Control Software Reuse Strategies with IEC 61499

Mika Strömman, Seppo Sierla, Kari Koskinen
Inf. and Computer Systems in Automation
Helsinki University of Technology
P.O Box 5500, FIN-02015 TKK
Mika.Strömman@tkk.fi

Abstract

The IEC 61499 gives developers considerable freedom and offers several features that support reuse. Although this can help the adoption of the standard into an organization’s development process, the range of design decisions can also harm the reusability of software, if components are developed with incompatible design principles. Guidelines are needed, but the unique backgrounds and market positions of industrial players must be respected. The paper analyzes some alternative solutions that were developed by automation designers using IEC 61499; although the solutions differ considerably, it is very difficult to rate their reuse potential absolutely. The factors that must be addressed in a reuse strategy are discussed, concluding with the suggestion that it is not possible to present detailed guidelines that would apply to the entire industry.

1. Introduction

Reconfigurability and reusability are non-functional qualities of control software that are greatly affected by the software development process. The process defines activities and provides guidelines for performing them, but the use of particular programming languages, development tools, standards and modeling notations is usually not required [8]. The process is mainly concerned with producing software of predictable quality on schedule using activities such as requirements management and software component repository management [7]. However, tools, standards, languages and notations are based on certain paradigms [27] and will either support or hinder the use of any process. The goal of this paper is to explore the new IEC 61499 function block modeling and programming standard [19], and to identify its support for various reconfiguration and reuse strategies. We suggest that this understanding is a prerequisite for integrating the standard with an existing development process or proposing a new methodology around it.

Our view in this paper is that any industrial software development process and related standard is ultimately validated by the industrial experts that design the applications; any proposal must be perceived useful in the context and value system of the problem owner [21]. In order to exchange academic and industrial views concerning the standard, the Helsinki University of Technology organized a summer course on IEC 61499 that was lectured by Dr. Jim Christensen, chairman of the standardization work group, in 2004. The assignment was to automate a pallet lift in limited time using IEC 61499 tools and an embedded controller. The participants were professional automation designers and researchers with a solid background understanding of the industry’s needs, and they presented several working solutions with very different implications on reuse and reconfigurability.

The purpose of this paper is to analyze a number of these solutions as well as the written feedback of the participants; this material is complemented by interviews that have been conducted in Finnish companies concerning their development processes. The data indicates that the evaluation of design approaches is highly context-dependent. For example, the background of designers, the existence of legacy software, the possible use of subcontractors, the organization’s policy regarding open systems (see [24]) and the business goals of the organization all determine the best design approach – choosing a reconfiguration strategy will have far-reaching consequences as designs can be propagated into several generations of systems. Our intended contribution is to identify possible reconfiguration and reuse strategies with IEC 61499 as well as some of the context-related factors that can be used to evaluate the alternatives.

We want to stress that this paper focuses on one important aspect of reconfigurability. [3] describes the potential of agents in dynamically reconfigurable control systems using IEC 61499. [28] describes the role of real-time middleware in automatically setting up communication links between distributed fragments of an application. [29] discusses infrastructures for managing the lifecycle of a component-based automation application in a dynamic, multi-vendor environment. These approaches to reconfigurability are orthogonal and should be seen as being complementary.
2. Research Goal and Approach

We are now in a position to articulate our research question and the approach. In research on software process improvement, [34] and [20] point out that 3 kinds of knowledge are created: understanding of current practices, supporting artifacts and propositions, and improvements to current practices. The activities on these three fronts are interdependent and progress in any one of them can support advances in the others. The standard XML notation of IEC 61499, an open knowledge economy and repositories for reusable components [19] [24] [35] all address identified problems in designing reusable automation software, so they support practice. These normative propositions are tested in attempts to improve practice, and this experience in turn yields a deeper understanding of practice.

It is this understanding that is our research goal, so we organized a course that challenged experienced practitioners to make use of the FBDK (an IEC 61499 development tool) and its repository of reusable components. As new research results are obtained from pilots with new technologies, our goal is to advance their deployment into industrial contexts – we argue in this paper that any guidelines for this must be sensitive to the needs of different organizations.


3.1. General software development

Although this paper is aimed squarely at industrial control software designers, it is nevertheless useful to refer to the output of a broader community of software developers. A great volume of literature on software processes, their assessment and improvement has been published, but the results of practical case studies are confusing and contradictory. These experiences can be very useful for automation software developers, who are now faced with challenges that are in many ways similar to the problems that their peers in other domains have been struggling with for some time.

The majority of research on software development is presented in a highly normative fashion. Researchers often seem to expect that practitioners will abandon the established methodologies in their organization in favor of the new approach. Proposals such as [7][17][18] are based on considerable insights, and while the authors may concede that the approaches might need to be adjusted in practical situations, the difficult problem of how to perform this adjustment in a feasible way is glossed over. Even the established paradigms for software process assessment, improvement and maturity modelling are concerned with identifying certain activities in the target organization, instead of evaluating the suitability of the existing activities in the organizational context [8][26].

There are a smaller number of publications, which address the practical failures of the mainstream approaches. In a much cited article [4], Brooks argued that the creativity of designers cannot be replaced by any artefact. [16], [23] and [25] demonstrate that the successful use of methods or tools is highly dependent on the individuals that use them and their organizational context. [33] questions whether the available methods are actually used in realistic development situations. [21], [22] and [20] describe the benefits of greater coordination between researchers, practitioners, domain experts and end-users. [20] reports on a series of software process improvement initiatives in Danish companies; the established assessment and improvement frameworks [8][26] were considered too normative and were therefore rejected by the team. [1] rejects the idea of process phases with a fixed chronological sequence and propose the concept of mediation between four aspects of the development effort. Although the importance of requirements management is recognized in the mainstream of software engineering [7][17][18], the proposed solutions cannot be considered adequate, since of all the tasks in the process, poorly specified, understood or managed requirements are the greatest source of failure [13]. The ability to cope with dynamic requirements is of central importance in reconfigurable software; agile development methods [2][12] emphasize excellent communication, continuous customer involvement and the ability to reorient the project painlessly when a new understanding of requirements is achieved.

In summary, the many insights and findings of the software development community are contradictory, and it is difficult to say what kind of a development approach will enable organizations to develop reconfigurable software flexibly and reliably. It is our view that all of the cited researchers have produced results that are valid in some context, but that there is also a need for greater understanding concerning the deployment of these results to the industry. The purpose of this paper is to add to the understanding of the deployment of new normative approaches to the industry, focussing on the IEC 61499. The next section will describe the current situation in industrial control software design and set the context for our case study. In the analysis of the study, validity is sought by comparing the findings with a broader body of research, including the publications that were cited in this section. We believe that most contradictions result from poorly identified contexts that are not directly comparable across cases. This article aims to be as explicit as possible about the context of the case study and the generality and comparability of the results.
3.2. Industrial control software development

Several trends in the industry must be considered in order to assess the feasibility of open, standards-based reusable automation software development. In this section, some problems and advances in programming techniques and the industry’s development processes are described. Although new technologies would support the unification of development processes and global markets make it more difficult to adhere to proprietary practices, the industry remains hesitant in moving towards an open knowledge economy. We conclude this section by arguing that an automation software vendor’s development process is at the heart of its competitive advantage and has been adapted to that company’s niche in its market environment. Any effort to introduce techniques and practices that support reuse should respect this situation.

The most significant difference between developing industrial control software and general purpose software is that industrial software is often tightly bound to the physical hardware that is being controlled. This has implications on requirements definition, the partitioning software into components, the distribution of the components, the possibilities of reuse and testing.

The hardware is designed for the process that is to be automated. The software has to be designed according to the hardware and after the hardware design is adequately ready. Therefore, in industrial control software development the bonding between software and hardware affects not only to the actual design of the software but also to the process of design.

In recent research projects in the laboratory of Information and Computer Systems in Automation at the Helsinki University of Technology, the challenges in software development in automation projects have been observed by interviewing Finnish suppliers of automated systems. In every interview, the early phases of a project are emphasized. There are two major problem areas: the gathering of user requirements and the transfer of the project from the sales organization to the project team.

The difficulties in gathering user requirements are a recognized problem in general software engineering [30]. In industrial control software projects, the customer often has specialists in the manufacturing process itself. These have opinions about user interfaces, but lack sufficient knowledge of the automation software in between.

The project is often sold by the marketing department. Though the requirements specification is written, the “spirit” of the negotiations isn’t propagated to the automation designers.

Testing is also to be found problematic, because the SAT (Site Acceptance Testing) is always expensive to accomplish and usually the time schedule is tight. The FAT (Factory Acceptance Testing) is difficult to carry out comprehensively, because some of the instruments and actuators are often ordered directly to the site due to their size and delivery schedule. In manufacturing industries, the system often has to be built completely or almost completely for testing purposes, broken into parts for transportation and finally rebuilt on site [32].

The actual application development is nowadays mostly done by programming PLC’s (Programmable logic controller) with graphical languages [11]. The thorough understanding of low-level program details is crucial, because modifications to the physical system can lead to updates in the related software component.

The software development environment must make it possible to raise the level of abstraction, so that larger entities can be treated and reused. There’s an ongoing trend of creating intelligent mechatronic components for dealing with these problems [36]. The use of function blocks is also suggested for the early phases of the project for modeling the system [15] [36]. The new IEC 61499 standard introduces many improvements that support the broader use of function blocks than what was possible with the IEC 61131-3 standard, which is the earlier function block standard mainly for programming PLC (Programmable logic controller) devices. Such improvements are the transferability of components in XML format [14], better support for distributing the application and better support for separating the interface from the implementation (see [19] for more details). The transferability of the program components together with the features that enable the encapsulation of intellectual property make software component markets possible [35].

The processes of automation design are contingent on the field of industry, the size of the project and the enterprise and the amount of reusable knowledge. The increased networking of the supply chains makes demands on the design process; the enterprise subcontracting parts of its automation application development should require the subcontractor to follow similar programming styles and compatible design process.

Because of the high expenses of automation design, the design process is a matter of competition; with a good practice, which is suitable for the field of application, it is possible to achieve a more competitive position. Therefore, the goals to unify both the design process and the programming styles should be carefully examined. Solutions that allow the enterprises to use processes of their own and to develop the processes furthermore can be of great benefit, yet open markets and uniform development approaches that enable broad reuse can be very profitable to some actors. The problem of deploying the advances in industrial informatics to companies is multifaceted and requires research in close contact with the industry.
4. Case Study

4.1. Equipment

Before the course, DeviceNet I/O-modules were installed to the lifter. The DeviceNet was connected to a TILT-board, which had a SNAP processor attached to it. Function Block Development Kits (FBDK) were installed on computers of a class room. TILT and SNAP are commercially available products, while FBDK is a research tool for developing IEC 61499 applications. The system schematics for the arrangements on summer course are shown in Figure 2.

4.2. System requirements

The students on the course were given the following requirements for the system:

- The Lifter Unit should work as a part of a larger production system. It must be capable of receiving pallets from the upper conveyor, even several pallets with no gap in between. The Lifter Unit should also be safe, so that the vertical movement of the lift is stopped if something is in the shaft.
- The upper conveyor belt that feeds pallets to the lift will always be operational.
- The pallet is driven to the lift when all the following conditions are met:
  - a pallet is ready at the stopper on the upper belt
  - the lift is in its top position
  - there is no pallet in the lift
  - the pallet is driven in by lowering the stopper for the duration of the feed and starting the conveyor belt inside the lift (into the inward direction) until the pallet is inside
  - driving down is started if the lift is at the top position and a pallet is inside.

There is an optical shaft sensor that can be used to detect if an object is in the shaft where pallets go in or out of the lift. The lift is stopped if this sensor detects something.

When the lift is at the bottom, the belt inside the lift is activated into the outwards direction until the pallet is no longer inside. The lower conveyor belt that takes the pallet to the assembly line is also activated when the lift is at the bottom, and it will be stopped 6s after the pallet has exited the lift.

The lift is driven to the top when the pallet has exited the lift. The lift keeps pushing up against its top limits while it is at the top position. Lifting is suspended while the shaft sensor detects something.

4.3. Tutorials

At the beginning of the course, some relevant design patterns and useful function blocks from the FBDK repository were presented. In particular, the distributed application, proxy and MVC (Model-View-Controller) design patterns were covered [6]; these had some differences to more classical patterns [5] and were supported by function blocks in the FBDK repository of
The students had to use this repository to interface their control logic to the I/O handling that was built onto the SNAP by the lecturer.

5. Alternative Solutions

5.1. About the solutions

The students developed a number of markedly different solutions. This section presents three different solutions using IEC 61499 function blocks and one using IEC 61131-3 function blocks.

5.2. Reusing old programs

The first properly working solution was made by a programmer who had previously experimented with IEC 61131-3 languages for lifter unit [31]. This knowledge was exploited by reusing an existing solution in Structured Text. The code was simply copied into an algorithm of a 61499 function block. Some minor changes were made, because the tool that was used to develop the original ST program had allowed some non-standard expressions.

![Figure 3. A solution in which the previous ST-program was reused.](image)

The block had the I/O-variables in its data interface. It also had an event input for setting the initial values of the variables and another event input for initiating the algorithm.

The algorithm was executed periodically, like in the IEC 61131-3 solution. The leftmost block in Figure 3 is a proxy which is handling the communication with the hardware. In the middle is the cyclic event generator block and on the right side the actual controlling block.

The proxy block had an event output which notified if a variable state had changed. The next step was to try executing the control block without the cyclic event generator, so that the algorithm would only be executed when a physical I/O-variable changed its state. This worked as well.

The possibility for reusing old programs easily is one essential feature which helps organizations that are interested in adopting new standard.

5.3. State diagram

The Execution Control Chart (ECC) of a function block is used for controlling the execution of algorithms inside a function block. In this solution (Figure 4), the functionality of the lifter unit was divided into small algorithms, and the execution of these was controlled by the ECC state diagram. The conditions that trigger transitions between states are Boolean expressions involving event and data inputs.

![Figure 4. State diagram based solution.](image)

A solution of this kind works well when the problem to be solved is naturally sequential or the states and the transitions between them can easily be identified. Reusing such a solution can be possible if the states are typical in the application process. This kind of reuse can be compared to the use of design patterns [10], i.e. reusing the ideas rather than low level code; the reuse works best when the hardware-dependent part of the software is encapsulated in their own function blocks.

One significant advantage of this solution is that if the tool supports online diagnostics, the state of the program can be identified and therefore the troubleshooting is easier than in conventional solutions.
5.4. Mechatronical approach

The next paradigm for automation design could be the unification of design areas, so that the mechanics, instrumentation, data transfer and control software are developed more tightly together. The mechatronical approach suggests that the software is divided into components according to the physical composition of the target equipment. This approach is familiar from the process industry, where type circuits for different control loops are used in process industries to control e.g. the liquid material flow through a valve, the level of a tank or even more complex equipment and unit processes that are often found in process plants. Type circuits are often realized in the form of parameterized function blocks in the DCS (Distributed Control System) application programming interface. In the manufacturing industry, the type circuit concept is quite new, because the physical equipment changes more often.

In this solution (Figure 5) the program is divided into components according to the physical composition of the equipment to be controlled. The processing is no longer driven by changes in the I/O-state, but by events that indicate the successful completion of operations. The advantage here is the great potential for reuse if it is possible to assume that the physical components can be reused independently in other processes.

Figure 5. Mechatronical approach.

5.5. Solution according to IEC 61131-3 function blocks

As a fourth solution to the problem we present an "old style" function block program. The problem is divided into sub-problems which are solved in a sequential network. The solution is simple and uses only basic function blocks.

Figure 6. A solution using IEC 61131-3 Function Blocks [31]

6. Analysis

Most programming languages enable the use of a certain range of paradigms and programming styles in addition to the ones that they support explicitly [27]. Most automation designers are used to a very restricted programming style that is dictated by the PLC cycle of scanning inputs, executing programs and writing outputs. The students on the course quickly discovered that developers using IEC 61499 are faced with significant design decisions that should be addressed before anything can be implemented.

Participants perceived the standard as giving more freedom than traditional PLC programming languages, although it was more restricted than languages such as Java and C++. Everyone had to use events to some degree and typical PLC programming techniques were applied in various ways. However, this was a source of considerable confusion, because the underlying execution model in IEC 61499 is very different from PLCs. Although it is quite straightforward to chain a number of blocks behind a cyclic event generator, the reading and writing of inputs and outputs behaves
differently from a PLC where inputs and outputs are frozen during program execution. Our interviews with industrial practitioners revealed that understanding the run-time behavior of the programs depends on the tacit knowledge of experienced designers; the participants on the course expressed their concern about the lack of experience with the event-driven model.

Some students doubted whether the standard could support the same development approaches as C++ or Java. In particular, questions on the scalability of systems were raised. The idea of wiring ports between function blocks is derived from the centralized PLC model and might be problematic for large, distributed and frequently reconfigured systems. Techniques such as Web services, OPC and Java-based component platforms have been field-proven in enterprise scale systems, although their adaptation for industrial automation requires further work. It was suggested that IEC 61499 could serve as a base standard for such extensions because of its strong connections to factory-floor automation. The use of UML2 extension mechanisms for creating a domain specific profile was also proposed.

Returning to the actual solutions, it can be observed that the interfaces of most control blocks plug directly into the I/O interface. This means that many of these blocks with very different control solutions, such as the ST and ECC controllers, could be exchanged. While this is usually not needed during reconfiguration, it suggests that the various solutions can be encapsulated behind interfaces that enable straightforward wiring to other blocks. Reuse and reconfiguration are greatly facilitated by having common guidelines for defining interfaces, although doing this at the I/O level (e.g. having ports corresponding to digital inputs and outputs) can cause technical problems and is not robust in the face of changes at the hardware level. Defining more abstract events for the entire organization, such as TRANSPORT_PALLET, is a potentially elegant solution, but it is difficult to define such a generic set of events that would satisfy the needs of the entire manufacturing industry. One of the companies that we interviewed had succeeded in defining such an interface based on the functionality of their applications in warehouse automation, although not with 61499. This interface is also used by their subcontractor that supplies transportation modules.

What are the implications of these alternative approaches on reuse and reconfiguration within or between organizations? This depends on the interoperability of artifacts that have been designed using the different approaches; this in turn depends on the features of IEC 61499. For example, a block can encapsulate the entire control logic for the pallet lift in a state machine or ST and have the same interface and behavior. This implies that these two paradigms can be mixed in the same application. However, there were other design decisions concerning the granularity of blocks or the different ways of combining cyclic and event-based execution. The flow of control could be driven either by changes in the I/O or by propagating events in a network of blocks in push or pull fashion. These differences are not encapsulated behind IEC 61499 interfaces and the incompatible operational principles might cause problems even if one uses design patterns such as adapter, facade or bridge [10] to unify the interfaces.

In summary, IEC 61499 enables the use of a range of programming paradigms and design principles, and their coherent use is a major factor behind successful reuse and reconfiguration. This requires guidelines, but it might not be possible to define complete and generic recommendations that are directly applicable in all companies and application areas.

7. Conclusions and Further Work

These totally different kind of solutions show that the IEC 61499 gives many possibilities to solve a problem. The complexity of a solution can be inside an algorithm, in an ECC or it can be built as a network of function blocks, so that the event and data connections are complex. Alternatively, the problem can be hierarchically divided into several layers using composition function blocks.

Concerning our contribution, the paper has raised questions without providing any definite recommendations. Our research goal was to increase the understanding of the use of IEC 61499 by industrial practitioners. The need for guidelines has been demonstrated by identifying many specific design decisions that must be addressed before the commercial development of reusable software can begin. Further, there is no obvious way to define good guidelines and the application area and competitive advantage of a company must be taken into consideration.

The object of our further research is to advance the deployment of these solutions [24] into the industry, but any practical solution must be validated by practitioners’ success criteria [21]. This necessitates a research environment and method that fosters collaboration and the exchange of information between researchers and practitioners. The design and reuse strategies can be evaluated with a research environment consisting of an application development environment and devices or simulator models, against which the application programs can be verified. The goal of our further research is to construct such an environment and to invite industrial practitioners to perform design tasks in it with new technologies.

In order to compare different process models for automation design and to assess new technologies in these contexts, the concepts that comprise the design process should be explicitly defined. This should be based on the design tasks, not the process phases, for the
processes in enterprises are different from each other. Using such a general development process model, the design tasks that are assigned to practitioners working in the research environment are similarly understood and the results of tests are comparable.

References


