A Programmable Logic Controller under RTLinux Following Related Standards

I. Plaza, C. Medrano and C. Catalán
Escuela Universitaria Politécnica, Universidad de Zaragoza
C/ Ciudad Escolar s/n, Teruel 4403, Spain
{iplaza,ctmedra,ccatalan}@unizar.es

Abstract

In the present paper we show the guidelines of a project to run a PLC (Programmable Logic Controller) under RTLinux. The first version we present, called boolean PLC, includes a translator for the instructions list PLC programming language, following a subset of the IEC 61131-3 standard. The user gives information about the hardware configuration (such as I/O cards), the software configuration (number of memory bits, and functional blocks) and the task characteristics (period, instruction list related programs). Each task (as defined in IEC 61131-3) is mapped to a thread and they can have access to common objects (I/O images, memory bits, functional block parameters) by means of shared memory. For the whole project, we have adapted the Spiral Model describing development as an iterative six-phase process. The system is being developed according to the ideals of Quality, in agreement with applicable standards (IEC 61131, ISO 9001, ISO/IEC 12207, ISO/IEC9126 and ISO/IEC14598), and adapting them to our small university R&D&I group.

1 Introduction and motivation

The original idea behind this project raised from a collaboration between our university R&D&I group and a machine tool manufacturer company in our surroundings. In a first step, we designed some data acquisition hardware for them. Afterwards, the idea of improving their control systems arose. Four key terms were involved: Programmable Logic Controllers (PLCs), PC platform, Real-Time variation of Linux, and the ideals of Quality.

Programmable Logic Controllers are digital systems designed for automation and control. They are widely used in industry, based on proprietary solutions in the market [1][2].

There is also an increasing use of PC in industrial applications [3]. This fact can be due to the decreasing of the prices, the increase of performance, the possibility to integrate it in the company information systems and the appearance of control specific platforms (like PC/104) [4].

Another fact to be considered is the appearance of the Linux operating system and its real time variations. In this paper we will work with RTLinux [5]. It is stable, reliable and open source, thus facilitating the development of self-made applications.

Therefore, a protoype PLC is being implemented as a way to obtain the know-how in this field, to realise the possibilities of RTLinux and to show its true usefulness in industrial applications. In addition, we try to perform the work according to the international standards for the product and for the process development, in agreement with the ideals of Quality.

As differences between our project and MatPLC [6][7], we can note: i) Our system will perform the classic PLC control loops, while MatPLC can have, in general, different architecture; ii) It is based from the beginning on RTLinux; iii) It looks for the integration of several standards, including the management of the process; iv) Up to now, it is not open source due to the involvment of a company who has not assumed this philosophy.

In this context, we will show the guidelines of an application to run a PLC under RTLinux on a PC and the way we are managing the project. Our first version implements very basic functionalities.

2 Management considerations and related standards

Before we started coding the application, we made a research about the standards that could be applied to the process and the product (a list of standards is shown at the end of the paper). We point out that the application of some of the standards, specially those concerning the process, is not usual in university groups in Spain. We think it is a way to optimize the limited human and economic resources we have at our disposal. In addition, we would like to get continuous improvement, to get closer to industry trends, to obtain better results and to increase the possibilities of future certification of our prototypes-products.

We will mention in this section all the standards we have chosen to apply to our work, even though we will focus on IEC 61131. We can distinguish between the applications of quality concepts in the process, and the quality in the product. In our case the product is a complex system (software + hardware), but the development software phase is the most important part since the hardware (some kind of industrial or embedded PC) will be acquired in the market.

2.1 Process development

For the process management, we have chosen the ISO 9000:2000 family of standards. The reasons are the following: it is applicable to any kind of project (services, hardware, software), it enables us to work with a broad spectrum of Spanish companies (28690 certified firms according to ISO 9000 [8], among them several Machine Tool Manufacturers), and it facilitates the formalities required to obtain the later CE mark.

Thus, we have adopted the management system model of ISO 9001:2000 and ISO 9004:2000 to our small university group, integrating also the common framework for the software life cycle established by the ISO/IEC 12207, because it allow us to define and to implement a quality system that covers all the essential business processes in the product life cycle.

For the project development, we have adapted the Spiral Model [9] as an iterative six phase process: planning, system requirements definition and analysis, software and hardware design, system integration, final system testing and close. Then, we have defined a Quality model based on the quality characteristics of the software regulations (ISO 9126/ IEC 61131-8), user and manufacturer needs and trends, and the experience and knowledge of team members.

System requirements analysis was the next phase. We transformed user requirements into technical specifications and models to serve as basis for design and implementation. We have chosen to work with the classic house of quality (HoQ), combining also software aspects [10].

The following point is the System modelling. In our project we will adapt the model already described in IEC 61131-3.

Last step will be the definition of a framework for evaluating quality. Thus we have defined metrics to evaluate the fulfilment of the requirements, adapted to our small team. These metrics are simple and refers to process and product. They are based on the Quality model previously defined and in the related standards (ISO 14598).

Three versions have been foreseen. In this paper we will present the first one ('boolean PLC').

2.2 Product standards

The IEC 61131 is the main product standard. It describes all aspects of PLCs. Part 1 defines the general concepts. Concerning to software, two parts are to be considered, parts 3 and 8. Part 3 describes the PLC software model and the PLC programming languages. Part 8 describes the guidelines for the application and implementation of programming languages. PLCs are in principle real-time systems, since they are used for control applications. Of course, some of the applications are not stringent and could be supported by non real-time operating systems. However, part 8 clearly addresses several important points, like scheduling, priority and other real time issues, thus indicating the suitability of supporting real-time. It also explains the aim of the new capabilities for PLC included in part 3: to promote software engineering ideas.

The software model presented in IEC 61131-3 considers several basic elements. Very briefly, they are the following:

- Configuration: the whole programmable controller system.
- Resources: the elements that provide support for all the features needed to execute programs in order to perform a signal processing function, the Man Machine Interface and then sensor-actuator functions. A configuration can contain one or more resources.
- Programs: programmed in one of the languages we mention below. They will perform some intended signal processing.
- Task: these elements control the execution of programs and functional blocks, periodically or upon a signal trigger.

- Function blocks: a set of input/output parameters and internal variables, together with an algorithm executed when the function block is called. They can store values, so they have internal state.
- Functions: produces one single result, and has no internal state.
- Global variables: they can be defined at program, resource or configuration level.
- Directly represented variables: variables that are addressed directly at known memory locations of a PLC.
- Access paths: they provide facilities to transfer data between different configurations.

The most known part of IEC 61131-3 concerns the programming languages. They are the following:

- Instruction List (IL): it is a basic low level language whose appearance is like a machine assembler.
- Structured Text (ST): it is a high level language whose appearance is similar to Pascal.
- Ladder Diagram (LD): it is a graphical language representing relay logic.
- Functional Block Diagram (FBD): it can be seen as a set of connected software blocks, and it is similar to an electrical circuit diagram.
- Sequential Flow Chart (SFC): it is well suited to describe sequences in a graphical form.

IEC 61131 originated as an attempt to standardise control systems and to help the integration of systems. It also favours the software structure, the reusability (through the use of functions and functional blocks), the data structure and execution control; in a few words, it points toward software quality. Even if the standard define more advanced languages, LD or IL are still widely used and are part of the knowledge of many maintenance engineers. Thus, they must be included in any prototype to be shown in industry [11]. The adoption of the standard by the main manufacturers is still unequal.

For the first version we show in this paper, we have used IL. IL can be regarded as the base language for IEC compliant PLCs. We will focus on it and give more details.

IL consist of instructions in different lines. The most part of instructions adopts the format of an operator followed by an operand. There are instructions to perform boolean, comparison and arithmetic operations. Of course, there are also load and store instructions. In addition, it is possible to call functions and functional blocks and to jump in the program. Parentheses are also allowed in some operators, thus deferring the corresponding operation. A simple program looks like that:

```
LD %I1
AND %M5
ST %Q2
```

It loads input bit number 1, and it with memory bit number 5 and stores the result in output number 2. The general meaning of a pair (Operator, Operand) is:

Result:=Result Operator Operand

To follow the standard helps you because it gives specified goals. However, it is very wide and the designer has to tailor it enormous possibilities. Besides, the definition of the standard is often ambiguous. We remarked above the external similarity between IL and a machine assembler. However, the behaviour of the virtual machine that could execute IL instructions is not clear. Any ambiguity must be removed before we proceed to the implementation. We cite some of the points:

- There is no definition of the register(s) that could store different data types. We decided to have a 64 bit register. It is wide enough to contain any kind of bit string types defined in the standard.
- The possibility of mixing data types is also unclear (for instance, to store a word, 16 bits, in a byte, 8 bits). We do strong data typing, as in ST so implicit extensions and truncations are not allowed. Literals can also be loaded into a variables if they are in the variable range.
- While not explicitly mentioned in the standard, we think it would not make too much sense to allow some operations inside parentheses (jump, load and store).

In addition to PLC specific standard IEC 61131, we have used the generic ISO/IEC 9126 (Software engineering - Product quality - Quality model) and ISO/IEC 14598 (Software product evaluation), as we mentioned in section 2.1.

Finally, we should refer to the POSIX standard concerning the code of our application. It would allow easy portability between operating systems. RTLinux is a right candidate since it is compliant with the minimal POSIX real-time operating system (1003.13).

3 First version

We named our first version 'boolean PLC', since it implements only boolean functions and use bit string types. It is a basic version to show the feasibility of the whole project, that, in fact, contains a single configuration on the single PC we are considering, and only one resource with a fixed task structure. The application runs from the command line. We should make a difference between the application manager and the RTLinux module. The former is a normal Linux application in user space that manages all the information given by the user, interacts with the translator and any other additional programs. The later is obtained from a standard compilation with *make*, after the preparation of the files by the manager. It can be inserted and executes the PLC programs. The information flow inside the application is shown in figure 1.

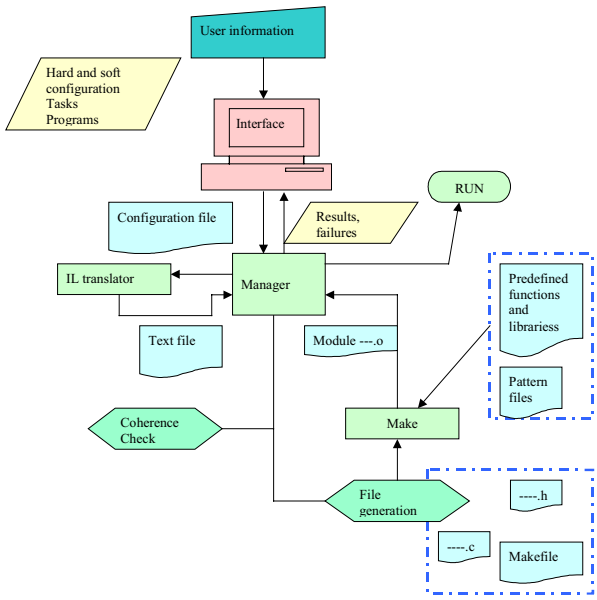

FIGURE 1: *Information flow in the 'boolean PLC'*

The user gives information about the following aspects:

- The software configuration (number of function blocks, size of internal memory).
- The hardware configuration (this depends on the I/O cards inserted; for simple test the parallel port can be used). Each hardware needs predefined files to be included.
- The instruction list programs associated to each task.
- The task characteristics. Up to now, the number of task is fixed. There are two periodic task and one event task, associated with an interruption.

The information is stored in a configuration file used by the manager. It is also possible to directly edit this configuration file.

Taking into account the information contained in the configuration file, the manager proceeds to translate the IL programs, associating them to given tasks, checks some coherency (for instance that we do not try to access an input bit not present in the hardware) and prepare the files for the module compilation. The translator reads every line of the IL programs and transforms it into an intermediate format, which holds codes for the operation and type of operand, and the address. Then, this intermediate format is translated into C code. Thus, the outputs are C files that contain mainly the job to be performed by every program, as well as the Makefile to perform the usual C compilation under Linux. The Makefile refers to some predefined, modified and new files (those generated by the manager).

When the final module obtained after the compilation is inserted, it creates two threads for periodic task. The handler of an interruption is used by the event task. In the module initialisation, interruption and I/O regions are demanded, depending on the hardware used. In addition, the memory regions are allocated: input image area, output image area and internal memory. Each task has its own input image and the rest are operated as shared memory. The resources are freed when the module is removed.

A user Linux program communicates with the RT threads so that it can monitor the state of the PLC memory areas. Up to now, the PLC system is used as any other RTLinux program, thus the PLC will work when the module is inserted and will lose its state when the module is removed.

For instance, the schematic structure of a periodic task is:

```
void *task1 (void *arg){
 struct sched_param p;
 volatile char *pin;

 pin=pi[0];
 p.sched_priority = 2;
 pthread_setschedparam (pthread_self(), \
  SCHED_FIFO, &p);
 pthread_make_periodic_np (pthread_self(), \
  gethrtime(), PERIOD1);
 while (1) {
    pthread_wait_np ();
   /* Read physical inputs */
   update_input_image(pin);
   /* Execute user program */
```

```
    do_plc_program[0](pin);
    /* Set physical outputs */
    update_output();
  }
  return 0;
}
```

For the shake of simplicity, we have shown some non-posix functions. The threads executes the classic PLC cycle, reading the inputs, doing some calculations and writing to the outputs. The functions related to input and output are hardware dependent and must be predefined in an external file. The do_plc_program[0] function is also defined in an external file, which is generated after the translation of the IL user programs, executing them. The scheduling priority is assigned inversely to the period of the task, PERIOD1, that is defined in a header file (.h). Pin is a pointer to the input memory of the task.

The IL translator accepts the IEC 61131-3 subset represented in Table 1. The variables are directly represented (as the example we showed above).

Operator	Modifier (*)
LD	N
ST	N
S	
R	
AND	N,(
OR	N,(
XOR	N,(
GT	
GE	
EQ	
NE	
LE	
LT	
JMP	C,N
CAL	
)	

TABLE 1: *IL operators implemented in the first version. (*) N=Negation, C=Conditional*

CAL instruction can invoke standard functional blocks: RS and SR bistable, up/down counters, timers and edge detection blocks.

4 Conclusions and future work

In this paper, we have shown the standards that should guide the implementation of a PLC, looking for the application of the philosophy of quality to the product and to the process. We have chosen a Spiral Model with six phase development process. The first version presented in the present paper implements a PLC that can perform boolean operations on bit and bit string variables, having two periodic task and one event task. The programming language follows a subset of IEC 61131 Instruction List.

As future lines of work, we list:

- The complete definition of a virtual machine (instruction set architecture) that could execute IEC 61131 programs. We noted above the existence of an intermediate format in our version, obtained from the translation of IL programs. This format could be extended and completed to accomplish the mentioned goal. Thus, instead of the passage from IL to C code, a binary file should contain the 'machine language executable' compiled from the user IL program. It would be charged in memory and interpreted by the PLC during every scan (this is a more classic operation).
- To provide a graphical interface including programming, operation and diagnostic facilities (like watching and setting of variables).
- To include all instructions in the IL language and also the LD language.
- To support full definition of the configuration as stated in IEC 61131.
- To install the system in a suitable format for industrial purposes (embedded PC), adding all the capabilities of commercial PLC: connection via TCP/IP or serial port, download or upload of programs, direct booting without the need of keyboard or screen. Thus, a black box where the user does not know that the system is supported by RTLinux.

Acknowledgements

We acknowledge the financial support of the Diputación Provincial de Teruel (OTRI 2001/0333), and the Diputación General de Aragón (P097/2001).

References

[1] *http://www.sea.siemens.com/default.asp, last visit on September 2003.*

[2] *http://www.schneiderelectric.com, last visit on September 2003.*

[3] G.A. Mintchell, February 1999, *PCs are gaining control*, Control Engineering available at http://www.manufacturing.net, last visit on June 2003.

[4] *http://www.pc104.org/technology/pc104_tech.html, last visit on September 2003.*

[5] *http://www.ocera.org/, last visit on September 2003.*

[6] *http://mat.sourceforge.net/, last visit on September 2003.*

[7] M. De Sousa, May 2001, *Linux-Based PLC for industrial Control*, Embedded Linux Journal.

[8] Forum, April 2003, *Eighth report from Forum Calidad,*, Forum Calidad, 140 (Spain), pp7–14.

[9] R.S. Pressmann, 2002, *Software Engineering. A Practitioner's Approach*, Mc Graw-Hill Companies.

[10] R.E. Zultner, 1994, *Software Quality Function Deployment-the North America experience*, Proceedings of the 4th European Conference on Software Quality, Zurich, pp143–158.

[11] R.J. Jafrate, 2001, *The Linux PLC: requirements to gain acceptance in industry*, Third Real Time Linux Workshop.

Standards

UNE-EN 61131-1:1996 Programmable controllers. Part 1: General information.

UNE-EN 61131-2:1997 Programmable controllers. Part 2: Equipment requirements and tests.

UNE-EN 61131-3:1993 Programmable controllers - Part 3: Programming languages.

IEC TR 61131-8:2000 Programmable Controllers - Part 8: Guidelines for the application and implementation of programming languages.

ISO 9001:2000 Quality management systems. Requirements.

ISO 9004:2000 Guidelines for performance improvement.

ISO/IEC 12207:1995(E) Information technology - Software life cycle processes.

ISO/IEC 12207:1995/Amd.1:2002(E) Information technology - Software life cycle processes. Amendment 1.

ISO/IEC 9126-1:2001(E) Software engineering - Product quality. Part 1: Quality model.

ISO/IEC 14598-1: 1999(E) Information technology - Software product evaluation - Part 1: General overview.

ISO/IEC 14598:2000(E) Information technology - Software product evaluation -Parts: 12,3, and 6.