Proceedings of the Software Innovation and Engineering New Zealand Workshop (SIENZ'07), Auckland, 26-27 November, 2007

Taming the Complexity of Software Design Process in Industrial Automation

Valeriy Vyatkin, John Grundy, Partha Roop
The University of Auckland, New Zealand

Abstract. **In this paper we discuss the challenges raised by the growing complexity of software in industrial automation systems. This trend is combined with the diminishing border between embedded systems and industrial automation. The solution is sought in using open standard component architecture, combined with object-oriented design patterns and model-based engineering. A new open-source tool FBench that implements these ideas is presented and discussed.**

I. INTRODUCTION

Industrial automation has been long associated with Programmable Logic Controllers (PLCs) which, as the title suggest, are computing devices capable of being programmed. The spectrum of languages used for programming PLCs includes some languages very similar to the traditional high and low level procedural languages (Structured Text language is similar to Pascal and C, Instruction List resembles Assembly language), along with others, which are quite strange 'animals', which are graphical and resemble electric circuitry. The family of graphical languages is represented by Relay Ladder Logic, Function Block Diagrams, and Sequential Function Charts.

Most common PLCs are microprocessor based devices with a modular architecture of peripherals, interfacing industrial environments. Their size, performance and interface capabilities vary greatly. There are PLCs capable of controlling huge machines with thousand of sensors and actuators, and there are micro-PLCs implementations on a chip, or on customized integrated circuits. However, the programming architecture makes these devices similar despite the huge visible differences.

The spectrum of physical systems where Industrial Automation technologies apply is huge. It covers all sectors of production systems, energy generation and distribution, material handling, transport (ships, trains), work machines (e.g. in mining), civil engineering (building automation, wastewater management), space exploration (launching pads), etc.

Recently the automation business has changed significantly. Automated products and systems need to be more flexible, both in their ability of manufacturing new products with a minimum re-configuration (run-time flexibility), and in terms of their own structure (design time flexibility) enabling their vendors to create new similar systems by re-using design solution from previous projects.

As a result, PLCs need to be networking, the software design needs a higher degree of re-use, and the role of standards is increasing in order to improve interoperability. Naturally, with the growing complexity of projects, the need for efficient higher level design tools and architectures is increasing as well.

II. OVERVIEW OF THE CURRENT TRENDS IN INDUSTRY

The market of automation hardware is shared by a number of large vendors, the biggest of which are: SIEMENS, Rockwell Automation, Schneider Electric, Mitsubishi Electric and OMRON. They also supply full range of software tools required to develop and maintain projects. Also, there are some independent software tool vendors, such as ICS Triplex ISaGRAF, and 3S software.

There is a major standard for PLC programming IEC 61131-3 [1] which has been tremendously successful. The standard defines a programming architecture of PLCs and provides five programming languages. All major providers of automation devices and tools support this architecture, at least, in part.

III. CHALLENGES

A. Distributed systems

As the automation hardware becomes more distributed and the software, consequently, is getting executed on networking computing devices, the importance of architecture independent languages, preserving semantics after being mapped to a particular distributed architecture, is growing.

The situation is aggravated by the mix of PLCs and embedded chips (programmed in an arbitrary way), and the mix of event-triggered and time-triggered networks.

B. Higher level design

System integrators and machine vendors need a design and implementation language of a sufficiently high level, supporting efficient encapsulation of intellectual property with its subsequent re-use.

C. Domination of the bottom-up approach

Software projects in automation tend to be designed in the bottom-up way. The reason of that is the need to re-use machines and mechanisms with existing software functions.

D. Flexible systems

Current markets dictate adaptability of automated manufacturing to ever changing product orders, produced in small batches. Besides, in many cases automated production machinery is surrounded by teams of workers and both adaptability and safety become the paramount concern.

IV. NEW GENERATION OF ARCHITECTURES AND DESIGN METHODOLOGIES

A. Similarities and Differences with Business applications

There are many similarities in the process of software design in business applications and automation.

The most essential differences have their roots in the nature of processes being dealt in automation. Direct interaction with machines asks for reactivity, response under hard real-time constraints, compliance with various industry-specific requirements (like the "operating mode support in chemical industries defined by the ISA-88 standard"), safety requirements, etc.

B. Component architectures

As an obvious solution to the accumulated problems, component architectures have been considered in the last decade. An example of a proprietary component architecture is Component Based Automation (CBA) of SIEMENS, which uses DCOM as an underlying IT architecture and PROFINET (a combination of Profibus and EtherNet) as a communication mechanism. This architecture, however, has not become hugely successful for many reasons.

An alternative open approach has been formulated as the IEC 61499 reference architecture [2], which is in early stages of adoption. A comprehensive description of the IEC61499 concepts can be found in [3]. Some concepts related to IEC61499 are briefly described as follows.

Basic Function Block is an atomic program unit. Basic function blocks are used to encapsulate *algorithms* written in one or several programming languages. Execution sequence and semantics of the algorithms depend only on the internal state of the block and on the input events but not on the properties of a particular execution platform. Thus, Basic Function blocks will exhibit equivalent execution semantics on different computing platforms, regardless of a sequence they are listed or called in the program. An example of a Basic Function Block description is shown in Figure 1. Every function block consists of an *interface, an execution control chart (ECC)* and of a set of *algorithms that are invoked in specific states of the ECC.* The *interface* is explicitly separated on the event and data parts.

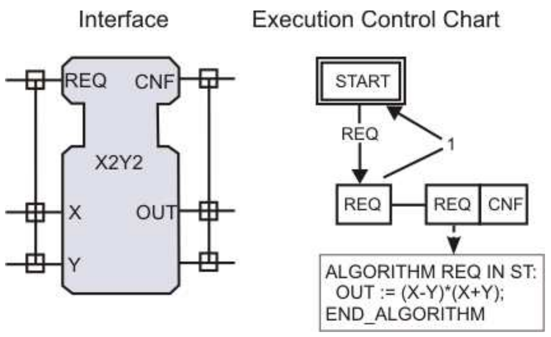

Figure 1. An example of a function block according to IEC 61499.

The event inputs and outputs are connected to the *head* part of the interface shape, while the data are connected to the lower *body* part. Basic function blocks are activated by event inputs. The ECC is a state machine that decides in each state which algorithms to call and which output events to emit.

Composite Function Block is an encapsulation of a network of other function blocks (a set of function blocks that are interconnected using their respective interfaces in a point-to-point fashion). The example in Figure 2 implements the same computation as the block in the previous example, and has exactly the same interface. Its internal logic, however, is defined as a network of three interconnected function blocks, each performing basic arithmetic operations instead of ECC and algorithms.

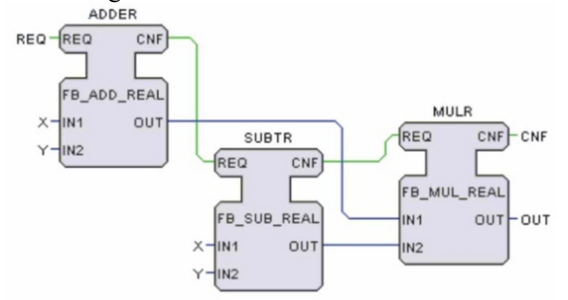

Figure 2. Network of function blocks instances - its behaviour is completely defined by the event interconnections.

An *application* in IEC61499 is also a network of function blocks. A *distributed application* is a number of applications allocated across different computing devices. The inter-block connections in an application are extended across boundaries of devices in distributed application using communication function blocks.

Thanks to explicit event connections between function blocks the execution semantics of the distributed application does not depend on a particular number and topology of computational resources or on the order of components' execution as heavily as in the case of networking cyclically scanned PLCs (certainly some dependencies caused by different speed of the resources and networks still will remain). As a consequence of this no additional synchronization between devices would be required to ensure consistency of the data transmitted between them. This will be true also for more complex applications that are represented as hierarchical networks of function blocks. Thus, system development can now be done in a high-level platform independent form of function block applications which can be mapped to different distributed structures of devices (and used computing platforms) and any legitimate mapping will preserve the semantics of the original application.

C. *Design patterns*

The growing complexity of automation systems and their safety-critical nature make especially beneficial the use of design patterns. In particular, the modified Model-View-Control (MVC) object-oriented design pattern adapted by Christensen in [8] to industrial control has proven its usability in conjunction with the IEC 61499 architecture. The pattern is represented graphically in Figure 3. The core part of the pattern is the closed loop object – controller interconnection. In software an object is represented an *interface* to its data sources (say sensors) and signal consumers (actuators). The object can be substituted by its model – a software entity having the same interface and simulating object's behaviour. Several models can be used depending on the required accuracy and the purpose of modelling.

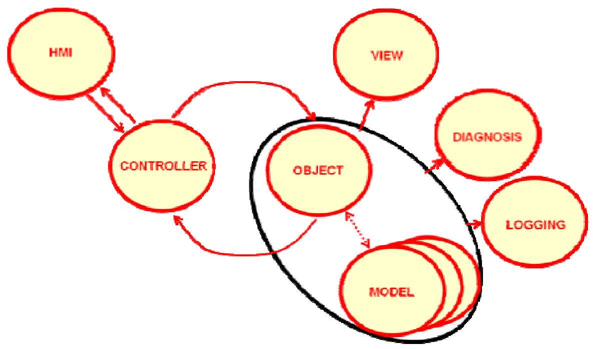

Figure 3. Components used in the MVC design.

The model and the object are sources of data for a number of other optional components, such as View – implementing rendering, Diagnostics and Database logger. Contrary, the human-machine interface (HMI) is connected in closed loop with the controller.

D. *Object-Oriented Design and Model-driven engineering*

In industrial automation the idea of object-oriented design has been explored in two forms:

a) In the classic software engineering sense (i.e. the combination of encapsulation, polymorphism and inheritance)

b) Suggesting architectures where structure of software can resemble the structure of physical components of the machinery being controlled. Taking this idea to extremes, an appropriate architecture must be applicable to any automated machine: from a complex production cell that consists of several machining centres, robotic arms, etc. down to very basic pneumatic cylinders, with a couple of position sensors, as illustrated in Figure 4.

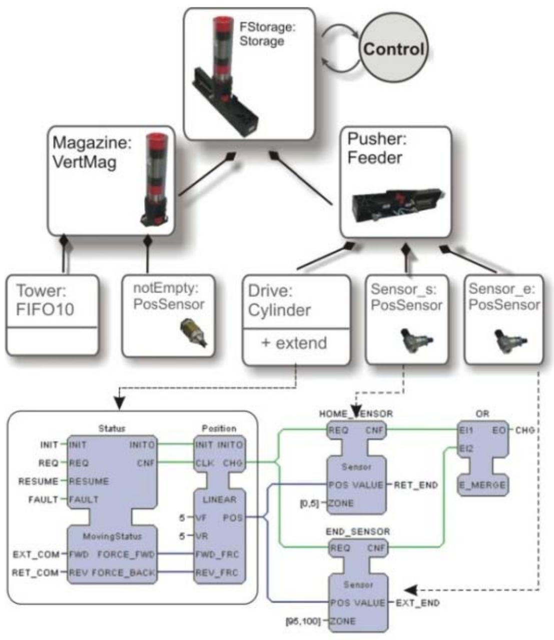

Figure 4. Physical structure of a mechatronic object is directly reflected in the structure of the software (function blocks) controlling and simulating it.

The idea of model-driven engineering in industrial automation has been explored from different perspectives, such as the use of the Unified Modelling Language (UML) and development of Automation profile for UML 2.0, the use of Petri net–like formalisms for program prototyping, code-generation and verification, developing of languages based on state machines (e.g. HiGraph of SIEMENS [6]).

V. FIRST EXPERIENCES AND NEW CHALLENGES

A. *Experiences*

The author is dealing with the IEC 61499 architecture for the past 8 years and has accumulated a few experiences and observations. A few small to medium size automation projects have been completed using the FB technology and the design methodology described above.

In particular, recently in Auckland the control of the reconfigurable manufacturing testbed (see Figure 5) has been implemented using the MVC pattern with function blocks.

While applying the abovementioned software design approach the following problems were encountered and identified:

a) "Spaghetti" of inter blocks connections – explicit drawing of all links makes the schema unreadable. A solution is known and described in [3], but requires seamless tool support;

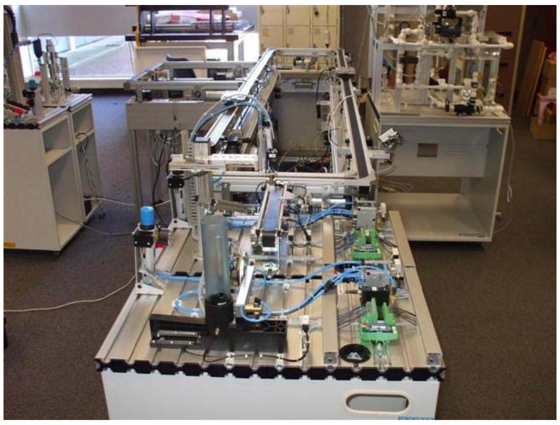

Figure 5. Reconfigurable manufacturing testbed in the infoMechatronics lab of the University of Auckland.

b) A new component is created for every slight modification in interface, structure or functionality. A sort of "inheritance" dependency within a group of 'relative' components would be desirable;
c) As in the general programming, the question of an optimal component size (especially in terms of provided methods) is open;
d) Migration from the legacy systems (IEC 61131) to the new (61499) architecture is a challenging research issue.

B. Some solution ideas

The concept of software product line [4] seems to be promising. Keeping a family of components related to each other via the design-time inheritance can be supported by the engineering tool. This can provide for easier variation of "parts". The FB architecture of IEC 61499 provides a good mechanism to subdivide this issue on categories. Thus for a basic FB the categories may include:

- Interface elements:
  - Blocks of Inputs/Outputs
  - Extra attributes of I/Os
- Algorithms
- State machines

C. Tools development

The IEC 61499 standard is already almost 2 years old. There have been enormous amount of research done worldwide, however, industrial adoption is still very limited (although the dynamics during the last 2 years is encouraging).

Given the considerable expertise on IEC 61499 accumulated in academia, it seems feasible to develop a mature tool as an open source software. Such a project has been started in 2006 by o3neida – the international community for intelligent industrial automation.

The first version of the tool called FBench was released in Auckland in 2006 [7], and as of October, 2007 the tool (Figure 6) is open-source on the Sourceforge.

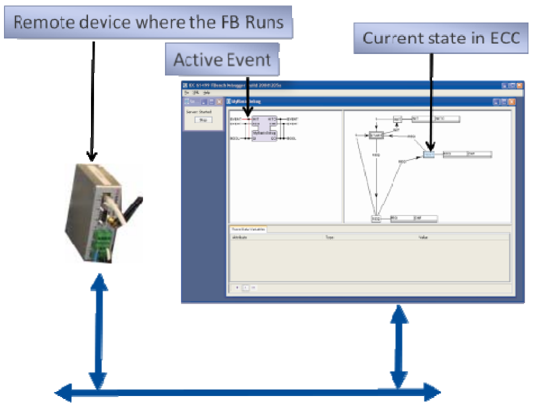

Figure 6. FBench provides a full set of remote debugging services: status of the program executed in a remote control device is rendered.

VI. CONCLUSIONS

The domain of industrial automation software faces similar challenges to the business software design. Application of advanced software engineering techniques can be beneficial there.

However, the engineering techniques cannot be simply planted, they need to be modified according to the strong legacy and traditions of automation systems design.

VII. REFERENCES

1. *IEC 61131-3 International Standard,* International Electrotechnical Commission, Geneva, 1993
2. *IEC 61499 - Function blocks for industrial-process measurement and control systems* - Part 1: "Architecture", International Electrotechnical Commission, Geneva, 2005
3. Vyatkin V., *IEC 61499 Function Blocks For Embedded and Distributed Control Systems Design*, 297p., Instrumentation, Systems and Automation society publishers, USA, 2007
4. Software Product Line: http://biglever.com/overview.html
5. ISaGRAF Workbench: www.isagraf.com, access May, 2007
6. SIEMENS S7-Hi-Graph, http://www.automation.siemens.com/simatic/industriesoftware/html_76/produkte/software-s7-higraph.htm , online, October 2007
7. W. Dai, A. Shih and V. Vyatkin, *Development of distributed industrial automation systems and debugging functionality based on the Open Source OOONEIDA Workbench*, Australasian Conference on Robotics and Industrial Automation, ACRA 2006, Auckland
8. J. Christensen, "Design patterns for systems engineering with IEC 61499," *Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, Ch. Döschner, ed.* Magdeburg, Germany: Otto-von-Guericke-Universität, 2000.