way that the activation, or turning on of one device, would turn on another according to a predetermined sequence of events. The original ladder diagrams were established to represent hard-wired logic circuits used for the control of a machine or equipment. Because of wide industry use, it became a standard way of providing control information from the designers to the equipment users. As PLCs were being introduced, this type of circuit representation was also desirable; not only was it easy to use and interpret, but it was also widely accepted in the industry.

Our customers demand a choice of PLC based on their private analysis, which is often limited to the consideration of maintenance skills and corporate buying opportunities. Each PLC has its own language, requiring its own hardware and software expertise. New types of PLCs are demanded by our customers at the rate of one every second year; old PLCs seem to wither every fourth year.

The withering of PLC also means the death of whatever software standards evolved during its lifetime. The arrival of a new PLC means creating new standards at the same time the new PLC is being learned. In this environment, it is impossible for practical software standards to evolve. For example, the validation of standard software for one PLC could not imply validation for any other PLC. Because of this situation, system designs used to be costly, sometimes incomplete, and often error prone. Startup costs were unpredictable because the Field Service Department often made substantial software changes in the field, some that were unnecessary and many that went unreported to the design engineers. By the spring of 1985, the corporate bottleneck created by all these factors was obvious.

Although by mid-1986, our chief executive officer (CEO) had identified the electrical engineering department as our most serious problem, we continued accepting orders as Lamb flourished in history's greatest capital equipment spending cycle, experiencing a threefold increase in business activity. However, our capacity to do PLC software was floundering; this situation could have led to unthinkable consequences if not handled successfully.

Description of AI Technology Use in the Application

A fundamental relationship exists between the expertise of control engineers and their ability to meet the needs of a specified system with quality and on schedule. Lamb could realize a fantastic competitive advantage by capturing the skill of the world's best experts in our specific business arena, then automatically applying these design skills as needed. This possibility became more attractive as concern grew about the difficulty of training many new control experts in the current, fast-paced business cycle.

The automation of control engineering design would not only solve an immediate problem, it would also follow some of Lamb's individual strategic pathways. These pathways required experimenting with the automation of engineering design and hopefully gaining strategic business advantages at the front end of the corporation. The solution of a strategic problem using expert system technology could open the entire corporation to legitimate exploitation using similar concepts.

We rejected the possibility that major PLC manufacturers could somehow work together to solve the problem of multiple dialects and hardware. We believed that each PLC manufacturer needs to differentiate its product as superior and, therefore would be hostile to any tool that seemed to reduce PLC products to commodities. We could find no commercially available design tool that would meet Lamb's needs and, in fact, still cannot. We had to begin work on our own and did so at an affiliate location, the Lamb-Cargate Engineering Department in Vancouver, B.C., Canada. This department had been losing money but had the skill, energy, and enthusiasm to undertake a new challenge.

The first task was to find a method of representing the knowledge carried in the heads of expert control engineers. Because experienced control engineers could talk to each other and verbally explain important generic concepts, the first attempt at capturing their conceptual and generic knowledge was based on a stylized form of English, for example:

```
(on this-machine "start-motor"
(if (previous-machine "seal-in-
contact" on) and
      (next-machine "safety-interlock"
      off)))
```

This statement explains that if the previous machine's (pm) seal-in coil was energized, and the next machine's (nm) safety interlock coil was not energized, then we want this machine's (tm) start-motor coil to be energized. This representation of knowledge was entirely independent of PLC to be used; the project itself; and the identification of actual machines involved, either current, upstream, or downstream.

The first CALES prototype experimented with our ability to represent large, generic chunks of control knowledge with a stylized English language. It experimented with the possibility of linking many chunks of generic knowledge into a specific system made of many examples of the generic knowledge. It also continued to enforce the separation of generic knowledge from the design engine so that normal control engineers could capture their knowledge and make subsequent designs without any skill in Lisp. The results of this initial work were encouraging. The work was conducted in the first few months of 1985 by the chief engineer who was emotionally committed to creating a substantially better method of doing business and was a self-taught Lisp enthusiast. His commitment to Lisp beat the AI insertion barrier that many organizations experience. Further responsibilities for CALES development were transferred in July 1985 from the chief engineer to the chief electrical engineer. He continued to code in Lisp. As line managers who are measured on their performance in creating industrial engineering designs and who are not required or expected to do anything with AI, we both found it impossible to explain new creations, visions, needs, and solutions to other people and, sometimes, even to each other. It was just much easier to code the novel or normal engineering solutions than to find some way of communicating with another human being. During the six-year development of CALES, we had to rely on faith, stubbornness, endurance, and a little luck for our success.

We started to use this tool in August 1985 and completed our first project within three months, for an Allen-Bradley Series 2 PLC. We named this new creation CALES. It soon became evident that although the stylized English form was perfectly adequate for the computer, it was too difficult for a human operator to use. The ladder-logic language was accurate for relays but could not handle the generalization or generic content needed for current-day PLCs. Many modern-day control engineers still prefer this ladder language and totally reject the idea of using any other new language. Because of this preference, we next wrote an intelligent editor to guide our detail-oriented engineers in capturing their generic knowledge in a manner almost identical to simple ladder logic (figure 1).

What we actually captured was enough information to translate their input into our stylized English. We considered this development somehow perverse because the computer was using English to direct its behavior, but English-speaking humans were rejecting this language in favor of their own specialized, ladder-logic language. We now realized we were dealing with a problem that needed multiple translation tasks, both in parallel and in series. We became more formal in specifying the emerging CALES language, which we required to be generic and

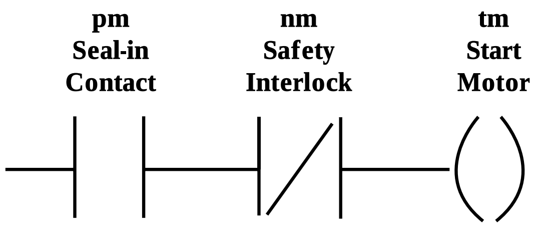

Figure 1. An Intelligent Editor Captured the Engineer's Knowledge in a Manner Similar to Ladder Logic.

still use the ladder-logic format. We now were glad we were using Lisp as our coding language. We would probably have abandoned our efforts at this stage of translation had we not already been fluent in Lisp and had confidence in it.

The Lisp we were so dependent on is called Waltz Lisp. It was inspired by Franz Lisp, cost $169 in 1982, and runs in a DOS environment on an IBM PC with 640 kilobyte (KB) and a hard disk. Languages used by Lamb for expert system tasks include Golden Common Lisp, Ibuki Common Lisp, GoldWorks, Genesis (a neural network tool), Turbo Pascal, and Powerhouse (a 4GL). Our most recent language is Top Level Common Lisp and Common Lisp Object system (CLOS), a parallel Lisp and object system that runs on a Sequent Symetry S27, the same parallel hardware that supports our corporate information system based on relational database management systems. We expect to integrate future expert system work with our conventional information system by implementing SQL calls from inside CLOS.

By January 1986, we had to confront the problem of handling PLCs other than the Allen-Bradley Series 2. This problem was handled by translating from the generic CALES language into the target PLC language before continuing with system integration and design. Our next PLC target was the Modicon 984.

By March 1986, we were no longer self-conscious that we weren't knowledge engineering our developments or performing system analysis. The development and use of CALES was in the hands of one organization and was subject to daily change as real production work proceeded. With respect to CALES, our best expert and our Lisp expert were one and the same person. He did not need to do knowledge engineering and came to believe his achievements would have been impossible had he needed to deal with any other person for system analysis or knowledge engineering. Furthermore, it seemed counterproductive to separate the CALES development from its regular daily productive work.

The year 1986 was devoted to using CALES in production and developing other translators for CALES so that our growing list of generic modules could be used on other projects. Allen-Bradley Series 3 and General Electric Series 6 translators were added in 1987 and 1988. We were now confident that we could add multiple translators in the future and, through this technique, use ancient control knowledge (translated to CALES language) on future PLCs (see figure 2 for the CALES flowchart).

The year 1987 was devoted to training more individuals in the use of CALES and in promoting this home-grown system at other affiliate locations in the corporation. CALES met with initial opposition because of conceptual strangeness, its inability to handle large systems, and translation skills that were not as good as a skilled human (perhaps only 70 percent as good).

We were now forced to set a goal of designing big systems with several thousand coils while working within 640-KB random-access memory. We were again tempted to abandon our efforts at this stage until we modified our programming style by limiting the number of Lisp symbols used, externally storing most of the intermediate data, and splitting Lisp translation and system integration programs into tens of relatively small Lisp programs executed by a management program written in Pascal.

The year 1988 was devoted to accelerated project work; the development of other translators and the refining of those we had in operation; and the continued selling of the concept of knowledge-based engineering to conservative, skeptical, or hostile control engineers. However, "the best expert's expertise" requirement now became a major stumbling block. A huge overload of normal work (perhaps combined with a touch of human politics) started to slow the diffusion of this technology within our company. This slowing sometimes took the form of vetoes and foot dragging on the official version of our best expert knowledge. Now, there were differences over who was the best expert or what combination of experts could formulate consensus-best expertise. While this friction continued, the original developers and users of CALES continued to create and deploy their own expertise in the absence of an official version by building on the previous work of the company's best experts. Most PLC experts outside Lamb would not believe that efficient translation from generic ladder logic to specific PLC languages could be done by a computer.

The year 1989 was devoted to further refinements of our generic CALES language, including nested branches and conditional logic rungs. By early 1989, CALES could translate 90 percent of the PLC instructions, including math, block, and other advanced functions. In terms of the required PLC memory and efficiency, the CALES translation was now about 90 percent as good as a best expert. All these refinements reduced user opposition. CALES was deployed within the

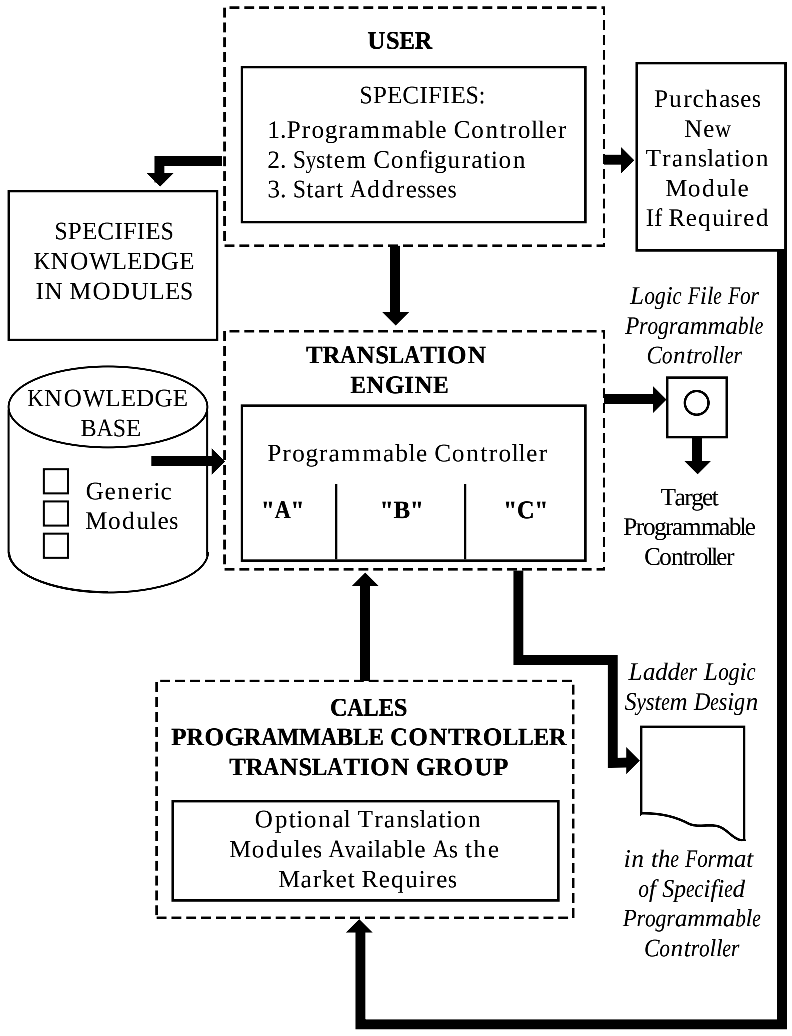
USER
SPECIFIES:
1.Programmable Controller
2. System Configuration
3. Start Addresses
Purchases New Translation Module If Required
SPECIFIES KNOWLEDGE IN MODULES
KNOWLEDGE BASE
Generic Modules
TRANSLATION ENGINE
Programmable Controller
"A"
"B"
"C"
Logic File For Programmable Controller
Target Programmable Controller
Ladder Logic System Design
in the Format of Specified Programmable Controller
CALES PROGRAMMABLE CONTROLLER TRANSLATION GROUP
Optional Translation Modules Available As the Market Requires

Figure 2. Examples of the Cales Translation from Generic Ladder Logic to Modicon 984 and Allen Bradley Programmable Logic Controllers.

company and was gradually accepted as a valuable tool by all experienced control engineers in the company. Corporate best experts now promised to establish machine control standards (that is, knowledge) for CALES. The original developers and users of CALES created their own references by translating old, proven software into CALES language using the company's best and proven past project references.

The original developers and users of CALES then started getting feedback from projects that had started up and began to systematically refine their CALES knowledge base. Also, the field service group was now regularly seeing the CALES designs in the field. Field-service comments were initially negative, and a number of unrelated happenings were blamed on CALES. It took several projects to develop some understanding about what CALES could and should be blamed for. By year end, some field-service engineers were reporting to their superintendent that "this CALES-supplied software is the best software the company has ever provided."

By the end of 1989, the CALES engineers had developed and validated their own generic modules and had documented some successful start ups. These CALES engineers then announced that in the interest of high quality and on-time scheduling, they would refuse to use any generic module other than that which they had created, refined, and validated. Lamb's best experts then accepted CALES as an efficiency tool and agreed to use the developed generic modules as the temporary company standards.

In 1990 an Allen-Bradley Series 5 translator was added to CALES. The improvement in speed and user interface was achieved when the CALES intelligent editor was rewritten in Turbo Pascal. By the end of 1990, more than 150 CALES generic modules had been created by Lamb experts, and were being used, as company standards, by Lamb Control Engineers.

One of the first Lamb high-speed roll finishing systems designed with the CALES software was started mid-1990 at Canadian Pacific Forest Product newsprint mill in Gold River, B.C., Canada. This system consists of all main Lamb machines, including wrapper dispenser, header, crimper, headmaster, label applicators, band dispenser, kickers, and slat and belt conveyors. The system was started up successfully and provides a 20-second roll machine cycle time. CALES has now been used in 20 Lamb roll and bale finishing systems.

Lamb Engineering Department finances have dramatically improved, and CALES is credited with being a major contributor. The department improved from a loss of over 10 percent of sales to a sales profit of 17 percent (sales had grown by over 200 percent [$ = pretax profit]); these figures are even more amazing when you consider CALES was developed as a "skunk works" project with little overt funding and substantial development and standards costs buried in the financial figures. (We think of a skunk works project as one that is allowed freedom from the normal checks and balances of bureaucratic organizations. It is a project driven by a fanaticism condoned by strategic policy makers because of the hope of making radical improvements in important, difficult, and intractable problems.) Therefore, it's only fair to conclude that the true financial benefits of this technology are substantial but not yet fully quantified.

The human problems surrounding CALES continue into 1990. Not surprisingly, the greatest problems in implementing knowledge-based engineering systems have been and will continue to be those dealing with people and the management of their work.

Innovations Brought by the Application

Many of the innovations created in developing CALES were related to concepts, with unconventional system design, implementation, and Lisp programming, and did not require sophisticated the use of AI technology. The most important AI contribution was the belief (false?) that we could handle any knowledge-based problem by coding in Lisp if we thought the problem important enough and spent enough time on it. Our development group did not have access to expensive AI hardware or software but never felt limited in creating the CALES language or translation packages, although our software did suffer from the infamous 640-KB DOS barrier.

Looking back, we merely created a language and a compiler, an achievement that might not seem particularly innovative. However, we did feel innovative and felt we were relied on fundamental concepts while we developed and deployed CALES. We believe the work would not have been completed without the combination of (1) using the Lisp language and having a rapid prototype mentality, (2) working on development and production at the same time with the same group of people, and (3) having a knowledge engineer and real engineer in one person.

Criteria for a Successful Application

This section elaborates on the strategic goals for CALES that were described at the beginning of this chapter. The first thing that happened on the way to lower cost and good schedules was an improvement in quality. Only after realizing almost perfect quality was it possible to automate the design for cost and schedule.

Another unexpected benefit was a management attitude change. Everyone concerned with the management of the electrical engineering function at Lamb believes that this expert system has given the company a competitive advantage in cost, quality, and rapid response time. Everyone recognizes the genuine technical and human risks and problems and our inability to maintain exact project control with these problems.

The fact that we're still unable to schedule or estimate the costs of future phases of the development, does not greatly concern us. The quotations that follow are from two experienced individuals: Warren MacFarland in the 1989 video *Competitive Advantage through Information Technology* said, "If you as a senior manager cannot handle great volatility, uncertainties, and overruns in project costs or schedules, just do not

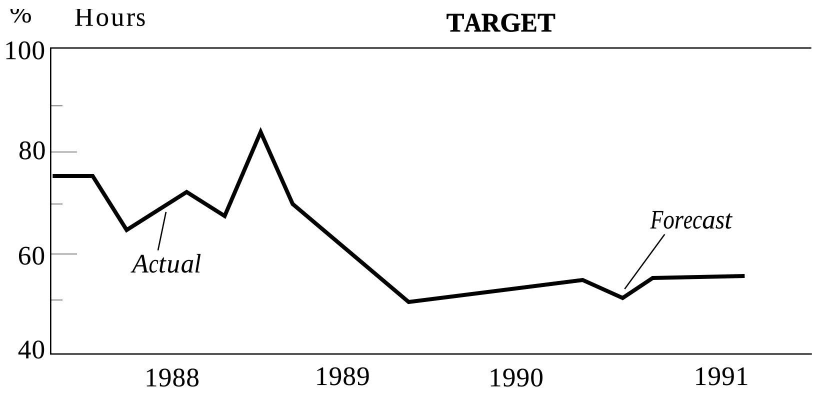

Figure 3. PLC Software Engineering Hours

start projects that are very unstructured and that use technology which is very advanced for your project team." George Rusznak of the Index Group, Los Angeles, said "We don't think of strategic moves as being primarily cost justified."

The same CEO who originally worried about the electrical engineering department now says it is no longer on his list of critical corporate problems and is allowing expert system development in other areas having corporate strategic benefit potential. These efforts are Caces (started in 1986 for schematic and wiring diagrams), Cames (started in 1987 for mechanical configuration and drawing preparation in Autocad), and Capes (started in 1989 for proposal costing, pricing, and writing). Our company might begin the implementation of a five-year plan to restructure the entire front end of the company so it will get maximum leverage from the evolving Capes-Cames-CALES capabilities. Many of these changes were never part of our goals for CALES but are indications that CALES is influencing the structure of our company.

Nature and Estimate of Payoff to Our Organization

We have not been able to quantify the benefits of CALES, but here are some preliminary data: Intelligent standards reduced software design costs by 50 percent and design software errors by 80 percent. Customer satisfaction with software improved by 50 percent. The peaceful dreams of the Lamb chief electrical engineer were improved in quantity and quality by 100 percent because he is much less concerned about litigation. The Lamb chief electrical engineer reports he would need CALES even if it didn't save money and improve response time; he needs it most for its quality work. He also says that he doesn't know how our company could do the required work if CALES were suddenly taken away. However, he says CALES is still in its early days; as it diffuses further into the departments and gradually takes on more and more work, it will continue to change the essence of his business.

Information technology is beginning to be seen as more than a computer program; it can be a new way of doing business by using knowledge in the form of intelligent standards. Perhaps the greatest payoff is that the phrases engineering standard and business standard are not automatically considered inflexible or bad because intelligent standards can be made as flexible as the business or engineering conditions require. If a human expert needs to intervene because CALES cannot handle certain details, s/he only has to intervene in small portions of a large task because the work is of a high and predictable quality.

Although proof is not available, CALES might have allowed the company to take 20-percent more work for at least two years by removing a corporate bottleneck. If so, the contribution to overhead and profit will be measured in the millions of dollars and will clearly be a strategic benefit for a company of our size. If our cyclical market suddenly becomes soft, we'll be well positioned to compete on price if we choose, another clear strategic gain. Our original goal was to reduce person-hours by 80 percent of the budget. We've only reduced it by 50 percent but haven't abandoned our original goal.

Deployment Times, Costs, and Technology Transfer Problems

CALES development started in 1985 and continues. We don't know our development costs because CALES development became so intimately tied to Lamb's production work that normal cost accounting was not possible. We no longer think of CALES as in development and, therefore, fund further work either from the jobs or as an overhead charge because "it's just our way of doing normal department business." It's also a way of continuing our skunk works mentality, which we believe is valuable.

Technology transfer was no problem for the first location because the developing group was also active in using the tool; therefore, if something wasn't right, the group fixed it or worked around it until it could correct the problem. Technology transfer problems were more important with other engineering groups within the company, but the benefits of quality and productivity have gradually been selling themselves.

Acknowledgments

We would like to thank the AI community for its contributions to our company's work in strategic planning and implementation.