Incorporating Industrial Experience to IEC 61499 Based Development Methodologies and Toolsets

M. Strömman¹, K. Thramboulidis², S. Sierla¹, N. Papakonstantinou², K. Koskinen¹

¹Helsinki University of Technology
P.O. Box 5500
FI-02015 TKK, Finland
{mika.stromman, seppo.sierla,
kari.o.koskinen}@tkk.fi

²University of Patras
Electrical & Computer Engineering
26500, Patras, Greece
{thrambo, npapakonsta} @ece.upatras.gr

Abstract

The successful use of IEC 61499 in the industry requires software development processes that utilize the standard’s support for best practices of software engineering, such as CORFU. The adoption of such a process would involve upgrading existing practices, tacit knowledge and legacy software, which constitute much of the core competence and essential assets of a company. This paper reports on a two part research that aims to manage the risk and cost of such a venture. The first part has been performed in collaboration with professionals in order to identify industrially familiar work practices for IEC 61499 based development. The second part investigates the possibility of adapting the CORFU development process into the direction of the work practices discovered in the first part.

1. Introduction

The function block standard IEC 61499 gained its standard status in the beginning of year 2005. So far, the commercial application of the standard in design tools has been quite exiguous. The deployment of the IEC 61131-3 standard, which covers the programming languages for programmable controllers, was also quite slow even though it was based strongly on practices that were industrially established at that time.

There are several reasons for the slow adaptation of the new standard. First of all, the factory automation field is rather conservative with new technologies, before their reliability has been proven in industrial scale operations. Secondly, clients expect to be able to maintain the delivered control applications, so the tools and design practices must be adequate and easy to learn; switching costs from current practices must be smaller than the benefits of the new practices [1]. If customers are not demanding tools that support the new paradigm, it is unlikely that any tool producer will promote it.

An experiment that has been performed independently with two development processes and toolsets is described. One approach imitates established industrial practice while the other seeks to incorporate and support the best practices of the software engineering community. A comparison of these two approaches is used to identify a development process that would optimally support the use of IEC 61499 in an industrial context. On one hand, it is important to incorporate best practices of software engineering in order to improve current practices; on the other hand, in order to minimize switching costs, industrial software development expertise should not be rendered obsolete.

The research was done in two parts. An event for obtaining industrial experience was arranged at the Helsinki University of Technology (TKK) in June 2006; invited professionals and researchers worked in teams to automate a batch process. This resulted in an understanding of how industrially established work practices and team organizations could be applied when IEC 61499 development tools are used. Participants in the event learnt the necessary work practices easily, but it is questionable whether this approach makes full use of the IEC 61499 standards potential to support the best practices of software engineering. The second part of the research consisted of developing the same batch control application with the CORFU toolset and development process in the University of Patras (UniPat).

By contrasting the work practices in the first and second parts of the research, it is possible to identify retraining needs for upgrading current industrial practice to modern software engineering practice supported by IEC 61499. Skillful definition of work practices can be used to make the new development process more accessible to practitioners with a traditional automation engineering background. It is important to define work practices in such a way that a minimal amount of IEC 61499 knowledge is needed. This will make it possible for a team of
developers to make progress even if the majority of its members have limited background in computer science or IEC 61499, which is often the case in the industry.

2. Research questions

Our goal is to propose a development process that takes advantage of the new features of IEC 61499 in such a way that its industrial adoption will involve minimal changes in established work practices and personnel. In order to understand the obstacles to adopting a new process, current industrial practice is summarized in the next paragraph, based on interviews in 8 automation companies. The interviews concentrated on the organizations’ software development process and the challenges in those processes.

Software development in the process and batch control industries is divided among R&D projects and delivery projects. A delivery project aims at a one of a kind installation for a client, such as the automation system for a paper mill. The main criteria for these projects is delivery time, so it should be possible to perform software development activities mostly or even completely by retrieving, parameterizing and wiring existing function blocks. R&D projects maintain repositories of function blocks and often provide baseline solutions, which are templates or skeleton applications containing functionality that is common among applications. In this case, much of the architecture design and detailed design has been performed by the R&D and is taken as a starting point in a delivery project.

A fundamental research issue in proposing any software process for industrial use is the tradeoff between generalizability and contextuality. In order for a contribution to be of scientific interest, it must be generalizable rather than limited to solving the problems of one company. However, a general software process is not directly applicable in an company context, which is unique in terms of toolsets, educational backgrounds of developers, application domain and available legacy code and tacit knowledge [2][3][4][5]. The pursuit of competitive advantage will continue to create such differences between companies in the contexts for software development processes.

The first part of our research aimed at proposing and assessing work practices for IEC 61499 development that are close to the real world context of our industrial partners. The following two research questions were formulated:

1. Can current automation design work practices be supported by 61499 tools?
2. What new or modified work practices need to be introduced?

In order to propose and evaluate new work practices, an existing tool (the Holobloc FBDK) was augmented with guidelines in the form of example solutions that incorporated a batch domain-specific standard. An event that resembled a delivery project was organized in this development environment. Positive experiences and problems are discussed in section 3.

In section 4, this experience is used to adapt the CORFU development process into the direction of current industrial practice. CORFU is based on best software engineering practices, so it is of interest to discover how far the adaptation can be taken before the best practices suffer. A skillful reorganization of work practices within the CORFU process can be attempted to align the process with the outcomes of research questions 1 and 2. The research questions for this second part of our research are:

3. How can the proposed work practices be used in the context of another 61499 toolchain and process?
4. What kind of roles need to be manned in this process and what kind of competences are required in these roles?

3. Experimental evaluation of proposed work practices.

3.1. Research approach

Because IEC 61499 is not yet used by our industrial partners, industrial experience has been obtained by inviting professionals to participate on automation design courses [6][7][8]. A one week course was now organized for the third time, with main goal to achieve a working solution for a batch process system presented in Figure 1. The process was a simplified version of the liquor circulation in pulp cooking. It was also much more complex than our previous projects, so significantly more effective work practices were required in order to complete the assignment successfully.

An important observation from our previous research is that the current practice of professional designers is to modify example solutions rather than create original ones. The lack of such legacy software has made it difficult to decide on design principles or to appreciate the potential efficacy of IEC 61499 based development
Therefore, partial solutions and design principles were provided by organizers. Before the course, this background information was transferred to invited participants, who were given the roles of project manager, system engineer and test engineer.

The sequential logic for the automation of the process was specified with the graphical Procedure Function Chart (PFC) notation [9] [10]. The PFC specifies four levels of sequential control: procedure, unit procedure, operation and phase. Each level is implemented with one or more constructs from the lower level. A unit procedure contains the sequence for one part of the process such as filling a tank or cooking under certain temperature and pressure. The procedure was composed of a sequence of five unit procedures: impregnation, black liquor fill, white liquor fill, cooking and discharge. The PFC notation for White Liquor fill unit procedure is shown in Figure 2. The solution was developed with the Function Block Development Kit (FBDK) [11].

Designers were shown how to implement PFC specifications by modifying existing FBs that realized similar PFCs. After this, designers were independently able to develop FBs according to PFC specifications by copying and modifying existing blocks. No formal rules were presented to designers, since professional designers also rely on tacit knowledge instead of formal rules [1]. In the examples, operations are represented with composite FBs and the fork and join operations are represented as event-split and event-rendezvous FBs. The transition condition is represented as DI_gate, which is a special purpose FB. A detailed description of the mapping from PFC to FBs is given in [12].

Empirical data was gathered during the course by three researchers doing field notes based on a question list. Based on the research questions 1 and 2 and our previous experience, the empirical material was first coded with four codes: Team organization, Task and resource allocation, Coordination and knowledge integration, Technical issues. Items related to software development methodology were coded according to the following software development activities: Architectural Design, Detail Design, Implementation and Integration and Testing. This analysis was performed for both teams and the results were compared to the interviews conducted in companies. A three dimensional set of qualitative data was obtained, and most relevant parts are displayed in several two dimensional tables. It was discovered that the “Task and resource allocation” code overlapped with the “Team organization” code, so these were merged into one table due to limited space in this paper. Coordination and knowledge integration will be addressed in another paper with a different focus.

3.2. Results of the event

The 16 participants of the course were divided into two teams. Each team’s project manager, system engineer and test engineer were required to divide the work into tasks and to make a plan for using the resources, pairs of designers, to implement the PFC specifications in the given time. The PFC specifications and some example solutions were given to the groups but otherwise the design decisions were left to groups.

The following three subsections (Software development method, Team organization and Technical issues) address some difficulties and problems that the teams encountered. Each subsection includes a table in which the subject is further divided according to the development activities (Architectural design, Detail design, Implementation & Integration and Testing). The work practices on the course are then compared with corresponding industrial practice.

3.2.1 Software development method. The architectural design in both teams was strongly based on the PFC specification which was given as initial data. In architectural design the system is decomposed hierarchically into smaller units. The interfaces between these units are specified. This approach doesn’t require that architectural design and implementation are done in the same tool, but a common tool will minimize the amount of work. Team A trusted the accuracy of the PFC specification and started with the lowest level operations, which were all implemented before proceeding with the integration of higher level unit procedures.

Team B had a more goal-driven method: they started with the operations that were needed in order to accomplish the first unit procedure, so they could integrate and test unit procedures earlier.
In the industry, application design consists of configuring and wiring existing blocks that have a greater level of abstraction than the blocks in the FBDK libraries. This was considered to be a major difference between the course and industrial delivery projects. There should be libraries for function blocks like motor controls, level control with a valve and a pump etc. Without such a library it is difficult to compare the tool with the existing tools, and the IEC 61499 standard can also easily be perceived as low level programming.

The decisions over the testing policy were difficult because the development environment was not familiar. Team A decided to first test all operations by constructing corresponding tester function blocks. Editing the tester block was straightforward and fast, but the testing of such low level blocks didn’t expose any faults. When the team decided to change their testing policy and skipped the tests for remaining operations, one critical fault was immediately found when testing with the real equipment. The fault was in the wiring of the blocks and it would have been identified in the module testing.

Team B decided not to test operations separately, with the exception of more difficult control loops. When the operations were integrated into a unit procedure, the sequence was tested with the real equipment. Although team B gained some advance by skipping operation testing and performing only unit procedure testing with the real equipment, it has to be noted that real equipment is usually not available for tests in a delivery project.

### 3.2.2 Team organization

The team roles were fixed before the event. A project manager, a system engineer and a test engineer were selected for both teams and they were given a two-day introduction. In the beginning of the course, the project manager divided the rest of the team into pairs of designers. The team organization matched real projects, although it is quite common that the testing is carried out by the designer himself, not a separate testing engineer, and the title “commissioning engineer” would suit this role better.

The only thing that wasn’t realistic from the industrial point of view was pair programming. Designers are used to working alone. Pair programming was nevertheless advocated by course organizers in order to overcome the additional communication and problem solving challenges that were imposed by the unfamiliar tool and standard; pair programming has been used successfully by agile teams especially when the project and designs cannot be planned at the beginning [13]. Pairs on the course performed satisfactorily and only one pair decided to divide tasks, which resulted in an increased need for supervision and guidance from the system engineer.

The programmers who worked alone didn’t follow the guidelines as carefully as those programming in pairs. Any burden on the system engineer can be a serious problem with this industrially accepted team organization, because the ability to integrate the system relies on the vision of this individual for the system-wide design.

---

Table 1. Software Development Method in course and in industry divided in software development activities; (A)rchitectural design, (D)etail design, (I)mplementation & (T)echnology.

<table>
<thead>
<tr>
<th>Team A</th>
<th>Team B</th>
<th>Industrial Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan: top-down design by complementing the skeleton-model and defining interfaces of the FBs, then implementation. Team ended up trusting PFC specification fully and started from lowest level components.</td>
<td>Identification of difficult tasks and dependencies between tasks. Example solution’s architecture needs to be extended (controllers can’t be inside the operation).</td>
<td>Architecture design is based on previous, similar projects. Legacy code defines extensively the architectural design by library elements and their interfaces.</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choice about the state of reuse should be made before designing components. Designers have more responsibility, because sufficient libraries aren’t available.</td>
<td>In automation delivery project, only wiring of the blocks from libraries, parameterizing and testing should be enough.</td>
<td>Modifying internals of a FB, like algorithms, or implementation of low level FBs belongs to R&amp;D.</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiled and tested components are saved in common folder. Test engineer gets the proper version of components to launchable system for equipment testing.</td>
<td>The example solution didn’t address all the architectural issues involved. Integration failed because developers used architectural elements differently.</td>
<td>In automation delivery project, only wiring of the blocks from libraries, parameterizing and testing should be enough.</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software testing before testing with the process. Test engineer did the first example of the test-FB module. The same pair programmed the FB, corresponding test-FB and performed the test. The use of test-FBs straightforward but took some time.</td>
<td>Skipped module testing: not very important and takes too much time. Later they decide to test controller modules. The testing environment is built up gradually during testing. The testing policy was ok, but some more thought could have been given to test environments.</td>
<td>Testing with real equipment is often impossible before delivery. Three levels can be identified: 1) component testing with debugging features, simulator or formal verification 2) system testing with final execution environment with simulated I/Os and 3) testing with the process equipment.</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2. Team Organization in course divided in software development activities; (A)rchitectural design, (D)etail design, (I)mplementation&Integration, (T)esting.

<table>
<thead>
<tr>
<th>Team</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>System engineer is responsible for the architecture.</td>
</tr>
<tr>
<td>B</td>
<td>Much of the architectural design was already made in PFC and mapping rules.</td>
</tr>
<tr>
<td></td>
<td>Project manager and system engineer roles often the same person. Separate PM is more business oriented.</td>
</tr>
<tr>
<td>D</td>
<td>Designers worked in pairs.</td>
</tr>
<tr>
<td>I</td>
<td>System engineer observed the implementation.</td>
</tr>
<tr>
<td>T</td>
<td>Test Engineer did test-FB template.</td>
</tr>
</tbody>
</table>

Table 3. Technical Issues in course divided in software development activities; (A)rchitectural design, (D)etail design, (I)mplementation&Integration, (T)esting.

<table>
<thead>
<tr>
<th>Team</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Unfamiliarity of the tool and new method caused difficulties. The concept of composite block some what unclear.</td>
</tr>
<tr>
<td>D</td>
<td>Pairs tend to model concurrent execution as a series of FBs, not with E_Splits.</td>
</tr>
<tr>
<td>I</td>
<td>The concept of instance somewhat unclear for some of the designers.</td>
</tr>
<tr>
<td>T</td>
<td>Unfamiliarity with the tool caused uncertainty about the proper testing method.</td>
</tr>
</tbody>
</table>

4. Experiment with Corfu ESS

A different approach was adopted by the UniPat team using Corfu ESS (http://seg.ece.upatras.gr/corfu) and Archimedes System Platform [14]. The team, which was not aware of the results of the event that was organized by TKK, was composed of two members: one familiar with Corfu ESS and IEC61499 and the other with limited knowledge on IEC61499 and Corfu ESS, but a good knowledge on software engineering. The requirements for the UniPat team were given by the organizers of the TKK event; it included: a) the description of the Festo MPP mechanical process, and b) the PFC specifications of the process to be automated. The first attempt was to apply the hybrid approach [15] as it was used in previous case studies and experiments, such as the Festo MPS and the Robotic Arm case studies (http://seg.ece.upatras.gr/seg/dev/iec61499.htm). It was recognized very early that the PFC specifications should be the basic source information for software development, instead of the textual specifications. This means that a different approach could have been adopted if the PFC specifications were not available.

PFCs were used as the source for use case identification and description. The first approach was to define a use case for each event of any PFC describing a unit procedure. However, this approach resulted in a rather false application of the use case concept in the Festo MPP, since no modular reusable units of functionality were defined. This was the motivation for proposing a
slightly different technique of using use cases in batch systems [16]. A use case, according to this approach, may be composed of a set of sub-use cases that results in a hierarchical realization of the system’s control. The control logic of each sub-use case is captured by a corresponding controller, while the controller corresponding to the main use case captures the logic for the coordination of sub-use case controllers.

The so defined process is, of course, quite unknown to industrial engineers so they have to be trained on using it. However, given the simplicity of the use cases concept, their wide adoption by the software industry and their support by any available general purpose CASE tools, such training could be affordable when appropriate tools are adopted by the industry. System engineers are required to identify and describe use-cases, while the other designers in the project have a much simpler task that is to read use cases and understand their relation to the corresponding automatically generated FB network (FBN) diagrams.

Each mechanical unit will be accompanied by its FB type and this simplifies the task of system engineers during the development process. FB’s of mechanical units will be constructed by the unit’s vendor to a) encapsulate the logic required to control the specific units and b) provide an IEC61499-compliant interface. These vendors’ FBs will be located in FB type repositories organized in categories, so the toolset and the developer can easily locate and use them. The need to search for required FB types based on many different search criteria on FB type characteristics is very crucial and should be supported by the toolset. However, in contrast to the TKK event, there were no other FB types available, so the designers were responsible for identifying and describing the required FB types. Unfortunately, the experience gained from already developed case studies was not applicable to the batch system domain for the identification of FB types. The methodology was required to be modified to this direction too.

Implementation and integration are semi-automatically supported by Corfu. The implementation model is obtained automatically by using model-to-model transformers [16]. Integration of FB instances to form the control application is also done automatically by the tool, merging selected FBNs.

The job of testing was much simpler, compared to the TKK experiment, since Corfu fbdk provides a run-time environment, the Corfu FBRTE, that allows for the step-by-step execution of FBNs without the need to construct extra testing FBs or additional user interface screens. This environment can be used not only by testers to verify the design models, but also by the developers of FB types and FBNs, since it provides a very simple user interface. Both simple and composite FB types are supported and a flexible way is provided to define the input events to be triggered, as well as the internal data of FBs and the FBN outputs to be monitored. This means that the industrial practices on testing captured in table 1, can be fully applied. FB instances as well as FBN diagrams were tested using Corfu FBRTE. The implementation model of the control application was tested next using an application specific simulator that was developed for this purpose.

During the definition of the design methodology, several constraints imposed by the development toolkit were identified. The most important limitation was the tools inability to capture different levels of abstraction in the form of leveled FBN diagrams. This strengthens our finding from the Festo MPS case study that such a leveling should greatly simplify the development process.

It should be noted that the proposed leveling in FBNs facilitates the discrimination of activities performed by team managers, system engineers and industry’s professional designers. These roles work at different levels of abstractions that are captured by the different levels of the leveled FBN. According to the team organization described in subsection 3.2.2, system engineers produce the higher levels of abstraction that capture the architectural decision. They also decide which lower layer design decisions to lay aside for professional designers. Team managers mainly use the higher layers of abstraction to understand and coordinate the progress of the entire team. Industry’s professional designers can, as is the case with the TKK event, reuse existing example solutions from the projects’ library and also generate new FB types by modifying existing ones. Similar applications from previous projects can also be utilized to reuse and exploit their architecture for the definition of the architecture of the new system.

The need for a leveled FBN diagram was recognized immediately after the merging of the FBN diagrams. These diagrams were automatically generated from the OIDs (object identifier) that were constructed for the system’s use cases. The resulting FBN was so complicated that it would be impossible for the industrial engineer to understand it, to check it for correctness, or even to make minor modifications. The exploitation of the corresponding findings from Festo MPS was the first choice.

Corfu ESS in its current version does not automatically support leveling, but this can be supported through the creation of the higher layer FBNs from existing FB instances. This is like having different views of the same flat FBN. Each view represents only selected FB instances and is created to serve a specific purpose. Such a FBN is the Impregnation FBN that is created by fetching into the FBN workspace the instances of the controllers of each one of the constituent sub-use cases, i.e., the fill the digester, pressurize and depressurize controllers, along with the impregnation controller. The composite FB type can also be used with some restrictions for the realization of the leveling concept [15].

To get more simple FBNs, the concept of using- and implementation-interfaces can be exploited. The using-interface for a FB is the part of the event and data interface that is used in the upper level diagram that the FB appears in. The implementation-interface, that is the
remaining part of the interface, will be used in the lower level of abstraction where the FB type will be implemented to provide the functionality assumed in the upper layer. [15]

Implementation-interfaces such as the open and close event outputs of the fill digester controller can be switched off since they provide no significant information to this view (level). The implementation interface plays the primary role in the fill the digester view of the FBN that represents the FB instances participating in this case use and the way they collaborate under the coordination of the fillDigesterController. By hiding some information, the tool offers better support for different roles in the development team. System engineers perceive the architecture easier when the details are hidden, while the designers need only to handle using-interfaces in normal situations.

The whole Festo MPP control application can be downloaded from http://seg.ece.upatras.gr/seg/dev/FestoMPP.htm to get a better view of the way that layering is currently supported by Corfu ESS.

5. Discussion

In this paper, a three dimensional set of qualitative data has been analyzed to contrast emerging work practices with IEC 61499 with established industrial work practices; parts of the data have been presented in several tables. There are several ways to organize an experiment for obtaining answers to research questions 1 and 2. The chosen arrangements build on previous experience to obtain positive answers; the goal was not to obtain an impartial evaluation of industrial applicability but to propose, pilot and evaluate practices that would be considered most feasible by organizers. This resulted in an experiment that resembled industrial practice much more than our previous experiments with the same IEC 61499 development tool. The work practices that were used successfully are presented in the tables; practices are divided into the categories of architectural design, detail design, implementation&integration and testing.

Research question 1 was “Can current automation design work practices be supported by 61499 tools?” According to our previous experiments, it is not realistic to expect professionals to independently apply familiar work practices with fbdk. Based on this most recent experiment, a positive answer to research question 1 has been obtained by training team members in the practices that are performed by the role that they man in the software development team. This strategy avoids debates on design alternatives that have no place in a time pressured delivery project. Tool support for current automation design work practices requires that application domain specific blocks are found in the library. Industrial work practices rely on such blocks, since designers work very quickly if they only need to retrieve blocks from a library, wire blocks and configure blocks; R&D maintains the library. A skeleton library was provided for the course, but a fully positive answer for research question 1 can only be obtained by doing the work of the R&D department and providing a more extensive library.

Research question 2 was “What new or modified work practices need to be introduced?” The tables show a match between practices on the course and industrial practice. The main differences were due to a lack of an industrial strength FB library of legacy software, and this resulted in the delivery project team doing some tasks that normally belong to the R&D department. Another difference was that the manager only needed to watch schedule and resources, since no client negotiations about system requirements were needed. New work practices were needed for testing, because these are contingent on the tool support for debugging rather than on features of the IEC 61499 standard; a module testing work practice with the fbdk naturally relies on the user interface that the tool creates for a FB that is launched.

Research question 3 was “How can the proposed work practices be used in the context of another 61499 toolchain and process?” The experiment with Corfu ESS helped move the CORFU methodology from one man projects to team projects performed by actors with limited and heterogeneous expertise. The layering of the FB network and the possibility to hide unnecessary information from the interface of FBs supports the different roles in the team; such means are essential for limiting the amount of architectural, application-specific and IEC 61499 related knowledge required from personnel manning the various roles. It is thus possible to divide the project into simpler work practices as performed by these roles in the industry.

Research question 4 was “What kind of roles needs to be manned in this process and what kind of competences is required in these roles?” The team organization that was used in the TKK event consisted of project manager, system engineer, test engineer and designer pairs. This was commented to be realistic from the industrial perspective. Project manager has to be business oriented and have managing skills. System engineer needs knowledge of the controlled process, the runtime system and component libraries. He is also responsible for architectural decisions. All designers don’t have to be experts in IEC 61499 if there is a proper library of components, good example solutions and the supervising of the designers is organized. Pair programming strengthens the capability of designers even if both the designers are inexperienced; for efficiency reasons, pairs are unlikely to be used in the industry after the designers are experienced with the IEC 61499 toolset.

The roles and competences required when using Corfu ESS are quite similar, but instead of strong experience from function blocks, a more object oriented profile of the designers is preferred.

The key to managing any unfamiliar work practices is that responsibilities are clearly allocated to the team’s leader roles, and that these roles have been given sufficient IEC 61499 related training for these work practices. For example, system engineers were presented the rationale for design alternatives, so they were able to
adapt the architecture to emerging problems without being able to resort to legacy solutions from previous projects or a general purpose architecture provided by R&D.

6. Acknowledgment

The research part done in Finland was carried out as a part of USVA research project founded by Tekes (National Technology Agency of Finland) and several Finnish automation enterprises. Thanks are due to Dr. James Christensen who has taught us facts and practices about IEC 61499. We especially thank the participants of the IEC 61499 summer course for taking part in our research.

References