

Running Head: APPLYING CONTROL PROGRAMMING STANDARD IEC 61131-3

Applying the
Control Programming Standard IEC 61131-3
to Software Engineering
Fernando E. Doylet
Nova Southeastern University

Software Engineering Project
Graduate School of Computer and Information Systems

Abstract

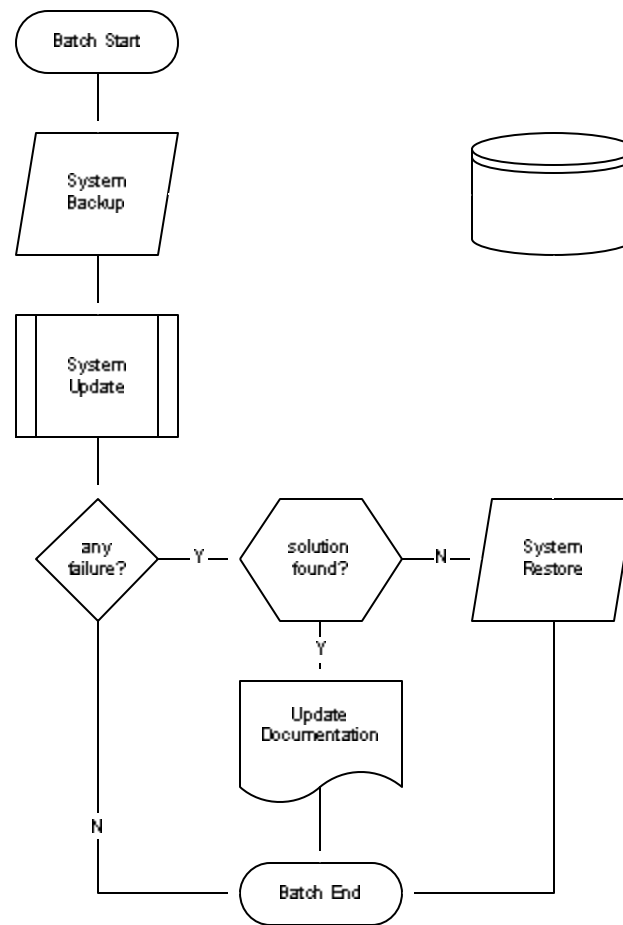
Preparing periodic batches to run updates or get reports from mainframes, requires a good understanding of the requirements of each component, to avoid the cancellation of any batch process and its resulting lost of time and resources. To improve that understanding, an industrial Programmable Logic Controller (PLC) was set to simulate a batch process; the industrial components included in the existing tutorial, were used based on their functionality, as equivalent software components.

Applying the
Control Programming Standard IEC 61131-3
to Software Engineering

Legacy systems are usually under-documented, written in dated languages and the original design blurred by several maintenance operations, contains a lot of – partially undocumented and implicit – knowledge. Because of the growing importance of legacy systems, long-term software evolution has become a maintenance concern and a significant topic within software engineering. (Gall, Jazayeri & Klosch, 1995). Furthermore, “for many long-lived systems, software maintenance accounts for more than 75% of the total cost of the software during its lifetime.” (Leach, 2000, p. 365)

Problem Investigated

To maintain mainframes updated, computer departments schedule their recurring jobs in daily, weekly, bi-weekly, monthly, quarterly, yearly or on-request batches; sometimes processing two or more batches together or in sequence; usually backing up one or more systems before starting, to protect their critical files, their relationships and the whole processing structure. Batches are given priority and protection, for the risk they represent. (Applegate, McFarlan & McKenny, 1999, p. 211).



When something goes wrong, restoring the whole system is the last alternative, because it represents declaring all that batch processing time and resources wasted (see Figure 1). To determine if a failure may be ignored, overcome, or justify a restore, requires good documentation, the support of an expert on the failing software or system, or plain good luck.

“In most instances, business managers underestimate the degree to which their firms are dependent on IT. Even small interruptions in service can cause massive customer defections or significant—and costly—operational disruption.” (Applegate et al., 1999, p. 11)

“Maintenance can be very complex, particularly for older systems. It requires highly competent professionals to safely perform necessary changes in a way that does not bring the system *and the firm* [italics added] to a crashing halt.” (Applegate et al., 1999, p. 36)

With the pass of time, there are more reports to produce, legacy software to maintain (some in parallel with new systems to compare to) or software to update, test systems, changes to existing modules and so on. This variable load of required processing is usually done overnight or on weekends, while users are inhibited from their regular processing.

Need for the Study

Since the expert on a failing software may be unavailable, the documentation is usually updated (formally or informally) when a new unspecified failure occurs. More significant than the documentation on how to fix the failing component (which may not fail again) is the knowledge about how that software component relate to other programs or procedures, to have a better sense of the whole system structure and improve our decision making process.

“The challenge in IT, conversely, has been that of harnessing an exploding body of knowledge within a very short time frame.” (Applegate et al., 1999, p. 7)

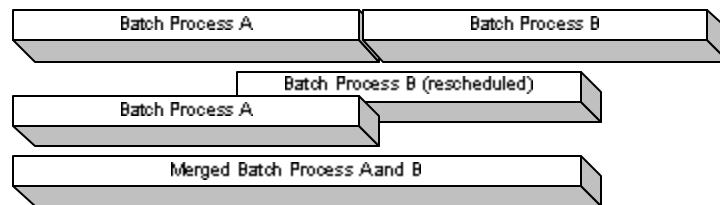
By the time a technology is reasonably well understood by both IT personnel and key users, standards and controls are developed “to ensure that the applications are done economically and can be maintained over a long period of time. Formal standards for development and documentation, change control, cost-benefit studies, and user charge-out mechanisms are all appropriate for technologies in this phase. Failure to develop and

maintain these standards can be extraordinarily expensive.” A common pitfall during maturity is that enthusiasm for the technology dies while there is still opportunity to use it to add value. (Applegate et al., 1999, p. 28).

The concept of *dependent composition* applies: the dependencies among parts are multidimensional; this inherent complexity requires a mental shift in the way that the software development process is viewed. (Valenti, 2002, p. 250). After all, the whole system is usually developed (updated or upgraded) with each batch process.

Main Relevance

Knowing the dependencies and relationships between software components also help us restructure the Batch process to increase reliability and performance, every time a job is added or retired, or when merging two or more batches in one. The better control we get on the whole process, decreases our possibility of errors and minimizes our batch processing time.



For example, if instead of scheduling two batches in sequence, we merge them in one and run a portion of their jobs concurrently (like on Figure 2), we are saving the time those concurrent jobs would take to run independently.

Goal or Objective

Beyond an isolated graphic representation of each software component dependencies or restrictions, which could be part of each component documentation, our goal is to obtain a intuitive graphic alternative representation of the batch process, a

dynamic model to show each software component with its restrictions and dependencies; for that purpose, we use an industrial PLC simulator, based on the International Electrotechnical Commission (IEC) 61131-3 Standard.

“Computer-based simulation is a well-established technique that allow systems (e.g. manufacturing, business processes, computer networks) to be modeled in a dynamic way and permits controlled experimentation on the model that would not be possible on the real-life system. This enables various alternative scenarios to be compared without the expense and disruption of fully implementing the proposed changes.” (Henderson, 2002, p. 151)

Elements Investigated

The IEC 61131-3 Standard, first released in 1993, has been widely accepted by the international user and vendor community, for programming and configuring industrial control devices; their main products are PLCs connected to actuators (like switches and timers) and sensors, mostly programmed using Relay Ladder Logic (RLL) programming for its intuitive graphical interface. (PLCopen, 2001)

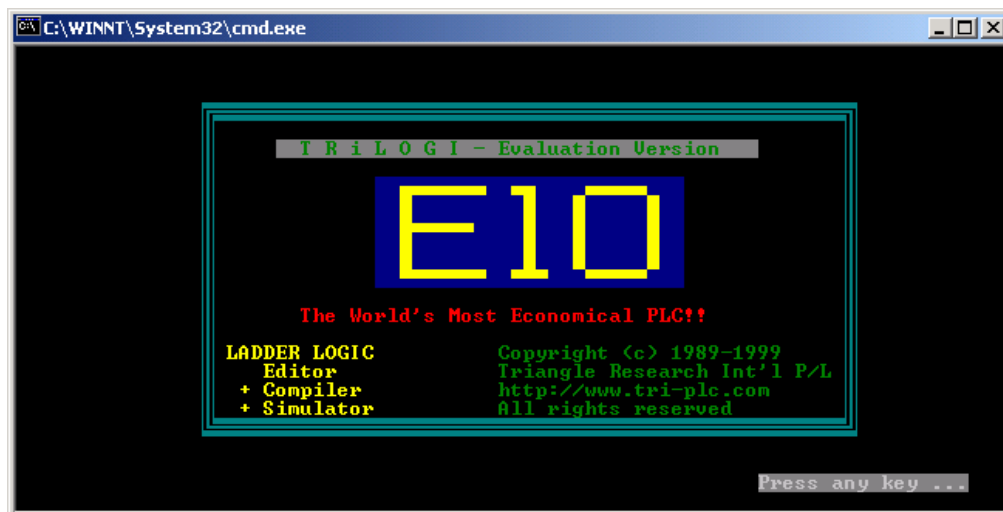


Figure 3. TRILOGI software presentation screen

The PLC simulator shown on Figure 3, named TriLOGI (International Control Components [ICC], 1999) is presented as a quick to learn and implement graphical control alternative; the first challenge is to adapt it to work with software components and their requirements, sequence of events, warnings and other details; then we can use this simulator to communicate with other computer developers, operators and customers.

There is no technique that is best for all projects and this technique is flexible enough to be presented in two ways for requirements validation: as a Manual model to show functions and relationships, or as an Automated model to validate functions (Pfleeger, 2001, pp. 175, 179).

Approach

We will start by learning the basic components in the simulator, followed by the understanding of how to represent simple decision making and sequences of events using the RLL programming language; then to complete our introductions, simple warnings, holders and the six workflow primitives will be represented.

Limitations of the Study

The scope of this study will be limited to adapting the functionality of the simulator to work with a software batch process, typical of a business computer department, to improve communications and accelerate the learning process. Our main concern will be to avoid the stereotypes that the industrial background of the simulator may cause.

Summary

In brief, adapting an industrial simulator to represent a software batch process, including each component with their restrictions and requirements, we will significantly

improve practical learning and communications, facilitating time savings and decision making efforts.

Review of the Literature

To justify and make sense of the proposed adoption and adaptation of an existing simulator, to aid in the administration or troubleshooting of obsolete technologies, in this chapter we expose the reasons for this study and our sources of information.

Overview

Maintaining legacy software is a continuous concern for the businesses that prefer their proven reliability, to the uncertainty of replacing their most critical software components with newer versions; consequently, those businesses need to keep developing their legacy systems, creating new modules or programs and upgrading or updating their existing software.

“With the introduction of microcomputers and the proliferation of end-user computing and decision support applications in the early to mid-1980s, users began to take back control of their information that had been trapped in mainframe systems.” (Applegate et al., 1999, p. 215)

To deal with the natural change of times and people, extensive documentation is accumulated to understand how each component fits into the whole system; that written documentation is usually understood only by senior programmers and senior computer operators, because it takes time to teach the newer employees about the available software documentation, functionality, requirements and relationships.

Research Literature

“Given the communications technology available in the 1960s and 1970s, extended work on the mainframe was essentially limited to those residing in the same building that housed the computer; those in remote sites were limited to very brief

sessions in which the user could access a single record or piece of data. All other data were 'batched' together to be shipped to the mainframe at one time, frequently during evening hours." (Applegate et al., 1999, p. 211)

The preparation of batches is usually a copy-from-previous routine, with little modifications as needed; those schedules are so critical (as the files updated by them) that the whole system is backed up, in case an unexpected failure force us to restore. To avoid possible mistakes, junior programmers and junior operators have the tendency to follow the examples instead of innovating, with little insight about the details, making naive structural mistakes that accumulate with the pass of time; those mistakes could go undetected for years, until the whole system starts failing. Take the year 2000 as a recent example.

The problem of understanding how each piece of software fits into the whole system is the reason of this study, which will provide an option to build a simulator from adapting an existing software, based on the Control Programming Standard IEC 61131-3.

Contributions to the Field

Contrary to other techniques like Data Clustering, Reengineering, or Software Modularization; that require additional analysis and resources; the careful use of a simulator will provide a test environment and a learning tool where to try possible changes and determine the benefits or complications, without the risk of experimenting with the real system.

The simulator will be TriLOGI, the industrial most user-friendly Ladder Logic Editor + simulator (free evaluation version) to demonstrate the examples (ICC, 1999)

with free to download step by step instructions and tutorial in its Programmer's Reference (Triangle Research International [TRi], 1999).

Resources Available

Our central source of information will be PLCOpen, an independent worldwide association, with one of its core activities focused around IEC 61131-3, the only global standard for industrial control programming (PLCOpen, 2001)

Other resources will be: a real-life example from the Production Control area at The School Board of Broward County in Florida (SBBC), the Institute of Electrical and Electronic Engineers (IEEE) Standards Information Network, the Software Engineering Standards Committee, PLCS.net , IEC.ch , ANSI.org and the like.

Conclusion

In the same way obsolete mainframes and their exclusive batch jobs are maintained and continue to evolve, it is important to obtain a troubleshooting and learning tool to better understand the system, and helpful when such tool is free, easy to learn, standardized and kept updated due to its primary industrial applicability.

Review of Methodology

Our Scope in Perspective

To investigate fluctuations in batch jobs performance at the School Board of Broward County, processing times and delays are extracted weekly from the automatically generated mainframe reports, to end up with simpler reports in Spreadsheet format.

To understand the structure of the information obtained, several interviews were needed with the experts in charge of scheduling the weekend batches, and those in charge of reviewing or translating those scheduled requests into computer instructions.

After finding that most jobs were scheduled in sequence instead of concurrently, the list of all available jobs have been transferred to a database table, to later include the specifications of each job design, related to other jobs; meanwhile, a relay ladder logic simulator is being explored, to produce a graphical representation of those specifications.

This study will provide a detailed explanation of how to represent each scheduled batch job in the simulator, including its requirements and restrictions, where the batches could be modified and tested for training, or in search of possible benefits or inconsistencies.

As all other test environments, the proposed simulator will be as reliable as the parameters included in the simulations. The challenge will be to make the simulation as close as possible to the production process, eliminating the inconsistencies that may cause unexpected failures.

The test and learning tool here introduced, will be adapted from its proven original use for industrial controllers (like motors and switches) to work with software

components instead; a careful analysis of corresponding components will guide our efforts to continue using this or a similar simulator.

Understanding the Simulator

PLCs work with RLL programs, that take their names from the electric switch called *relay* and the appearance they have, similar to a ladder. A relay is a switch that is closed only when it is energized. Switches could be normally open or normally closed and change to their opposite state when energized; switches are also used to energize other components like timers, counters or outputs.

The continuous vertical line on the left on Figure 4, is continuously energized from top to bottom, so we read the steps of the ladder from top to bottom and from left to right.

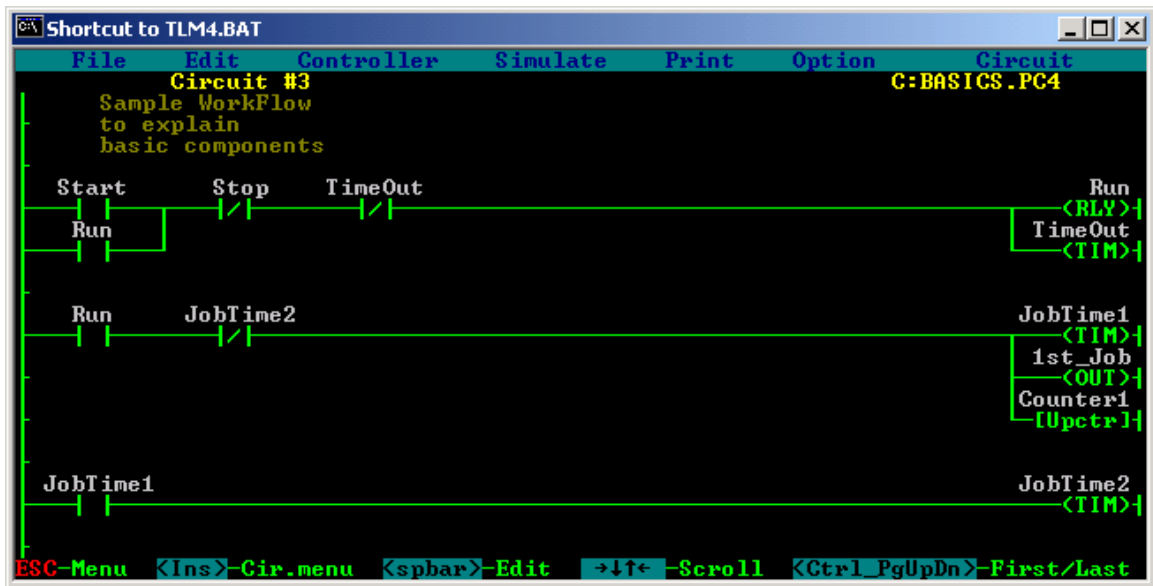


Figure 4. Relay Ladder Logic circuit example

Understanding the Components

On the right side are represented the components to be energized, and towards the left are the resulting states of those same components (with the same names) or other

manual switches, placed as needed; the same component could be represented as a normally open or a normally closed.

At creation time, the components can be created as...

- INPUT -| |- : like a manual switch or button;
- OUTPUT -(OUT)- : a light or energy signal;
- RELAY -(RLY)- : a switch that changes its state while it is energized;
- TIMER -(TIM)- : a switch that changes its state while it is energized and during a set time only;
- COUNTER -(__ctr)- : a value that goes Up or Down one, each time it is energized.



Figure 5. Basic components on Self-latching circuit

The first step of our basic ladder demonstrate two button inputs (*Start* and *Stop*) and a Relay (*RLY*) and a Timer (*TIM*). When the *Start* button is pressed, the *Relay* and the *Timer* get energized; the *Relay* closes its switch with the same name *Run* in parallel to the *Start* button, so it keeps energy continuously flowing to the Relay and the Timer, even after the *Start* button is released and its switch or path is opened. When the *TimeOut* Timer finishes its set time, its normally closed switch opens and the energy stops flowing to the right, disconnecting the Relay and the Timer. This is called a *Self-latching* circuit (TRi, 1999, pp 3-15) and is shown on Figure 5.

The *Start* and *Stop* buttons are the typical ways to initiate and end the simulations; on the other hand, the *TimeOut* Timer could be more useful triggering a warning signal

instead of ending the simulation, unless a continuous cycle or endless loop is purposely set, like in the simulation now being explained.

Timers will represent the time each computer program takes to process and their switches will allow other processes to start, producing sequence of events; the second and third steps shown on Figure 6 will alternate two timers, counting the cycles with a counter, only while the *Run* Relay is active, which is until the *TimeOut* Timer runs out of time.

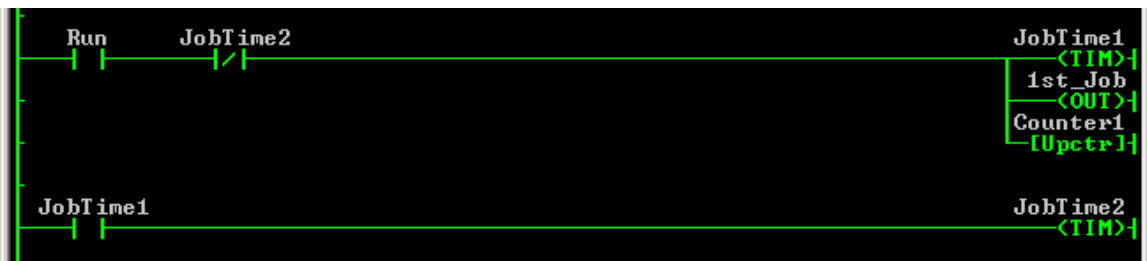


Figure 6. Example of circuit to count cycles.

Once the *Run* relay is energized, energy flows to the *JobTime1* timer, the *1st_Job* output and the *Counter1* counter; at the end of the *JobTime1* timer time, its *JobTime1* switch on the left closes, activating the *JobTime2* timer; likewise, when the *JobTime2* timer time ends, its *JobTime2* normally-closed switch on the left opens.

The moment when the *JobTime2* switch opens, energy stops flowing to the three components on its right, so the *JobTime1* switch on the third step is reset to its normally open state, cutting the flow of energy towards the *JobTime2* timer, which resets its *JobTime2* switch to its normally closed state, starting the sequence again.

IN 1	IN 2	TIM 1	CTR/SEQ 1	RLY 1	RLY 2	OUT 1
Start	.	JobTime1* 0	Counter1 9	Step_1	.	1st_Job *
Stop	.	JobTime2 65	Counter2 -	Step_2	.	2nd_Job
.	.	JobTime3 -	Counter3 -	Step_3	.	3rd_Job
.	.	JobTime4 -	.	Step_4	.	4th_Job
.	.	JobTime5 -	.	Step_5	.	5th_Job
.	.	TimeOut 2240	.	Run *	.	.
.

Figure 7. Simulation of circuit to count cycles.

Running the simulation (observable with the software simulator on Figure 7) we can see that the *Counter1* counter is incremented when each cycle starts, the *1st_Job* output is continuously energized and the simulation ends when the *TimeOut* timer ends its set time. We can also observe that the *JobTime1* timer is continuously energized, even after it runs out of time, and that the loop is reset in apparently no time at all.

Results

Based on the training in PLCs received by the author of this document a decade ago, concluded with the simulation of the traffic lights on a busy intersection, the updated free version of the simulation and its programmer's reference were retrieved and reviewed to explain its components; in this chapter, we will examine other example simulations to prove the applicability of this tool to various loops and other basic conditions.

Renaming the components and the option of adding comments in as many rows of the simulation as needed, gives us easier ways to understand and explain the process and its internal requirements; but we still have to account with assumptions, like the proposed study of the first three chapters of the Trilogi Programmer's Reference (TRi, 1999), to get hands-on experience with the simulator.

Before diving into more detailed simulations, we will explore the convenience presented by Trilogi (our selected simulator) and its IEC 61131-3 certification; that will assure us of the validity and importance of completing the remaining of the learning curve presented in this study, with the final and basic simulations included in this chapter.

Analysis

Trilogi is a certified IEC 61131-3 programming system that have an agreed degree of source code compatibility and similar look and feel in relation to its PLC competitors. Initially, the IEC 61131-3 Standard was seen primarily as a PLC standard, until its popularity increased when the first PC-based "soft"-controllers (that made PCs behave like PLCs) were defined. (PLCopen, 2001, Mission section).

While a conventional PLC contains one resource, running one task, controlling one program, running in a closed loop, the “IEC 61131-3 adds much to this, making it open to the future. A future that includes multi-processing and event driven programs. And this future is not so far: just look at distributed systems or real-time control systems. IEC 61131-3 is suitable for a broad range of applications, without having to learn additional programming languages.” (PLCopen, 2001, Intro section).

To relate to the most typical circumstances during batch programming, slightly more advanced simulations will be shown with the same basic components, to demonstrate how we could setup Holders, and represent Warnings and Workflow Primitives.

Warnings and holders

Timers could be set to produce warning signals while the process has not reached a certain job in a specified time limit, as set on Figure 8 and simulated on Figures 9 and 10; in other words, the warning signal is energized after a Timer times-out, until the required job ends; so the end of the required job energizes the selected job being monitored and de-energizes the Warning signal. In this example we also demonstrate how each line of the circuit could be documented.

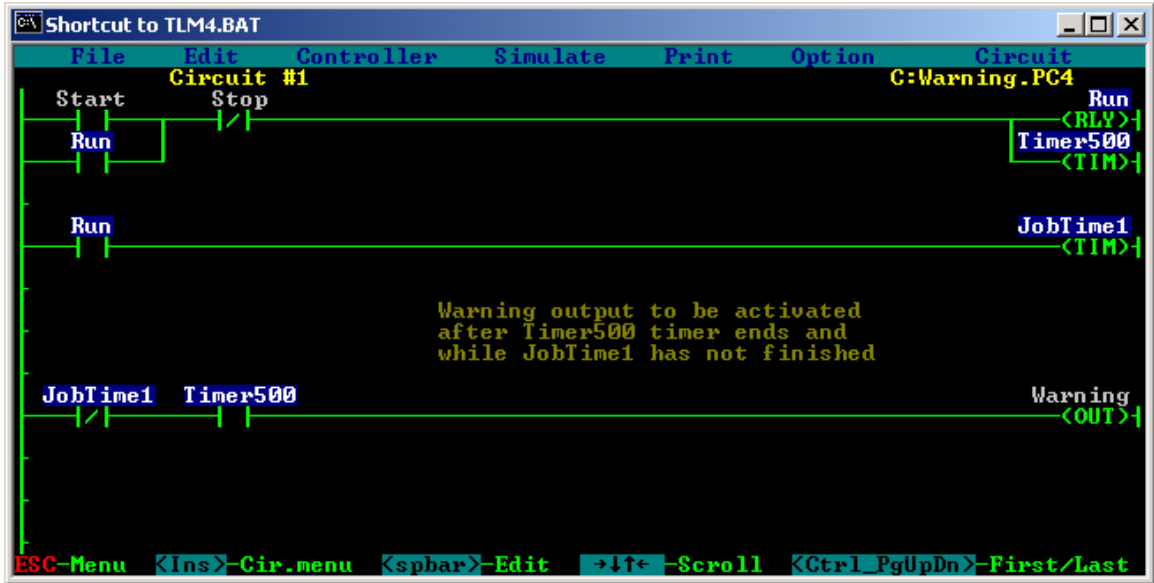


Figure 8. Example of RLL circuit to produce a warning.

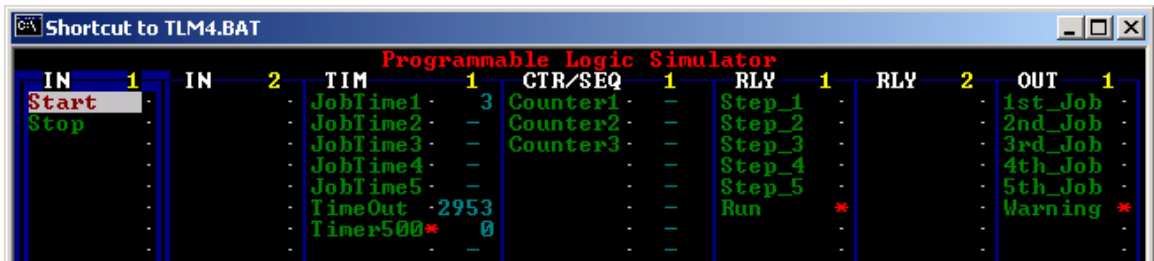


Figure 9. Simulation of circuit producing warning signal.

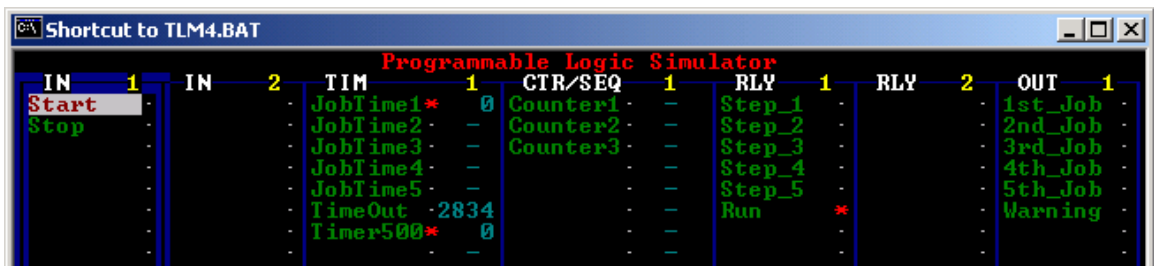


Figure 10. Same circuit after producing warning signal.

And just like the *Run* relay holds its state after the *Start* button is released, demonstrated as the *self-latching* circuit in the previous chapter, we could set our *Warning* signal as a relay, to hold its own *Warning* signal continuously, as set on Figure 11 and simulated on Figure 12.

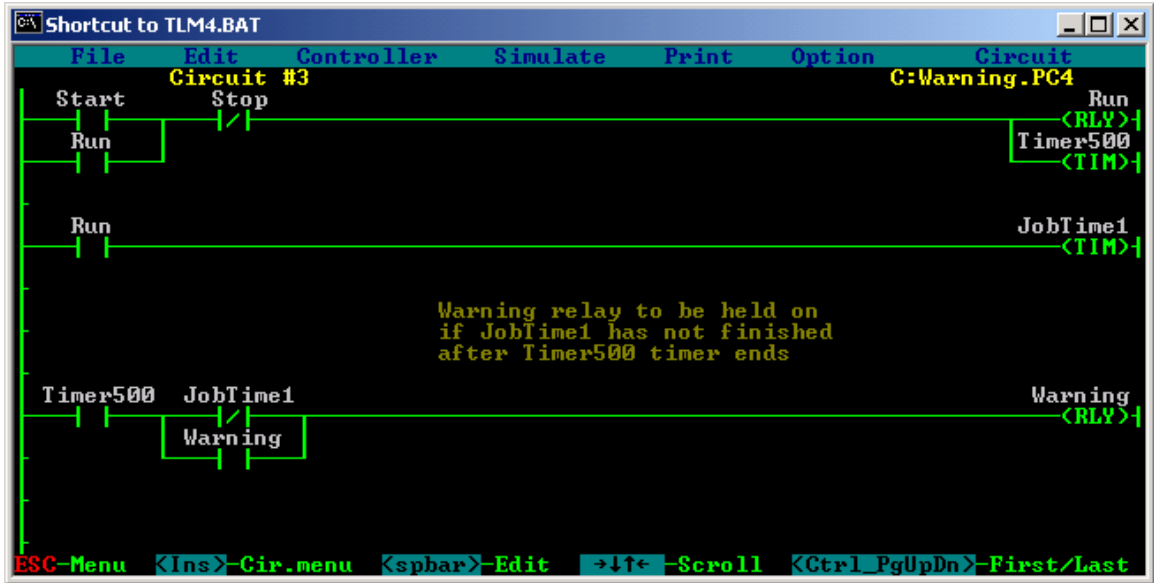


Figure 11. Example of circuit with a continuous warning.

The screenshot shows a "Programmable Logic Simulator" window with a table of simulation data. The table has columns for IN, TIM, CTR/SEQ, RLY, and OUT. The data is as follows:

IN	TIM	CTR/SEQ	RLY	OUT
Start	JobTime1*	Counter1	Step_1	1st_Job
Stop	JobTime2	Counter2	Step_2	2nd_Job
	JobTime3	Counter3	Step_3	3rd_Job
	JobTime4		Step_4	4th_Job
	JobTime5		Step_5	5th_Job
	TimeOut		Run *	
	Timer500*		Warning *	

Figure 12. Simulation of circuit with continuous warning.

Workflow primitives.

The usefulness of this Simulator is demonstrated setting the 6 Workflow primitives identified by the Workflow Management Coalition (WFMC) (Wakayama, Kannapan, K., Khoong, Navathe, Yates & Kannapan S., 1998, pp 170-171) and here replicated.

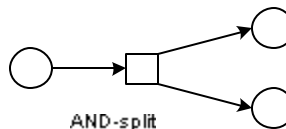


Figure 13. AND-split Workflow Primitive

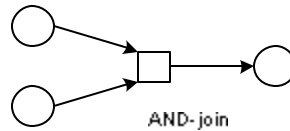


Figure 14. AND-join Workflow Primitive

First, the AND-split mapped on Figure 13, starts two jobs at once on the second row of Figure 15, then with the AND-join mapped on Figure 14, a third Job waits until its two predecessors are finished at the end of Figure 15.

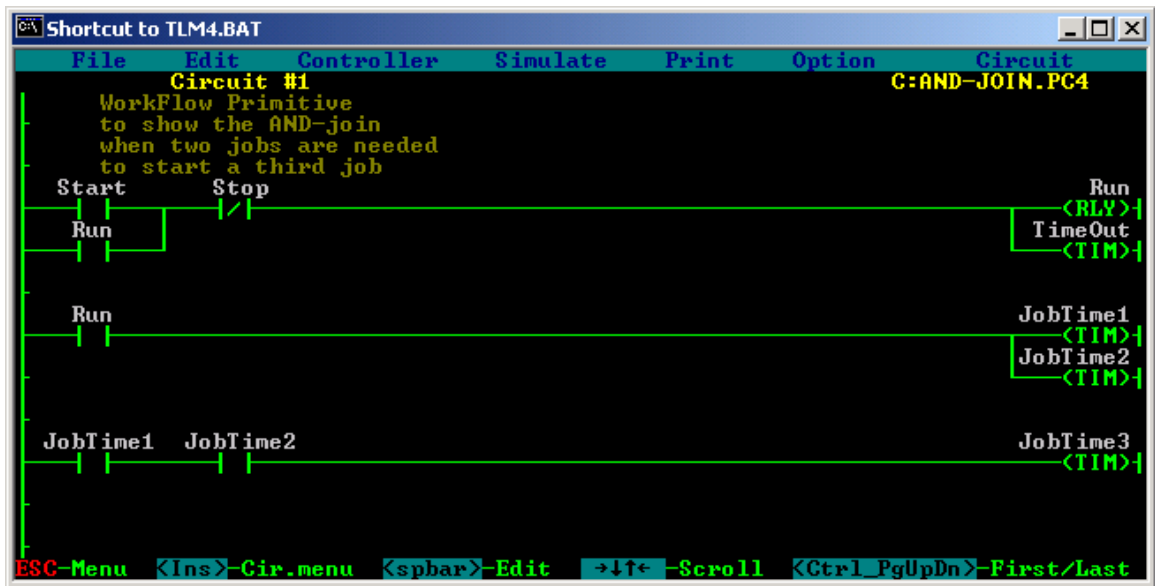


Figure 15. Circuit with AND-split and AND-join workflows.

Running the simulation as shown on Figure 16, we can see how the first two jobs (*JobTime1* and *JobTime2*) have to end, before starting the third job named *JobTime3*.

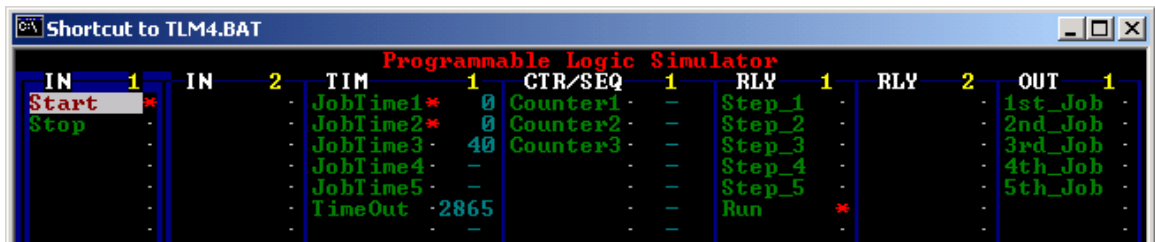


Figure 16. Simulation of AND-split and AND-join circuits.

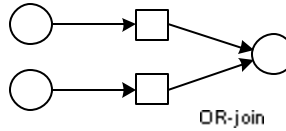


Figure 17. OR-join Workflow Primitive.

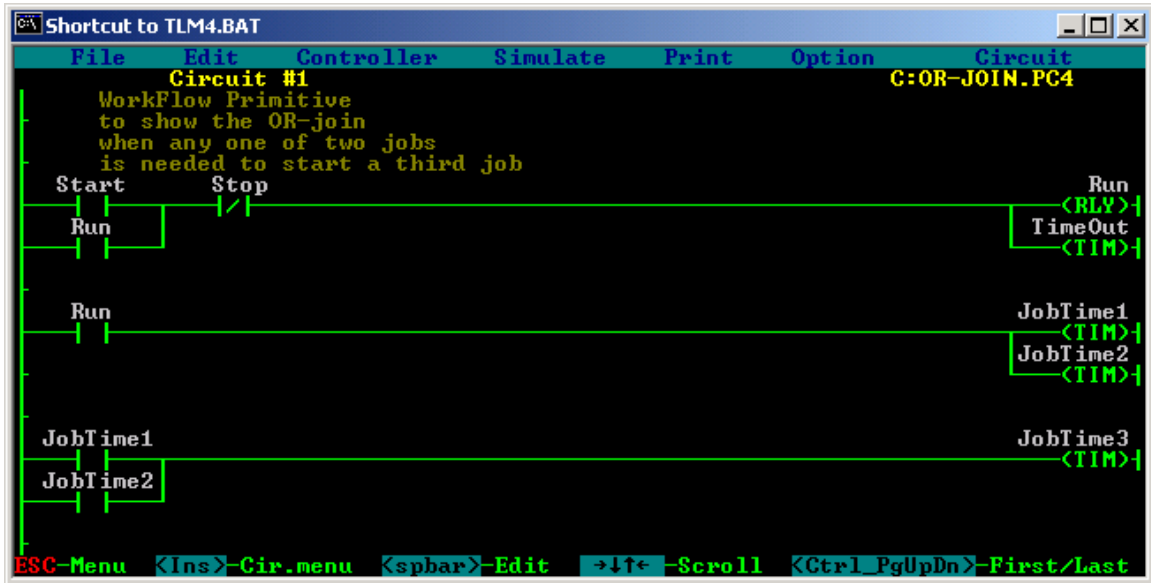


Figure 18. Circuit with an OR-join workflow.

With the OR-join mapped on Figure 17, set on Figure 18 and simulated on Figure 19, as soon as any of the two jobs (*JobTime1* or *JobTime2*) in parallel ends, the follow up job (*JobTime3*) starts.

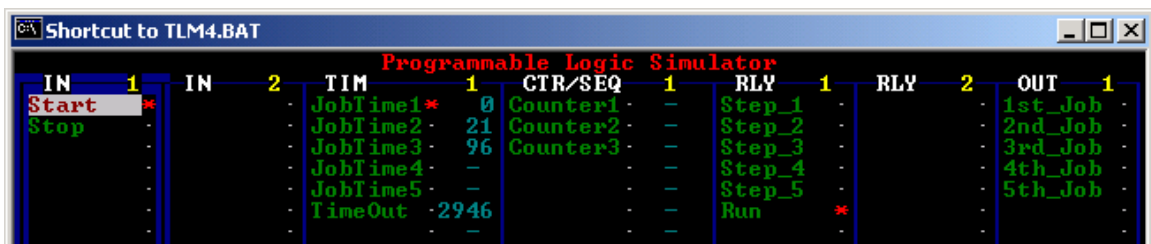


Figure 19. Simulation of an OR-join circuit.

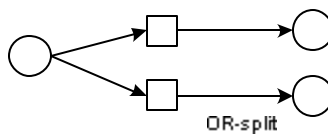


Figure 20. OR-split Workflow Primitive.

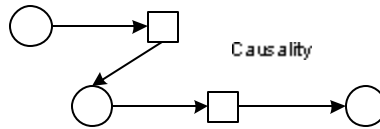


Figure 21. Casualty Workflow Primitive.

The OR-split mapped on Figure 20 works like the AND-split, with each of the two started jobs starting an independent sequence of events, and the Causality workflow primitive mapped on Figure 21 is only one sequence of events where a job has to finish to start the next.

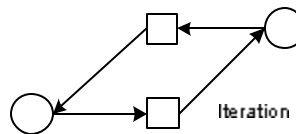


Figure 22. Iteration Workflow Primitive.

An iteration mapped on Figure 22 is modeled on Figure 23 and simulated on Figure 24, adding a feedback transition. In this setting, the three jobs are processed in sequence and restarted when the last job (*JobTime3*) ends; a counter shows how many times the cycle has been repeated.

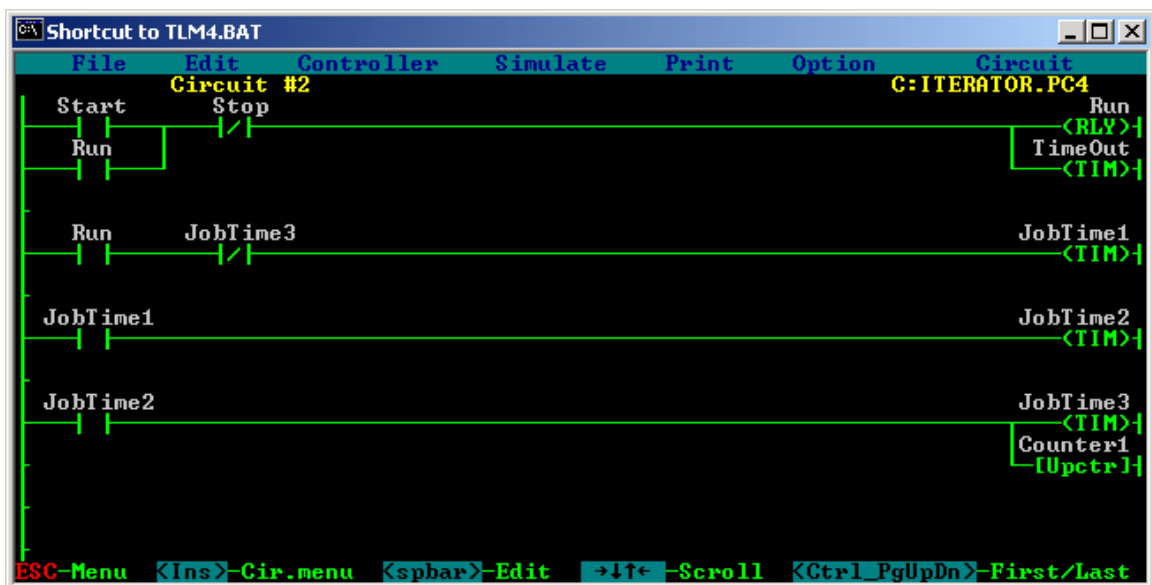


Figure 23. Circuit with an Iteration workflow.

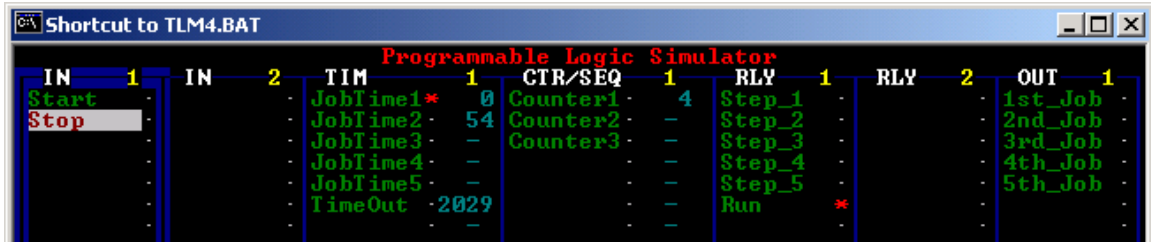


Figure 24. Simulating circuit with an Iterative workflow.

Findings

While building this simulations, other programmable options of the simulator were found (like custom functions and variables, flags, clocks and sequencers) that could represent or not other software components or conditions; also, the ability to introduce up to twelve consecutive and ten parallel components and four lines of comments on each step or row, provides adequate flexibility in the formulation of alternatives.

The IEC 61131-3 Standard is being revised and enhanced to eliminate the inconsistencies and contradictions detected with practical experience (PLCopen, 2001). The United States National Committee (USNC) represents the IEC in the United States, with the administrative support of the American National Standards Institute (ANSI) (American National Standards Institute [ANSI], 2002).

Summary

PLCs not only provides a way to replicate the conditions in which the components of a batch process collaborate, but also opens the possibility of automating the control or monitoring of routine software processes via the signals interaction with hardware components; in other words, an interpreter could send variable signals to a hard wired PLC, to communicate when each job finishes or encounter any problem; likewise, the PLC could tell the interpreter program when to start the next job, or turn on flashing lights or an audible alarm to alert the Computer Operators for immediate action.

The same way we could go from an isolated simulation to simple hardware relationships, we have gone from a practical or applicable understanding of the simulator and its components in the previous chapter, to how we could represent temporary or continuous warning signals, or any of the six workflow primitives identified by the WFMC (Aalst, n.d.).

Conclusions

Decomposing a system to understand each of its components and how they relate to each other, or what each component produce to (or require from) other systems, is undeniable important to troubleshoot failures; that is how the responsible component may be identified. A detailed documentation of each component and its relationships or dependencies, facilitates our understanding, or the effort of other developers that may be working with the same or related systems in the future; producing or updating that documentation is equally important but frequently postponed or forgotten, for unpractical and time consuming.

Producing a quick simulation of the actual system components, with a free simulator built on a proven standard, gives us the options to save the simulation as documentation, or to delegate its formal documentation to a less experienced developer; sharing the responsibility with a junior developer, also provides the much needed in-house practical training, more if the resulting formal documentation (which could include a formal or final version of the simulation) is reviewed for accuracy and completeness.

Conclusions

The simulator here demonstrated produces a logic representation of a system, highly useful to replicate actual or possible batch processes. As a quick to learn and implement graphical control alternative, this technique provides a clear way to communicate sequence of events, requirements, warning sources and other details, between developers and the processing or maintenance teams. If consistently applied, it may also help us reach agreements with our customers.

Industrial simulators are capable of hardware interaction, receiving or stopping or delivering signals, to sense and react as indicated by the settings of the simulation; so we could install a PLC, connect the components and transform the simulator in the controller, which could increase the possible functions available; connecting the components may be challenging, and that is why this study was limited to the simulation and its standard.

The four programming languages defined by the IEC 61131-3 Standard are two textual and two graphical...

- Textual: * Instruction List (IL) * Structured Text (ST)
- Graphical: * Ladder Diagram (LD) * Function Block Diagram (FBD)

only one of the four, the Ladder Diagram programming language is explored in the PLC simulator, which simplifies learning and training. “Ladder Diagram has its roots in the USA. It is based on the graphical presentation of Relay Ladder Logic.” (PLCopen, 2001, Intro section).

Sequential Function Charts (SFC) is one of the common elements of the Standard, that “describes graphically the sequential behavior of a control program. It is derived from Petri Nets and IEC 848 Grafcet, with the changes necessary to convert the representation from a documentation standard to a set of execution control elements. SFC structures the internal organization of a program, and helps to decompose a control problem into manageable parts, while maintaining the overview.” (PLCopen, 2001, Intro section).

Because of its general structure, SFC provides also a communication tool, combining people of different backgrounds, departments or countries. Like the three

other programming languages not included in the Trilogi PLC simulator, users not need to be aware of the details of the Standard to enjoy the user-friendly interface, quick learning curve, among other benefits.

Implications

All assumptions must be identified and documented to understand the difference between any simulation and reality, for example...

- All processes should finish successfully; to approximate reality, the software components should be flagged as critical or optional and failures could also be simulated to determine the implications.
- There is no significant delay between processes, so we should compare the time it takes a batch in the simulator against the time it takes in real life; all significant delays found should be explained.
- Computer Operators monitor each batch process, to determine if something goes wrong at any time.
- The System may continue working normally after a failed batch, followed by a System Restore, unless the updates are required for normal processing.
- If a batch process fails, the failure is part of the batch. On the other hand, changes to the system since the last similar batch could also be causing the failure.

Recommendations

An interesting challenge would be the identification of programs, functions or other software components, to determine their equivalence to the other components found in the simulator like Sequencers, Clocks, Functions, Flags and Variables.

This short study may also be presented to the PLC producers community, to suggest the possibility of creating a version with symbols related to Software control instead of the usual industrial components.

Summary

The simulation studied in this report, while developed for industrial control devices, may become a valuable communication resource, if and when adapted to our specific needs. The small effort required is easily outweighed by the benefits here explained.

By making sure the IEC 61131-3 Standard is implemented in the program development environment, like in the case of the Trilogi PLC, “users can move between different brands and types of control with very little training and exchange applications with a minimum of effort.” (PLCopen, 2001, Intro section).

Reference List

- Aalst, W.M.P. van der. (n.d.). *Petri-net-based Workflow Management Software*.
Department of Mathematics and Computing Science, Eindhoven University of
Technology. Retrieved August 1, 2003, from
<http://lstdis.cs.uga.edu/activities/NSF-workflow/wfm.html>
- American National Standards Institute. (2002). *United States National Committee of
International Electrotechnical Commission (USNC/IEC)*. Retrieved August 1,
2003, from
http://www.ansi.org/standards_activities/iec_programs/overview.aspx
- Applegate, L., McFarlan, W., & McKenney, J. (1999). *Corporate Information Systems
Management. Text and Cases. (5th ed.)*. Boston, MA: Irwin, McGraw-Hill.
- Gall, H., Jazayeri, M., & Klosch, R. (1995). *Research Directions in Software Reuse:
Where to go from here?*. Vienna University of Technology, Italy. (ACM 0-
89791-739-1/95/0004)
- Henderson, P. (2002). *Systems Engineering for Business Process Change*. London,
England: Springer.
- International Control Components. (1999). *FREE Downloading of TRiLOGI (Evaluation,
420K)*. Retrieved April 9, 2003, from
<http://www.icc-gb.com/triangle/trilogi.htm>
- Leach, R. J. (2000). *Introduction to Software Engineering*. Boca Raton, FL: CRC Press.
- Pfleeger, S. (2001). *Software Engineering – Theory and Practice (2nd. Edition)*. Upper
Saddle River, NJ: Prentice Hall.

PLCopen. (2001). *PLCopen – Standardization in Industrial Control Programming*.

Retrieved April 9, 2003, from <http://www.plcopen.org/>

Triangle Research International. (1999). *Trilogy Programmer's Reference*. Retrieved

April 9, 2003, from <http://www.icc-gb.com/triangle/trilogi.htm>

Valenti, S. (2002). *Successful Software Reengineering*. Hershey PA: IRM Press.

Wakayama, T., Kannapan, K., Khoong, Ch. M., Navathe, S., Yates, J., & Kannapan, S.

(1998). *Information and Process Integration in Enterprises, Rethinking*

Documents. Norwell, Massachusetts: Kluwer Academic Publishers.

Table of Content

Abstract2

Introduction.....3

Problem Investigated.....3

Need for the Study.....5

Main Relevance6

Goal or Objective6

Elements Investigated.....7

Approach.....8

Limitations of the Study8

Summary8

Review of the Literature10

Overview.....10

Research Literature10

Contributions to the Field.....11

Resources Available.....12

Conclusion12

Review of Methodology13

Our Scope in Perspective.....13

Understanding the Simulator.....14

Understanding the Components14

Results.....18

Analysis18

Warnings and holders19

Workflow primitives.....21

Findings25

Summary25

Conclusions27

Conclusions.....27

Implications29

Recommendations29

Summary30

Reference List.....31

Table of Content33

Table of Figures34

List of Acronyms35

Table of Figures

Figure 1. Batch Processing.....4
 Figure 2. Merging two batches with common components.....6
 Figure 3. TRILOGI software presentation screen.....7
 Figure 4. Relay Ladder Logic circuit example.....14
 Figure 5. Basic components on Self-latching circuit.....15
 Figure 6. Example of circuit to count cycles.16
 Figure 7. Simulation of circuit to count cycles.....17
 Figure 8. Example of RLL circuit to produce a warning.....20
 Figure 9. Simulation of circuit producing warning signal...20
 Figure 10. Same circuit after producing warning signal.20
 Figure 11. Example of circuit with a continuous warning.....21
 Figure 12. Simulation of circuit with continuous warning...21
 Figure 13. AND-split Workflow Primitive.....21
 Figure 14. AND-join Workflow Primitive.....22
 Figure 15. Circuit with AND-split and AND-join workflows...22
 Figure 16. Simulation of AND-split and AND-join circuits...22
 Figure 17. OR-join Workflow Primitive.....23
 Figure 18. Circuit with an OR-join workflow.....23
 Figure 19. Simulation of an OR-join circuit.....23
 Figure 20. OR-split Workflow Primitive.....23
 Figure 21. Casualty Workflow Primitive.....24
 Figure 22. Iteration Workflow Primitive.24
 Figure 23. Circuit with an Iteration workflow.....24
 Figure 24. Simulating circuit with an Iterative workflow...25

List of Acronyms

American National Standards Institute (ANSI)25
 Function Block Diagram (FBD)28
 Institute of Electrical and Electronic Engineers (IEEE)12
 Instruction List (IL)28
 International Control Components [ICC]8
 International Electrotechnical Commission (IEC).....7
 Ladder Diagram (LD)28
 Programmable Logic Controller (PLC)2
 Relay (RLY)15
 Relay Ladder Logic (RLL) programming7
 School Boar of Broward County in Florida (SBBC).....12
 Sequential Function Charts (SFC).....28
 Structured Text (ST)28
 Timer (TIM).....15
 Triangle Research International [TRi]12
 United States National Committee (USNC).....25
 Workflow Management Coalition (WFMC)21