2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

# A Comparison of Sequential Function Chart and Object-Modelling PLC Programming

Vivek Hajarnavis and Ken Young

*Abstract*—**This paper analyses the use of two control software design methodologies for developing a solution to control an automated manufacturing cell. Two separate approaches, Sequential Function Charts and an object-modelling tool called Enterprise Controls are compared and contrasted with a view to establishing the effectiveness of each approach. These experiences are illustrated by comparing software to operate a simple clamp. Enterprise Controls provided a simpler solution, which reduced both the time taken for implementation of process changes as well as the need for these to be carried out by trained personnel. The experience showed that the object modelling approach provides benefits to manufacturers by aiding manufacturing flexibility, though this comes with a processor overhead. Concerns remain as to whether or not Enterprise Controls will be accepted by practicing control engineers in industry. Further work to quantify these benefits is planned.**

## I. Introduction

Automated production lines in the car industry rely extensively on computer control in order to achieve the required manufacturing function. Typically, a computer called a programmable logic controller (PLC) is used to implement the algorithm required to control this machinery. Usually, this algorithm is digital logic and has historically been achieved using a programming language called ladder logic. Ladder logic is a graphical language, and a method of expressing the "if…then" construct used extensively in structured programming methods in mainstream programming of information systems. The background behind this language is based on its users, who were historically plant electricians with responsibility for maintaining the hard-wired control systems that had been in existence prior to the advent of the PLC.

The advantage of well-structured ladder logic is that programs are easy to debug. A maintenance technician can look at a ladder program and immediately identify whether the status of an individual input or output bit is correct. Consequently, manufacturers in the car industry have been reluctant to adopt alternative programming techniques.

Manuscript received August 31, 2004, revised February 14, 2005. This work was supported by the EPSRC and Rockwell Automation.

Vivek Hajarnavis is a Research Engineer at the Warwick Manufacturing Group, University of Warwick, Coventry, CV4 7AL England (E-mail: V.Hajarnavis@Warwick.ac.uk)

Ken Young is a Reader at the Warwick Manufacturing Group, University of Warwick, Coventry, CV4 7AL England (Phone: +44 24 76522764, Fax: +44 24 76573743, E-mail: K.W.Young@Warwick.ac.uk)

In recent years, a desire for standardisation of programming approaches across PLC's supplied by different manufacturers has led to the creation of the IEC61131-3 standard. This defines five languages – ladder logic, sequential function charts (SFC), statement list (STL), function blocks and structured text. SFC was derived from Grafcet, a graphical language based on a French national standard and itself an evolution of a Petri-net. The main advantage of an SFC is that it allows visualisation of the main states in a system together with all possible changes in state and the reasons why these changes could occur [1].

More recently, a desire by manufacturers to offer a wider range of products has placed greater emphasis on the ability to modify the machines carrying out the manufacturing process. This, coupled with the availability of more powerful processors has opened up the possibility of alternative programming techniques. Two examples of new methods of PLC programming are object-oriented software engineering and automatic generation of code. These new approaches aim to increase the ease with which a process can be changed, suggesting that there is scope for identifying whether a particular logic design methodology or approach is suitable for use by a manufacturer wishing to implement quick rapid process changes. This paper compares and contrasts the experience of using one of these new tools with SFC, one of the languages defined by the IEC61131-3 standard. This draws on the experiences gained from programming a sequence of operations within a test cell at the University of Warwick. Differences are illustrated further by describing the code operating a simple clamping operation.

## II. Logic Design

Previous work on logic design methodologies has centred on trying to estimate the time and effort required to create the software needed to operate a piece of equipment correctly, comparing ladder logic with academic tools such as Petri-nets and modular finite state machines. Tests have suggested that programs written with implementations of Petri-nets are easier to modify than those that are based on structured ladder code [2]. However, these results are based on implementations of Petri-nets using traditional ladder programming and may not necessarily be appropriate for use in industry as it would undermine the debugging methods used by designers at present [3]. This is owing to the requirement for latching logic, a situation in which outputs are turned on and off in different locations. Conversely, ensuring that each output is triggered on a single rung aids understandability of the code.

An alternative approach is the implementation using a SFC, as this possesses many of the characteristics of a Petri net [4]. In view of the origin of the SFC language, this result is not surprising, though the experiences suggest that there is a difference in performance, suitability and ease of use of the five programming languages defined in the IEC 61131-3 standard.

The development of software design techniques for use in PLC's has taken place independently of software engineering techniques in the commercial environment because of two main differences. Firstly, software for industrial use requires good quality diagnostics in order to maintain production in the event of a plant fault. Secondly, there is often a need to allow users to modify plant functionality quickly and accurately. All the same, technology transfer from one environment to the other may help overcome some of the difficulties encountered by PLC programmers [5].

One area in which traditional software engineering techniques are becoming more visible is in the area of formalisation of programming tools. This can serve various purposes, such as verification and validation and suggests that like traditional industrial programming techniques, there are various alternative approaches for developing a solution for the same program using formalised design techniques [6]. Furthermore, formalisation techniques can stimulate the development of translation tools, such as mechanisms to express programs written in statement list in the Extensible Markup Language (XML) [7]. Given that formalisation could be done in different ways, identification of a formalised approach most suitable for use in the automotive manufacturing environment would be an interesting exercise.

A new idea which is starting to come into use is the concept of offline verification of control software, making use of existing information to generate automatically the PLC code required to operate a process [8]. As part of this process, there would be benefits gained from integrating diagnostic capability into the offline models. Inclusion of diagnostic capability within the model means that any system changes lead to a corresponding modification within the Human Machine Interface (HMI). This ensures that control code and diagnostic mechanism are synchronised, eliminating the need for programmers to look at code for debugging purposes. A move away from reliance on ladder for error correction could make ladder-based implementation of Petri-nets or finite state machines more acceptable, especially if the actual implementation is hidden from the user.

Offline programming tools also present the opportunity for adopting an object-oriented approach to developing control programs with their modular approach allowing re-use of existing work. There are instances of successful implementations of object-orientation for industrial control. One example is an object-oriented front end for a robotic manufacturing cell, which introduced a level of abstraction and allowed operators rather than specialists to program machinery [9]. Another example describes the implementation of a control system to operate the under-body welding line at VW Portugal [10]. This work again refers to a reduction in development time rather than describing process change time. The programming tool described in this paper, a commercial application called Enterprise Controls (EC) makes use of a similar approach and software structure to that described in these examples.

## III. FACT CELL

The FACT (Future Automation Control Technology) cell (Figure 1) is a test environment for new automation techniques. The cell is controlled using a number of Allen-Bradley ControlLogix PLC's, all of which have network connections to Ethernet, ControlNet and DeviceNet. These three networks are used primarily for programming, peer-to-peer communication between PLC's and data links to and from hardware respectively, though some communication with control hardware does take place over each network. A separate Pilz PLC provides safety functions, communicating with remote devices over SafetyBusP.

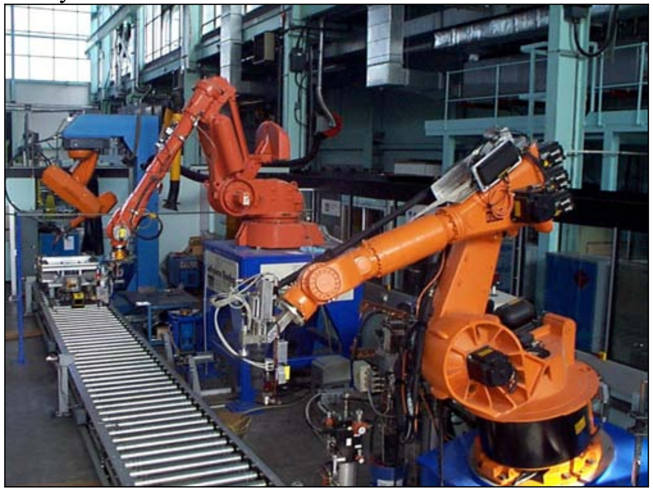


**Figure 1: FACT Cell**

This facility contains three robots and a conveyor system, on which a tooling pallet assembly is mounted. The pallet can be moved to a station in front of each robot, where it is clamped in place using a pneumatically operated locking pin and a Stäubli quick-fit connector which provides compressed air, DeviceNet and SafetyBusP connections to the tooling pallet. The connections allow twelve clamps used for holding space-frame components on the pallet to be operated. The cell provides an example of many of the operations typically used in a Body-in-White facility in a car factory, and allows different sequences of operations to be programmed and tested. Similar equipment in each cell, as well as multiple clamps makes this facility particularly suited for demonstrating reuse of control code.

## IV. Comparison of Programming Techniques

### *A. Sequential Function Charts*

SFC's consist of step and transition pairs, as can be seen in Figure 2. Steps are depicted by rectangles and transitions by horizontal lines. A SFC step corresponds to a place in a Petri-net, and represents process status. The difference between a SFC and a Petri-net is the ability to assign a control function to a SFC step; a Petri-net place only provides an indication of whether or not a state is active based on the presence or absence of a token. Petri-net and SFC transitions are alike in that they represent the occurrence of events. In this sense, Petri-nets can be seen as the method by which SFC's operate rather than being a programming language in their own right.

The functionality of SFC steps and transitions is achieved by associating code written in one of the other IEC61131-3 languages with each step and transition. The principle is based on carrying out the operation in each step until such time as the state of the transition changes. Figure 2 also shows that SFC's can aid the debugging process. The step highlighted indicates that the program is awaiting logic in one of the transitions below the active step to become true.

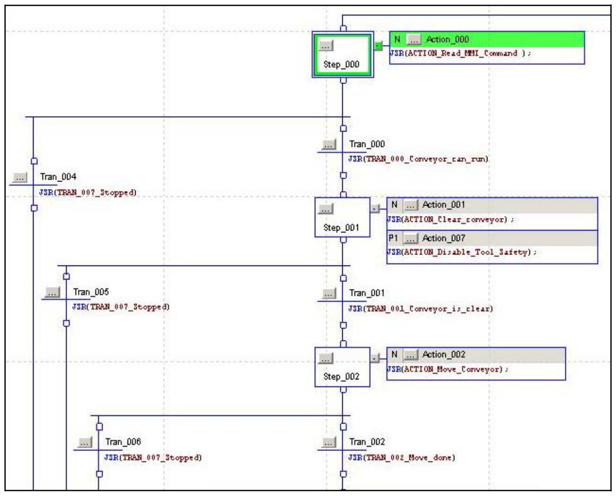
The software to operate the FACT cell was structured such that SFC's provided the overall framework, with small sections of ladder code used to give the functionality. The program was designed such that operations associated with a piece of equipment could be easily identified. This architecture allows the program to be developed such that each component can be operated manually from a HMI or as part of an automatic sequence, consisting of a number of manual operations called consecutively by the code in the PLC. This automatic sequence is based on a set of command words held within the data table of the PLC.

System flexibility was improved by adding an opportunity for a user to modify the steps in the automatic sequence from the HMI screen instead of entering numeric values manually into a PLC data table. This facility was quite limited and only allowed an operator to select from a limited, pre-defined set of operations. Any further modification of the sequence required changes to the SFC code. Similarly, the inclusion of a new operation required extensive work to analyse and perform the function indicated by the operation code. In some cases, it required the creation of a new SFC.

Commissioning revealed that further work was required to aid handling of errors and emergency stop situations. In order to address this, diagnostic features were added to the SFC to provide feedback to an operator as to the status of the machinery, and additional step-transition pairs were added to provide a "bypass" feature. Prior to the inclusion of this code, there were instances when an operator had to trigger a transition manually in order to recover from a fault.

As the program size increased, the length of time taken to transfer the program from the PC to the PLC also increased. This added to the time taken to get the plant operational again after a change.

The overall impression gained of the SFC-based software was that it provided a neat method of developing and maintaining a control system but still required a good understanding of the PLC, hardware and manufacturing system in order to allow effective modification of the cell. Potential for re-use of code was very limited and realistically only proved possible in a parallel branch dealing with the emergency stop condition. A further limitation was that the software structure required the intervention of programmers familiar with the facility and so it did not prove possible to delegate operation and maintenance to technical staff.



```
JSR(ACTION_Read_HMI_Command );
JSR(TRAN_000_Conveyor_can_run)
JSR(TRAN_007_Stopped)
JSR(ACTION_Clear_conveyor);
JSR(ACTION_Disable_Tool_Safety);
JSR(TRAN_007_Stopped)
JSR(TRAN_001_Conveyor_is_clear)
JSR(ACTION_Move_Conveyor);
JSR(TRAN_007_Stopped)
JSR(TRAN_002_Move_done)
```

**Figure 2: Sequential Function Chart**

### *B. Enterprise Controls*

EC is an application running on a standard PC, communicating with a ladder program running in an Allen-Bradley ControlLogix PLC through an OPC server. The PC application allows a user to create software device templates containing the functionality required to operate a piece of equipment. Instances of device templates can be defined and tied to specific input / output (I/O) points on the plant. Each device template contains diagnostic capability and messaging functions to aid fault finding in use. Configuration of the equipment, physical I/O points and machine specific code are defined in a master ladder file. This is the only time in which a programmer might need to make use of a ladder-logic programming tool.

The standard operating principle within EC is the idea of setting a set of outputs (command behaviour) until an end condition (status behaviour) is reached. The device template also allows the definition of signals which would create static errors (such as emergency stop cases) as well as input conditions for creating running messages to provide feedback on the progress of the process.

Definition of the hardware is followed by an automatic code generation process which creates a ladder logic file based on the pre-defined master file. This generated file can be downloaded into the PLC. A sequence of operations can then be created, calling the actions previously created by the programmer. This sequence can then be downloaded into the data tables of the PLC, a step which takes considerably less time than the transfer of a complete ladder program from offline PC to PLC. The code in the PLC can be seen as an interpreter, taking the operations input by the programmer and converting them into the ladder code required to operate the real equipment.

Visually, the screen used to program sequences in Enterprise Controls looks quite similar to an SFC but with one main difference: there are no transitions (Figure 3). Each sequence consists of a set of cells, and operations relating to a device can be called from within a cell. The end condition for that device is defined within its template, and on completion the sequence will automatically drop into the next cell.

The FACT cell control algorithm was implemented by the same programmer who had previously programmed the cell using SFC. The initial observation was that despite limited training, operations in EC could be created with very little effort and did not require a good understanding of the PLC code. To give an example, code to operate a single clamp was implemented within an hour of first using the software, and extended to controlling a sequence of twelve clamps a few minutes later. Creation of code to perform the same operation in SFC took about a day with replication required in order to operate each of the twelve clamps in turn. In [3] it is stated that expertise plays a part in the development of control logic, and in order to account for the programmer's experience when writing the EC code, the software was subsequently re-written from scratch in SFC and EC. Creation of a sequence with comparable functionality to open and close two clamps in order took approximately three times longer with SFC than was the case with EC, showing that the more abstract approach provided by EC gives a considerable time saving

Development of the device templates to create more sophisticated devices such as robot interfaces took considerably more effort and required an understanding of the core functionality of the EC ladder program. However, the advantage of this was that it only needed to be done once. Following completion of code, creation of diagnostic messages and verification that the operation was correct, there was no further need to look at the underlying ladder code and the template could be reused for other devices simply by defining a new instance with different I/O points. Within the FACT cell, this meant that the same template could be used to operate six distinct locking pins, each with different input and output points.

As well as the time saving noted, a further benefit observed as a result of using EC was that it no longer required a skilled operator to start, maintain and modify the machine. The method of presenting a sequence as a set of operations relating to a machine meant that a technician with an understanding of the mechanical process could make changes to the operation sequence without the need for worrying about making changes to the underlying code. Adding additional steps took little additional effort, and transfer of the new sequence to the PLC typically took less than a minute. The in-built diagnostics provided more feedback than had been possible to implement in the SFC software, eliminating the need for understanding and debugging the underlying ladder code. This would be of great benefit to an industrial user who has to implement an engineering change at short notice, not least because this could be done ensuring that diagnostic messaging and appropriate interlocks are modified automatically.

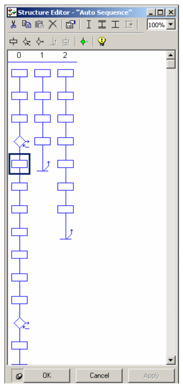


**Figure 3: EC Structure Editor**

## V. Example: Simple Clamp

A further insight into the differences between the two programming techniques can be seen by considering the operation of the simplest device in the FACT cell: one of the twelve pneumatically operated clamps (Figure 4). Each clamp is operated by signalling one of two outputs which in turn open valves to open or close the clamp as appropriate. Two sensors fitted to the clamp provide an indication as to whether or not a clamp has opened or closed properly.

The differences between the interfaces presented to the programmer by the two programming tools can be seen by comparing Figure 5 and Figure 6. The principal variation is the approach: the SFC shows a breakdown of the process, complete with diagnostic functionality whereas EC provides a set of operations associated with a device.

Whilst the code to operate a single clamp using an SFC-based approach was comparatively simple, creating a sequence was more challenging. Initially, code to operate all twelve clamps in a sequential manner was created on a single SFC screen. The limitations of this approach became apparent when the order of operation needed to be changed, as it required an extensive redesign of the SFC, a process which took at least an hour. Subsequently, separate SFC's to operate each clamp were created with sequence data held in the data table controlling the automatic sequence. Once configured, this allowed limited sequence changes to be implemented within a few minutes. A third option made the use of a single SFC with user-defined data types, or structures, to allow operation of twelve distinct clamps from a single SFC.

Conversely, the more abstract approach followed by EC made the implementation of sequence changes significantly easier as it presented a programmer with the concept of a mechanical operation (open or close clamp). This in turn meant that major sequence changes could be implemented and committed to the cell within a few minutes. Additional operations could be added simply by adding new cells to hold the new functions. Increasing the length and complexity of sequences did not have a significant effect on the length of time taken to download the operation to the PLC Data table. Given diagnostic messaging was already included within the device template, no additional effort was required by the user and it did not prove necessary to investigate the underlying PLC code.

Performance of the system was compared by taking measurements of the processor scan time when each program was running. The EC program in the PLC is structured in two parts, one part consisting of the operating system and the other section specific to the system to be controlled. The operating system was found to have a stable scan time, taking approximately 7 milliseconds to complete a cycle whereas that of the system-specific part varied according to the number of devices controlled. A simple sequence to open and close two clamps in turn gave a scan time of 3 milliseconds. In comparison, SFC code to provide the same control function was found to have a scan time approximately 20 times faster than that of the EC variant. The SFC software was found to use 5% of the memory required to run EC.

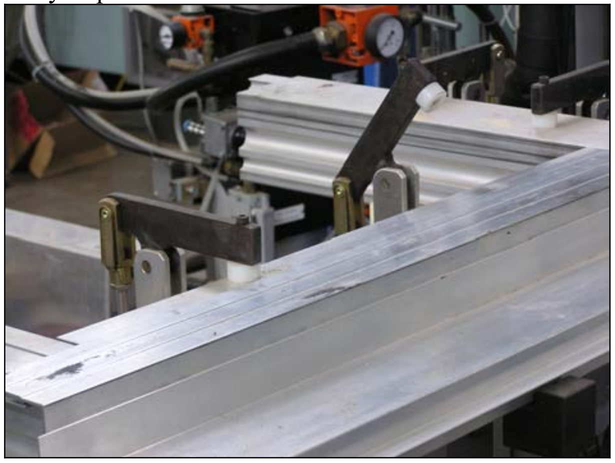


Figure 4: Clamps

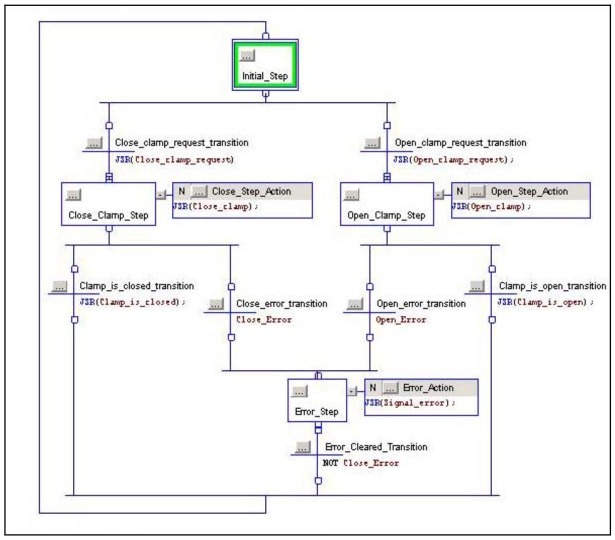


Figure 5: Clamp Operation in SFC

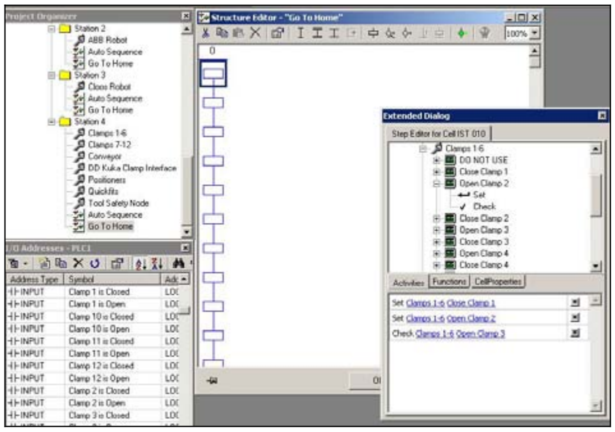


Figure 6: Clamp Operation in Enterprise Controls

## VI. Survey

Following a seminar to disseminate the findings of this research, the delegates, all control engineers in industry were asked a set of questions asking about their perception of this type of product. All were positive about the concept with some expressing concern about the inability to use these tools with legacy equipment as well as the commitment to hardware manufactured by a single supplier. Further concerns were raised as to whether the underlying PLC code was sufficiently reliable, as was the issue of taking responsibility for any problems arising with this code. Other end-users stressed the need for accurate diagnostics within the devices as the issue of incorrect or incomplete device templates would seriously hinder operation.

The main benefits of interest to the delegates were the standardisation of control code, the ease of programming and commissioning, the mechanism for ensuring that the PLC and HMI link together, the transparency of the approach and the speed of determining the root cause of intermittent faults. All but the last factor are points which are essential if a control program needs to be modified by anyone other than the original programmer. End-users state that implementing a change in a PLC without a corresponding modification within the HMI is a common problem, leading to incorrect diagnostic messages. Thus, integrating the diagnostic features into the functionality of the device will overcome this problem. Acceptance by system programmers that this "black box" approach will still provide sufficient diagnostics was highlighted as a potential difficulty.

## VII. Conclusions

Contrasting two very different approaches to developing a control system has shown that there is a considerable benefit to selecting a methodology appropriate for the situation. This work has shown that using design automation tools provides advantages for a manufacturer wishing to adopt a flexible manufacturing process in terms of reduced development time and the ease of incorporating diagnostic capability into the process, as well as reducing the reliance on skilled programmers for implementing simple process changes. There is potential for testing the level of abstraction of a tool with a view to capturing the level of complexity of a software module that provides process flexibility without affecting diagnostic ability. The next step in this area will be to design metrics to allow measurement and quantification of each programming technique in order to identify the benefits and pitfalls of moving from programming with traditional IEC61131-3 languages to a more modular "black box" approach.

The advantage of additional flexibility and ease of use comes at the cost of a processor overhead in both scan time and memory usage. The values obtained suggest that this overhead should not be a hindrance in most situations, although applications such as high-speed machinery which requires very fast response times would find EC unsuitable.

Presenting this work to end users has been generally favourable, though the reaction has stressed the need for good, rigorous design to ensure that device templates have been sufficiently tested and validated for use in real production environments.

There is scope for looking at the level of training and skill required of an operator or programmer in order to achieve a process modification using a particular programming tool. Undertaking a program of tests over a large sample size will be required to validate this work.

## VIII. Acknowledgment

The authors would like to thank Ray Daniels and Phill Hannam of Rockwell Automation for their support and guidance during this work.

## IX. References

[1] Lewis, R.W.; Programming industrial control systems using IEC 1131-3, Revised edition, The Institute of Electrical Engineers, 1998, pp185-186

[2] Lucas, M.R. and Tilbury, D.M.; Quantitative and qualitative comparisons of PLC Programs for a small testbed with a focus on human issues; Proceedings of the American Control Conference, Anchorage, Alaska, May 8-10 2002, Vol. 5 pp4165-4171

[3] Lucas, M.R. and Tilbury, D.M.; A study of current logic design practices in the automotive manufacturing industry, Int. J. Human-Computer Studies 59 (2003) pp725-753

[4] Carpanzano, E, Cataldo, A, and Tilbury, D; Structured design of reconfigurable logic control functions through sequential function charts, Proceedings of the American Control Conference 2004, Boston, Massachusetts, June 30- July 2 2004, pp4467-4471

[5] Edan, Y and Pliskin, N; Transfer of software engineering tools from information systems to production systems, Computers & Industrial Engineering 39 (2001) pp19-34

[6] Frey, G; Formal methods in PLC control at a flexible manufacturing line, Proceedings of the 5th IFIP International Conference for Balanced Automation Systems in Manufacturing and Services, Cancun, Mexico, September 2002, pp501-508

[7] Bani Younis, M and Frey, G; Visualization of PLC programs using XML, Proceedings of the 2004 American Control Conference, Boston, Massachusetts, June 30- July 2 2004, pp3082-3087

[8] Richardsson, J and Fabian, M; Automatic generation of PLC programs for control of flexible manufacturing cells, Proceedings of the IEEE Conference on Emerging Technologies & Factory Automation, Lisbon, Portugal, Sept 16-19 2003, pp337-344

[9] Norberto Pires, J and Sá da Costa, J.M.G.; Object-oriented and distributed approach for programming robotic manufacturing cells, Robotics and Computer Integrated Manufacturing Volume 16, Issue 1, 2000, pp29-42

[10] Flores, L and Barata, J; Object oriented software engineering for programmable logic controllers, A successful implementation, Proceedings of the IEEE Conference on Emerging Technologies & Factory Automation, Lisbon, Portugal, Sept 16-19 2003, pp116-120