Helsinki University of Technology
Information and Computer Systems in Automation

Espoo 2007 Report 13

Adapting Problem Solving Strategies in Control Software Development to New Standards: Case Examples IEC 61499 and ISOBUS

Seppo Sierla

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D´HELSINKI

Helsinki University of Technology
Information and Computer Systems in Automation

Espoo 2007 Report 13

Adapting Problem Solving Strategies in Control Software Development to New Standards: Case Examples IEC 61499 and ISOBUS

Seppo Sierla

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Automation and Systems Technology, for public examination and debate in Auditorium K at Helsinki University of Technology (Espoo, Finland) on the 14th of December, 2007, at 12 noon.

Helsinki University of Technology
Department of Automation and Systems Technology
Information and Computer Systems in Automation

Distribution:
Helsinki University of Technology
Department of Automation and Systems Technology
Information and Computer Systems in Automation

P.O. Box 5500

FIN-02015 HUT, Finland

Tel. +358-9-451 5968

Fax. +358-9-451 5394

ISBN 978-951-22-9104-5

ISSN 1456-0887

Picaset Oy

Helsinki 2007

ABSTRACT OF DOCTORAL DISSERTATION

HELSINKI UNIVERSITY OF TECHNOLOGY
P.O. BOX 1000, FI-02015 TKK
http://www.tkk.fi

Author Seppo Sierla

Name of the dissertation

Adapting Problem Solving Strategies in Control Software Development to New Standards: Case Examples IEC 61499 and ISOBUS

Manuscript submitted 16.10.2007 | Manuscript revised 22.11.2007

Date of the defence 14.12.2007

☐ Monograph | ☒ Article dissertation (summary + original articles)

Department Department of Automation and Systems Technology
Laboratory Information and Computer Systems in Automation
Field of research Software Development
Opponent(s) Prof Seppo Kuikka, Dr Harri Happonen
Supervisor Kari Koskinen
Instructor Kari Koskinen, Janne Hukkinen

Abstract

New standards in software development can significantly constrain teamwork, and practical success in applying a new standard in a pilot project is therefore more than a technical problem. This dissertation focuses on teamwork specifically from the aspect of problem solving. In two case examples in which the new standards IEC 61499 and ISOBUS were adopted, it is observed that preferred or familiar problem solving strategies can be unsuccessful when work is constrained by a new standard. Adaptations to problem solving strategies are required for successful control software development, yet these changes should permit a team to continue working in a way that has been successful in the past. A compromise is needed between familiar practices and new practices that would best support the use of the new standard. This dissertation studies the emergence of such practices in the context of two standards: IEC 61499 and ISOBUS.

Case studies focus on projects, in which there was a requirement to adopt a new standard: either IEC 61499 or ISOBUS. It was presumed that the teams would make attempts to use problem solving strategies that would match the nature of the standard adoption problem. Such attempts were observed only in some situations; participants were inclined to use problem solving approaches that had been successful in the past, even though the nature of problem to be solved had changed due to the new standard. Hence, there are two disparate and possibly conflicting forces at work in a standard adoption project: firstly, there is an attempt to orient problem solving according to the new problem, and secondly there is a tendency to cling to familiar practices that have been successful in the past.

Two theoretical frameworks, knowledge networking and technological frames, are used to analyze these two disparate aspects of standard adoption in a software development project. The frameworks are first applied separately and then some conclusions are drawn regarding how these analyses can complement each other to address the two aspects of standards adoption mentioned above. The results suggest how problem solving strategies can be assessed before a project, taking into consideration both the nature of the standard adoption problem and the capability and willingness of the team to follow a particular problem solving approach.

Keywords Standard, software development, problem solving strategy, IEC 61499, work practice

ISBN (printed) 978-951-22-9104-5 | ISSN (printed) 1456-0887

ISBN (pdf) | ISSN (pdf)

Language English | Number of pages 130

Publisher TKK Helsinki University of Technology, Information and Computer Systems in Automation

Print distribution TKK Helsinki University of Technology, Information and Computer Systems in Automation

☐ The dissertation can be read at http://lib.tkk.fi/Diss/

VÄITÖSKIRJAN TIIVISTELMÄ	TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi

Tekijä Seppo Sierla

Väitöskirjan nimi

Ohjausjärjestelmien ohjelmistonkehityksen ongelmanratkaisustrategioiden soveltaminen uusiin standardeihin: case esimerkkeinä IEC 61499 ja ISOBUS

Käsikirjoituksen päivämäärä 16.10.2007	Korjatun käsikirjoituksen päivämäärä 22.11.2007

Väitöstilaisuuden ajankohta 14.12.2007

☐ Monografia	☒ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)

Osasto	Automaatio- ja systeemitekniikan osasto
Laboratorio	Automaation tietotekniikka
Tutkimusala	Ohjelmistonkehitys
Vastaväittäjä(t)	Prof Seppo Kuikka, TkT Harri Happonen
Työn valvoja	Kari Koskinen
Työn ohjaaja	Kari Koskinen, Janne Hukkinen

Tiivistelmä

Uusien standardien käyttö ohjelmistonkehitysprojektissa voi asettaa merkittäviä reunaehtoja projektin ryhmätyön ongelmanratkaisulle. Väitöskirjatyössä tutkittiin uusien IEC 61499 ja ISOBUS standardien käyttöönottoa ja havaittiin, että nämä voivat rajoittaa tuttujen ongelmanratkaisustrategioiden käyttöä siinä määrin, että standardeja pilotoiva projekti ei saavuta kaikkia tavoitteitaan. Strategioita muutettaessa tarvitaan kompromissi uuden standardin määrittelemään ongelmaan sopivien ja entuudestaan tuttujen ongelmanratkaisustrategioiden välillä. Väitöskirjatyössä tutkitaan tällaisten strategioiden syntymistä kahden standardin kontekstissa.

Case tutkimukset on tehty projekteihin, joissa oli vaatimuksena ottaa käyttöön uusi standardi: joko IEC 61499 tai ISOBUS. Projektiryhmien oletettiin pyrkivän käyttämään ongelmanratkaisustrategioita, jotka soveltuisivat uuden standardin käyttöönotto-ongelmaan. Tällaisia yrityksiä havaittiin vain joissain tilanteissa, koska tekijöillä oli taipumusta käyttää entuudestaan tuttuja ja hyväksi havaittuja ongelmanratkaisustrategioita. Näin kävi siitä huolimatta, että ongelman luonne oli muuttunut uuden standardin käyttöönoton takia. Tutkimuksessa pyritään huomioimaan nämä kaksi ongelmanratkaisuun vaikuttavaa tekijää, jotka eivät ole yhteismitallisia: toisaalta esiintyy pyrkimystä orientioda projektin ongelmanratkaisua uuden ongelman mukaan, mutta toisaalta on taipumusta pitää kiinni tutuista toimintamalleista.

Yllämainittuja kahta ongelmanratkaisun aspektia analysoidaan kahdella teoreettisella viitekehyksellä: tiedon verkottumisella ja teknologisilla kehyksillä. Näitä on sovellettu erikseen ja tämän perusteella on päädytty johtopäätöksiin kuinka kumpaakin viitekehystä soveltamalla voidaan vastata molempiin yllämainittuihin uuden standardin käyttöönottoprojektin ongelmanratkaisun aspekteihin. Johtopäätöksissä esitetään, miten tulokset voidaan ottaa huomioon uuden standardin käyttööottoprojektia suunniteltaessa.

Asiasanat Standardi, ongelmanratkaisustrategia, ohjelmistonkehitys, IEC 61499, toimintaprosessi

ISBN (painettu) 978-951-22-9104-5	ISSN (painettu) 1456-0887
ISBN (pdf)	ISSN (pdf)
Kieli Englanti	Sivumäärä 130

Julkaisija TKK, Automaation tietotekniikka

Painetun väitöskirjan jakelu TKK, Automaation tietotekniikka

☐ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/

Preface

A doctoral student at this university will face some compromises before the dissertation is finished. One approach is to concentrate primarily on the dissertation and to worry about issues like project funding after graduation. Another approach is to give greater emphasis to project work, even when these activities do not directly contribute to a cohesive thesis. Now that the permission for publication of this dissertation has been obtained, I can be pleased about having chosen the latter approach.

Espoo, 20 Nov 2007

Seppo Sierla

Table of Contents

List of Publications

Author's individual contribution for each publication is stated in underlined italics

I. Bruun, Henrik; Sierla, Seppo (forthcoming): Distributed Problem Solving in Software Development: The Case of an Automation Project. Social Studies of Science. 26 pages. *Responsible for case study and empirical work, primarily responsible for analysis and jointly responsible for discussion.*

II. Strömman, Mika; Thramboulidis, Kleanthis; Sierla, Seppo; Papakonstantinou, Nikolaos; Koskinen, Kari: Incorporating Industrial Experience to IEC 61499 Based Development Methodologies and Toolsets: ETFA'2007 12th IEEE International Conference on Emerging Technologies and Factory Automation, September 25-28, 2007, Patras, Greece. 8 pages. *Responsible for research problem, literature review and discussion.*

III. Sierla, Seppo; Christensen, James; Koskinen, Kari; Peltola, Jukka: Educational Approaches for the Industrial Acceptance of IEC 61499: ETFA'2007 12th IEEE International Conference on Emerging Technologies and Factory Automation, September 25-28, 2007, Patras, Greece. 8 pages. *Jointly responsible for technical work. Fully responsible for the rest.*

IV. Thramboulidis, Kleanthis; Sierla, Seppo; Papakonstantinou, Nikolaos; Koskinen, Kari: An IEC 61499 Based Approach for Distributed Batch Process Control: IEEE International Conference on Industrial Informatics, July 23-27, 2007, Vienna, Austria. 6 pages. *Responsible for literature review and case study.*

V. Peltola, Jukka; Christensen, James; Sierla, Seppo; Koskinen, Kari: A Migration Path to IEC 61499 for the Batch Process Industry: IEEE International Conference on Industrial Informatics, July 23-27, 2007, Vienna, Austria. 6 pages. *Responsible for literature review and discussion.*

VI. Strömman, Mika; Sierla, Seppo; Peltola, Jukka; Koskinen, Kari: Professional designers' adaptations of IEC 61499 to their individual work practices: ETFA'2006 11th IEEE International Conference on Emerging Technologies and Factory Automation, September 20-22, 2006, Prague, Czech Republic. 7 pages. *Responsible for literature review and research method.*

VII. Peltola, Jukka; Sierla, Seppo; Strömman, Mika; Koskinen, Kari: Process Control with IEC 61499: Designers' Choices at Different Levels of the Application Hierarchy: IEEE International Conference on Industrial Informatics, 16-18 August 2006, Singapore. 6 pages. *Responsible for literature review, research method and discussion.*

VIII. Strömman, Mika; Sierla, Seppo; Koskinen, Kari: Control Software Reuse Strategies with IEC 61499: ETFA'2005 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catalania, Italy, September 19-22, 2005. 8 pages. *Responsible for literature review, research method and discussion.*

IX. Sierla, Seppo; Peltola, Jukka; Koskinen, Kari: Real-Time Middleware for the Requirements of Distributed Process Control, 3rd IEEE International Conference on Industrial Informatics INDIN'05, Perth, Australia, August 10-12, 2005. 6 pages. *Responsible for literature review, research problem, technical work and conclusions.*

1 Introduction

1.1 Background

New standards can offer better support for certain technical problems in software development, but the benefits that are realized by the development team depend on the adoption process. Successful use of new technology often involves creating new work practices or modifying existing ones. If this cannot be accomplished in a limited time, the developer community might not accept the new standard regardless of the potential benefits. New standards seldom come with problem solving and troubleshooting instructions; many researchers agree that this is not even possible due to the situational nature of software development (Goulielmos, 2004; Truex et al., 2000).

The creation of new practices as an outcome of collective learning has been extensively studied, for example, in the literature on communities of practice (Wenger, 1998) and distributed cognition (Hutchins, 1995). While such research is used as background information, the focus in this dissertation is on software development and the work practices involved. In particular, how are successful practices established? The goal is to support developers of new technology who would like to see their work accepted by practitioners. The objects of study are small teams of developers, and the processes of creating, modifying or appropriating new work practices is approached from the perspective of interaction, communication and division responsibility within the team.

Two standards have been chosen as cases. The IEC 61499 function block standard for industrial process measurement and control systems (IEC, 2005AB; see also Lewis, 2001) aims to be a successor to the IEC 61131-3 function block standard that has gained industrial acceptance (Lewis, 1998); IEC 61499 has incorporated many object oriented concepts (Lewis, 2001) and thus represents a paradigm shift for professional automation designers. The ISOBUS standard for agricultural implements (ISO, 2003) defines standardized interfaces and data communications between system parts, thus enabling plug and play integration of parts developed independently by different vendors. The developer of a software module will therefore have no need or possibility to interact with the developers of other modules that need to be integrated together, so new work practices for testing are required.

1.2 Summary of the Publications

The following summaries focus on the author's individual contributions.

Publication I aims at establishing correlations between problem solving strategies that are chosen by a team of developers and the structure of the problem to be solved. The focus is on integrating knowledge that is distributed among people and artifacts, and the theoretical framework of knowledge networking is used to distinguish between three problem solving strategies for integrating this knowledge. Empirical material has been

gathered with semi-structured interviews in order to obtain the participants' views on the structure of the problem that they were solving and their strategy of collaborating with colleagues. The analysis uses this material extensively to identify the knowledge networking approaches at various stages of the project. Problem structures changed over the course of the studied software development project, and the analysis revealed correlations between problem structure and knowledge networking mode. This work is used as a basis for the material in sections 4.2 and 6.

Publications II, III, IV and V refer to the 2006 project experience with IEC 61499 that is described in sections 4.1.3 and 5.2. The literature reviews of these publications assess the obstacles and opportunities for industrial adoption of the IEC 61499 standard. The reviews have identified the minimization of switching costs to the new standard as a research goal:

- Publication III addresses the element of the switching cost that is caused by the need to retrain software developers to use the new standard and related tools. The theoretical framework of technological frames is applied here, and sections 2.3 and 5.2 are directly based on publication III.
- New work practices are proposed for the purpose of reducing switching costs, and their relevance to other IEC 61499 compliant development environments is addressed in publications II and IV.
- Publication V aims at decreasing the switching cost by extending the IEC 61499 compliant development environment in order to support work practices that resemble industrially accepted practice. This work focused on setting up a development environment for the event described in section 5.2.

Publications VI and VII refer to the 2005 project experience with IEC 61499 that is described in sections 4.1.2 and 5.1. However, 5.1 is directly based on publication III, which is based on a more thorough analysis of the empirical material. The literature review in section 2.1 is directly based on publication VII. Publication VII also contains detailed technical descriptions of some software module design and integration problems, which should be read by those interested in a more thorough understanding of the discussion in 7.1 and 7.3.1.

Publication VIII presents the initial experience with IEC 61499 and is based on the observation that the new standard offers a much broader range of design options than the conventional PLC programming paradigm. It is concluded that there is no obvious way to rank alternative designs, and this provided the motivation for the explorative project arrangements described in publications VI and VII.

In publication IX, the publish-subscribe and client-server communication mechanisms were studied; these were used extensively in the IEC 61499 based application development described in publications II-VIII. These publications emphasize the importance of being able to integrate software modules that have been developed by different team members. The software integration is part of knowledge integration within the team, and it is constrained significantly by the technical details of interfaces. Many of these interfaces were realized with the publish-subscribe or client-server communication

mechanisms. Publication IX addresses how these mechanisms can be used to address the communication requirements of process control applications, and how they can be mapped to IEC 61499 service interfaces.

2 Literature Review and Theoretical Frameworks

This section presents the body of literature against which the dissertation's contribution will be made. Section 2.1 explores the scope on non-technical issues that affect the acceptance of an IT standard, and concludes by narrowing the scope to a research problem; the problem is further refined to some research questions in section3. Section 2.2 addresses software development processes, which are the work practices that need to be modified during standard adoption. Section 2.3 presents technological frames as the theoretical framework for the IEC 61499 case study. Section 2.4 presents knowledge networking as the theoretical framework for the ISOBUS case study.

2.1 The Adoption of Emerging Standards

This section explores the scope on non-technical issues that also affect the acceptance of a standard by a market, using IEC 61499 as an example. The industrial automation market involves players ranging from manufacturers of small software-intensive devices to system integrators working with various factory and enterprise level information systems. IEC 61499 is a new standard for software architecture and interfaces that enables the exchange of components between different development tools and organizations. Such standards can facilitate the design of marketable software modules (Hahn and Turowski, 2004) that can be reused in a number of applications, and IEC 61499 underpins a vision for an open knowledge economy for industrial automation (Vyatkin et al, 2005). This economy is centered on repositories of intellectual property that is encapsulated into components and accessible to all stakeholders. Vendors from device manufacturers to enterprise system integrators will be able to make their products available to a broad range of potential clients via the repositories; clients will be offered a greater selection of interoperable software products without being locked into a single vendor.

However, despite the promising vision of a market that is based on open global standards, the IEC 61499 has not yet gained the necessary support from industry, even though a number of academic players have contributed solutions ranging from run-time platforms to development tools (Ferrarini et al., 2005; Thramboulidis and Zoupas, 2005). Current industrial practice is still based on the earlier standard, IEC 61131. This was created at a time when several proprietary solutions had already been developed, so major vendors differentiate their products with extensions to the standard that prevent the straightforward integration of software components from several sources. Differentiation based on nonstandard extensions or the standardization of ones own proprietary practice are typical strategies for blocking competitors from the market (Egyedi, 2001; Lee and Oh, 2006), resulting in decreased market value for the entire industry (Hurd and Isaac, 2005). However, it is difficult to bring about changes after significant investments have been made (Browne and Menon, 2004), and IEC 61499 will need to replace established

proprietary extensions of IEC 61131-3. Single players are likely to realize that it is in their advantage to use proprietary solutions that have begun to dominate markets, even if such collective behavior leads to long-term social losses when the benefits of open standards are not realized (Browne and Menon, 2004).

It is helpful to consider the problem of standard adoption from the perspective of one company or a domestic industry. Even if the benefits of the new standard are unquestionable after international acceptance (Browne and Menon, 2004; Hurd and Isaac, 2005), the adoption process is unlikely to start unless individual companies have sufficient short term incentive that outweighs the risk of the standard failing. A firm has a strong motivation to hold onto existing investments in knowledge, tools and legacy software based on familiar standards and practices (Zhu et al., 2006), so the switching cost to IEC 61499 will be lowered significantly, if some of this investment can be retained.

IEC 61499 is a general-purpose architecture, while industrial practice is based on tools that explicitly support a specific domain such as manufacturing, batch or continuous process control. In order to discover possibilities for industrial adoption of IEC 61499 in one of these domains, a feasible migration path with minimal switching costs from current industrial practice to IEC 61499 based practice should be identified. The batch control domain has been selected for more detailed investigation. One strategy for minimizing switching costs is to use the established standards, concepts and terminology from this domain.

Any decision to upgrade practices will be heavily influenced by the existing investments of a company in terms of knowledge, development tools and legacy software (Zhu et al., 2006); the cost benefit analysis of a company depends on the extent to which this investment can be utilized in the process of adopting a new standard. Much of this domain knowledge is embodied in the industrially accepted framework for batch process control defined in the IEC 61512 (IEC, 1997) family of standards. The leading commercial batch control systems, InBatch, VisualBatch and OpenBatch, are fairly compliant to these standards as far as models and terminology is concerned (Kuikka, 1999), so any IEC 61512-based design decisions should be easily understood and accepted by professionals.

In this subsection, a number of factors impacting standard acceptance on the macroeconomic level have been identified. One important way to overcome startup problems is to minimize switching costs, which will be the focus in this dissertation. In particular, how can familiar and proven concepts and practices be applied even if new development tools are adopted? The next subsections shift the focus to the software development process, which encompasses the development tasks or work practices as well as the workflows between them. The switching cost problem that is studied here relates to the challenge of developing and testing software based on an unfamiliar new standard; elements of the cost that will be studied in detail are the effort involved in changing attitudes, establishing new work practices and adapting proven teamwork arrangements.

2.2 Adapting Software Development Methods in the Context of Standard Adoption

In order to assess how software development methodology can be adapted to support the use of a standard, it is first necessary to state what is meant by software development methods or processes in the context of this research. Process differs from method in the sense that some authors require that a process is a repeatable way to develop software, so that projects are instances of the process. However, according to SPICE (Software Process Improvement and Capability Evaluation), the requirement for repeatability is only made at level 2, in which repeatability has become a characteristic of the organization. At level 1, any chance of repeating the success of previous projects depends on the experience of individuals (Emam and Madhavji, 2001). There are obvious incentives for reaching level 2, but it can be argued that downplaying human problem solving skills and constraining human judgment by the requirement to follow repeatable processes can result in suboptimal performance in some situations. Agile methods are an example of a different kind of philosophy that has proven successful in many real projects (Beck and Andres, 2004).

Although these is no universally accepted definition of process and method, the term "process" is often used to describe a practice that follows documented guidelines and results in well documented artifacts. Such a process can be the object of an audit and it can be certified. Software development method, however, is a broader term that also encompasses problem solving approaches that are not even intended to stand the scrutiny of an audit. In this sense, using the term "methods" to describe practices that exploit various kinds of internalized, tacit or spontaneous human problem solving is consistent with a large body of literature. This dissertation is very much concerned with individual, undocumented and informal human factors and interactions in software development, so the context of this research is software development methods as described above.

In this dissertation, no prejudice is taken for or against the sets of values that underpin systems such as SPICE or agile methods. The terms process and method are used interchangeably only to refer to the collection of tasks that comprise the software development effort; some workflow between these tasks must naturally exist, but the use of terms process and method hereafter will not imply that the team is following any preplanned or repeatable flow; it is not even assumed that the team is conscious of workflow between different tasks. The remainder of this subsection summarizes a number of research perspectives on software development methods and concludes with a narrowing of the scope for this dissertation.

The focus of much software development literature is on the processing of artefacts: the process consists of tasks or phases in which certain artefacts are received as inputs and other artefacts are delivered as outputs (Sommerville, 2004). The work in any task is performed by a skilled professional working in a role such as a system architect (Jacobson et al., 1999). Several authors have presented models of development processes that identify important tasks and their mutual dependencies (Kruchten, 1995; Douglass, 2004). These analyses extract the context-independent aspects of software development,

so they are highly generalizable. However, the application of such knowledge in a real-life project requires considerable experience and sophistication from the developers (Introna and Whitley, 1997). A context insensitive application of methodology, standards and supporting tools can even hinder efficient problem solving (Nandhakumar and Avison, 1999; Rost, 2005). The potential benefits of introducing CASE (Computer Aided Software Engineering) tools that support and enforce a software development process depend on the competence of the developers (Coupe and Onodu, 1996) and the organizational responsiveness (Orlikowski, 2002).

Other authors have found that software development should not be viewed as a rational engineering process because the team is affected by a number of external factors. Agile software development methods downplay the importance of plans and documents in order to facilitate quick responses to the changing needs of customers (Beck, 1999; Beck and Andres, 2004; Highsmith and Cockburn, 2001). Truex (2000) and Avison (1998) consider it wiser not to propose workflows that can be used with limited adaptation in different projects. Lundell and Lings (2004) present a comprehensive discussion of why software development methods, and the tools that support those methods, are rarely used as intended by the method developers.

These differences might simply reflect a difference in focus rather than contradictory recommendations, since it is broadly accepted that tools and methods need some adaptation by practitioners. However, what is the nature of the adaptation and can it be in any way straightforward? This is a fundamental and longstanding problem in software development research, and it can only be tackled here by defining a specific perspective. The problem of adapting development practices to a new standard is a focus that is shared by both case studies. Each case is focused further on specific software development problems. The case study on IEC 61499 identifies the mental frameworks of software developers and investigates how new standards can be introduced with limited changes to these frameworks. The ISOBUS case looks at how a team's preferred way of solving problems and interacting is impacted and constrained by the requirement to use the standard. The theoretical frameworks for the cases are presented in the next two subsections.

2.3 Technological Frames: A Theoretical Framework for the IEC 61499 Case Study

Professionals, academics, standardization workers, managers and designers have different views on the nature, usefulness and potential of IEC 61499. Educational backgrounds that range from electronics and mechanics to object-oriented software engineering can also deeply mould an individual's appreciation of a new innovation such as IEC 61499. Technical and human aspects of adopting and applying IEC 61499 need to be considered in the same framework. In this section, one possible theoretical apparatus is presented.

It cannot be assumed that an artifact developed by one social group, such as an IEC technical committee, will be adopted and used as such by all other stakeholders. The

social construction of technology (SCOT) rejects a linear view on the development of technological artifacts (Pinch and Bijker, 1987). For example, IEC 61499 is presented as a new and better standard than IEC 61131-3 that should replace or encapsulate the old one in the sense that 61131-3 code can be used as algorithms inside 61499 function blocks. However, it is very possible that an emergent mixture of these and other alternatives such as the UML (Unified Modeling Language) will eventually establish itself as mainstream industrial practice.

According to SCOT, several technological solutions can be offered to social groups that face a problem, such as developing control software. These solutions are embodied in artifacts which are appreciated and adopted differently by the various social groups. In our case, IEC 61131-3, IEC 61499, UML and CASE (Computer Aided Software Engineering) tools are relevant artifacts. Tool vendors can implement concepts in a partial way with additional proprietary twists, and users can violate intended design principles as they adhere to the established practices in their organization (Lundell, and Lings, 2004).

The concept of interpretive flexibility refers to how individuals view and apply an artifact according to their understanding and needs (Bijker, 1987). An important artifact in our experiment was the IEC 61499 compliant CASE tool (FBDK), which was a prototype, since no commercially mature tools were available at that time. Academic participants typically perceived the tool as an environment for experimenting with the new standard and programming paradigm, and this attitude was also encouraged by the course organizers. Industrial participants nevertheless were strongly inclined to evaluate the CASE tool instead of the possibilities of IEC 61499. One possible explanation for these different interpretations is that professionals are only directly involved with tools and are often not conscious about how standard compliant these are.

Due to interpretive flexibility, no objective assessment of the IEC 61499 standard can be expected from professionals. Factors such as switching costs, retraining needs, tool support and investments in legacy code and in-house tools should be taken as a starting point for practical efforts to deploy the standard to industrial development processes. The adoption of IEC 61499 implies that established practices and paradigms are changed or abandoned at least partially. The scope of this change and barriers to it can be understood in greater detail by the concepts of technological frame and inclusion.

A technological frame is comprised of the goals and the methods for reaching the goals, which are shared by members of a social group (Bijker, 1987). Software reuse is one goal; programming paradigms, CASE tool expertise and tacit knowledge are some relevant methods. An individual can be included in one or more technological frames to various degrees; inclusion is a measure of how that person’s definition of goals and knowledge of methods matches the frame. Consequently, strongly included individuals are likely to apply that frame to all problems, since (s)he is impervious to other kinds of goals and methods. A person who is partially included in several frames can combine them in a creative way, but the solution will not necessarily be more elegant or efficient than the strongly included person’s.

In the case study, frames are used to describe the unique experiences of participants. This focus is significantly different from researchers who use technological frames to study organizational change and project management associated with the adoption of new IT (Orlikowski and Gash, 1994; Linderoth and Pellegrino, 2005). Such research treats software professionals as one social group, so the concept of inclusion is not used for understanding the difference between individuals. In our case, considerable heterogeneity was identified among software developers. The analysis of individual interviews suggests that particular programming paradigms are strongly entrenched in the minds of designers. If this situation is not addressed, IEC 61499 will not be used as intended, and negative practical experiences will be obtained.

2.4 Knowledge Networking: A Theoretical Framework for the ISOBUS Case Study

The activity of bringing together knowledge, skills and information residing in a group of people and artifacts has been called knowledge networking by Bruun et al. (2005). See also (Langlais et al., 2004). The meaning of the concept can be clarified by comparison with another, similar term, that of “distributed cognition” (Hutchins, 1995). Whereas distributed cognition refers to collective problem solving in general, knowledge networking specifically designates the knowledge integration or coordination that goes on in such a process. In the literature on distributed cognition, the latter are generally referred to with the concept of “coordination” (Hutchins 1995). The knowledge networking framework allows us to identify different modes of collective problem solving, so it provides a theoretical apparatus for studying changes in the problem solving strategy.

The knowledge networking framework addresses problems that have been studied by many other researchers. The originality and value of the framework as a new scientific contribution is argued in (Bruun et al., 2005; Langlais et al., 2004; Bruun and Sierla, forthcoming). This introduction concentrates on applying knowledge networking as a theoretical framework to analyze the research question described in the next section. This subsection will only present the framework for the purpose of understanding the analysis.

A knowledge network is formed by the interaction within the group and between group members and outsiders. Bruun et al. (2005) identified three different modes of knowledge networking, labeled modular, integral and translational. Modular knowledge networking (MKN) is based on the modularization of knowledge work: each group member focuses on tasks for which they have expertise, and then the outcome of the separate inputs of work is integrated by a co-ordinating agent. For example, in the Unified Software Development Process, component engineers are responsible for developing and maintaining software components or subsystems, but the integration of these artefacts is beyond the scope of any individual component engineer (Jacobson et al., 1999). This is the responsibility of a system integrator, who functions as a MKN co-ordinating agent.

Integral knowledge networking (PKN), in contrast, implies that the agents communicate directly with each other, and thus make knowledge work a joint effort. Different tools and strategies such as boundary objects are used for mediating between the distinct knowledge perspectives. Boundary objects must, according to Star and Griesemer (1989), be both plastic enough to allow divergent uses by different social groups, and robust enough to maintain a common identity across these groups. The diagrams and notations of the UML (Unified Modelling Language) could function as boundary objects, since they can be used to represent business processes and software constructs (Eriksson and Penker, 2000; Marshall, 2000). Shared understandings and mental models develop as individuals are forced to explain and defend their ideas (Saferstein, 1998). Still, in the details of interpretations, there is space for divergence: people do not have to share a mental model completely in order to communicate successfully. (For more on MKN and PKN in science, see Bruun, Hukkinen, Huutoniemi and Klein, 2005).

Translational knowledge networking (TKN), finally, is a kind of compromise between MKN and PKN. It is based on a modularization of work, just like MKN, but combines this with the PKN-feature of direct communication (Bruun et al., 2002). Thus, in contrast to MKN, TKN has the modules in frequent communication with each other. In contrast to PKN, however, the purpose of this communication is not to create a synthetic, shared mental framework, but rather to influence work in the modules. This will allow a greater co-ordination of the contents of modular knowledge production than in MKN, while still preserving the advantages of modularization. The challenge in TKN is to develop a shared language, or a so called interfacing device, for communication between different knowledge perspectives (D'Adderio, 2001). In PKN, time is invested in developing such a language, because the latter is seen as an output with a value in itself. In TKN, however, the shared language has a more instrumental role, and specialized work has priority. In contrast to the boundary object, the interfacing device must be used in a similar way by all communicators, because deviations for the standard will cause communicative problems. An example of such a shared language is the RosettaNet standard, which supports B2B (business to business) systems integration among players in a supply chain (Sundaram and Shim, 2001). Now that the specifications are stable, RosettaNet defines business processes and technical specifications for data exchange, so developers within an enterprise do not need to negotiate with their colleagues in the other organizations in the supply chain.

Methodical work, task modularization, using CASE tools, informal communication and socialization have been emphasized by different authors. Many researchers agree that all of these are needed in a software development project. Methods, tools and task modularization can be used to minimize interdependencies between tasks, thus increasing efficiency (Crowston et al., 2005), but communication and socialization will build collective mind (Weick and Roberts, 1993) and therefore help manage the interdependencies that cannot be avoided (Crowston and Krammerer, 1998). This results in a potential dilemma, since teamwork strategies that reduce interdependencies also tend to eliminate opportunities for developing collective mind. Practices that minimize interdependencies might overlook critical differences between the technological frames of key social groups, such as software professionals and users (Orlikowski and Gash,

1994). Factors that are external to the problem solving can also constrain the problem solving; for example, pressure from deadlines and the re-evaluation of project priorities by various stakeholders can motivate a change in teamwork arrangements and the use of any ICT that supports it (Ciborra and Patriotta, 1998). With so many conflicting requirements, any fixed problem solving strategy is unlikely to yield the best combination of efficiency and reliability.

The debate for and against various software development approaches is avoided, since many of them succeed in some circumstances. However, a single approach is rarely optimal for the whole project, and it is difficult to make a switch because the underlying problem solving and knowledge sharing strategy needs to be changed. The categories of knowledge networking are used to describe these strategies as they appear in the case, focusing on shifts in software development approach. In real projects, there is a complex flux of problem situations, and the team is often required to engage in several knowledge networking modes partially, and to shift between modes. The interest in this research is on how the team maintains its shared understanding when shifting to another mode of knowledge networking. Activities that are beyond the scope of the dominating knowledge networking mode at any time are identified in order to study how they can facilitate a modal shift in knowledge networking.

3 Research questions

The following research questions are addressed in the discussion after the case study descriptions.

3.1 Research Questions for the IEC 61499 Case

The introductory section 2.2 that used IEC 61499 as an example ended with the following statement: elements of the switching cost that will be studied in detail are the effort involved in changing attitudes, establishing new work practices and adapting proven teamwork arrangements. This phrasing is too vague for a research question, and the theoretical framework using technological frames in section 2.3 can be used to identify specific questions:

1. What obstacles impede professional automation designers from accepting the technological frame associated with IEC 61499?
2. How can this new frame be presented in the context of familiar work practices in order to facilitate adoption?
3. How can the resulting work practices support teamwork that leads to effective software development?

3.2 Research Questions for the ISOBUS Case

The need to adopt the unfamiliar ISOBUS standard was one factor that considerably impacted the use of problem solving and knowledge sharing strategies. After analyzing these strategies as they occurred in the case, it is possible to assess this impact. The discussion of the case should address the following set of research questions:

1. How did the need to adopt ISOBUS interfere with the team's preferred problem solving and knowledge sharing strategies?
2. To what extent and how successfully were these strategies modified for the sake of ISOBUS?
3. In retrospect, how could the strategies have been chosen better for successful standard adoption without compromising the goals of the software development project?

3.3 Research Questions for Comparing the Cases

The theoretical frameworks for both cases address the problem of adapting work practices to goals and methods in a standard adoption project. The main work practices in these cases are software design, implementation, testing and integration, and the term "problem solving strategy" is concerned with how the team configures these practices. Project specific configurations include factors such as division of responsibility, communication and reliance on tools. These are addressed by both theoretical frameworks. The difference between frameworks is that knowledge networking seeks to identify correlations between problem structure and chosen problem solving approach; the extent to which this can happen depends on the team's capability and willingness to configure itself to the problem at hand. Technological frames are not concerned with an optimal way of working that is independent of the team members' history; any upgrading of work practices is contingent on the upgrading of the technological frames or on the inclusion of individuals in frames.

It can be misleading to make assumptions about whether the development team will adapt its practices to project goals, or whether the goals will be adapted to familiar and preferred work practices. The successful adoption of a new standard can be prevented if:

- the team cannot establish effective and efficient work practices to use the standard as intended by the standard creators, or
- the standard is applied according to familiar problem solving practices, thus ignoring the paradigm on which the standard is based

Either of the above two conditions is sufficient to cause the standard adoption to fail. The two theoretical frameworks together address these issues. At the time of planning the research, it was not clear to the author which framework will address relevant problems. The discussion based on the questions in sections 3.1 and 3.2 will identify the adequacy of the framework in explaining the observations. In order to assess the scope into which these observations can be generalized, the framework of each case can be applied to the other case. This results in the following two research questions:

1. Are technological frames and knowledge networking overlapping or complementary approaches to understanding the same phenomena?
2. Are both frameworks relevant or could one of them be adequate?
3. Do the conclusions based on questions in sections 3.1 and 3.2 imply orthogonal or contradictory recommendations to managing the project? Does a comparative study with two cases and two theoretical frameworks suggest that there are tradeoffs that need to be made to the goals of the project that involves a standard adoption?

4 Research Method

4.1 Research Method for the IEC 61499 Case

4.1.1 Using Action Research Cycles

As mentioned in section 3.3, the IEC 61499 case was proactive in the sense that the software project was arranged to make standard adoption as easy as possible, with minimal problems emerging during the project. In order to be able to control the preparation and goals of the project, it was arranged at the university, and professionals and researchers were invited to participate as software developers. Three one week intensive events were held in the summer of 2004, 2005 and 2006. In each event, the assignment was to automate laboratory equipment according to specifications.

These events together with their preparation and analysis of observations can be seen as three action research cycles. The outcome of each cycle is used to focus research questions and data gathering methods for the next cycle. The relevance of the research arrangements for practitioners is also improved on each cycle. The process is continued until satisfactory results are obtained (McKay and Marshall, 2001). In our case, it took three cycles to obtain the proactive approach mentioned in section 3.3.

This research on IEC 61499 is presented as a case study even though it occurred on university premises. It is not an experiment, because aspects of experimental methodology such as control groups were not used. The purpose was to imitate the industrial context at the university as much as possible, and this was accomplished with three action research cycles with professionals participating in each cycle. A typical case study in a company was not possible due to the emerging nature of the standard.

The event in June 2004 provided specifications without any further design guidelines. An analysis of solutions and informal questionnaire feedback made it clear that there are several design alternatives that can seem equally good to designers without prior experience with IEC 61499. This can make it difficult to proceed quickly or to integrate solutions with modules that are provided by another team (Strömman, Sierla, Koskinen, 2005). This motivated more thorough planning of goals and empirical data collection in subsequent events, as described in the next subsections.

4.1.2 An Event in 2005 Using In-Depth Interviews

The course in 2005 emphasized teamwork arrangements for confronting the range of design alternatives; it was reasonable to assume that the participants' considerable experience in automation design would enable them to produce interesting solutions, so course organizers did not want to suppress their creativity by providing a solution beforehand. As recommended in (Mathiassen and Purao, 2002), the chosen educational approach exposed participants to design decisions instead of enforcing canonical practice. Course arrangements encouraged debates and discussions, and small groups were expected to integrate their modules with other groups' modules. The problem with this democratic team organization is that the process might not converge in the given time (Tomayko and Hazzan, 2004), and this is also what happened during the one week intensive course. Different attitudes and design philosophies were revealed in 16 recorded and transcribed in-depth interviews (Seidman, 1997). The conflicting design approaches are described in (Peltola, Sierla, et al., 2006; Strömman, Sierla, et al., 2006).

Using in-depth interviews to gather empirical material had significant strengths and weaknesses. Interviews were conducted very soon after the experience, as recommended (Seidman, 1997), but participants were nevertheless often unable to give richer descriptions when pressed for details, because they were not able to recall their design choices for specific system parts. In these cases, it was not possible to probe further without resorting to leading questions. However, the inability to give details on specific issues is in itself useful information, since it indicates the extent to which certain choices have been considered against alternatives. The range of solutions that are considered suggest the extent to which the participant is included in the technological frames into which those solutions belong. This information would have been difficult to collect with real time observations, not even with extensive video recordings, because many of the considerations are not verbalized; it is difficult and error prone to infer design decisions based on edits that can be observed in the software development environment. Detailed technical information could be readily obtained from issues that had been given conscious thought by the participant, and a variety of justifications were given by different interviewees; the arguments ranged from computer science to professional experience.

Teamwork related issues were remembered well, as each interviewee could easily identify those participants with whom it was not necessary to argue, as well as those who seemed to obstinately cling to their own ideas. In the latter case, the extent to which the justifications of the other party were understood varied greatly among interviewees, and this could be understood by comparing the interviews of both parties. This comparison could be used to identify the extent to which the interviewee could be included in the technological frame of the other participants. It also shows if the interviewee can accept solutions belonging to those frames after hearing justifications in their favor. This knowledge is crucial for answering the research questions, since further efforts to deploy

IEC 61499 into this social group must either succeed in including individuals in new frames or in proposing work practices that are sufficiently familiar to the existing frame.

4.1.3 An Event in 2006 Using Real Time Observations and Field Notes

The feedback from these events has been used to increase the industrial relevance of the event in June 2006 (see Peltola et al., 2007; Thramboulidis, Sierla et al., 2007). An important observation is that professional designers are habituated to modifying example solutions, and the lack of such legacy software has made it very difficult to decide on design principles or to appreciate the potential efficacy of IEC 61499 based development (Peltola, Sierla, et al., 2006; Strömman, Sierla, et al., 2006). For the event in June 2006, application architectures, design principles and example solutions were provided. The team organization relied on experienced participants in the roles of project manager, system engineer and tester, with inexperienced participants working as designers. According to industrial partners, this arrangement is similar to industrial projects. Sufficiently experienced participants with experience from our 2004 and 2005 events could be recruited for these roles, and they were given a two day briefing before the course about the development environment and the example solutions. This finally resulted in successful teamwork, even though the assignment was much more complex than during the previous courses. After the event, one industrial partner commented: “You have gotten very close to a real delivery project. Only one thing was purely academic. In real deliveries at the factory, when something goes wrong, someone has to take a shovel to get rid of the material that gets dumped onto the floor.”

The empirical material was collected as field notes based on real time observations by the author and two other researchers. A list of questions similar to an interview guide was prepared beforehand by the author to be used by all three researchers. The difference between this method and interviewing was that the researcher silently matched the observed behavior to the question list and noted issues when there was a match. The advantage of this approach over interviews is that the researcher is unobtrusive and there is no danger of leading questions. Unobtrusiveness was much more important in 2006 than in 2005, because the 2006 event aimed at assessing a development approach that had been proposed by the researchers.

The question list was organized by topics to guide the researchers to observe specific issues; as an example, the following excerpt is from the topic of validation:

How does the group determine whether their task has been successfully completed? Again, is this done from a software or process control perspective? Is validation planned before implementation? When is validation performed? Are tests planned systematically and is coverage considered, even in an informal way?

Important criteria for the guidelines that were provided by organizers were conciseness, ease of learning and minimal reliance on documentation. Such guidelines cannot cover every eventuality, so the empirical work gave special attention to situations in which

developers were not able to solve problems by a straightforward application of guidelines. Field note observations focused on how well the assigned team organization, which imitated industrial projects, was maintained as people had to exercise some creativity in finding solutions. The following excerpt is from the field note guideline:

Do the individuals carry out all the responsibilities of their roles? Do they try to go beyond these roles? In problem situations, was everyone working in their role? Was something neglected because it was not clearly someone's responsibility? Was there overlap so that several people tried to do a one-man job? Were there situations where the expertise of several roles would have been needed?

These questions are not related to the IEC 61499 standard that was being adopted; team organization is completely out of scope of the standard. Nevertheless, in 2005 it was discovered that an inappropriate team organization could result in an unfavorable experience and thus a negative appraisal of the new standard. Therefore it was necessary to consider the impact of team organization to limit the technological frame that was being imposed on participants working in specific roles on the team. The frame includes goals as well as methods and techniques for reaching those goals, and these could be defined for each role in the team by making an explicit division of labor among the roles.

4.2 Research Method for the ISOBUS Case

Two methods were used for studying knowledge networking in the case project (AGRIX): action research and interviews. The author participated in AGRIX from September 2003 to May 2004. Action research implies that the researcher is involved in the daily activities of the community that is being studied, just as in ethnography. In contrast to ethnographers, however, action researchers are active members of the social system that they are studying. This is true also of the study of information systems. While the ethnographers of this field generally aim to behave in an unobtrusive and passive way (Harvey and Myers, 2002; Nandhakumar and Avison, 1999), action researchers participate in problem solving activities (McKay and Marshall, 2001). For example, Mumford (2001) worked on facilitating the communication between software developers, users and other stakeholders; Mathiassen (2001) improved software development processes in several companies. The author piloted the CASE (Computer Aided Software Engineering) tool that was used in AGRIX, proposed a methodology for module testing, and acted as a tester. The experience from this work and numerous discussions with people involved in the project were used as a platform for thirteen semi-structured, recorded and transcribed interviews, which focused on the technical, cognitive and organizational aspects of the project.

Action research was necessary preparation for the interviews, which addressed participants' everyday experiences in the software development environment. A deep contextual understanding was imperative for encouraging the respondents to talk about their thoughts and actions during the development process. It was not possible to ask directly about distributed problem solving; authentic information could only be obtained

by discussing the work as the participant had experienced it (Kvale, 1996), using the terminology that was familiar to them. Concrete references to system parts, artifacts, and tool properties were used as anchoring points during the interviews, as though the interviewer and interviewee could both look at them on a screen. The interviews were facilitated by the interviewer's action research experience, which involved intensive work with the design and testing of the development tool.

Recordings of real-time actions and interactions of the project members would have contributed to an in-depth understanding of the circumstances through which knowledge networking solutions were produced. This could have been accomplished through video recording, but many of the interactions, decisions and deliberations in research projects were difficult to capture in real time, even with a video camera, because they were not fixed in time and space. They occurred during lunch breaks, when walking to the bus, when meeting by accident in the corridor, or behind closed doors. Further, in software development much of the crucial interaction occurs when engineers browse, study, modify and integrate artifacts that have been developed by colleagues. These activities dominate the experience of most software engineers and constrain many of their decisions, but there is little overt, bodily behavior to be observed: only mouse and keyboard use.

The strength of the method is that the action research gave a good, contextual understanding of the development process as a whole, as well as detailed knowledge about the technical issues of the project. This was important for focusing the discussion with each interviewee. The interviews allowed us to gather data on how the engineers framed and experienced problem-solving situations, individually and collectively. Respondents were first asked to describe, with their own words, the structure of the development work, and this structure was then used to guide the rest of the interview. For each phase, interviewees described the problems that they faced and how they used tools and worked alone or with others to solve the problems. The theoretical framework was then used to identify and classify instances of knowledge networking. Special attention was given to changes in the types of knowledge networking, and to how the informants explained those changes.

The use of the empirical material in the analysis is illustrated extensively in (Bruun and Sierla, forthcoming) with direct quotations. These are used to motivate how the author has identified the interviewee's perception of the problem and how the participants' descriptions of episodes in the project are used to justify the classification of problem solving approaches according to knowledge networking modes. A more limited use of quotations for illustration purposes has also been performed in section 6 of this introduction.

5 Case Study with IEC 61499

This section is structured as follows. The results of the June 2004 experiment are described in (Strömman, Sierla, Koskinen, 2005); they were used to design the June 2005 experiment and are not explained in further detail here. Section 5.1 describes the June

2005 experiment (see also: Peltola, Sierla et al., 2006; Thramboulidis, Sierla et al., 2006), and the material is used to address research question 1 (see section 3.1) in the discussion. Section 5.2 describes the June 2006 experiment (see also Peltola et al., 2007; Strömman et al., 2007), and the material is used to address research questions 2 and 3 in the discussion.

5.1 An Initial Project for Identifying Obstacles

The research question for the first event in June 2005 was to identify how participants with diverse backgrounds were included in different technological frames, and how this affected the successful application of IEC 61499. The research arrangements described in section 4.1 revealed different attitudes and design philosophies.

Table 1 shows the extent to which seven participants were included in three technological frames 'cyclic PLC', 'object-oriented' (OO) and 'IEC 61499'; these participants were chosen to represent the different social groups on the course. Cyclic PLC (Programmable Logic Controller) refers to the PLC cycle consisting of reading inputs, computation and writing outputs (Lewis, 1998), which is currently the industrially accepted practice in factory automation. The column for IEC 61499 summarizes the reflections of the participants on using the standard in their work, based on the interviews after the course. Participants are treated anonymously with names A-G, and their background is indicated as a professional working for a company (P), a researcher or consultant (R) or as a member of the committee that produced the IEC 61499 standard (S). Each item in the table is based on recurring themes that were emphasized by the interviewee.

Table 1 Inclusion of participants A-G in technological frames. (P) professional; (R) researcher; (S) standardization worker

	Technological frame		
	Cyclic PLC	Object oriented (OO)	IEC 61499
A (R)	Cyclic paradigm is useful for some parts of the application.	Fully aware of encapsulation, interface design, structural and dynamic modeling.	Supports most aspects of SW development. Dynamic modeling needs e.g. sequence diagrams (UML)
B (P)	Strongly included: all computation is based on data flows.	Encapsulation not fully respected. Functionality not recognized as separate from data.	Standard can be used in the context of the cyclic paradigm, but extra work is needed to design a level of events.
C (S)	Aware of control loops importance in module design. Vendor should encapsulate.	Encapsulation, modularity and reusability need to be introduced to traditional automation design.	Member of standardization committee
D	Cyclic paradigm is	Modularity and	Admits possible

(P)	used in all application areas. Visual FB notation industrially accepted.	encapsulation are familiar from non-OO techniques.	advantages in events, though execution flow is difficult to handle. Suggests events be hidden from designers.
E (P)	Prefers textual. Manages outsourcing of diagnostic tools to IT houses.	Encapsulation and interface design are extremely important in outsourcing OO work.	Encapsulation and event based execution are useful. Using function blocks doesn't feel natural.
F (R)	Cyclic paradigm is one of many. Visual SW components match mechanical modules.	Experience with various OO-modeling techniques.	Events give better execution control than conventional PLC languages. IEC 61499 and OO largely equivalent.
G (R)	Very limited experience with cyclic paradigm.	Strong computer science background.	Events help understand flow of control. FB interface should offer services that encapsulate signals.

Industrial participants B and D had electrical engineering educational backgrounds and had learnt to program on the job. Programming was viewed as processing various kinds of signals such as data, parameters and status. The importance of interface design was acknowledged, but not from the perspective of encapsulation: the implementation of a FB (function block) was considered the definitive documentation of its outputs. The use of IEC 61499 was only considered in the context of the cyclic PLC execution model, so with IEC 61499 the designer has to explicitly add events that just replace the functionality of the PLC execution environment. Unlike B and D, E had not worked as a control software developer, but managed the outsourcing of diagnostic tools to a software house that used modern OO (Object Oriented) techniques. Many OO principles have been incorporated into IEC 61499, so these were readily appreciated. The visual FB notation was considered unnatural due to E's background in textual programming. E was included in the OO frame while B and D were strongly included in the cyclic PLC frame; these differences resulted in nearly opposite appraisals of IEC 61499.

Researchers were all included in the OO frame, and some were also partially included in the cyclic PLC frame. The OO background clearly helped appreciate the features of IEC 61499, but it did not prevent participants with this background from encountering problems in interface design and module integration during the course. The following questions could not be answered easily during the one week event: what exactly should be encapsulated, how should it be made available via the interface, and how should the functionality be distributed among several function blocks?

Based on this experience, it is not obvious that IEC 61499 brings the benefits of object-orientation to designers with more traditional automation programming backgrounds (as claimed in e.g. (Thramboulidis, 2005)), since even researchers on the course had disputes about how to apply OO principles. For further research, it was proposed that these benefits could be obtained with project arrangements that conform to current industrial automation software development practice.

5.2 A Case Aiming at Industrial Acceptance

5.2.1 Project Arrangements and Engineering Environment

It was observed that a much more successful development project with IEC 61499 could be obtained if all participants could be included in the IEC 61499 technological frame. The management of the project should unify participants' technological frames to avoid conflicts, decrease the need to resort to fixed solutions and facilitate inclusion in the IEC 61499 frame. It might not be necessary to obtain full inclusion, which would involve an appreciation of difficult issues such as execution semantics. According to industrial participants, it is very desirable that development tools and methods can be quickly learned, and that a strong computer science background will not be required from designers. It was then decided to imitate the organization of teams in real industrial delivery projects. Participants were assigned to roles with well defined responsibilities, and each role required various amounts of IEC 61499 related knowledge, i.e. partial inclusion in the IEC 61499 frame.

The teams consisted of three leader roles (project manager, system engineer, tester/system integrator) and two or three pairs of designers. Industrial partners explained that real projects depend on experienced and competent professionals in the leader roles; the team can then make progress with much less experienced staff manning the designer roles. Leaders were given a two day training prior to the course, involving a presentation of our reference application, discussions on design principles and a hands-on session in the development environment.

It was clear that the general purpose architecture of IEC 61499 could not be directly applied to a particular domain, such as manufacturing, process automation or batch process control, without first introducing a set of domain-specific design principles. Batch processes were chosen, since there is a lack of IEC 61499 related research in this domain. As described in section 2.1, much of this domain knowledge is embodied in the industrially accepted framework for batch process control defined in IEC 61512 (IEC, 1997). Design guidelines for structuring IEC 61499 function blocks and compositions were therefore formulated with the goal of obtaining a practice analogous to batch control design with the PFC (Procedural Function Chart) of IEC 61512. It was presumed that this would lower the learning curve of IEC 61499 for participants who are familiar with batch control. An example of the PFC notation and its mapping to IEC 61499 according to our guidelines is presented below.

The equipment to be controlled was a simplified laboratory version of the liquor circulation in pulp and paper processes (Fig. 1). The liquor circulation was first described textually and then modeled with the graphical PFC notation. The PFC provides four hierarchical 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.

Fig. 1. The equipment to be controlled

The impregnation unit procedure in Fig 2. is used to illustrate the application and the PFC notation. The internals of the unit procedure are described as a sequence of operations such as EM2_OP1 and EM5_OP1. The parallel lines are split and join conditions that permit the concurrent execution of operations; the flow of control will stop at the lower set of lines until all operations have been completed. The first three operations in impregnation are responsible for opening a route from tank T200 to the digester T300 and for turning on a pump. Then, the execution pauses at the condition LS+300 activated until the surface level inside T300 reaches the LS+300 sensor. Then the outlet of T300 is closed, so pressure will build up inside the tank since the pump is still on. After the delay of length Ti, the pump is turned off and the route that was opened previously is closed.

After these operations are completed, the digester is depressurized by opening the exit valve at the top for 2 seconds.

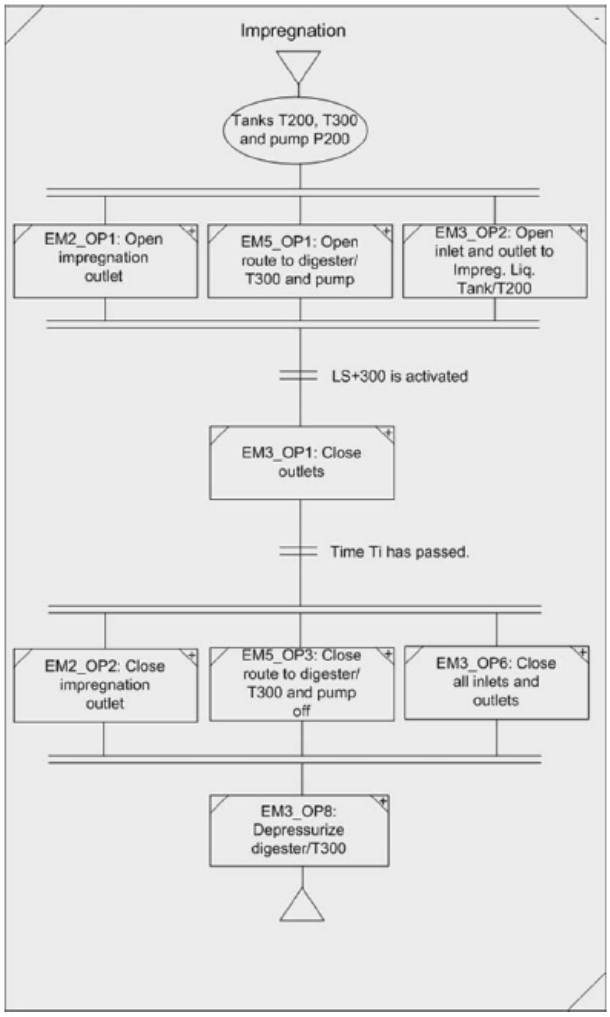

Fig. 2. The Impregnation unit procedure

In industrial projects, methods and work practices are in place for moving from specifications to software designs; for the course, it was necessary to provide a method of obtaining IEC 61499 designs from PFC specifications [19]. The PFC's hierarchical structure and sequential behavior prompted us to choose a straightforward one-to-one mapping from PFC elements to IEC 61499 function blocks. The PFC Procedures, Unit Procedures and Operations are hierarchical application elements, and the design can easily be transformed into a hierarchy of IEC 61499 function blocks, using composite blocks which are implemented by other function blocks. The sequential control flow of PFC follows a type of token passing semantics, which can be modeled with IEC 61499's event mechanisms. Splitting and joining the token for parallel execution of activities (e.g. operations) and their synchronization requires event handling function blocks. Also, various other recurring PFC elements were implemented as function blocks, such as time delays and conditional delays to wait for a sensor input condition. These blocks were placed in a "PFC Elements" library to increase productivity and to enforce the preferred

programming style. Together with some example unit procedures, operations and brief training, this library constitutes the needed guidelines for the designers to quickly start participating in a team development effort.

Fig. 3. shows the beginning of the function block network implementing the PFC unit procedure in Fig. 2. The unit procedure begins with three parallel operations, which all take the REQ token straight from the composite block interface. The execution is then synchronized by an “E_REND3” block, which implements a rendezvous for the output events of all three preceding operations. Then the token is passed forward to a conditional delay pattern “DI_Gate”, which waits for a digital process measurement from subscriber “SUBL_1” to pass a true value (meaning that surface level has reached a high water mark). After this the token is passed to another PFC operation, and so on.

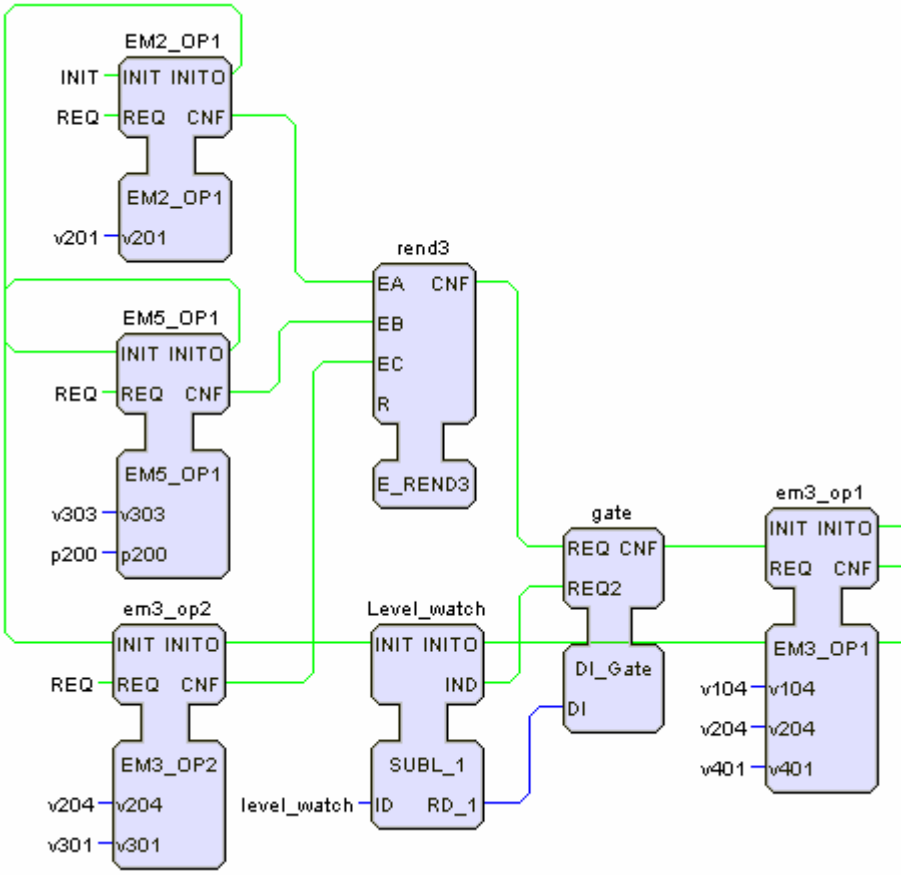

Fig. 3. Part of a function block network implementing the Impregnation unit procedure in Fig. 2

After browsing the example function block networks and creating the first ones themselves, the designers quickly learnt to read the PFC and find the matching function blocks to fit the application layout. Although the flow of control is top-down in the PFC and left to right in the function block network, the two notations are otherwise similar enough to allow quick comparison and visual inspection of the two application models.

5.2.2 Qualitative analysis

The arrangements described in section 5.2.1 were used to organize a one week project in June 2006 for professionals and researchers. Qualitative empirical material was gathered in the form of field notes by three researchers. Observations were made according to a list that focused on various potential problem sources in four areas: teamwork, knowledge integration, software development methodology and technical problems in the development environment. During analysis, the field note items were coded according to these four areas; similar items were categorized together and relationships between categories were identified. The analysis methodology is adapted from grounded theory (Urquhart, 2001; Orlikowski, 2002), but the purpose is to evaluate technology and methods rather than to build theory (Lings and Lundell, 2005): four areas that impact industrial acceptance and inclusion in the IEC 61499 technological frame were identified a priori based on previous experience. Only the final analysis results are presented here.

5.2.2.1 Team organization

The successful industrial adoption of new design tools depends on the ability to manage inexperienced developers who need to learn new skills on the fly. This was achieved by a team organization that defined roles and responsibilities. The team leaders used their technical expertise to inspect the work of the designers, so that common design guidelines were understood and used. It was crucially important that leaders refrained from solving technical problems themselves, so developers could learn to apply the design skills independently. The manager had decomposed the project into tasks and assessed the difficulty of each task, so developers were then given progressively more difficult work as their capabilities improved.

The team organization was challenged when dealing with technical problems that were not anticipated in the initial guidelines. A certain amount of opportunism from the tester and system engineer resulted in adequate solutions, which were then enforced within the team. This enforcement required a significant amount of communication, and it is important to note that this could not have been accomplished if the leaders would have forgotten their roles and engaged in engineering work on some part of the system.

5.2.2.2 Technical challenges

The team was required to embark on the project with an incomplete knowledge of the standard, tool and development environment. For example, version control for the various components that were produced by the team had to be addressed during the project after an integration had failed due to an obsolete version of one component. However, further problems surfaced later when resources (containers of IEC 61499 function block networks) that had been developed independently of the system could not be integrated by copy-pasting XML. Other problems were caused when it was not clear which blocks should contain the cyclic control loops or how the event mechanisms should be used to activate them from the sequential control logic. The idea was to implement these into controller modules in separate resources, so that the publish-subscribe communication mechanism would be used to activate the loops. This made it difficult to test the controller modules, because a great number of publish subscribe channels had to be

configured for that purpose. These problems were not covered by the examples or the preliminary training before the course; significant technical problems were not encountered with issues that had been covered.

5.2.2.3 Software development method

The software development method is concerned with workflows between tasks such as design, implementation, module testing, integration and system testing with the physical equipment. Of interest are situations that forced the team to deviate from the manager's plan. Leaders decided to avoid module testing to save time, relying on inspections. However, controller modules that contained continuous control loops were identified as a risk, since there were no examples about how they should be combined with the sequential batch control logic. The risk was addressed by building module testing environments.

The risk assessment was correct, but the approach did not succeed satisfactorily for two reasons. First, there was so much work involved in setting up the test environment and modifying the architecture, that it was not possible for the leaders to supervise this and ensure that all designers were occupied with productive work. Second, the technical limitations of the tool made testing a time consuming process since many publish-subscribe channels had to be set up for each test. A significant amount of time was spent debugging the test environment, because of subtle data type conversion issues in these channels.

These problems were addressed in an ad hoc way by the tester, who took control of the team, worked with developers and gave them further assignments. This resulted in a number of iterations that were not anticipated in the project manager's plan and during which parts of the team proceeded without adequate supervision. The delays barely permitted the team to complete their task successfully during the course.

5.2.2.4 Knowledge integration

How much overlapping knowledge is needed among team members? The efficiency benefits that can be obtained from modular software development suggest that overlap should be kept to a minimum. However, such overlap can help overcome problems when individual team members are not able to complete tasks on their own (Crowston et al., 2005). In this experiment, shared knowledge was maintained in the form of guidelines, which were example solutions that could be modified and applied by all team members. In order to save time and lower the learning curve, guidelines were not accompanied by formal or detailed documentation.

This strategy led to a division of labor. Leaders engaged in creative problem solving and designers learnt to apply guidelines. A significant amount of communication and knowledge sharing was required between leaders and designers to make sure that everyone used the same guidelines and that the guidelines addressed emerging problems. However, no communication or interaction was needed between the various pairs of designers; since the pairs were coordinated indirectly by the leaders. In this way, the

benefits of modular development were also realized, since the need to communicate and negotiate about solutions was reduced, and the responsibility for creative problem solving was limited to a few roles that were manned by individuals with sufficient experience.

6 Case Study with ISOBUS

6.1 Project Background and Goals

The case project concerns the development of an agricultural information system that links actors from various social groups, such as farmers, precision farming experts, agricultural planning experts and fault diagnostics experts. The system was to be distributed and use several data communications technologies. The systems development team of the case project, called AGRIX (The automation system for agricultural implements), was requested to build an interoperable, reliable and general-purpose automation system for agricultural implements such as seeder-fertilizers and pesticide sprayers (Oksanen et al., 2004; Öhman et al., 2004). The motivation for this was an increased demand for customer options combined with pressure to decrease the engineering costs involved in reconfiguring the software for new machines.

The system development was supported, or constrained, by the requirement to adhere to the ISOBUS standard (ISO, 2003), which decomposes the system into several parts and specifies standard interfaces and data communications between them. The idea is to enable plug and play interoperability of system parts developed independently by different vendors. Examples of such parts are tractors, implements, user interfaces and precision farming task controllers. The standard was an emerging one and in 2004 there was very limited practical experience on using it.

The study focuses on activities in the spring of 2004, which was the critical phase of the AGRIX project. New software development technology had to be adopted and a reconfigurable architecture for a broad range of implements had to be designed based on the existing domain understanding. The industrial partners also wanted the researchers to validate their general-purpose system by configuring it for three different machines that they had provided; the first one was expected to be ready by the end of May, which is the latest time for sowing crops in a Nordic country. The team had worked on a similar machine in 2003, and the project manager of AGRIX saw this as a valuable experience for understanding the application context and improving the internal communication of the group. However, the project goals for 2004 required that the system be open, reconfigurable and interoperable, so it was necessary to adopt some methods, development tools, and standards. Since there was no established methodology in the industry of agricultural machines, the suitability of the state of the art in the broader context of industrial software development was evaluated (e.g. Lewis, 2001, Douglass, 2004).

The case is fairly analogous to certain kinds of R&D teams. For example, in the field of factory automation, separate client projects are typically needed to tailor a software-based

control system for the machinery at a client's site, but these projects are usually very similar, so that considerable cost savings are possible if software can be reused or reconfigured (Strömman et al., 2005). A R&D department typically maintains CASE (Computer Aided Software Engineering) tools and frameworks or repositories of reusable software components that are utilized in the client projects. The R&D teams are also responsible for piloting new standards, tools and methods. R&D staff usually does not use a formal requirements specification, since they continuously receive informal requirements from colleagues who are involved with clients. In AGRIX, the developers built a basic architecture that could be reconfigured easily for different machines; maintained a base of reusable software in the repositories of the CASE tool; and obtained requirements in discussions with their industrial partners during meetings and field tests.

6.2 The Case Project

AGRIX involved two Finnish research units and several industrial partners. This case study focuses on one of the research units, located at a Technical University, which was responsible for developing the software-based automation system. The members of this unit were specialists in automation, control and software technology and had a few years of work experience. Everyone had a reasonable level of understanding about each of these areas, so their knowledge perspectives were different but overlapping. The other research unit had expertise in the application area, and contributed with domain knowledge and system requirements, as well as usability and field tests with the prototype machines. The industrial partners provided the actual machines and contributed with some more requirements and insights about market trends. A steering group was formed containing representatives from all parties. The steering group meetings were used to focus the goals of the project as requirements were understood more fully, and to make decisions on proposed designs and technical issues of greater significance. No formal decision making procedures based on power relations were used in these meetings.

The internal organisation of the group at the university emphasized maintaining excellent communication at all times. The interviewees did not feel that there were any social problems. There were no hierarchical decision making procedures. According to a junior developer:
'Of course we have discussed everything together and made the decision together. If there were several propositions, we discussed them together to see which of them is best. In the end we have listened to the one in the highest position, but there has been little commanding from above.'
[Interview 11]

The group met weekly with the supervising professor. The developers considered these discussions useful even when some things were repeated, because they made everyone aware of what others were doing. The meetings were seen as being important for communication as well as problem solving, especially since the number of developers was not large:

‘It’s better not to have a lot of people with a lot of opinions. When there are a few opinions it is possible to discuss all of them and make a decision.’
[Interview 11]

In addition to this, developers had daily discussions among themselves in which they described their solutions and got feedback. Everyone felt that it was worth spending time to update the others’ detailed knowledge of the overall situation. Although this is not the fastest way of decomposing and executing the work, it updated the mental model that group members had of the development activity, and thus made them prepared to engage in more intensive, shared problem solving, whenever problems or new requirements were discovered.

In this organizational context, the team was free to choose its balance between formal and informal practice. The development process model existed as a mental model in the mind of the main programmer who was most experienced in this respect; the real model was a combination of concepts that had been learned on software engineering courses, proposals by the author and practices that had been used in the preceding project. Using the terminology of Introna and Whitley (1997), the developers and technology were merged seamlessly, since the methodology never became the focus of attention. It was not necessary to give an impression that formal or well-documented procedures were adhered to. According to the main programmer:
‘In principle we have followed [the V-model], but in this research project we are not making a commercial product, so there has not been a formal requirements specification. [The prototype that was made in 2003] also served as a requirements specification.’
[Interview 1]

The other developers carried out smaller tasks such as component development, but they were not responsible for coordinating the development effort or integrating the system. They communicated with the main programmer in order to make sure that their system part was being built correctly. They also participated in group discussions concerning requirements definition and architecture design, but the main programmer was allowed to implement his understanding of the outcome of these discussions with the CASE tool.

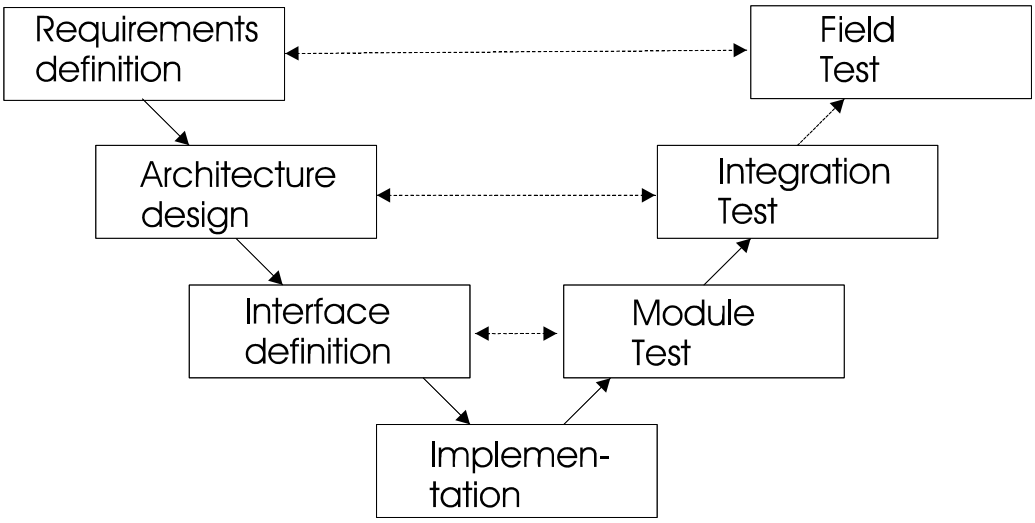

Fig. 4. The V-model

Fig. 4 shows the classical V-model, which is based on the idea that system requirements that have been elicited from the end users are used to design an architecture with parts that correspond to the structure of the desired application. The modules that are identified in architecture design have well-defined responsibilities and interact to provide meaningful functionality to the user. The interface design simply refines this work so that all modules, large and small, have a detailed interface that permits an individual developer to provide an implementation. As an effect, modules can be implemented without detailed technical knowledge about the system as a whole. Artefacts are tested and integrated against specifications that were developed in previous phases, as shown by the horizontal lines in Fig. 4. The model can be seen as a chronological sequence of tasks or only as a collection of relationships and dependencies among the artefacts that are produced. The next section illustrates how the model was applied in practice.

6.3 Analysis

It is broadly accepted that methodical and tools-based practices need to be combined with informal interaction. Still, software development efforts continue to fail, and one significant reason can be that this combination is not being performed appropriately (Goulielmos, 2004). In order to understand why this is so, we begin with the observation that both types of activities tend to occur in parallel, and that shifts of emphasis between the two do occur. In some cases, such shifts seem to be a requirement for success in the development work; in other cases, an unsatisfactory problem solving strategy was abandoned. In both types of shifts, the chosen strategy may support or restrict the development of shared understanding. We use the knowledge networking categories to identify and characterize both the dominating problem solving strategy and background activities.

6.3.1 Using Boundary Objects in Requirements Definition

The AGRIX project could not start by defining a stable set of requirements, because the application area experts needed to see various prototypes before they could make concrete specifications. The main programmer explained the problem at the beginning of the project as follows: 'The machines were not designed for precision farming. During the project, it has been necessary to modify the mechanics and this has resulted in new software requirements… The software group had discussions about machine functionality, then we looked at existing user interfaces [UI] and then we built a first version of our UI that we could show to application area experts. It's very important to have something to show because otherwise it's very difficult to get feedback.' [Interview 13] Requirements were captured in a document that associated screenshots from a prototype user interface with textual descriptions of machine operations. This artefact turned out to be a good boundary object, since 'application area experts were able to correctly associate machine operations with visual user interface components.' [Interview 13] However, there were problems 'because we did not want to copy old machines but to add new automated functionality… the new operations were difficult to explain.' [Interview 13]

This resulted in a dilemma. The final requirements could not be defined before actually building a prototype; at the same time, a prototype could not be built before some requirements specification was obtained. In order to make progress, the team worked iteratively: presenting designs, receiving feedback, designing a system, implementing a prototype, and taking snapshots from the prototype user interface. A document with these snapshots was used as a basis for the next requirements discussion, so there was iteration between the requirements, design and implementation activities. 'When we got feedback to our requirements proposal, we updated the design on the CASE tool. Then we implemented the design and took screenshots from the new prototype UI for the next requirements negotiation with application area experts. The CASE tool supports this mode of working, since it is possible to automatically generate program code from the design.' [Interview 13]

The AGRIX team's iteration between requirements, design and implementation activities was influenced by both CASE tool and notations. The design used a graphical state-machine notation which describes the system's reactions to events such as user commands. Developers were conscious of two different uses for this notation: it could serve as a basis for communication as well as for automatic program code generation by the CASE tool: 'Some of the state diagrams only need to be understood by developers. Others were used for high level functionality, and these were useful in discussions with application area experts.' [Interview 13] Some state diagrams and the pictures from the prototype user interface functioned as boundary objects in the discussions with application experts, and they also had a more specific meaning for the software developers who used the CASE tool to implement that functionality.

During these iterations the team switched between two kinds of knowledge networking. First, there was the problem of specifying the detailed functionality of next generation machines in communication with application experts. This was followed up by the well-

defined technical task of implementing a prototype. The first problem was solved in integral mode with open communication in a heterogeneous group of experts, and boundary objects were used to promote mutual understanding. For the second problem, the desired system functionality was well defined as far as the next prototype was concerned. The implementation task was modularized, since developing a module did not require the combined knowledge of several individuals. The engineering task could be simplified by avoiding such communication that was no longer necessary: 'if one person does one module, it gets done properly. Most of the problems arise when many people work on the same software and it has not been decomposed properly. You don't know what the others are doing and might fail to go through everything during testing. When you are building software, it is quite efficient to divide it into clear-cut modules, so that everyone has a well-defined responsibility.' [Interview 10]

The difficult requirements problem was not solved simply by supplementing methodical development work with informal interaction. For example, the developers did not just proceed with design and implementation, contacting application area experts whenever they were not sure about what the system should be doing. Instead, a collectively shared understanding was developed and maintained by switching between integral work with boundary objects and modularized software development work. The state diagram and the user interface evolved in a gradual way, so the understanding that had developed around them at earlier stages was maintained in later stages, and across the different knowledge networking modes. These artefacts could function, on one hand, as boundary objects in the requirements definition sessions with application area experts, and, on the other, they constituted the actual development artefacts that were manipulated by the software developers. The double-function of the state diagram and the UI implied that software could be developed efficiently and reliably, without worrying about the effects on the global level of requirements; these were addressed after each prototype had been implemented. This strategy satisfied the seemingly contradictory needs of the project, that of knowledge sharing and that of independent, focused work. All project participants were able to contribute to fruitful requirements discussions that were based on a prototype, at the same time as software design and implementation could be carried out without constant interaction with application experts.

6.3.2 Using the CASE Tool for Module Testing

One goal of AGRIX was to pilot methods that could be used across geographical and organizational boundaries. A module testing methodology that relied on the component repository of the CASE tool was developed. Developers were responsible for contributing software modules and interface documentation into the repository, and a tester was in charge of creating test applications, plans and reports, which could be used by developers and integrators. The tests were developed in a 'black-box' fashion, in which the tester only relies on interface documentation. This was an attempt to use the CASE tool as an interfacing device for translational knowledge networking, so that actors in different roles would communicate by exchanging standard document and software component types via the repository.

This practice of communication over the interfacing device was used for a few components, but as field test deadlines drew near, the developers decided to visit the tester for speedier fixes. A real-time debugger of the CASE tool, which plotted the values of selected signals versus time, was used. The tester's special expertise was drawn on to identify places where the curves did not behave as expected, and the developer associated the curves to variables in the internal implementation of the component. The developer and tester continued with 'white-box' testing, in which an understanding of implementation details is used to design tests. A subtle error could be found and fixed in a fifteen minute session. Translational knowledge networking was thus quickly replaced by integral problem solving in direct face-to-face interaction. The CASE tool was now being used as a boundary object, since the visual representations could be associated with the partly different and partly shared mental models employed by the tester and developer.

If the tester would have been unaware of implementation details, he would have needed to obtain this understanding to build up capacity for focused integral knowledge networking. However, the developer had discussed this thoroughly with the tester on previous occasions. Such discussions were not required for the black-box testing methodology based on communication over the CASE-tool interface, but it permitted an immediate shift to very efficient integral white-box testing when that was needed. The shift was made when the project started slipping behind schedule.

6.3.3 Module Development

Module development was managed by the main programmer, who acted as a coordinator by defining detailed interfaces and connections between modules. The development of individual modules was assigned to developers who worked independently. The project manager also worked on a few modules himself. In one case—a module for the analysis of network data—he chose to involve one of the other developers, a junior one, rather than to do the work independently. The ensuing discussions were unnecessary from the perspective of modular knowledge networking, because the project manager could have done the work alone. The reason for this arrangement was that the project manager needed time for his other duties. The two developers did programming in turns, which meant that they had to have frequent communication about the modifications. The project manager admitted that fewer man-hours would have been spent on the task if he had worked alone, but the integral strategy of problem solving enabled him to attend to other tasks that could not be delegated.

6.3.4 Using the ISOBUS Standard for Independent Development of System Parts

The steering group of AGRIX decided to adopt the ISOBUS standard in the development work. The standard defines data communications and functionality between system parts, such as the user interface, precision farming task controller and the software control system of the agricultural implement. The goal of the standard is to enable the independent development of any of these parts. This quality of the standard enabled

AGRIX developers to use the ISOBUS as an interfacing device in a process of distributed design involving the university and an industrial partner. System parts such as the task controller and the user interface were to be implemented according to detailed ISOBUS specifications. In the ideal case, developers working on the different modules would have to communicate with each other only about the application of the specifications, and about the division of labour in the implementation work. The purpose of the interfacing device was to remove any system integration problems.

The ISOBUS-based translational knowledge networking was successful in some integrations, but only when preliminary module tests had been performed quite thoroughly against the specifications of the standard. When developers at the university encountered problems, they resorted to an integral mode of knowledge networking, i.e. shared problem solving:

‘Of course it helps that we sit in the same room and have two computers open and can make verbal decisions about what to test next. If something doesn’t work, we fix it and make another test.’
[Interview 12]

Problem detection required an understanding of both modules that were being integrated, so it was necessary to break the modular principle of translational knowledge networking. The university team had built its shared understanding in many technical discussions, neglecting the fact that the designers of the interfacing device (ISOBUS) had aimed at eliminating such communication. As a result, it was easy to move onto shared problem solving when that was required. This additional communication had taken place within the university team only, however, while coordination between organizations relied completely on the standard. When problems occurred in this latter scenario, developers needed to communicate directly with each other, in natural language. This happened for instance when a module had been incompletely tested in one organization, which led to failure in its integration with another module produced by the university team. It was not possible to even identify the faulty module before a shared understanding of the workings of both was obtained via email and telephone.

The ISOBUS supports a very pure form of translational knowledge networking, which can be successful if all parties commit themselves to mastering and using the standard. This is advantageous for overcoming geographical and organizational barriers, but serious breakdowns can occur because the interfacing device does not encourage the development of shared mental models, or collective mind. The team’s ability to work around a breakdown of coordination depends on such shared models being developed by other activity. This had occurred within the university team but not over organizational barriers, so a quick transition to integral knowledge networking was possible only in the former case.

Based on these observations, it can be difficult to draw definite conclusions regarding the desirability of establishing pure translational knowledge networking. The results of this

case indicate that the most practical solution depends on the unique constraints and goals of the case. This issue is addressed further in the discussion.

7 Discussion

The discussions in sections 7.1, 7.2 and 7.3 address the research questions in sections 3.1, 3.2 and 3.3, respectively.

7.1 IEC 61499 Case

Technological frames of industrial participants indicated that familiar automation design approaches were attempted with IEC 61499, and that this did not lead to appreciation of the new features of the standard. The impressions regarding IEC 61499 for participants with object oriented backgrounds were very different, showing the impact of educational and professional backgrounds when all other factors are constant. The disagreements even among researchers with similar backgrounds, and the fact that the different opinions could not converge during the week, reveal the difficulty of obtaining design guidelines that are understandable and acceptable for all participants. Therefore, the reflective educational approach (Mathiassen and Purao, 2002) is unlikely to produce good results with busy professionals, who want concrete results in a week's time.

IEC 61499 is a general purpose architecture for industrial process measurement and control systems, so an attempt was made to introduce it in the context of an industrially accepted domain specific standard, the SP88 standard for batch control. Preparatory work was performed to obtain straightforward instructions for developing IEC 61499 control applications based on SP88 PFC specifications. The instructions were presented as example solutions to be copied and modified, thus exploiting skills that professional designers rely on heavily. The hierarchical PFC architecture was used to decompose the development effort into design and testing activities with limited scope.

The decomposition of the application development task into units of work made it possible to assign specific responsibilities to team members. The team structure resembled real projects with experienced staff manning project manager, system engineer and tester roles and inexperienced members working as automation designers. The former were responsible for supervising the latter until they had learnt the necessary copying and modification skills in the new environment. This learning process was accomplished quickly, so team leaders were free to concentrate on issues such as enforcing common design principles across the team and on managing resources so that all designers were employed with work units that matched their capabilities. (Sierla et al, 2007; Strömman et al., 2007).

7.2 ISOBUS Case

The AGRIX team clearly recognized the value of communication, shared problem solving, and spontaneity over following plans and rigid division of labour. They were

therefore able to solve difficult problems using integral knowledge networking. Their preferred working style was also poorly compatible with translational knowledge networking. A first attempt was made to set up the translational mode in the module testing methodology, but this was quickly replaced by work in the integral mode, because the latter was better suited for solving the problem at hand; the module testing methodology would have enforced the kind of discipline that would have been needed for the application of ISOBUS, especially across organizational boundaries. When two developers in the AGRIX team tested and integrated ISOBUS components, they were able to proceed efficiently with their preferred way of working. However, when ISOBUS components developed in a different organization needed to be integrated, there was no translational mode to enforce the necessary discipline, and the use of integral knowledge networking over organizational boundaries was very inefficient.

The team held onto its preferred way of knowledge networking as much as possible. ISOBUS related problems were solved in this way, because the team was successful in the integral mode and because the nature of the project did not have all of the constraints of commercial projects. The presence of a constraint such as limited or no access to developers in another organization would have made translational knowledge networking more attractive. One goal of the project was to pilot practices that could be used under such constraints, so the knowledge networking strategies were not satisfactorily modified for the sake of ISOBUS, even though other project goals were met.

In retrospect, a more disciplined use of the proposed module testing methodology would have established the kind of development and testing practices that are required for successful use of ISOBUS over organizational boundaries. This could have decreased the need for some of the communication and interaction that consumed much of the time. However, it is very difficult to know what interactions will prepare the team for solving emerging problems. One problem with the project was that it had conflicting goals, such as developing demonstrations in the short term and developing methodology in the long term. A clear prioritization of goals would have been needed in order to motivate the team to establish the mix of problem solving strategies that would have addressed the short term goal as well as the long term goal of standard adoption.

7.3 A Comparison of the Cases

7.3.1 Applying Knowledge Networking to Findings in the IEC 61499 Case

In this subsection, an attempt will be made to express the findings of the IEC 61499 case in terms of knowledge networking. The project arrangements that were planned by organizers in 2005 emphasized integral knowledge networking. The goal was to bring together the experience of people with experience in computer science, object oriented programming, cyclic PLC programming, process automation and current industrial practices. The project arrangements encouraged the communication of proposals and

their rationale, so that the best solutions could be obtained with negotiations in small groups and within the whole team.

The goal and arrangements match well with the definition of integral knowledge networking, but this mode did not emerge satisfactorily. There was no boundary object to mediate between the different knowledge perspectives, so many attempts at communication failed when an unfamiliar paradigm was confronted. In particular, the object oriented and cyclic PLC programming paradigms were perceived as incompatible. IEC 61499 could serve as a boundary object between these paradigms, since it combines event based processing and encapsulation of methods (algorithms) and data behind an interface, but also supports cyclic programming if periodically generated events are used to execute function block networks. Many industrial participants were unwilling to consider the IEC 61499 concepts as a boundary object, because their in house tools seemed more attractive.

The goal for integral knowledge networking, to draw from several knowledge perspectives in order to obtain novel and creative solutions, did not motivate some participants. Further, the task was too difficult to accomplish in the short time, which drove participants to attempt less elegant solutions that might get the equipment working. Project organizers expected the timeframe to be realistic due to the trivial nature of the project assignment, but the effort to become familiar with the standard and tool, coupled with the demanding debates with other designers, exceeded the capabilities of participants.

The project arrangements also had some characteristic of modular knowledge networking, since the assignment was decomposed into three modules, and the team was divided into three smaller groups that were each responsible for one module. Groups were expected to negotiate the module interface details and to perform the integration, which are tasks that belong to the modular knowledge networking coordinator or coordinating group. Representatives from each module group did meet in order to coordinate the work, but these negotiations were not able to bridge the diverse knowledge perspectives. In one team, there was curious triple modular knowledge networking arrangement, when each group finally decided to perform their own integrated system by using the modules from the three groups. The only working application that was produced during the event was produced by one of these groups.

Successful knowledge networking would have required skillful iterations between modular and integral knowledge networking. The coordinating group would have needed the capability to work in the integral mode in order to obtain mutually acceptable designs, which could then have been implemented by the three groups and then integrated. If the initial task for integral knowledge networking would have been too complex, the team could have started by defining a prototype of limited complexity, which would have then been implemented in the modular mode. An example of such a successful iteration between integral and modular work is described in the ISOBUS case in section 6.3.1. That work relied on well chosen boundary objects, and no such objects were available in the IEC 61499 case. The knowledge networking framework is not very helpful in

answering the question why the IEC 61499 standard notation did not emerge as a boundary object, but the analysis with technological frames identifies explicitly many obstacles, in addition to the limited time. Both the cyclic and object-oriented paradigms would need to be appreciated by all users of the boundary object even if the technical details are glossed over, and this would require inclusion in several technological frames.

Another weakness of the knowledge networking framework is the lack of attention towards motivation, which can result in changing or poorly defined goals. Langlais et al. (2004) do emphasize how goals can change during a project as specific problems are solved and new problems emerge; the goal of knowledge integration (Langlais uses the term transepistemic challenge for this purpose) is perceived according to the structure of the problem to be solved, and the team is expected to choose a knowledge networking mode that is suited for that kind of a problem. In the IEC 61499 case, it was clear that different team members did not agree on the nature of the problem that they were expected to solve together, and there was often a lack of motivation to consider arguments that were based on an unfamiliar paradigm. Knowledge networking is not a useful approach for probing deeply into these motivational issues, but an analysis with technological frames makes the participants' behavior seem rational: the effort needed to learn and apply an unfamiliar frame must be compensated by a sufficient incentive. This effort is unlikely to be made under time pressure, and many participants decided to press on within the context of their familiar frame with the hope of getting something working before the deadline.

Knowledge networking is useful for identifying a collective problem solving strategy, i.e. iteration between integral and modular modes and incremental prototyping to broaden the problem scope with manageable additions to complexity. The importance of establishing the IEC 61499 standard as a boundary object, at least within the coordinating group, is highlighted. Technological frames suggest how the boundary object could be created within the team. Representatives from each module development group would need to be partially included in the frames listed in table 1, so suitable individuals could be chosen for the coordinating group in modular knowledge networking. This group could then be quickly given an overview of the three technological frames involved, and this would enable them to communicate successfully with their colleagues as well as the members in their module development team. The members of the coordinating group would then be alternating between integral and modular modes, while the software developers could work in a purely modular fashion. In order to make the initial iteration in integral mode succeed, a sufficiently simple prototype should be defined at first.

Based on the above discussion, the research questions in section 3.3 can be addressed. In the IEC 61499 case, the frameworks appear complementary: both can be used to explicitly identify different kinds of problems that prevented the development work from proceeding smoothly. Each framework only suggests an incomplete solution to a project manager, but both frames could be used to outline a promising development approach. Knowledge networking addresses the structure of the problem that has been assigned to the team, and technological frames address the team members' perceptions of the problem.

7.3.2 Applying Technological Frames to Findings in the ISOBUS Case

The ISOBUS case has a notable difference to the IEC 61499 case: the technological frames of the software developers were much more homogenous. Therefore, problems in using the ISOBUS standard cannot be sufficiently understood by identifying differences in the developers' frames. The technological frame of the ISOBUS developers needs to be considered, even though the AGRIX team did not interact with standard developers, because this frame is incorporated into the ISOBUS specifications. This discussion therefore poses the question that to what extent were the AGRIX team members included into the ISOBUS frame and did this clash with their own frame; i.e. their accustomed approach for control software development. Can an identification of incompatibilities between these two frames give additional insights to the analysis with knowledge networking?

Technically, the AGRIX developers were able to implement the ISOBUS specifications; they were also aware that the standard was intended for independent development of system parts that can be integrated in a plug-and-play fashion. Using Orlikowski and Gash's (1994) definition for technological frames, the standard creators and users had congruous expectations and views on the nature of the standard. Therefore, it does not seem that technological frames give additional insights to the analysis based on knowledge networking in section 7.2.

Other problem solving activities in AGRIX that were unrelated to ISOBUS can also be examined with technological frames. The use of the boundary objects in requirements definition (section 6.3.1) was successful because the created boundary objects established commonalities between the software developers' and application area experts' technological frames. Software elements in the user interface as well as in the underlying control logic were successfully associated to corresponding machine functionality. A more thorough identification of these social groups' technological frames could have increased the understanding of the mental effort that was required to reach mutual understanding.

Regarding the research questions in section 3.3, it seems that knowledge networking was a more powerful way to analyze the findings, because the technological frames among AGRIX developers were homogenous. The study focused on software development and the application area experts were not interviewed; the interactions with these experts were described by the software developers that were interviewed. These findings can be used to guide future empirical work for knowledge networking in cases which involve successful or unsuccessful attempts to create boundary objects. For each such attempt the technological frame of the participants should be identified; if interviews are used it would also be desirable to elicit the interviewee's view on colleagues technological frames.

The findings can also be used to guide practice: a manager who is responsible for a project that adopts a new standard can identify the need to create boundary objects, so it is possible to also identify the technological frames of relevant project members and other stakeholders involved in creating the boundary object. A manager who actively facilitates knowledge integration would anticipate situations in which an integral knowledge networking mode is needed; staff with sufficient inclusion in relevant technological frames could then be provided or quickly trained for work with the boundary object.

8 Conclusion

8.1 Research Implications

8.1.1 Generalizability and Limitations of the Contribution

In order to support generalization, case studies should either be representative of current industrial practice, or they should pilot practices that are likely to establish themselves as mainstream (Kvale, 1996). In this dissertation, it is proposed that universities should take a more active role in deploying standards to industry, so the cases are not chosen for the purposes of a researcher who is content with a passive observation of the course taken by the standard adoption process. It has been demonstrated that leaving the industrial deployment of standards to market forces can result in billion dollar losses for the industry as a whole (Hurd and Isaac, 2005) as well as undesirable constraints for future standardization efforts (Browne and Menon, 2004).

The cases for this dissertation have been chosen to demonstrate how academics can work together with industry to promote the understanding of a standard and to ease its adoption. The AGRIX project piloted the ISOBUS standard on agricultural machines that were provided by industrial stakeholders, and ISOBUS components developed at the university were integrated to others that were provided by an industrial partner. The IEC 61499 studies invited professionals to participate in projects on university premises in order to avoid having to wait for industrial projects in which the standard would be used.

The global standards adoption process is contingent on many factors, so it is important to identify the possible impact from performing the kind of studies that have been described here. IEC 61499 underpins a vision for an open knowledge economy for industrial automation, and ISOBUS supports plug-and-play interoperability of system parts developed by different vendors. In such an environment, the benefits that a single firm receives from supporting one of these standards are based on network effects, i.e. the benefits are proportional to the number of other parties that use the standard (Browne and Menon, 2004; Weitzel et al., 2006). The benefits of open standards to the industry as a whole are unquestionable, yet the emergence of such standards can be prevented due to startup problems: the first firms that make the commitment face a very high risk, since

the benefits of network effects will not be realized if clients and competitors commit to other standards or proprietary solutions (Weitzel et al., 2006).

Individual research efforts cannot sway the above mentioned forces at the macroeconomic level; however, it is possible to decrease the cost and risk involved for a company that adopts a standard. The essential concept here is the switching cost, which includes the loss of investments that are rendered obsolete as well as the effort to obtain and set up the technology that implements the new standard. In this research, special attention has been given to minimizing the element of the switching cost that is caused by the need to retrain software developers to use the new standard and related tools (Sierla et al., 2007). New work practices are proposed for this purpose, and their relevance to other IEC 61499 compliant development environments is addressed in (Thramboulidis, Sierla et al., 2007) and (Strömman et al., 2007). Another way to decrease the switching cost is by extending the IEC 61499 compliant development environment in order to support work practices that resemble industrially accepted practice (Peltola et al., 2007).

The generalizations that can be made from these cases are the approaches for decreasing the switching costs for the adoption of standards which, once established, have the potential to benefit the industry greatly. It is proposed that in order to realize this potential, larger numbers of researchers should make similar efforts. Research funders are in a position to emphasize these issues in research programs; such programs can coordinate the efforts of several research institutes and industrial players to significantly lower switching costs to attractive standards.

8.1.2 Scientific Novelty of the Findings

The primary original contribution of this work is a method for assessing the obstacles to industrial deployment of standards for control software development. This has been demonstrated for IEC 61499. The assessments engage representatives from several companies and research organizations in order to bring together a broad range of expertise to work on a standard adoption problem. Participants have been chosen so that their joint expertise encompasses both a solid understanding of the potential of the new standard as well as an appreciation of the obstacles for industrial deployment. The challenge that is encountered in this approach is that the knowledge perspectives of the participants are heterogeneous and potentially contradictory; further, there is also no obvious way of integrating this knowledge.

The theoretical approach in this dissertation focuses on knowledge integration challenges and therefore supports the systematic design and analysis of the above mentioned assessments. The method and theoretical approach to assessing the industrial deployment of standards is a novel contribution to the IT standards literature which has been reviewed in section 2.1; the author is not aware of similar efforts.

The impact of this work on research on a particular standard has been demonstrated for IEC 61499, which has a very active academic community; publications II, III and VI

have appeared in special sessions devoted to IEC 61499. The presentation of these results has challenged unjustified optimism in the results of other papers at these sessions. The assessment method in this dissertation offers an approach for identifying the obstacles for industrial deployment of any of the above mentioned other results.

The ISOBUS case study was the first application of a relatively new theoretical framework, knowledge networking, to software development. The contribution of publication I therefore aimed at fundamental science and it has been published at an appropriate journal. It has been demonstrated that this approach can be used to address the knowledge integration challenges in a software development project. As described in section 7, the knowledge networking framework can be applied in conjunction with technological frames to strengthen the theoretical basis for further standards assessments.

8.2 Practice Implications

The comparison of the two cases in section 7.3 already discourages attempts to propose recipes for problem solving strategies that could serve the needs of diverse standard adoption projects. A working knowledge of the theoretical frameworks that have been used in this dissertation could prepare project managers and system engineers to coordinate the work of their team and to assess the realistic chances for success in a standard adoption project with specific goals, resources and schedule. In the case of a project that is required to adopt a new standard, it is not possible to rely only on experience from previous projects. The following considerations are based on the most salient issues that were encountered in this research; they need to be explicitly addressed in order to be able to plan a realistic project schedule and to avoid knowledge integration attempts that have a high probability of failure.

8.2.1 Identify the structure of the problem and the knowledge networking mode that can effectively tackle that problem.

For example, are there architectural solutions that can be applied to this project in a straightforward way, and are the developers on the project familiar with these solutions? If yes, a modular mode of working can be efficient; otherwise, integral knowledge networking should be considered in order to identify and satisfy the range of requirements for the new architecture. Section 7.3.1 illustrates the kinds of problems that can occur when an integral knowledge networking cannot be established to resolve design issues at the level of the entire system. It is not enough to identify that there is a need for integral knowledge networking, and this motivates the second topic:

8.2.2 Identify the capacity for integral knowledge networking based on inclusion of project staff in relevant technological frames.

The frames that are relevant depend on the nature of the problem; they can involve understanding of application domains, programming paradigms or engineering

disciplines such as mechanical modeling and simulation, control engineering and software development. Active project leaders should not assume that appropriate boundary objects will emerge from the interactions between project members and other stakeholders. Possible boundary objects should be identified early based on the nature of the standard and the goals of the project; good candidates for such objects include visual modeling notations, user interface designs, graphical simulators or algorithms that can be expressed in terms of business logic or control logic.

An understanding of the technological frames that are encompassed by the boundary object as well as the level of inclusion of project staff and stakeholders in these frames can be used to set up a group of people with the capacity for integral knowledge networking. If such personnel is not available, it is possible to avoid initiating an activity that is likely to drag on for a long time without succeeding, after which it might be too late to attempt another approach. If the proposed analysis is performed before or in the beginning of the project, it is possible to enlist more suitable experts to the project, provide quick training to key people, lengthen the schedule or reconsider the project goals.

8.2.3 Identify the need and capacity for translational knowledge networking.

This issue depends on the nature of the standard and the problem that it aims to solve. Several standards that are incorporated into protocols or software development environments are intended to enable the integration of systems developed by people who do not necessarily communicate with each other. ISOBUS is such a standard, but the same problem is inherent in all standards that support plug-and-play. The structure of the problem indicates that translational knowledge networking should be used, but if a more direct communication between developers is possible, the constraints of the translational mode might be bypassed, resulting either in concurrent use of another knowledge networking mode or a degradation of translational networking.

An obvious question at this point is that if spontaneous communication solves the problem at hand, would adhering to the translational mode have any value in itself? This depends on what constraints on communication and collaboration might be encountered in the future. The capacity for translational knowledge networking can take time to build since it involves disciplined work practices, possibly over organizational boundaries. For example, work practices related to testing and software integration need to be more rigorous if a module must be treated as a black box when there is no access to the module developer.

If a capacity for translational knowledge networking is desired, a project manager should actively encourage and justify this approach to the team, because the discipline involved is difficult and can seem counterproductive from the perspective of immediate problem solving. It requires a greater mastery of the technology that serves as the interfacing device and developers might expect to get away with only a partial mastery if they can resort to bypassing the device by communicating with other developers.

8.2.4 Based on the above, assess the probability of meeting short and long term goals

Examples of short term goals are passing a test or a milestone or developing a product. Longer term goals are maintaining the product or developing and maintaining a line of similar products. The short term goals might be met passably without developing an architecture or translational knowledge networking capacity that would be required for some longer term goals. Having project managers focus only on short term goals can therefore prevent the organization from developing the collective problem solving capability that would be required for initiatives that could result in serious competitive advantage, for example setting up mature product line practices. All levels of management in the organization need to agree on a prioritization of short and long term goals, so that appropriate problem solving strategies can be established. This means that projects with realistic goals, resources and schedules are planned by managers and approved by their supervisors.

The approach presented in this dissertation is one way to concisely justify goals and project plans for meeting them. It is very important to explicitly include longer term goals, such as building the capability for integral or translational knowledge networking for the purpose of improving the capability of architectural design or rigorous testing practices. Otherwise there is a danger that resources will not be allocated for these purposes and that the efforts of foresighted managers who pursue such goals will not be recognized.

9 References

Avison, D.E., Wood-Harper, A.T., Vidgen, R.T. & Wood, J.R. (1998). A further exploration in information systems development: the evolution of Multiview2. Information Technology & People, 11(2), 124-39.

Beck, K. (1999). Embracing Change with Extreme Programming. IEEE Computer, 32(10), 70-77.

Beck, K., & Andres, C. (2004). Extreme Programming Explained: Embrace Change. Boston: Addison-Wesley.

Bijker, W.E. (1987). The Social Construction of Bakelite: Toward a Theory of Invention. In W.E. Bijker, T.P. Hughes, & T.J. Pinch (eds.): The Social Construction of Technological Systems, MIT Press, Cambridge, MA, 1987, 159-187.

Brooks, F. (1987). No silver bullet: essence and accidents of software engineering. IEEE Computer Magazine 21(4), 10-19.

Browne, G.J., & Menon, N.M. (2004). Network Effects and Social Dilemmas in Technology Industries. IEEE Software 21(5), 44-50.

Bruun, H., Hukkinen, J., Huutoniemi, K., & Klein, J. (2005). Interdisciplinary Research and the Academy of Finland. Helsinki: Publications of the Academy of Finland, No 8/2005.

Bruun, H. Langlais, R. & Janasik, N. (2005). Knowledge Networking: A Conceptual Framework and Typology. Journal for Science and Technology Studies, 18(3-4), 73-104.

Bruun, H. & Sierla, S. (forthcoming). Distributed Problem Solving in Software Development: The Case of an Automation Project. Social Studies of Science.

Ciborra, C.U., & Patriotta, G. (1998). Groupware and teamwork in R&D: limits to learning and innovation. R & D Management, 28(1), 43-52.

Coupe, R.T., & Onodu, N.M. (1996). An empirical evaluation of the impact of CASE on developer productivity and software quality. Journal of Information Technology, 11(2), 173-181.

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2005). Effective Work Practices for FLOSS Development: A Model and Propositions, in Proceedings of the 38th IEEE Annual Hawaii International Conference on System Sciences (Hawaii, USA, 2005).

Crowston, K., & Krammerer, E. (1998). Coordination and Collective Mind in Software Requirements Development. IBM Systems Journal, 37(2), 227-245.

D'Adderio, L. (2001). Crafting the Virtual Prototype. How Firms Integrate Knowledge and Capabilities Across Organisational Boundaries. Research Policy, 30, 1409-24.

Douglass, B. (2004). Real Time UML – Advances in the UML for Real-Time Systems. Boston: Addison-Wesley.

Egyedi, T.M. (2001). Why Java was – not – standardized twice. Computer Standards & Interfaces, 23(4), 253-265.

Emam, K.E., & Madhavji, N.H. (2001). Elements of Software Process Assessment & Improvement. Wiley-IEEE Computer Society Press.

Eriksson, H.E., & Penker, M. (2000). Business Modeling with UML. New York: John Wiley & Sons.

Ferrarini, L., Veber, C., & Fogliazza, G. (2005). IEC 61499 Implementation of a Modular Control Model for Manufacturing Systems, 10th IEEE International Conference on Emerging Technologies and Factory Automation, September 19-22, 2005, Catania, Italy.

Goulielmos, M. (2004). Systems development approach: transcending methodology. Information Systems Journal, 14(4), 363-386.

Hahn, H., & Turowski, K. (2005). Modularity of the Software Industry: A Model for the Use of Standards and Alternative Coordination Mechanisms. Journal of IT Standards and Standardization Research, 3(2), 29-41.

Harvey, L.J., & Myers, M.D. (2002). Scholarship and Practice: the Contribution of Ethnographic Research Methods to Bridging the Gap. In M.D. Myers & D. Avison (eds), Qualitative Research in Information Systems, London: SAGE Publications, 169-180.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. IEEE Computer, 34(9), 120-127.

Hurd, J., & Isaak, J. (2005). IT Standardization: The Billion Dollar Strategy. Journal of IT Standards & Standardization Research, 3(1), 68-74.

Hutchins, E. (1995a). Cognition in the Wild. Cambridge, MA: MIT Press.

IEC (1997) Batch Control – Part 1: Models and terminology, IEC International Standard IEC 61512-1, Final draft, 1997.

IEC (2005A) Function Blocks - Part 1: Architecture, IEC Standard IEC 61499-1, 2005.

IEC (2005B) Function Blocks - Part 2: Software tool requirements, IEC Standard IEC 61499-2, 2005.

Introna, L.D., & Whitley, E.A. (1997). Against method-ism: Exploring the limits of method, Information Technology & People. 10(1), 31-45.

ISO Technical committee 23 (2003): ISO 11783-1 Tractors and Machinery for Agriculture and Forestry. Serial Control and Communication Data Network Part 1: General Standard, in ISO/TC23/SC19/WG1/No. 277/02E.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Software Development Process. Reading, MA: Addison-Wesley.

Kruchten, P. (1995). The 4+1 View Model of Architecture. IEEE Software, 12(6), 42-50.

Kvale, S. (1996). Interviews: an introduction to qualitative research interviewing. Thousand Oaks, CA: SAGE Publications.

Kuikka, S. (1999). A batch process management framework: Domain-specific, design pattern and software component based approach. Ph.D. dissertation, Dept. Automation and Systems Technology, TKK, Espoo, Finland, Espoo: VTT Publications, 1999, ISBN 951-38-5542-2, Available: http://www.inf.vtt.fi/pdf

Langlais, R., Janasik, N., & Bruun, H. (2004). Managing Knowledge Network Processes in the Commercialization of Science. Two Probiotica Discovery Processes in Finland and Sweden. Science Studies, 17(1), 34-56.

Lee, H., & Oh, S. (2006). A standards war waged by a developing country: Understanding international standard setting from the actor network perspective. Journal of Strategic Information Systems. 15(3), 177-195.

Lewis, R. (1998). Programming industrial control systems using IEC 1131-3. IEE Control Engineering Series. London: Institution of Electrical Engineers.

Lewis, R. (2001). Modelling Distributed Control Systems Using IEC 61499. IEE Control Engineering Series. London: Institution of Electrical Engineers.

Linderoth, H.C., & Pellegrino, G. (2005). Frames and inscriptions: tracing a way to understand IT-dependent change projects. International Journal of Project Management. 23(5), 415–420.

Lings, B., & Lundell, B. (2005). On the adaptation of Grounded Theory procedures: insights from the evolution of the 2G method. Information Technology and People, 18(3), 196-211.

Lundell, B., & Lings, B. (2004). Method in action and method in tool: a stakeholder perspective. Journal of Information Technology. 19(3), 215-223.

Marshall, C. (2000). Enterprise Modeling with UML. Reading, MA: Addison-Wesley Longman.

Mathiassen, L. (2002). Collaborative practice research. Information Technology and People, 15(4), 321-345.

Mathiassen, L., & Purao, S. (2002) Educating reflective systems developers. Information Systems Journal, 12, 81-102.

McKay, J., & Marshall, P. (2001). The dual imperatives of action research. Information Technology & People. 14(1), 46-59.

Mumford, E. (2001). Advice for an action researcher. Information Technology and People. 14(1), 12-27.

Nandhakumar, J., & Avison, D.E. (1999). The Fiction of Methodological Development. A Field Study of Information Systems Development. Information Technology & People. 12(2), 176-91.

Öhman, M., Oksanen, T., Miettinen, M., & Visala, A. (2004). Remote Maintenance of Agricultural Machines, in 1st IFAC Symposium on Telematics Applications in Automation and Robotics (Espoo, Finland, 2004); Helsinki: International Federation of Automatic Control. 149-54.

Oksanen,T. Öhman, M., Miettinen, M., & Visala, A. (2004). Open configurable automation system for precision farming, in ASAE Conference on Automation Technology for Off-road Equipment (Kyoto, Japan, 2004)

Orlikowski, W.J. (2002). Case Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development. In M.D. Myers and D. Avison, (eds.) Qualitative Research in Information Systems, London: SAGE Publications, 181-221.

Orlikowski, W.J., & Gash, D.C. (1994). Technological Frames: Making Sense of Information Technology in Organizations. ACM Transactions on Information Systems. 12(2), 174-207.

Peltola, J.P., Christensen, J.H., Sierla, S.A., & Koskinen, K.O. (2007). A Migration Path to IEC 61499 for the Batch Process Industry. IEEE International Conference on Industrial Informatics, July 23-27, 2007, Vienna, Austria.

Peltola, J.P., Sierla, S.A, Strömman. M.P., & Koskinen, K.O. (2007). Process Control with IEC 61499: Designers' Choices at Different Levels of the Application Hierarchy. IEEE International Conference on Industrial Informatics, 16-18 August 2006, Singapore.

Pinch, T.J., & Bijker, W.E. (1987). The Social Construction of Facts and Artifacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other. In W.E. Bijker, T.P. Hughes, & T.J. Pinch (eds.) The Social Construction of Technological Systems. MIT Press, Cambridge, MA, 1987, pp. 17-50.

Rost, J. (2005). Software Engineering Theory in Practice. IEEE Software. 22(2), 94-96.

Saferstein, B. (1998). Ethnomethodology. In W. Bechtel, & G. Graham, (eds.) A Companion to Cognitive Science. Oxford: Blackwell Publishers, pp. 391-401.

Seidman, I. (1997). Interviewing as Qualitative Research: A guide for Researchers in Education and the Social Sciences. Second Edition. New York: Teachers College Press.

Sierla, S.A., Christensen, J.H., Koskinen, K.O., & Peltola, J.P. (2007). Educational Approaches for the Industrial Acceptance of IEC 61499. ETFA'2007 12th IEEE International Conference on Emerging Technologies and Factory Automation, September 25-28, 2007, Patras, Greece. (SUBMITTED)

Sommerville, I. (2004). Software Engineering. Harlow: Pearson Education.

Star, S.L., & Griesemer, J.R. (1989). Institutional Ecology, 'Translations' and Boundary Objects. Amateurs and Professionals in Berkley's Museum of Vertebrate Zoology, 1907-39. Social Studies of Science. 19(3), 387-420.

Strömman, M.P., Sierla, S.A., & Koskinen, K.O. (2005). Control Software Reuse Strategies with IEC 61499. 10th IEEE International Conference on Emerging Technologies and Factory Automation (Catalania, Italy, 2005)

Strömman, M.P., Sierla, S.A., Peltola, J.P., & Koskinen, K.O. (2006). Professional designers' adaptations of IEC 61499 to their individual work practices. ETFA'2006 11th IEEE International Conference on Emerging Technologies and Factory Automation, September 20-22, 2006, Prague, Czech Republic.

Strömman, M.P., Thramboulidis, K., Sierla, S.A., Papakonstantinou, N., & Koskinen. K. (2007). Incorporating Industrial Experience to IEC 61499 Based Development Methodologies and Toolsets. ETFA'2007 12th IEEE International Conference on Emerging Technologies and Factory Automation, September 25-28, 2007, Patras, Greece. (SUBMITTED)

Sundaram, M., & Shim, S.S. (2001). Infrastructure for B2B exchanges with RosettaNet. Third International IEEE Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, (San Juan, USA, 2001); IEEE Computer Society Press. 110-119.

Thramboulidis, K. (2005). Model-Integrated Mechatronics – Toward a New Paradigm in the Development of Manufacturing Systems. IEEE Transactions on Industrial Informatics. 1(1), 54-61.

Thramboulidis, K., Sierla, S.A., Papakonstantinou, N., & Koskinen, K.O. (2007) An IEC 61499 Based Approach for Distributed Batch Process Control. IEEE International Conference on Industrial Informatics, July 23-27, 2007, Vienna, Austria.

Thramboulidis, K., & Zoupas, A. (2005). Real-Time Java in Control and Automation: A Model Driven Development Approach. 10th IEEE International Conference on Emerging Technologies and Factory Automation, September 19-22, 2005, Catania, Italy.

Tomayko, J.E., & Hazzan O. (2004). Human Aspects of Software Engineering. Hingham, MA: Charles River Media.

Truex, D., Baskerville, R., & Travis, J. (2000). Amethodical Systems Development. The Deferred Meaning of Systems Development Methods. Accounting, Management and Information Technologies. 10(1), 53-79.

Vyatkin, V.V., Christensen, J.H., & Lastra, J.L. (2005). OOONEIDA: An Open Object-Oriented Knowledge Economy for Intelligent Industrial Automation. IEEE Transactions on Industrial Informatics. 1(1), 4-17.

Urquhart C. (2001) An Encounter with Grounded Theory: Tackling the Practical and Philosophical Issues. In: E. Trauth (ed.) Qualitative Research in Information Systems: Issues and Trends. Hershey, PA: Idea Group Publishing, 104-40.

Weick, K.E, & Roberts, K. (1993) Collective Mind in Organizations: Heedful Interrelating on Flight Decks. Administrative Science Quarterly. 38(3), 357-381.

Weitzel, T. Beimborn D., & König, W. (2006) A Unified Economic Model of Standard Diffusion: The Impact of Standardization Cost, Network Effects and Network Topology. MIS Quarterly. 30, 489-514, Special Issue August 2006.

Wenger, E. (1998). Communities of Practice: Learning, Meaning, and Identity. Cambridge: Cambridge University Press.

Zhu, K., Kraemer, K.L., Gurbaxani. V., & Xin Xu, S. (2006) Migration to Open – Standard Interorganizational Systems: Network Effects, Switching Costs, and Path Dependency. MIS Quarterly. 30, 515-538, Special Issue August 2006.

HELSINKI UNIVERSITY OF TECHNOLOGY
INFORMATION AND COMPUTER SYSTEMS IN AUTOMATION

Report 1 Koskinen, K., Aarnio, P. (eds.),
Internet-, Intranet- and Multimedia Applications in Automation. June 1998.

Report 2 Koskinen, K., Aarnio, P. (eds.),
PC-based Automation Systems and Applications. June 1999.

Report 3 Mattila, M.,
Prosessilaitteen etätukijärjestelmä – ohjelmistoarkitehtuuri ja ohjelmistotekniset ratkaisut. March 2000.

Report 4 Strömman, M.,
Ohjelmoitavan logiikan ohjelmointi ohjelmistotuotantoprosessina. March 2002.

Report 5 Aarnio, P.,
Simulation of a Hybrid Locomotion Robot Vehicle. June 2002.

Report 6 Peltola, J.,
Uudet automaatiojärjestelmät - komponenttipohjaisen automaatiosovelluksen suoritusympäristö. September 2002.

Report 7 Fortu, T.,
Enterprise Resource Planning - Integration with Automation Systems. September 2002.

Report 8 Mattila, M.,
Condition Monitoring of an X-ray Analyzer. February 2003.

Report 9 Sierla, S.,
Middleware Solutions for Automation Applications - Case RTPS. June 2003.

Report 10 Honkanen. T.,
Modelling Industrial Maintenance Systems and the Effects of Automatic Condition Monitoring. February 2004.

Report 11 Seilonen, I.,
An Extended Process Automation System: An Approach based on a Multi-Agent System. February 2006.

Report 12 Tuukkanen, K.,
Representing Industrial Data Models in OWL Web Ontology Language. December 2006.

Report 13 Sierla, S.,
Adapting Problem Solving Strategies in Control Software Development to New Standards: Case Examples IEC 61499 and ISOBUS. December 2007.

ISBN 978-951-22-9104-5

ISSN 1456-0887

Picaset Oy, Helsinki 2007