VTT WORKING PAPERS 140

Juhani Hirvonen, Eija Kaasinen, Ville Kotovirta, Jussi Lahtinen, Leena Norros, Leena Salo, Mika Timonen, Teemu Tommila, Janne Valkonen, Mark van Gils & Olli Ventä

Intelligence engineering framework

ISBN 978-951-38-7480-3 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 4374

Technical editing Maini Manninen

Series title, number and
report code of publication

VTT Working Papers 140
VTT-WORK-140

Author(s)
Juhani Hirvonen, Eija Kaasinen, Ville Kotovirta, Jussi Lahtinen, Leena Norros, Leena Salo, Mika Timonen, Teemu Tommila, Janne Valkonen, Mark van Gils & Olli Ventä

Title

Intelligence engineering framework

Abstract
A number of advanced algorithms and mostly software-based technologies have been developed in recent decades in order to solve problems in complex technical systems. Examples that have been actively studied include machine learning, artificial intelligence, pattern recognition, neural networks, fuzzy logic, statistical methods, operation analysis and, most recently, sensor networks. The problem is that these techniques require considerable knowledge to be applied correctly, necessitating participation by skilled professionals. This makes applications expensive to design and maintain. There is therefore a common need for better engineering methods and tools. This paper describes start of the development of a systematic engineering discipline for algorithmic and knowledge-intensive intelligent systems and services. The rationale behind this idea is that advanced technologies and algorithms cannot be economically feasible unless standardised design practices, tools and system components are available. The focus of the research on the early stages of design gave rise to two issues of design reuse: 1) how to model the application's needs for intelligence and the features of potential solutions stored in solution libraries, and 2) how to help the designer search the libraries for solutions that provide the best match with the application needs.

ISBN
978-951-38-7480-3 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN		Project number
VTT Working Papers 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)		34570
Date February 2010	Language English	Pages 44 p. + app. 4 p.
Name of project Intelligent Engineering Framework	Commissioned by VTT	
Keywords Semantics, modelling, design, intelligent systems, ontology, design pattern	Publisher VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374	

Preface

The Intelligence Engineering Framework (IEF) project aimed to provide a systematic way to develop and maintain the next generation of "intelligent" systems. It was argued that such a framework was needed to take full advantage of the numerous advanced algorithms, methods, and solutions that have been developed in recent years. The intended applications of the IEF covered a wide range of areas in industry and society. Several research groups with different backgrounds were therefore involved in the project. This report summarises the results of the IEF project carried out during the period from September 2006 to December 2009.

The IEF is a part of the Complex System Design Technology Theme funded by VTT. The authors would like to acknowledge the Steering Group of the project for fruitful guidance and support. Moreover we would like to thank the members of the Industrial Support Group for rewarding discussions about the contents of the project. The members of the Industrial Support Group came from the following major Finnish companies: ABB, GE Healthcare, KCL, Metso Automation, Neste-Jacobs, Pöyry, and SavCor.

The project members came from various Knowledge Centres of VTT representing a broad scale of expertise in various fields of technology. Juhani Hirvonen D.Sc. (Tech.), Teemu Tommila M.Sc. (Eng.), Olli Ventä D.Sc. (Tech.), Janne Valkonen M.Sc. (Eng.), Antti Pakonen M.Sc. (Eng.) and Jussi Lahtinen M.Sc. (Eng.) represent knowledge in automation systems and their design. Leena Norros Res. Prof., Eija Kaasinen D.Sc. (Tech.), and Leena Salo M.Sc. (Eng.) are experts in human-technology interaction theories and practises. Mark van Gils Ph.D., Docent, Luc Cluitmans Ph.D. represents knowledge in applications of intelligent methods on medical instruments. Ville Kotovirta M.Sc. (Eng.) and Mika Timonen M.Sc. brought environmental monitoring expertise and Ville Vidqvist's M.Sc. (Eng.) speciality is rolling machine condition monitoring.

Espoo Feb 11th 2010

Authors

Contents

1. Introduction

The Intelligence Engineering Framework (IEF) project aimed to provide a systematic way to develop and maintain the next generation of “intelligent” systems. It was argued that such a framework was needed to take full advantage of the numerous advanced methods and solutions that have been developed in recent years. The intended applications of the IEF covered a wide range of areas in industry and society. Several research groups with different backgrounds were therefore involved in the project. This report summarises the results of the IEF project carried out during the period from September 2006 to December 2009.

1.1 The problem

A large number of advanced algorithms and mostly software-based technologies have been developed in recent decades in order to solve complex problems in technical systems (Figure 1). Examples that have been actively studied include machine learning, artificial intelligence, pattern recognition, neural networks, fuzzy logic, statistical methods, operation analysis and, most recently, sensor networks. The applications in which these techniques are employed are often referred to as intelligent systems. The problem is that these techniques require considerable knowledge to be applied correctly, necessitating participation by skilled professionals. This makes applications expensive to design and maintain. Extensive research in this field has generated numerous demonstrations and pilots but few practical solutions or commercial products. A further challenge is posed by the need for large-scale integration and continuous system modifications. Systems are becoming more complicated even if no advanced techniques are used. There is therefore a common need for better engineering methods and tools.

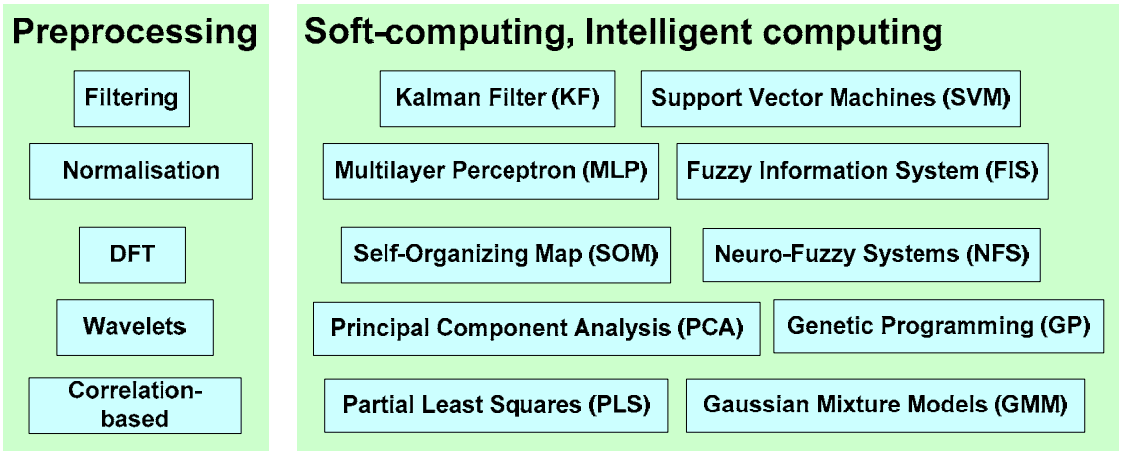

Figure 1. Well-known, popular, intelligent algorithms that have been extensively studied and piloted for decades.

The goal of the IEF project was to start the development of a systematic engineering discipline for algorithmic and knowledge-intensive intelligent systems and services. The rationale behind this idea is that advanced technologies and algorithms cannot be economically feasible unless standardised design practices, tools and system components are available.

Figure 2 clarifies the focus of the project. A product or system life cycle normally starts with user needs and requirements, and then goes into more depth on product design, realisation, verification and validation, ending up with a ready-to-deliver product or system. Effective and mature systems engineering processes are already in wide industrial use and are adequate to manage most of the product problem space and subsequent engineering tasks. Nowadays, however, it is more common for system and product developers to be faced with problems or requirements that cannot be solved using state-of-the-art methods. It is widely known that advanced algorithms may have great potential, often piloted in other contexts, but due to their impractical applicability they are neglected in the engineering processes.

The aim of the IEF project was thus to develop an engineering process (models and tools) which supports the selection and use of advanced solutions that match the specific "intelligence needs" of the application. The design process must be suitable for: 1) identifying the definition of the "specific" problem or opportunity to be dealt with, 2) performing the required engineering steps in outlining the design options, and 3) carrying along the actual design and implementing tasks, eventually ending in a high-end intelligent solution seamlessly integrated with the context.

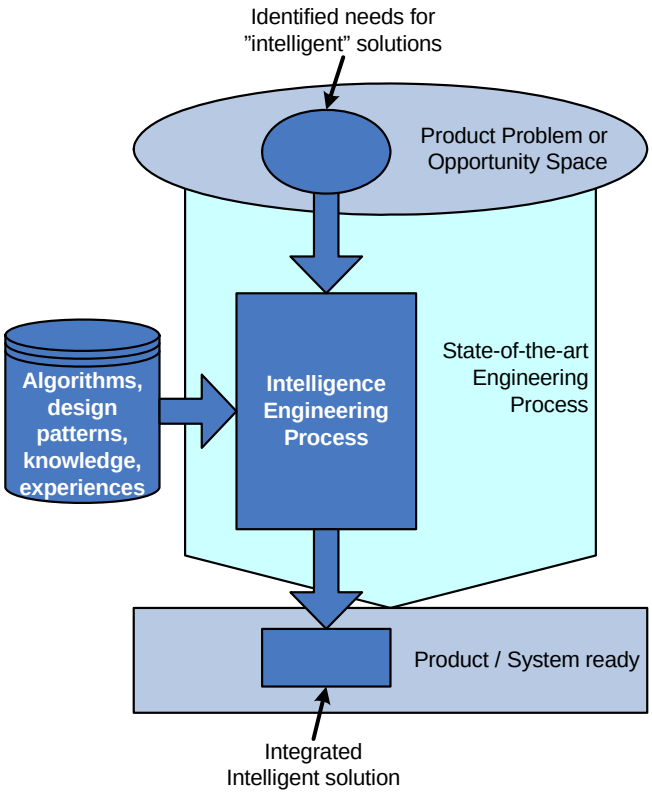

Figure 2. The research problem of the IEF project: how can advanced techniques be used to create systems that function in an intelligent way?

1.2 Document structure

The rest of this report is organised as follows. Chapter 2 gives an introduction to the problem domain of the IEF, i.e., a kind of “big picture”. The role of Chapters 3 and 4 is to discuss the current state of the art in design processes and implementation technologies that could be used to tackle the challenges created by intelligent systems. According to the project scope, the reviews focus on the issues considered most relevant to our purposes. Chapter 5 describes the developed IEF Design Pattern Language used to describe the solutions in the IEF knowledge base. The IEF toolset is described in Chapters 6, 7, and 8. Finally, Chapter 9 includes some conclusions and recommendations for the next steps. Appendices contain information about the ontologies and semantic modelling.

1.3 The definition of intelligence

Intelligence engineering is intended for applications used to monitor or control complex systems in areas like industry, health care, environmental monitoring and infrastructures of society. While the emphasis was on technical systems based on information and communication technologies (ICT), human-system interaction is foreseen as important

in these application areas. As fully automatic solutions are often impractical, successful intelligence emerges mostly from the joint action of people and computers. The technical system itself typically has a hybrid character, as it must combine various information systems, technologies and algorithms in order to implement the required functionality.

In order to outline the problem space and the design aim of intelligence engineering, we took a pragmatic view by interpreting the term "intelligence" in the following way:

"A system is intelligent if it behaves appropriately with respect to its purposes, not only in circumstances it was explicitly designed for, but also in rare, unexpected and even new kinds of situations."

Firstly, it should be noted that our definition of intelligence refers to the functionality of a system not its static features. Secondly, the appropriateness of the behaviour is evaluated in terms of system values and goals. Thirdly, intelligence requires the system to adapt itself to changing and unfamiliar situations. To enable intelligent behaviour, systems must have more concrete capabilities, such as:

- advanced computation and reasoning
- plug & play, reconfiguration, adaptation, learning, co-operation
- associating values with goals, data and alternative actions, rational decision-making
- proactive, goal-oriented planning
- management of conflicting goals, uncertainty and mistrust
- fault tolerance and recovery from failures.

As can be seen, the definition of intelligence and the required technical capabilities lead to a broad and challenging research area. We therefore selected a narrower scope by *focusing on intelligent systems that exploit computationally intensive algorithms for identifying system states and adapting to varying circumstances. With regard to design methods, we focused on model-based approaches, as they appear to offer the best tools for the design of complex entities.*

The aim of the Intelligence Engineering Framework was to define an efficient and practical design flow by augmenting current mainstream engineering methods with new concepts needed for intelligence, as defined above. It is largely based on design reuse, i.e., matching existing solutions to the specific design challenges of intelligent systems. Thus, the primary focus of IEF lies on the following issues:

- selection and application of advanced algorithms appropriate for the problem at hand
- modelling techniques
- solution libraries.

1.4 The development approach

The goal of the IEF is challenging, so we proceeded iteratively and focused on the early stages of design. In addition to reviewing related research, we selected four application cases to be analysed and used as the basis for the IEF. Although they come from different domains, they share the need for computationally intensive algorithms to analyse heterogeneous data sets. In fact, all the cases discussed the problem of identifying specific episodes, e.g., situations or anomalies (also called artefacts) in measurement data. The applications also represented complex entities that required advanced modelling techniques in their design.

2. Issues of algorithmic, intelligent systems and their design

The purpose of this chapter is to draw a “big picture” of the problem domain. The topics can be divided into two groups: 1) characteristics of intelligent systems and 2) the concept of joint intelligent systems.

2.1 System characteristics

The IEF focuses on the design process of intelligent systems, not their technical realisation. In order to discuss the current design issues properly, it is of course necessary to outline what is understood by an intelligent system. This need is more important still because some of the difficulties experienced in present design practice relate to insufficient insight into what the intrinsic characteristics of an intelligent system are. As indicated in section 2.1, our general definition of intelligence requires systems to have several capabilities. In the following we elaborate the most important ones.

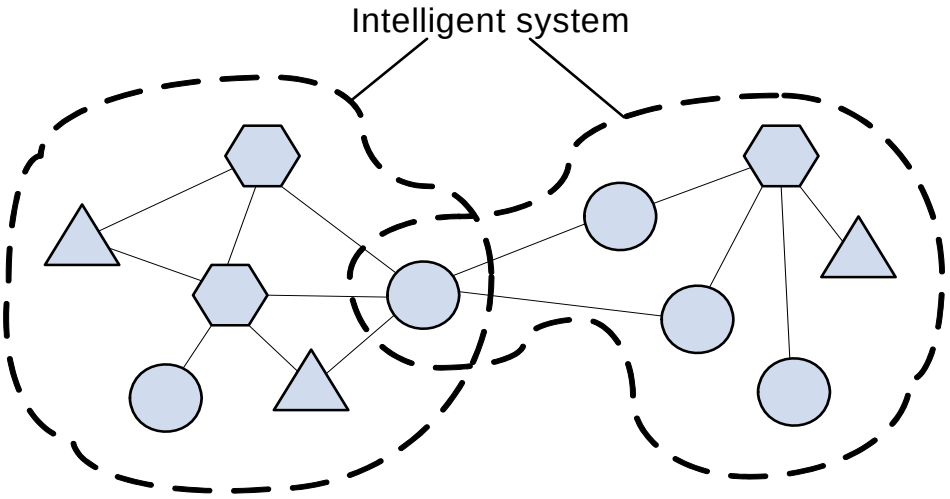

Figure 3. Intelligent systems typically connect various kinds of actors and subsystems.

Integration of heterogeneous components

Although there are stand-alone intelligent systems, there is often a need to combine the knowledge and services provided by several more or less intelligent systems (Figure 3). Such combinations can be referred to as “systems of systems”. They are characterised by large-scale networked integration of heterogeneous (technical and human) components (ARTEMIS 2006). Consequently, intelligent systems are typically geographically distributed and combine services provided by different nodes and subsystems developed, owned and maintained by separate companies. As a service can be used in more than one distributed application, various intelligent systems can overlap, i.e., system boundaries depend on the viewpoint.

Rationality

Intelligent behaviour should be guided by stated or implied goals. These goals may, however, be in conflict with each other. In addition, the actors within an intelligent system must base their actions on lacking and uncertain information. Rational decision-making and management of uncertainty are therefore features required for real intelligence.

Adaptation

Intelligence, by definition, requires a system to change its behaviour according to the current situation. In other words, adaptation is considered a key characteristic of intelligence. Adaptation can be achieved by, for example, modifying system composition (reconfiguration) or fine-tuning the functionality of existing parts. Adaptation can also be achieved by making use of the human actor’s capability to interpret situational requirements and change behaviour accordingly to maintain the results aimed for.

Encapsulated and emergent intelligence

The intelligence can be encapsulated within individual components or, perhaps more importantly, emerge from the co-operation and dynamic reconfiguration of less intelligent system elements. In both cases the mechanisms required for intelligent behaviour call for flexible system architectures, powerful data models and computationally advanced algorithms. In the case of “encapsulated intelligence”, the emphasis is often on the use of known algorithmic solutions, whereas “emergent intelligence” is based on complex interaction patterns.

2.2 The concept of the Joint Intelligent System (JIS)

As a conclusion of the characteristics above, systems consisting of multiple co-operating actors can be taken as a general framework for intelligent systems. Due to the limitations of any technology, however advanced, these systems have both technical and human components.

The concept of Joint Cognitive Systems (Hollnagel & Woods 1983, Woods & Hollnagel 2006, Norros & Salo 2009) focuses on the joint functioning of human beings and technology in intelligent systems. A joint cognitive system is composed of human and technical components that work together to produce certain behaviour. The essential feature of intelligence, adaptiveness, emerges from co-operation between the elements of the Joint Intelligent System. As we want to emphasise the *joint* nature of the human-technology system, on the one hand, and the *intelligent* features of the system, on the other, we have adopted the term *Joint Intelligent System.* Figure 4 illustrates the environment in which joint intelligent systems function. The joint intelligent systems are located inside the integrated system and can be seen as functional units that combine elements of human and technical intelligence.

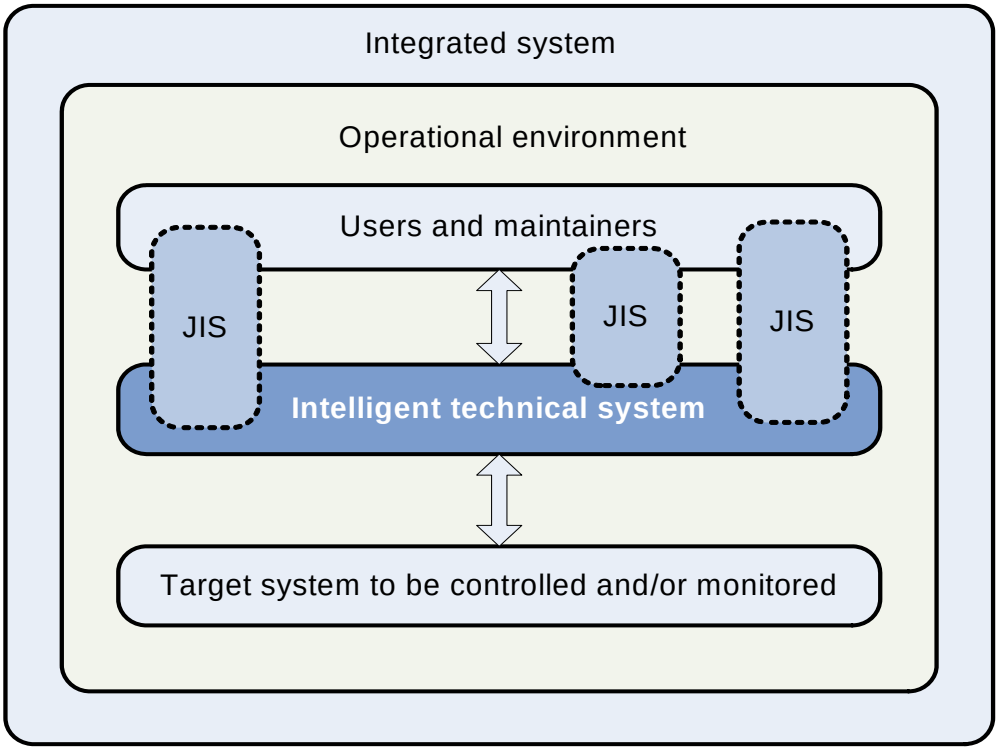

Figure 4. The environment of joint intelligent systems is the integrated system which consists of several technical and human elements located in the operational environment.

Adaptive patterns indicate general modes of acting and coping with the situational demands of Joint Intelligent Systems. The aim of the design is to identify appropriate adaptive patterns and develop generic and reusable technical solutions, or *design patterns*, for their implementation in different applications. *Situation awareness* is an example of an adaptive pattern according to which an intelligent system should function. Good situation awareness is not a result of simple recording of information. Instead becoming aware of the situation assumes active search for information and interaction between the technical system and the human user. Intelligent systems may also have other kinds of adaptive patterns, i.e., patterns of coordination, that relate to the temporal organisation of system functioning and the allocation of responsibility by different elements of the system.

3. Trends in design

The main goal of the IEF was to define a design process that supports the use of advanced techniques in developing complex software applications for intelligent systems. It was not realistic to start from scratch. Instead, the current mainstream design techniques had to be augmented with features that support our goals. We therefore had to consider the best practices of relevant areas of engineering. The following sections give a short overview of a few topics in design.

3.1 General systems engineering and design research

Historically, design has been a mixture of art, engineering practices and craftsmanship based on trial and error. With its growing complexity, more systematic approaches have become necessary. Design has also developed as an academic discipline in its own right, often termed “design theory”, “design science” or “design research”.

Design has been studied before, partly separately in different fields of science and engineering. General systems thinking, systems theory and cybernetics can be seen as a common basis. They bring together principles from ontology, philosophy of science, physics, biology and engineering. The focus of systems thinking, however, is on the analysis of existing (natural or technical) systems rather than on designing, i.e., synthesising new ones. *Systems Engineering* (SE) is an interdisciplinary approach to enable the realisation of successful systems (see, for example, http://www.incose.org and Sydenham 2004). It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements and then proceeding with design synthesis and system validation. Systems engineering considers both the business and the technical needs of all customers. In addition to the technical design processes, systems engineering includes activities like project management, customer agreement and procurement.

Systems engineering is a universal concept for managing complexity, and a way of thinking rather than a fixed set of rules. While technical designers tend to place strong emphasis on the details of a specific field, systems engineering tries to bring in more

abstract thinking and a generalist view. Traditionally, a design problem would be divided into sub-problems that could be solved. This reductionistic thinking cannot provide all the solutions to the complex problems encountered today. This has led to the emergence of so-called soft system methodologies that starts with the statement that the problem is unclear and cannot easily be decomposed (Checkland 1999). Such design problems typically contain a big human element in addition to the technical components.

Systems engineering is fostered, for example, by the International Council on Systems Engineering (INCOSE, http://www.incose.org). Although systems engineering had its start in the aerospace and defence industry, the universality of systems thinking has caused it to outgrow that realm. While the principles are intended for all application areas, the work seems to have some focus on software engineering (Tommila et al. 2007). For example, one of the important activities is the development of the OMG Systems Modelling Language (SysML) that has its basis in the Unified Modelling Language (UML). This makes sense, as systematic software development processes have been an important research area for many years.

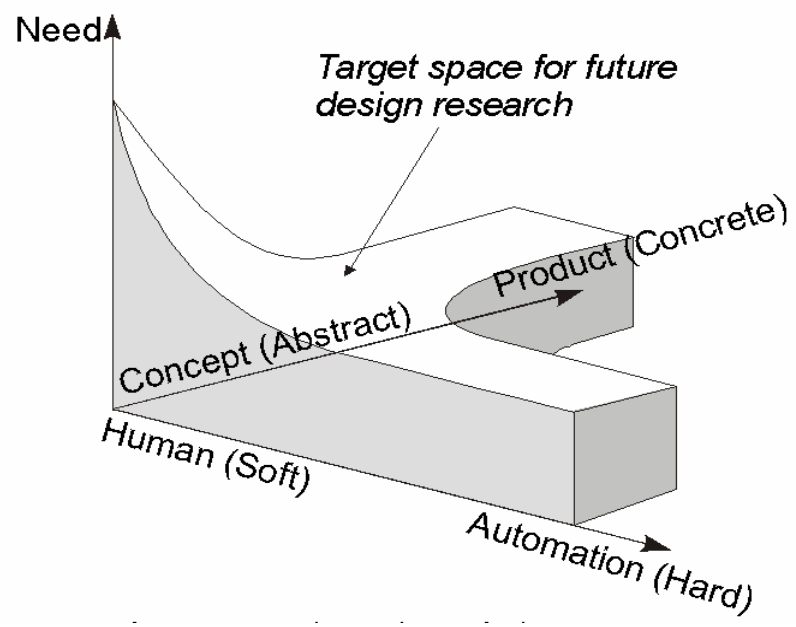

Figure 5. Research space as proposed by FIDR (1999).

Despite interactions between academic researchers and industry, research in general is not always well correlated with the realities of design practice (NSF 1996). In 1999, a group of researchers and industrialists (FIDR 1999) came to the conclusion that particular emphasis should be put on the conceptual and soft research areas (Figure 5) and, consequently, on inter-disciplinary co-operation in research.

3.2 Software development – trends and practices

3.2.1 Software development process

Software is typically a long-living product which is updated, improved and corrected along with its life cycle. The paradigm of software development can also be seen as a cycle, which includes requirements, design, implementation, testing and maintenance. These recognised life-cycle steps involve different kinds of analysis activities, which should also be identified in the life-cycle model.

The best-known life-cycle model is probably the one known as the waterfall model, but alternative models also exist, such as the traditional spiral model and several incremental agile processes, which are described in the sections below. An important starting point for any process model is to put emphasis on the requirements and their understanding through the different life-cycle stages. A framework for analysing the similarities and differences between the traditional life-cycle models is given in Davis et al., 1988.

Each life-cycle model provides a high-level view of phases and reviews and the order in which they occur. While they incorporate the same basic activities, the different models differ in the details related to their repetition, overlapping, etc. Notwithstanding these differences, the efficiency and success of the systems development effort probably depend more on a clear and complete definition of the products, processes and services expected at each step. An important task is therefore to establish the requirements engineering practices that yield such definitions.

Agile methods are lightweight processes that make use of short iterative cycles and involve users to actively establish, prioritise and verify requirements, and rely on a team's tacit knowledge and face-to-face communication over heavy loads of written documentation. Agile development is a general, descriptive term for approaches that are implemented by a number of different software development methods, such as XP (Extreme Programming), Scrum and DSDM (Dynamic Systems Development Method).

Agile methods are iterative and incremental, meaning that each iteration round is an entire software project including planning, requirements analysis, design, coding, testing and documentation. One iteration round does not necessarily bring much functionality to release the product to market, but the goal is to have a bug-free release at the end of each iteration. The agile way of developing software includes self-organising teams which determine the best way to handle work. The way of thinking and organising things is emergent, meaning that processes, principles and work structures are recognised during the project rather than being predetermined.

Agile processes have been criticised for their wild way of organising things and their lack of structure and necessary documentation. It has also been said that agile methods only work with senior-level developers and require much cultural change to be adopted. It all depends on the project and the organisation that try to be agile. The agile process is

definitely not the best solution for all kinds of projects, but the real challenge is to recognise the potential it has and where it can be used most effectively.

3.2.2 Requirements engineering

System requirements are capabilities that the system must supply or qualities it must possess in order to fit its intended use. Typically, requirements form a complex set of needs and wishes of people related to the system (stakeholders), and constraints and boundary conditions concerning the environment in which the system is supposed to operate. In traditional software development methods such as, for example, the waterfall model, the requirements play a very important role because they form the foundation on which the design and all the other parts of the system are based. If the requirements are changed during the later development phases (e.g., design or implementation), it is possible that the same changes will have to be carried out all the way through the development cycle and that parts have to be changed that have been finished once already. In agile methods, the development cycles are fast and the requirements can be updated in each round. Even in agile methods, however, the development is incremental and changes to the early phase requirements may cause as much trouble later in the project as in traditional methods. Agile methods have more and faster iterations, and the final design should be a result of several simple mock-ups, refactoring and tests (test-driven design).

There are several commercial requirements management tools that are meant to improve the requirements engineering process. The focus of the tools is mostly on the information management aspect of requirements management. They provide means for gathering, storing and editing information, as well as for managing the relations between different viewpoints and different parts of the requirements. People who do not know the area of requirements engineering well often think that a requirements management tool is the ultimate solution to the problems of requirements handling. This is not true, as the tool only offers a platform for requirements management. The tool itself does not solve anything. There is still a need to create a requirements process which best suits the purposes of the project at hand. The type and comprehensiveness of the requirements process depend on several things: the people and their experience, habits and working methods, application area and type and size of the project.

3.2.3 UML and SysML in software and systems modelling

UML has become one of the most popular modelling languages for software-intensive systems. UML consists of thirteen types of diagrams which are used for describing the architecture and design of the software. In addition to describing the software architecture, UML can also be used to model the requirements. Botaschanjan et al. (2004) suggest that the benefit of using UML for requirements modelling as well as for describing

the software architecture is that the developers do not have to handle different and incompatible modelling languages within the same project. Nonetheless, by using the same description language, there are some distinctions between the design and requirements models: requirements models are usually less detailed than design models, requirements models describe properties of the system as a black box and do not describe the internal structure, and requirements models often refer to the system as well as to the environment (neighbour systems as well as user behaviour), whereas design and implementation models concentrate on the system under development.

The Systems Modelling Language (SysML) is a domain-specific modelling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems of systems. These systems may include hardware, software, information, processes, personnel and facilities. (SysML 2007)

SysML is meant to be used for representing systems and product architectures as well as their behaviour and functionality. It builds on the experience gained in the software engineering discipline from building software architectures with UML. The architecture SysML describes and represents the elements, realizing the functional aspect of the system.

3.3 Broadening the human-centred design approach

Intelligent systems usually consist of multiple, co-operating human and technical components. Hence, the design of these systems must incorporate knowledge and design practices that deal with both technical and human issues. The integration of human factors in technical design already has traditions. Human-centred design as defined by the ISO13407:1999 standard (ISO 1999) is a well-established practice for the design of single software applications. With regard to the design of intelligent systems, the human-centred design approach has three major shortcomings:

1. An intelligent system may consist of several subsystems to be implemented gradually, and the users, contexts of use and tasks cannot be fully defined at the beginning of the design phase.
2. The design of intelligent systems should focus on the ways the system functions. Therefore, the understanding of cognitive and collaborative processes must be interwoven into the technical design in a much deeper manner than the human-centred design approach assumes.
3. The iterative prototyping defined by ISO13407:1999 considers iterations as more or less repetitive. As intelligent systems are complex, and the emergence of their capabilities are difficult to anticipate, they cannot be developed iteratively. Instead, the design should proceed via qualitatively different design steps in which targets and challenges vary.

Kaasinen and Norros (2007) propose a theoretical basis for a new design framework for intelligent systems. The ecological approach to designing intelligent environments focuses on the entity of people and technologies embedded in the environment – the modern ecosystem. According to the ecological approach, the aim of the design is not mere technology but the practices made possible by technology. Joint Intelligent Systems can be seen as units of intelligent environments, and their adaptive patterns can be considered to be the practices for which the design should aim. It appears that the basic ideas of the ecological approach could be applied to the design of Joint Intelligent Systems.

The ecological approach (Kaasinen & Norros 2007) states that the traditional product design approach needs to be extended with two new design levels: 1) immediate design and 2) remote design. Immediate design focuses on immediate user needs and local experiences, and emphasises the increasing role of users in the design. Design alone cannot create practices but offers possibilities that users utilise and shape into practices. Remote design aims to distance itself from the immediate needs and create more general solutions and infrastructures that provide possibilities for future services and applications. The identification of the two qualitatively different design levels provides the possibility to construct a new progressive (instead of iterative) design cycle.

Combining human factors with current model-based software engineering approaches is one of the key challenges of developing tools for a new design process for intelligent systems. Some attempts to broaden human-centred design towards the model-based design approach have been described (e.g., Constantine & Lockwood 2002). Agile design approaches often focus on customer rather than user participation. VTT's Mobile-D™ Agile computing approach (http://agile.vtt.fi/mobiled.html) allows active user involvement and supports gradual implementation of complex systems. Pattern-based design is a promising approach to connect human-centred design with model-based design approaches (e.g., Juristo et al. 2003; Bass & John 2003).

3.4 Reusable libraries and design patterns

In software development, design patterns offer the possibility of reuse. There are several established levels of reuse. The use of concrete software elements such as functions, classes and components is already well established and practised. However, if we speak about reuse at higher levels of abstraction, e.g., software patterns, reuse is still in its infancy.

Software patterns are usually presented in informal and loosely structured documents. These documents consist of several fields (e.g., name, context, forces, problem, resulting context, solution, implementation, etc) given as textual descriptions. They are intended to help developers in the understanding of the patterns, but there are no advanced knowledge management possibilities. There are also some semi-formal representations, usually based on UML. These are usually strongly tool supported enabling their inclu-

sion into solutions in a straightforward way. These semiformal representations are successful in capturing structures and behaviour, but they lack information and knowledge about high-level aspects such as intent, usability and consequences. To enable sophisticated use of design patterns, fully formal representations are needed. Within the design pattern community, the main goals in formalizing design patterns are:

- Better understanding of patterns and their composition. It helps to know when and how to use patterns properly in order to take full advantage of them.
- Resolving issues regarding relationships between patterns. It is not only relevant which design patterns are used to solve a problem but it is also important in which order they are applied.
- Allowing the development of tool support in activities related to patterns.

Following these lines there have been attempts to describe design patterns by semantic techniques. A sample approach, see Figure 6, is given in Pavlič et al. (2009).

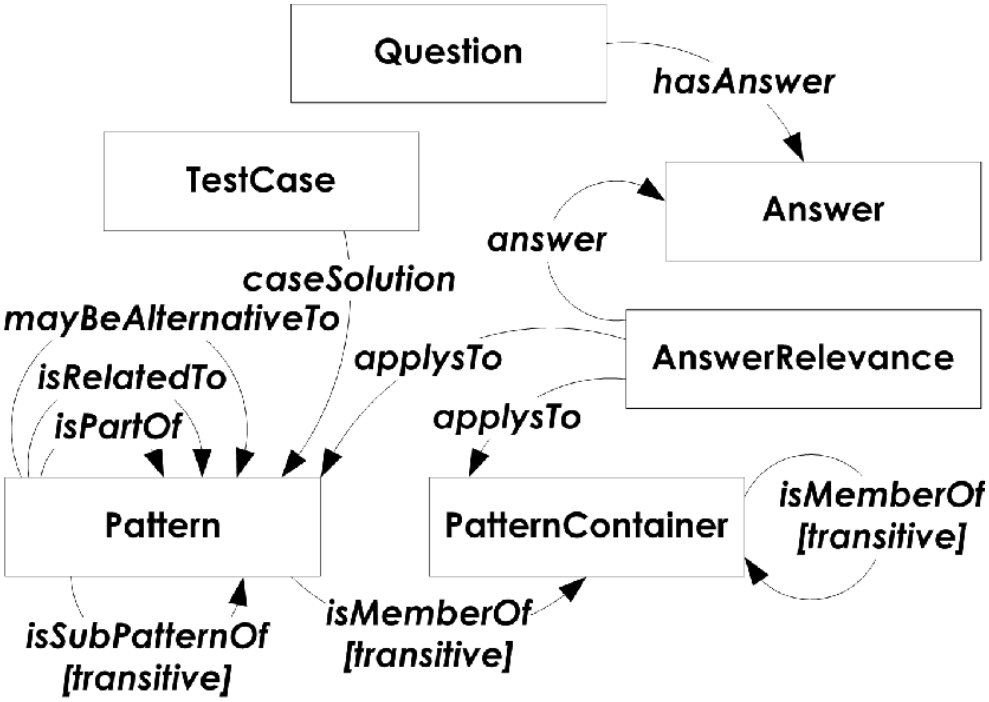

Figure 6. A sketch of Design Pattern Ontology.

4. Key technical solutions

Of the characteristics of intelligent systems, the IEF focuses on situation awareness and reuse of design knowledge. Situation awareness is an obvious pre-requisite for adaptation, and it links the human and technical agents in an intelligent system in a practical way. Reuse, in turn, seems to be the most appropriate way to turn advanced techniques into design practice.

4.1 Situation awareness of technical systems and their users

Adaptation to varying circumstances is essential in algorithmic, intelligent systems. This need is not new, and situation and context awareness have been widely studied in several domains, such as, for example, military operations, aviation, environmental monitoring, health care and monitoring of industrial plants. The two main communities looking at situational information are the human factors community and the engineering research community working on information fusion. The development of mobile and ubiquitous devices is an emerging area of context awareness.

Many definitions can be found for the terms “situation” and “context”. Endsley (1988 and 1995), for example, defines situation awareness as the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future. Dey (2001) sees context as any information that can be used to characterise the situation of an entity that is considered relevant to the interaction between a user and an application, including the user and the application themselves. A system is context aware if it uses context to provide relevant information and/or services to the user.

While these definitions have different backgrounds, they contain common elements, such as time, meaning and relevance. We took the freedom to combine existing formulations for the purposes of the IEF. We are interested in several application domains and looked at communities consisting of human as well as software agents, both of which should be situation aware. This led us to the following informal characterisations of situation and situation awareness to be used in the IEF (Figure 7).

- A *Situation* is a compilation of information that characterises the state of affairs of an entity or group of entities existing in space and time in a way that has meaning for a specific viewpoint.
- A system or component is *situation aware* if it maintains situational information and uses it to guide its actions towards its goals.

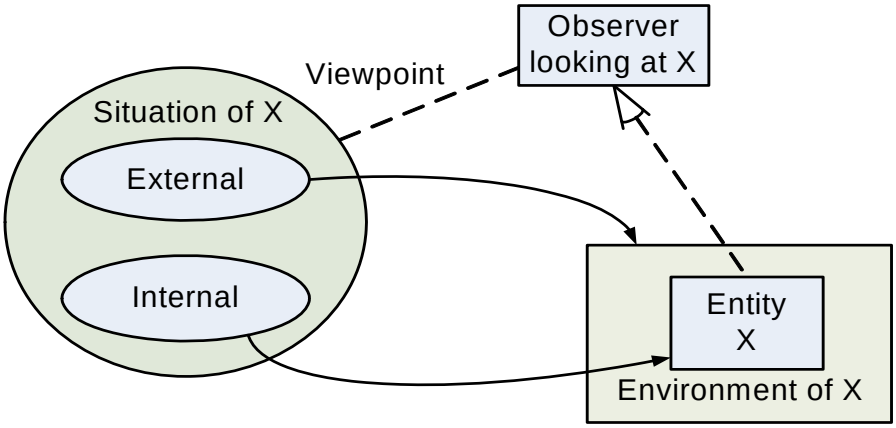

Figure 7. The IEF situation is understood as an information object that describes the status of an entity and its environment in the eyes of an interested observer (which in some cases is the entity itself).

In everyday language, the other widely used term "context" is understood as a set of circumstances or facts that surround a particular event. So, we might use it to refer to the world surrounding the entity of interest, e.g., the case of a mobile phone being aware of its user. Some researchers, e.g., Dey (2001), however, seem to understand context as lower-level observations that characterise situations. To avoid confusion, hereafter we limit our discussion to situation awareness, which in our minds also includes context awareness.

Due to our focus on software-based systems it is convenient to define situations as information objects, in other words, *models* that describe some relevant features of real-world entities. Situations contain information on entities as well as their environment (including users). They distil low-level information into knowledge that is meaningful from a determined viewpoint, for example, by a goal, activity or concrete domain entity. The meaning is created by understanding the relations between the available facts and their implications with respect to the goals associated with the viewpoint. Situations must therefore relate status information to goals in a way that helps in situation assessment. To make this possible, status information is not limited to the present state and should include recent history and anticipated future. Furthermore, to enable adaptation, situations should be linked to the means (e.g., resources and procedures) the actors can use to correct deviations between goals and the current status.

At the highest level, states of affairs can be abstracted to categories of states that are equivalent from the selected viewpoint, for example, “starting up” and “in maintenance” can describe the overall situation of a technical system. This leads to: 1) situations that can have names that refer to a category, i.e., they are instances of a situation type or template, and 2) situations that are occurrences, i.e., they exist for a limited period of time and are eventually replaced by new ones. In a similar way to the hierarchical structure of real-world entities, situations can be described in terms of lower-level situations. Consequently, here the evolution of situations is understood as a network of sequential and nested periods of time, each describing an episode in the history of the entity of interest (Figure 8).

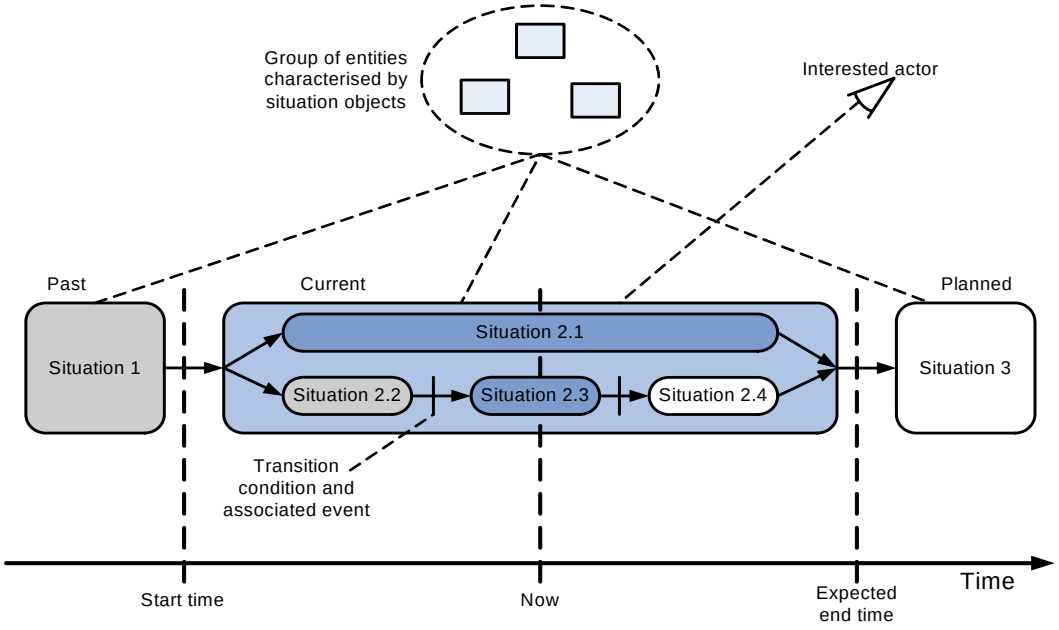

Figure 8. Evolution of situations in time.

Intelligent systems, as understood in the IEF, are typically communities of more or less situation-aware actors (humans and software agents) that share their intentions and situations. This joint activity improves the accuracy of the situational information of each actor. As a result, the situation awareness of the integrated system emerges from the co-operation of its less intelligent components.

In terms of technical implementation, situation awareness poses great challenges to developers, e.g., the evolution of situations over time, incomplete and uncertain information and the integration of heterogeneous data sources. Formal ontology languages and ontology-based reasoning are today’s approach to tackling these problems. As fundamental concepts, implementations and design processes are very diverse; no established design methods have been defined for situation-aware computer applications.

Important targets for further development are therefore concepts and methods that guide designers in situation management from requirements specification to system implementation and maintenance. The first step is to identify typical and important situations in the application domain and to analyse their features in terms of complexity, safety, relevance, etc. Operational states of subsystems and major disturbances detected in safety analysis are examples of good candidates for situations. As computers typically have a supportive role to carry out situation-specific tasks, known good principles of task allocation and information presentation are a starting point for situation support. "Computerised situation awareness" calls for new and more advanced solutions however. This has raised the following research issues for consideration in the continued work on the IEF:

- Formal situation ontologies and semantic reasoning
- Characterisation of situations in terms of features that allow the search for appropriate solutions for situation management (e.g., algorithms and design patterns)
- Technical images, i.e., a new pictorial language for making abstractions of situational information.

4.2 Semantic modelling

As the amount of stored data increases, the need for automated knowledge discovery, analysis, storage and utilisation is emphasised. These automated knowledge-handling processes face similar problems: heterogeneity of data sources, lack of common vocabulary and lack of support for reasoning. If the data are stored in a manner that holds knowledge about the hierarchical structure, relationships and usage of the data, automated knowledge discovery and utilisation will be easier.

Ontologies in information technology provide a common vocabulary for a specific problem domain. A common vocabulary holds the concepts, relations and axioms which give a formal definition of the problem domain. Knowledge of the domain can therefore be presented in an unambiguous way, readable both by humans and computer programs. If the ontology is well designed, programs in the same problem domain will be able to utilise each other's knowledge bases. Even if ontologies are not the same between two programs in the same domain, if the vocabularies are known, the knowledge bases can be shared and even integrated. The concepts in the ontologies can be mapped between different ontologies.

There are a few methodologies available for designing and implementing ontology. The three main methods are TOVE, EOM and Methodology (Fernandez, 1999, Gruber et al., 1995, Grüningen et al., 1995). Even though these methods are different they have the same basic process structure: identify motivation and scope (design/specification),

formalise concepts (conceptualisation), implement the ontology and evaluate the ontology. Timonen (2007) presents a general model for ontology development. The model integrates the process structures of all three main ontology development methods of our applications. We need a knowledge base in which we can query the semantic relations between applications, their data processing tasks, their design and usage, and the algorithms they use. By analysing the analogies between different applications, we can utilise the best architectural concepts of one application in the design of another, or algorithms utilised in one application to solve the data processing tasks of another application.

5. IEF pattern language

In the IEF, pattern discovery is about finding suitable guidelines for designing algorithm-intensive or intelligent systems. Basically, everything that can be found in the IEF knowledge base can be considered to be a pattern. A pattern is a composition of sub-patterns, and related and suggested patterns that give the designer guidelines in his/her task. In this document we describe a semantic pattern language related to signal validation. As an example, we concentrate on well-known approaches to diagnose instruments and/or actuators in industrial processes. These approaches are usually called "*Instrument fault detection and isolation*" (IFDI) (Betta, 2000) or simply "*Fault detection and isolation*" (FDI) (Mattila, 2008). These patterns contain, as a sub-pattern, data pre-processing, which is actually a very generic pattern and as such is applicable to the design of any intelligent system. As mentioned in a wider scope, part of the FDI can be seen as a special case of SIGNAL VALIDATION. In addition to validation, IFDI and FDI also contain isolation of the reason for the invalidity of the signal.

In describing FDI, we restrict ourselves to continuous processes, i.e., the process industry, but the pre-processing pattern is generic and not restricted to the process industry.

The discovery process itself should be requirement driven. This means that at least the most general patterns should be found according to a semantic description of the requirements of the design process at hand.

In addition to the Coplien form, we will model the patterns semantically, see Figure 9.

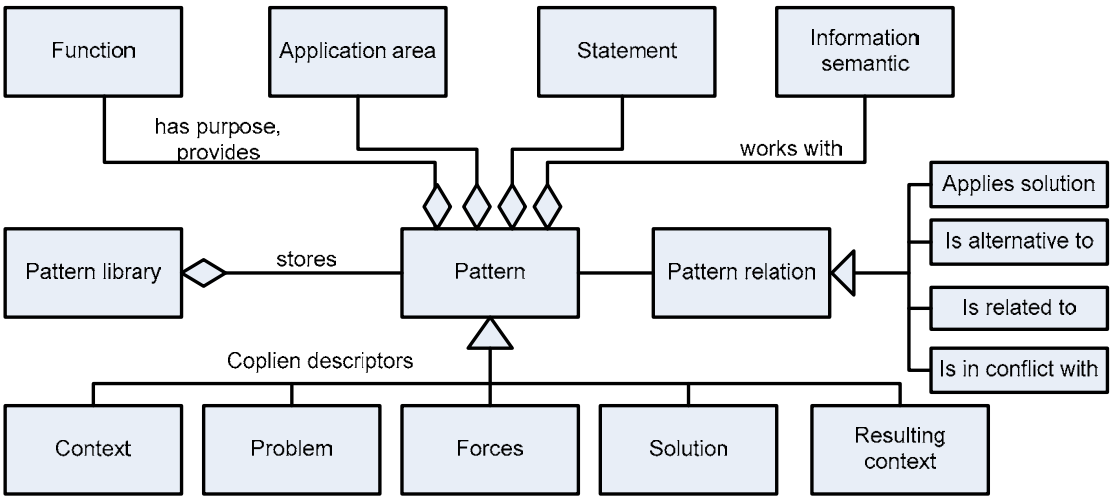

Figure 9. Pattern description.

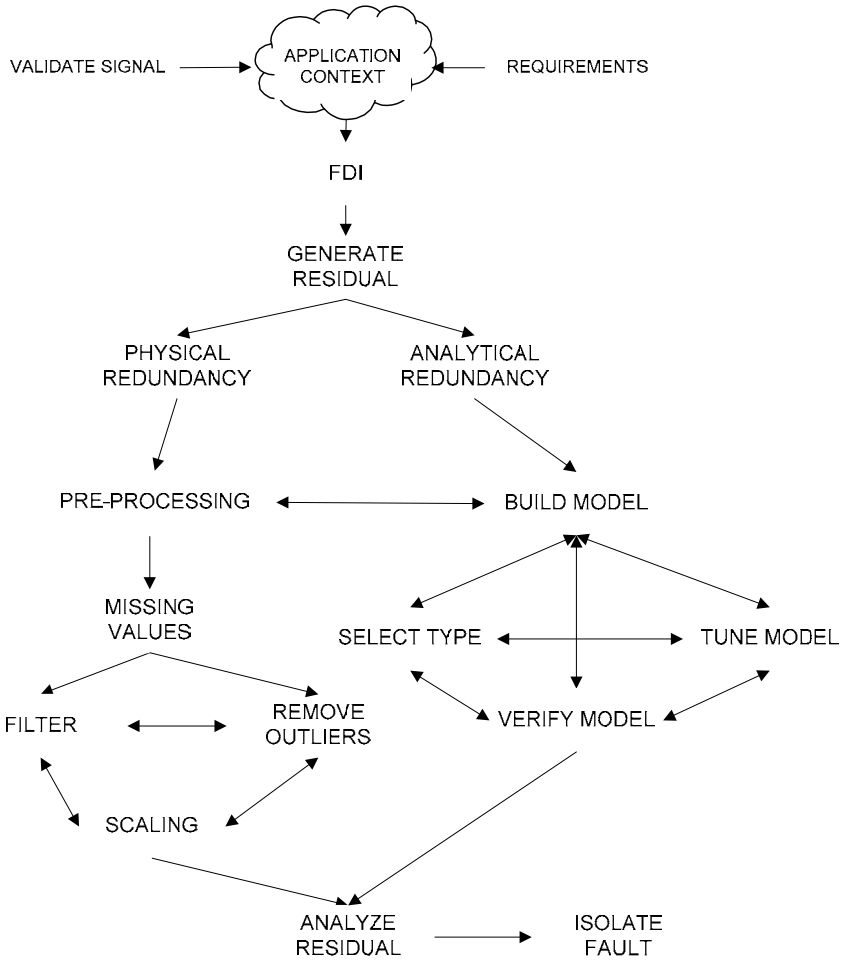

Figure 10. A sample IEF pattern road map.

A road map of the pattern language is shown in Figure 10. In this road map the starting point is the VALIDATE SIGNAL, REQUIREMENT and APPLICATION CONTEXT patterns (not described formally here). The number in parentheses after each pattern name is the number of the chapter below. We will also use other patterns which are not described here. All the patterns are written in capital letters, e.g., CONTROL SYSTEM.

In the pattern description, we will also refer to certain techniques and concepts defined in the IEF Core ontology. These will be denoted by ***UPPERCASE BOLD ITALICS***. Figure 11 depicts some modelling techniques used by the BUILD MODEL pattern.

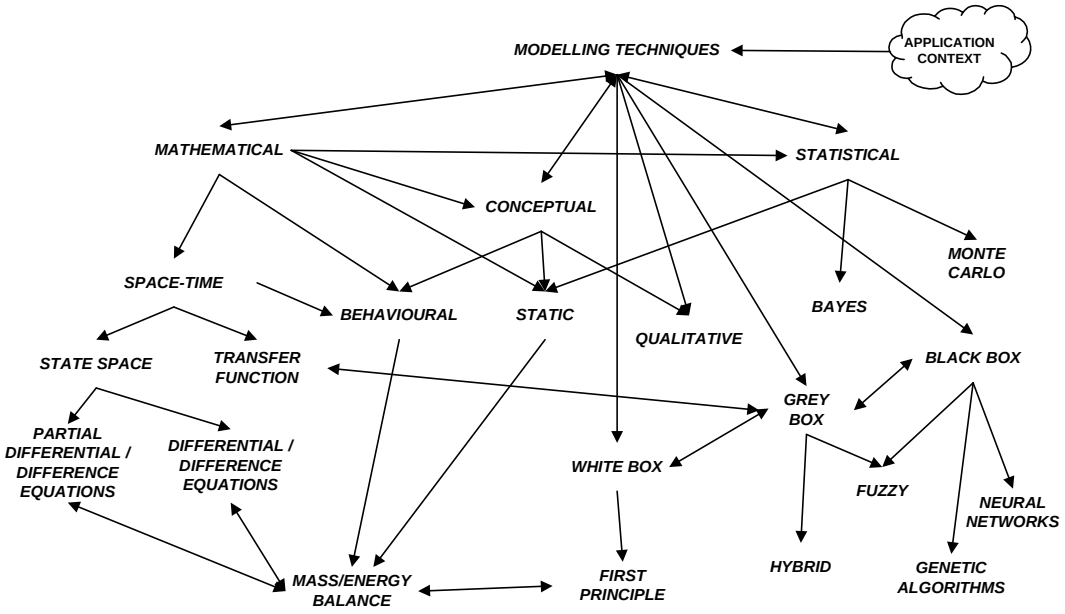

Figure 11. Simplified relationships between MODELLING TECHNIQUES.

1. BUILD MODEL

Context: APPLICATION CONTEXT: ANY

Problem: A ***MATHEMETICAL MODEL*** of a system is required for some reason.

Forces: Describing the whole essence of building a ***MATHEMETICAL MODEL*** is beyond the scope of this document. To build a dynamic model, the reader is advised to consult, for example, Fishwick (2007).

Solution: See above.

Resulting context: Model created and verified.

2. SELECT TYPE

Context: APPLICATION CONTEXT

Problem: While building a ***MATHEMETICAL MODEL*** of a system, one task is the selection of model type.

Forces: The selection of model type depends on the following factors:

1) The purpose of the model has a profound effect on the choice of it. It could be said that the opposite ends are qualitative models that can give a rough estimate of the behaviour of the system to be modelled. On the other extreme, there are ***FIRST PRINCIPLE*** models that are based on the laws of physics and chemistry.
2) Data availability: in the case of industrial processes there are usually enough data in the history database.
3) The amount of data required depends heavily on the model type, e.g., some neural network models require excessive amounts of data.
4) When the model is built for fault detection, the amount of data available can be scarce due to the rarity of the fault.
5) Some model types are applicable to steady state situations and some to transients; this has an effect on the amount of usable data.
6) To model the system according to FIRST PRINCIPLE equations (physics and chemistry) requires deep knowledge about the system and can be computationally too demanding.

Solution: In the case of FDI, a model should be chosen that predicts the behaviour of the system to be modelled as accurately as possible.

Resulting context: Model type selected.

3. TUNE MODEL

Context: APPLICATION CONTEXT

Problem: When the model type has been selected and the modelling work has been done, the model has to be calibrated so that its behaviour is similar to that of the real system.

Forces: 1) Different model types require different calibration methods.

Solution: 1) In the case of the ***NEURAL NETWORK MODEL***, use the TEACH NN pattern.

2) In the case of the ***PARAMETRISED MODEL***, use the IDENTIFY PARAMETERS pattern.

Resulting context: The model is operating as expected.

4. VERIFY MODEL

Context: APPLICATION CONTEXT

Problem: After the TUNE MODEL (3), the model's validity for the given purpose has to be evaluated.

Forces: 1) Verifying a model requires data that are independent of the data used in the TUNE MODEL (3) patterns.

2) Complete verification is usually not possible.

3) The developed models are usually valid in certain operational regions.

Solution: 1) Acquire enough data for the verification task.

2) Define test cases.

3) Analyse test results.

Resulting context: Model is verified.

6. Requirements for IEF modelling concepts

There will be different types of development projects using different versions (or profiles) of the IEF. Firstly, there is a difference between the components of stand-alone systems (e.g., an intelligent machine), typically developed by one company to be used within an overall (open or company-specific) framework, and integrated applications which need to integrate components from several vendors. Secondly, the development of products and platforms obviously calls for approaches that are different from those of application development.

The IEF will provide consistent concepts, modelling methods and tools, methods and criteria to analyse the quality of design, recommendations for their usage, and a recommended design process for various types of development projects.

A major part of the IEF is obviously based on the current best practices and standards of systems engineering and software engineering.

The IEF will use the model-based design approach (e.g., MDA) right from the domain analysis and requirements specification. The focus, however, will be on the specific issues related to the selection and use of intelligent algorithms.

It is claimed that design must be based on some kind of component architecture to be efficient in practice. This means that existing, intelligent algorithms could be encapsulated into the components. The intelligent functionality is generated not only by individual intelligent components, but also through collaboration between those components. The orchestration of the overall behaviour is based on the intelligent interaction mechanisms provided by the platform. Ways have to be created for joint functioning of human and technical elements, i.e., mechanisms of coordination and communication.

Consequently, there is no need for the IEF to know about the internal structure and algorithms of various components. It should be enough that the developers obey the platform rules and provide the information needed for using their components. This includes, at least, interface specifications and guidance on the application of the components (e.g., where it is good and where it is not).

The IEF design approach should include the idea of alternation between different modes of research and design during the development process of intelligent systems. This results from the need for both specific and generic solutions.

- The models created during the IEF-based design process should cover both the intelligent technical system itself and its environment (process to monitored etc.). In addition, they will describe the current situation and other input information to be designed.
- Models should also include various types of information at different abstraction levels, including, for example, requirements for engineering data (problems, goals, design constraints, etc.), system functions and implementation (e.g., HW and SW).
- Models and human readable documents generated thereof should be understandable by the relevant design parties.
- Models must be well structured (formal) and computer-processable, allowing electronic information management and exchange, as a minimum. As far as possible, however, the models should be "computer-understandable" so that they support advanced analysis, inference and integration of partial models.
- Complex applications are hard to understand, learn and test. The models should therefore be executable. This will allow system animation and simulation early during the design stage.
- The IEF must rely heavily on the reuse of design knowledge. For the early design stages it needs, for example, patterns and measurable features for application domains, requirements and available solutions.
- Powerful libraries and methods like similarity measures or case-based reasoning must be used to match application requirements with suitable solutions such as architectural patterns and advanced algorithms.

For subsequent design stages (detailed design and implementation), a combination of two approaches can be imagined: automatic model transformation and code generation, and the aggregation of applications from ready-made components.

7. IEF toolset

7.1 IEF toolset requirements

The future IEF toolset should provide several design-support activities. Some of these activities are listed below.

- Guidance in the design process
- Possibility to browse and edit context models
- Importing and exporting data from and to context models
- Adding and editing target system requirements, and tracking changes in them
- Searching for solutions (patterns, techniques) matching target system requirement(s)
- Browsing and evaluating found candidate solutions
- Generating application instances from patterns
- Storing design rationale (search criteria, user inputs).

7.2 IEF toolset usage scenario

The following example illustrates a simplified use case of the future IEF tool in the context of process diagnostics in a paper mill. The basic assumptions behind the use case are:

- The context has been modelled semantically in the IEF knowledge base, i.e., there exist models of
    - The target process and its physical entities (paper mill, paper machine, etc.)
    - The mission of the integrated system (produce paper)
    - The people (plant operators, maintenance personnel, etc.)
    - Other technical systems
- The operational environment is modelled in the IEF knowledge base.

- The context model in the IEF knowledge base is structured in a way that allows traceable references to
    - Requirements (original and updated design requirements)
    - Data and functions (logical view of the production process)
    - Hardware involved in the production (paper machine, control system, etc.)

During the night shift in normal operation, the operator of the paper machine PM1 makes a note in the operator's diary: "Noticed unwanted oscillation in headbox consistency. The frequency of oscillation varies from approx 1 cycle/min to 2 cycles/min. During my shift the amplitude seems to be increasing. The oscillation could not be dampened with standard procedures."

In the morning, the maintenance person reads the note in the operator's diary and sets up a new design project in the plant's IEF system.

- First he selects the system: PM1, headbox.
- Second, using the requirement description language tool, he writes the requirements of the project into the IEF system in natural language. The requirement reads: "Find reason for oscillation in consistency." He also inputs the oscillation frequency.
- Third, he inputs the situational context: normal operation, duration, PM1 speed, grade changes.
- Fourth, he inputs (selects) additional design constraints and rationalities like: "This is troubleshooting, fully automatic system not required, etc."
- After the input, he commands the IEF to create a solution. Based on the information that exists in the IEF knowledge base and fed by the designer, the IEF system proposes to use the FDI (Failure Detection and Isolation) pattern located in the Industrial Pattern Library. He creates an instance of the FDI in his design workdesk.
- According to the FDI pattern, the new system requires redundant information. Furthermore, according to the FDI pattern and context, the designer makes the decision that the future system must use analytical redundancy, i.e., he must create a reference model of the paper machine PM1.
- Consulting the Create Analytical Redundancy pattern, he moves to the Build Model pattern. According to Chapter 5 this pattern has several sub-patterns. The first is Model Type Selection. After checking the data availability (during last night PM1 was running in normal in a steady state), he notes that there are enough data for almost any model in the history database.

- Given this information, the IEF system recommends several possible modelling techniques, one of which is MAR modelling (Multivariate Autoregressive). After consulting the solutions stored in the IEF knowledge base, the designer decides to use MAR modelling and creates an instance of the MAR model in his desktop.

Continuing in an obvious way, after having selected the variables, time windows, and pre-processing algorithms, the designer creates a reference model of the PM1 which he verifies using data from similar operational states recorded in the plant history database.

The final steps in the FDI pattern are detection of the fault in the created residual and isolation of the fault. According to the design constraints and rationalities, he decides to leave these tasks to the human user, concluding his design task. The newly created system is stored in the IEF knowledge base as an example for possible reuse.

Figure 12 illustrates the functionality of the IEF system in finding matching solutions. The IEF system consults upper level ontologies and the knowledge base, suggesting solutions according to the context models and requirements.

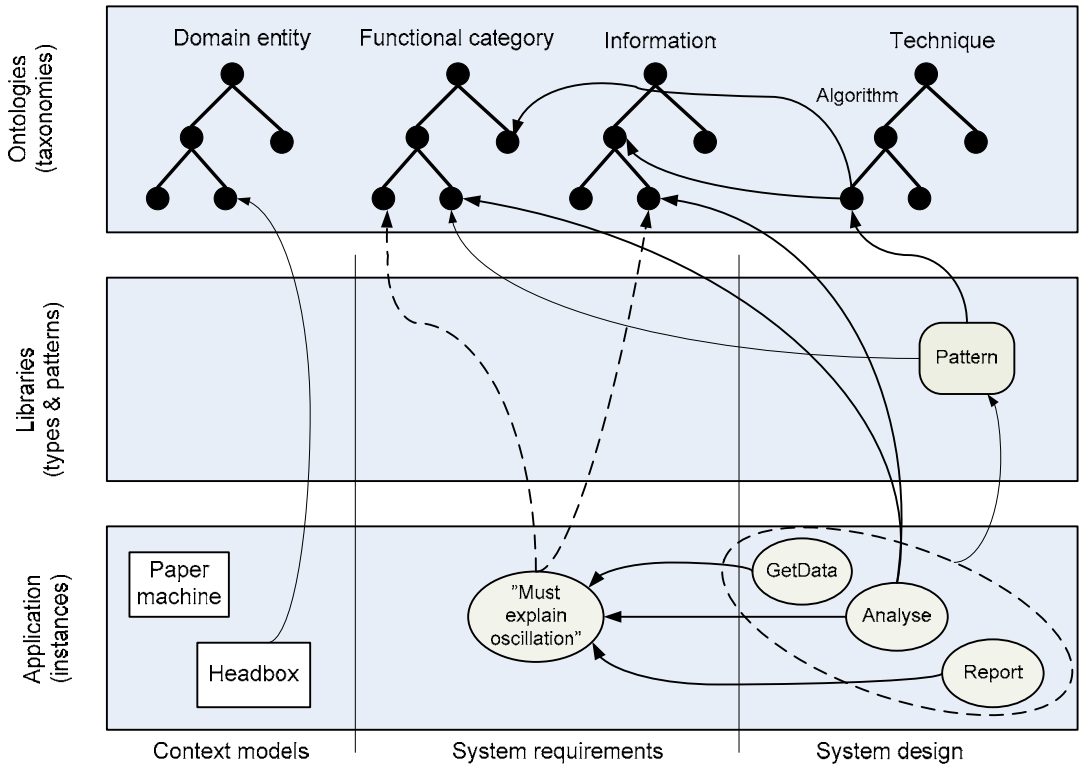

Figure 12. Ontologies binding problems and solutions.

8. IEF demonstration software

8.1 Structure of the IEF tool demonstration

The IEF Core and Application Area ontologies were developed with Protégé V3.4. The Protégé software provides a flexible base for application development. For these reasons the demo software was developed as a plugin to the Protégé ontology editor.

The demo software is a tab widget built on top of the plugin framework developed at Stanford University. The plugin framework resembles the OSGi technology, which is used in later Protégé versions (4.x).

The demo software realises the intermediate activities of the future IEF toolset:

- Possibility to browse and edit context models:
    - It is possible to browse a design database. The design database consists of projects and the requirements and other statements concerning the projects.
    - The software allows the user to open and edit the resources in Protégé (a separate Protégé window is opened).
- Importing and exporting data from and to context models:
    - Projects, requirements or some other statements concerning the projects can be chosen as the search basis. The proper search terms are added automatically.
    - Importing of data is not implemented.
- Adding and editing target system requirements:
    - The search criteria can be edited by the user. The search criteria hierarchies (application area, functional category, information semantic) can be browsed and each criterion can be added separately as a search term.

- Searching for solutions (patterns, techniques) matching target system requirement(s):
    - According to the search terms, the matching patterns, techniques and projects can be found.
- Browsing and evaluating found candidate solutions:
    - The candidate solutions can be browsed. Additional information on the solution is shown. Related solutions, related libraries and applied solutions are also shown. Links to external resources can be used to provide even more information.

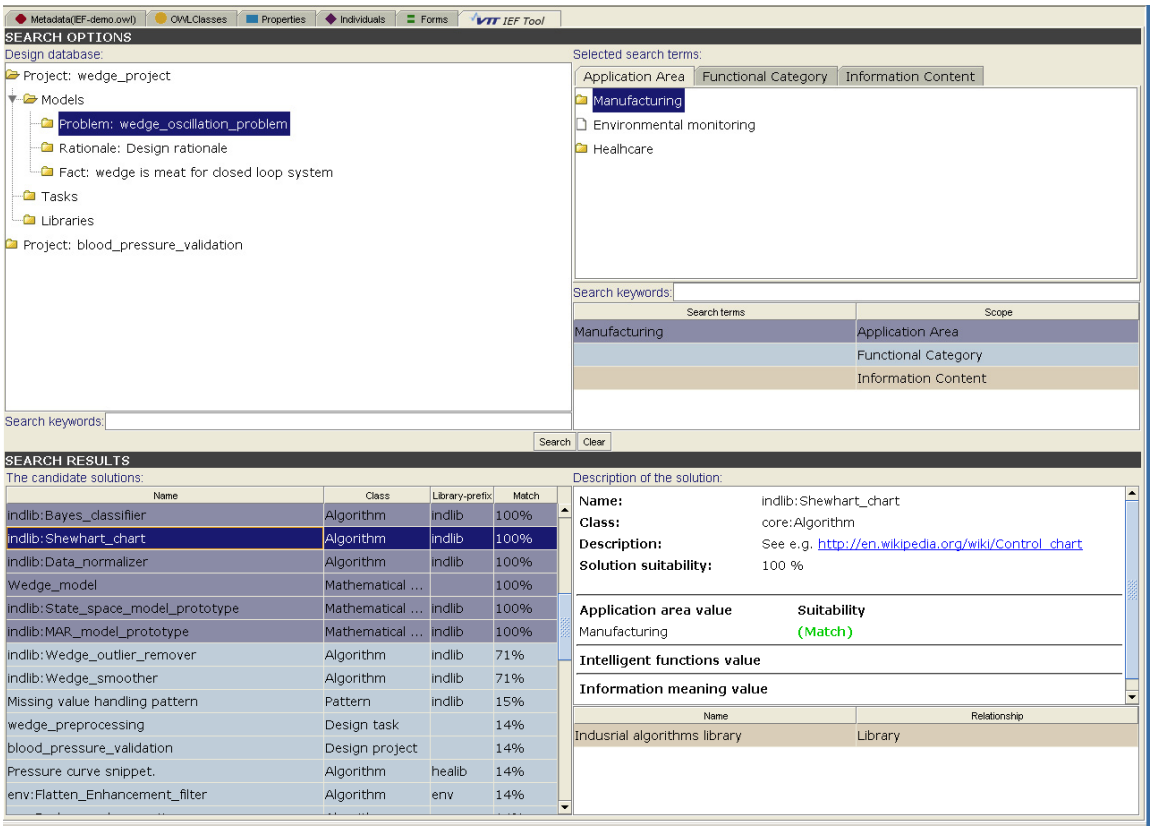

Figure 13. IEF tool demonstration, solution search.

8.2 IEF tool demonstration, search details

The IEF Core and Application Area ontologies exploit Full OWL description logic. Hence, for the time being, there are no suitable reasoning machines available. The IEF tool demonstration software implements the solution search as follows.

Search criteria

The search is conducted using three different search criteria: application area, functional category and information content (time series, image, list, etc.). The hierarchies of the criteria are extracted from the ontology and the hierarchies are graphically represented as trees in the software. An arbitrary number of search terms can be chosen from each tree. The three search criteria all carry the same weight (33%) to the search.

Match relevance

The search sorts the solutions according to their relevance to the chosen search terms. A solution is looked for that satisfies each criterion as well as possible. The candidate solutions are examined one by one. The suitability of each solution is evaluated with respect to the search terms. A search criterion is completely fulfilled if the candidate solution has at least one of the searched terms of this criterion. If an exact match cannot be found, a numerical suitability value is calculated for each of the solution's values. The highest value determines how well the solution corresponds to the search terms.

Suitability calculation

If an exact match is not found between a solution value and a search term, the suitability between these two values must be given a numerical value. The suitability is determined from the tree structure of the criterion. The search term represents one of the tree nodes. The solution value is also one of the tree nodes. First, a path between these nodes must be found. This is done by travelling towards the root node from both nodes until a common ancestor is found. This combined path is then travelled from the searched node towards the node that has the solution value. Each step in the path decreases the relevance of the node with respect to the searched node. The amount of reduction to the relevance is estimated according to a certain coefficient. If the path goes upwards in the hierarchy, a relatively small coefficient (0.3) is used because going upwards means that the new node also encompasses things that are irrelevant to the searched node. When the path goes downwards, a rather big coefficient (0.95) is used as the relevance with respect to the previous node does not disappear. The depth of the path in the hierarchy adds an extra coefficient to the suitability value.

Depth

The search logic takes into account the depth of the examined path in the hierarchy. This is because it is more likely that two closely located nodes in the hierarchy resemble each other if the nodes are located deep in the hierarchy. The depth of a path is determined by the ancestor node of the path (the distance from the root node to the ancestor +1). The depth of the path adds another coefficient to the numerical value. The depth coefficient is (1–0.5^depth).

Figure 14 shows the search implemented in the demonstration software.

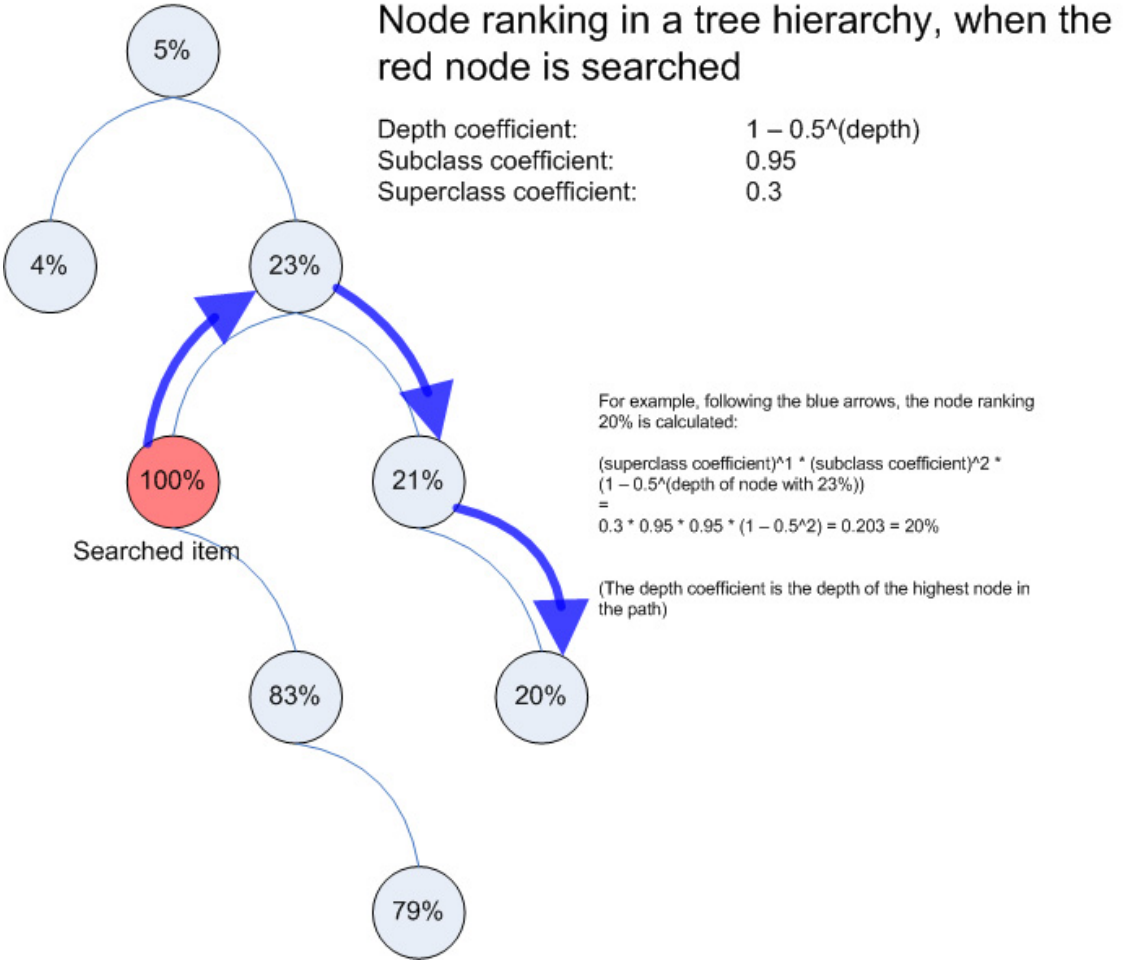

Figure 14. Search semantics.

9. Conclusions and next steps

The key research question was how the application of advanced algorithmic methods, such as neural networks or statistical analysis, could be made more efficient. This challenge was approached from different directions. Firstly, in addition to reviewing related research results, we analysed two application examples, namely supervision of an industrial process and patient monitoring in an intensive care unit. The aim was to maintain a practical touch and to combine the various competence areas available at VTT. The focus at this stage was on the early stages of the design process, a grey area of design widely known as problematic. Secondly, while the initial problem statement was rather technical, we first discussed the elements of "intelligence" and outlined the general characteristics and complex, sociotechnical systems.

One of the main conclusions was that rather than being contained by a single component, intelligence is often an emergent feature of several heterogeneous (technical and human) actors. It was obvious that the design of such systems is a big challenge. We therefore narrowed the scope of the IEF down to situation awareness, which, according to our definition, is one of the main elements of "intelligence". This directed our interest towards the design of intelligent systems that exploit computationally intensive algorithms for identifying system states and adapting to varying situations.

While the IEF cannot be given an exact shape at this stage, the overall picture started to emerge from our research. Firstly, as intelligent behaviour is typically embedded into a larger application, its design must be seen in the context of current engineering practices. In other words, the IEF augments current mainstream modelling methods and design processes with some new features. Secondly, the major challenge of the IEF lies in reducing the inherent complexity of intelligent systems. This is partly related to the known difficulties of applying advanced algorithms and maintaining their applications. The challenges are partly caused by the fundamentally new kinds of phenomena that result from complex interactions in large-scale dynamic systems. The obvious approach for the IEF is to enhance the reuse of design knowledge in all phases of the product or system life cycle, leaving it in the form of application examples, guidelines, design patterns or software components. This is a widely recognised problem area in which the

IEF should focus on issues related to intelligence, for example, system architectures and algorithms that support adaptation, reconfiguration, management of uncertainties and goal-oriented, value-driven behaviour. As design practices are partly domain-specific and change continuously, the core of the IEF should be rather general so that it allows more practical approaches and tools to be developed for different application areas.

The focus of the IEF on the early stages of design gives rise to two issues of design reuse: 1) how to model the application's needs for intelligence and the features of potential solutions stored in solution libraries, and 2) how to help the designer search the libraries for solutions that provide the best match with the application needs. This opens widely studied research fields such as case-based reasoning and the use of analogies and similarity measures to find solutions to the problem at hand. While recognising the difficulties associated with these approaches, we suggest them as a basis for the development of the IEF, especially for the early steps such as the requirements specification and conceptual design. We feel that the emerging techniques of the semantic web will be a good enhancement to the more traditional methods of design reuse. As appropriate modelling of the application domain and the overall integrated system are the basis of any design, we should augment the various domain ontologies with semantic information that serves the purposes of the IEF. Similarly, solution models in design libraries should be given additional information that describes their intelligence capabilities and limitations. Search methods and reasoning using this information would be the basis of new design processes and tools.

Keeping in mind that the IEF should support the whole life cycle of intelligent systems, we can list the following research issues to be considered in the continuation of the IEF:

- Ontology-based reuse of design knowledge, design patterns and algorithms
- Automated model transformations and code generation
- Analysis and model checking of intelligent system.

References

ARTEMIS (2006). ARTEMIS – European Platform on Intelligent Embedded Systems Strategic Research Agenda – Design Methods and Tools, version 1. http://www.artemis-office.org/DotNetNuke/Portals/0/Documents/RAPPORT_DMT.pdf.

Bass, L. & John, B.E. (2003). Linking usability to software architecture patterns through general scenarios. Journal of Systems and Software, Vol. 66, Issue 3, 15 June 2003, pp. 187–197.

Betta, G. (2000). Instrument Fault Detection and Isolation: State of the Art and New Research Trends. IEEE Transactions on Instrumentation and Measurement, Vol. 49, No. 1, February 2000.

Botaschanjan, J., Pister, M. & Rumpe, B. (2004). Testing Agile Requirements Models, Ed.: Wu Xiufang. Journal of Zhejiang University Science, 2004, Vol. 5, No. 5, ISSN 1009-3095.

Checkland, P. (1999). Systems Thinking, Systems Practice (Includes a 30-year retrospective). John Wiley & Sons Ltd. 330 p.

Constantine, L.L & Lockwood, L.A.D. (2002). Usage-centered Engineering for Web Applications. IEEE Software, March/April 2002, pp. 42–50.

Davis, A.M., Bershoff, E.H. & Comer, E.R. (1988). A strategy for comparing alternative life cycle models. IEEE Trans. Software Engineering, 14, 10 (Oct. 1988), pp. 1453–1461.

Dey, A.K. (2001). Understanding and using context. Personal and Ubiquitous Computing, 5, pp. 20–24.

Endsley, M. R. (1988). Situation awareness global assessment technique (SAGAT). Proc. of the National Aerospace and Electronics Conference (NAECON).

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. Human Factors, 1995, 37(1), pp. 32–64.

Fernandez Lopez, M. (1999). Overview of methodologies for building ontologies. Workshop on Ontologies and Problem Solving Methods (IJCAI99), Stockholm, Sweden, July 31–August 6, 1999.

FIDR (1999). Future Issues For Design Research (Fidr) Workshop, Bath 18.–20.2.1999, final report, http://people.bath.ac.uk/enssjc/fidr/fidr_title.html.

Fishwick, P.A. (2007). (Ed.). Handbook of Dynamic System Modeling, Taylor & Francis Group, Boca Ration, Fl, USA.

Gruber, T. (1995). Towards Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human Computer Studies 43, 5 (1995), pp. 907–928.

Grüningen, M. & Fox, M. (1995). Methodology for the Design and Evaluation of Ontologies. In: Proceedings of IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Canada.

Hollnagel, E. & Woods, D. (1983). Cognitive systems engineering: New wine in new bottles. International Journal of Man-Machine Studies (18), pp. 583–600.

ISO 13407:1999. (1999). Human-centred design processes for interactive systems. International standard. International Standardization Organization, Geneva, Switzerland, 1999.

Juristo, N., Lopez, M., Moreno, A.M. & Sánchez, M.I. (2003). Improving software usability through architectural patterns. Workshop Bridging the Gaps between SE and HCI – ICSE03. http://www.se-hci.org/bridging/icse/p12-19.pdf.

Kaasinen, E. & Norros, L. (eds.) (2007). Älykkäiden ympäristöjen suunnittelu – Kohti ekologista systeemiajattelua. Teknologiateollisuus ry. ISBN 978-951-817-944-6. http://www.teknologiainfo.net/default.asp?docId=12360&productId=14903&fromSearch=true.

NSF (1996). Research opportunities in engineering design – NSF strategic planning workshop, April 1996, http://asudesign.eas.asu.edu/events/NSF/report.html.

Mattila, M. (2008). Improving the robustness and reliability of industrial online analyzers using condition monitoring technologies and remote support, Helsinki University of Technology, Information and Computer Systems in Automation, Report 14, Espoo 2008.

Norros, L. & Salo, L. (2009). Design of joint systems – a theoretical challenge for cognitive systems engineering. Cognition Technology and Work, 11, pp. 43–56.

Pavlič, L., Heričko, M., Podgorelec, V. & Rozman, I. (2009). Improving Design Pattern Adoption with an Ontology-Based Repository. Informatica 33 (2009), pp. 189–197.

Sydenham, P. (2004). Systems Approach to Engineering Design. London, Artech House, Inc. 333 p.

SysML webpages (2007) (http://www.sysml.org).

Timonen, M. (2007). Implementation of Ontology-Based Biological Knowledge Base. Master's Thesis, University of Helsinki.

Tommila, T., Jansson, K., Karhela, T., Kiviniemi, A., Koivisto, R. & Sääski, J. (2007). Industrial engineering road map (INDERO) – A pre-study in the Finnish process industry. VTT working paper. Unpublished. 63 p.

Woods, D. & Hollnagel, E. (2006). Joint cognitive systems – patterns in cognitive systems engineering. Boca Raton: Taylor & Francis.

Appendix A: IEF Ontologies

IEF Core ontology

The IEF Core ontology was developed with Protégé V3.4. It contains all common concepts. Application ontologies extend the core ontology with application-depended concepts. A complete IEF tool will include the core ontology, application area ontologies and tool user's knowledge. Figure A1 depicts the top level concepts of the core ontology.

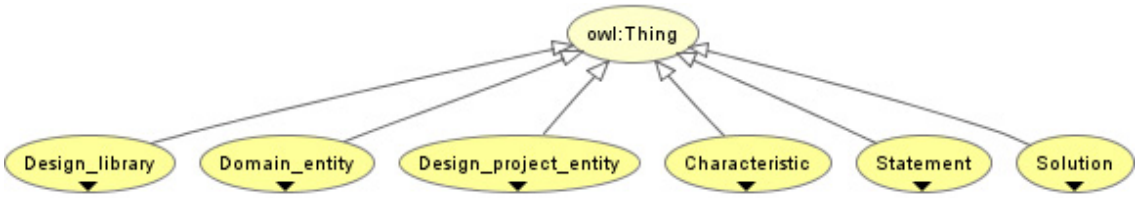

Figure A1. IEF Core ontology top-level concepts.

Domain_entity branch contains generic application modelling ontology and is depicted in Figure A2.

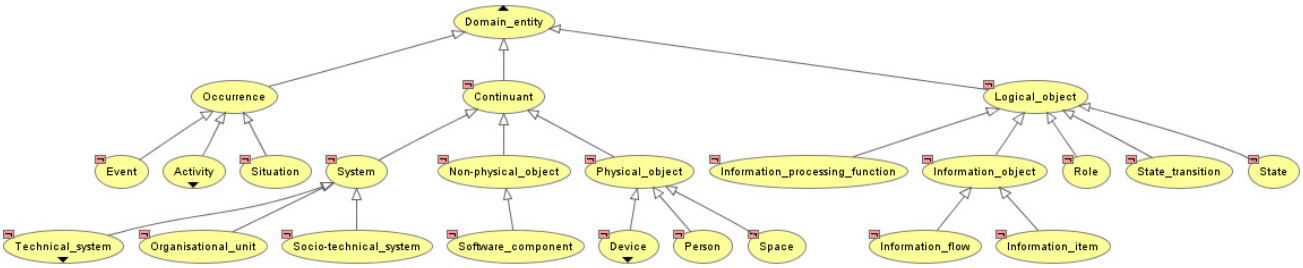

Figure A2. Domain entities.

Branch Characteristic (Figure A3) contains various dimensions characterising the design task. Furthermore, it contains taxonomy Functional_category for the "intelligent" functions that can be realised by advanced algorithms.

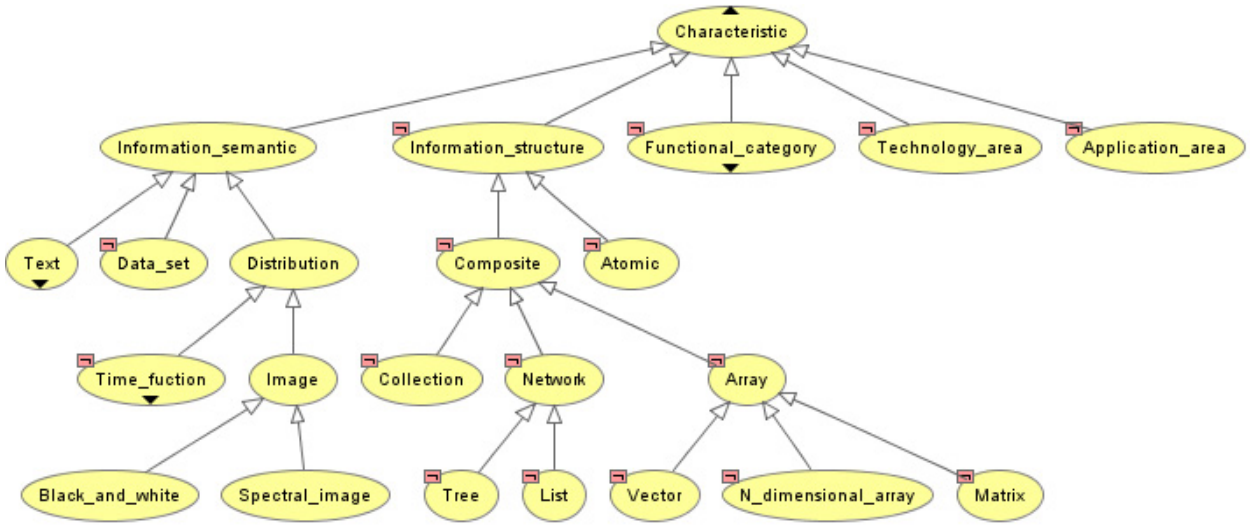

Figure A3. Characteristic concept with sub-concepts.

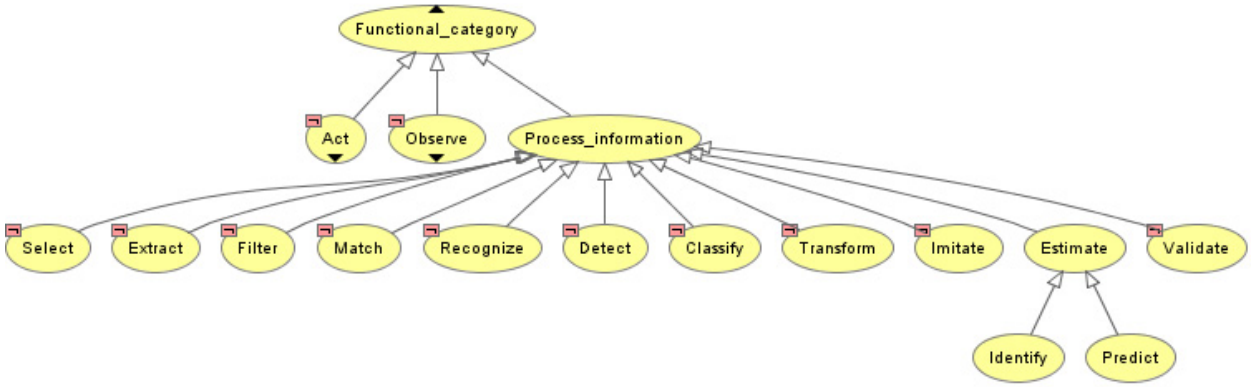

Figure A4. Intelligent functions concepts.

The functions in A4 are abstract and they are usually specialised in application area ontology. Some samples of the definitions of the concepts are given in Table A1.

Table A1. Examples of functions of algorithmic systems.

Concept	Definition
*Functional_category/**Observe***	Inputting information from external and/or internal SOA (SOA refers to “State of Affairs”)
*Functional_category/**Act***	Causing changes in external and internal SOA
*Act//**Produce***	Producing a desired SOA
Act/***Maintain***	Keeping a desired SOA
Process_Information/***Detect***	Becoming aware of a certain (predefined) kind of SOA
Process_Information/***Validate***	Ensuring that information or a statement about SOA is correct or true
Process_Information/***Classify***	Determining the category to which an entity being classified belongs
Process_Information/***Filter***	Changing, removing or reducing entities by amplifying relevant aspects or attenuating/removing irrelevant aspects
Process_Information/***Match***	Testing an entity for being similar to a template entity
Process_Information/***Estimate***	Calculating the approximation of (a part of) SOA which is usable even if input data may be incomplete, uncertain or noisy.
Estimate/***Predict***	A rigorous statement forecasting what will happen under specific conditions

Figure A5 shows the other branches of the core ontology.

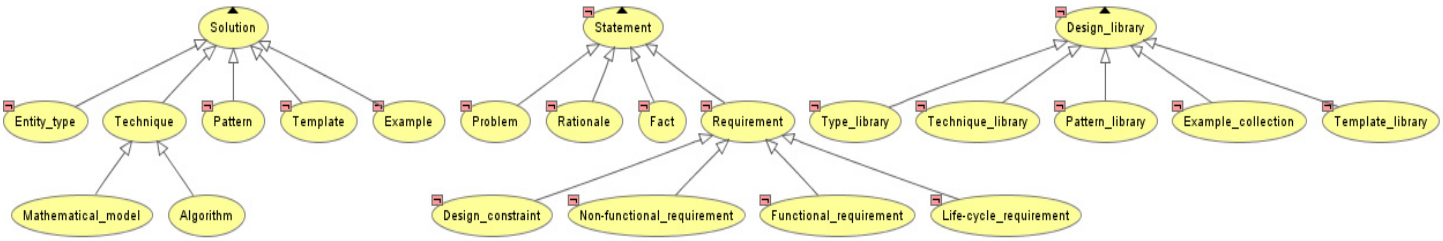

Figure A5. Design library, Solution and Statement concepts.

Application area ontologies

Application area ontologies apply the IEF Core ontology. As an example, some features of Industrial ontology are depicted here. The Industrial ontology imports the IEF Core ontology and extends it with various concepts typical to process control and monitoring.

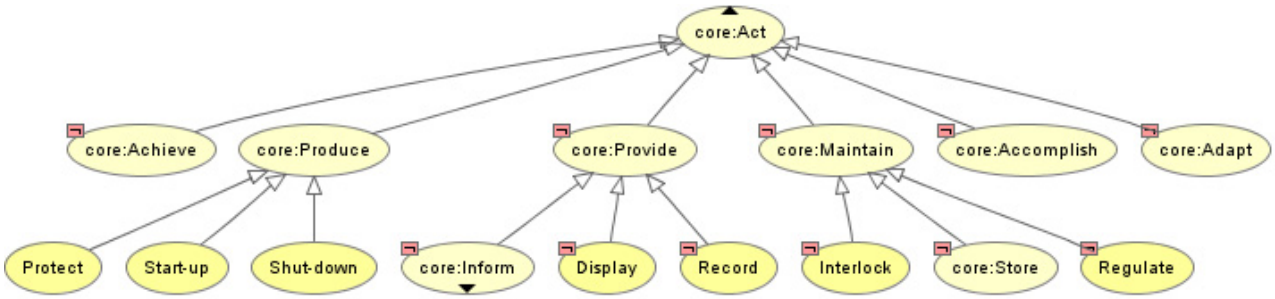

Figure A6. Some of the industrial extensions to the IEF Core Act branch.

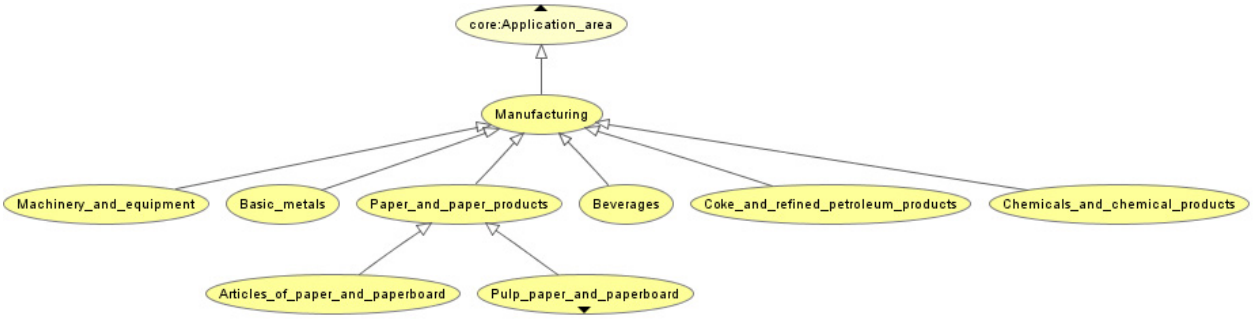

Figure A7. Example of industrial extensions to the Application area.

VTT Working Papers

122 Bettina Lemström, Juha Kiviluoma, Hannele Holttinen & Lasse Peltonen. Impact of wind power on regional power balance and transfer. 2009. 43 p.

123 Juha Forsström. Euroopan kaasunhankinnan malli. 2009. 80 s.

124 Jyrki Tervo, Antti Manninen, Risto Ilola & Hannu Hänninen. State-of-the-art of Thermoelectric Materials Processing, Properties and Applications. 2009. 29 p.

125 Salla Lind, Björn Johansson, Johan Stahre, Cecilia Berlin, Åsa Fasth, Juhani Heilala, Kaj Helin, Sauli Kiviranta, Boris Krassi, Jari Montonen, Hannele Tonteri, Saija Vatanen & Juhani Viitaniemi. SIMTER. A Joint Simulation Tool for Production Development. 2009. 49 p.

126 Mikko Metso. NoTA L_INdown Layer Implementation in FGPA Design results. 2009. 20 p.

127 Marinka Lanne & Ville Ojanen. Teollisen palveluliiketoiminnan menestystekijät ja yhteistyösuhteen hallinta - Fleet asset management - hankkeen työraportti 1. 2009. 65 s. + liitt. 10 s.

128 Alternative fuels with heavy-duty engines and vehicles. VTT´s contribution. 2009. 109 p. + app. 8 p.

129 Stephen Fox. Generative production systems for sustainable product greation. 2009. 104 p.

130 Jukka Hemilä, Jyri Pötry & Kai Häkkinen. Tuotannonohjaus ja tietojärjestelmät: kokemuksia sekä kehittämisperiaatteita. 2009. 37 s.

131 Ilkka Hannula. Hydrogen production via thermal gasification of biomass in near-to-medium term. 2009. 41 p.

132 Hannele Holttinen & Anders Stenberg. Tuulivoiman tuotantotilastot. Vuosiraportti 2008. 2009. 47 s. + liitt. 8 s.

133 Elisa Rautioaho & Leena Korkiala-Tanttu. Bentomap: Survey of bentonite and tunnel backfill knowledge – State-of-the-art. 2009. 112 p. + app. 7 p.

134 Totti Könnölä, Javier Carrillo-Hermosilla, Torsti Loikkanen & Robert van der Have. Governance of Energy System Transition. Analytical Framework and Empirical Cases in Europe and Beyond. GoReNEST Project, Task 3. 2009. 49 p.

136 Toni Ahonen & Markku Reunanen. Elinkaaritiedon hyödyntäminen teollisen palveluliiketoiminnan kehittämisessä. 2009. 62 s. + liitt. 8 s.

137 Eija Kupi, Jaana Keränen & Marinka Lanne. Riskienhallinta osana pk-yritysten strategista johtamista. 2009. 51 s. + liitt. 8 s.

139 Jukka Hietaniemi & Esko Mikkola. Design Fires for Fire Safety Engineering. 2010. 100 p.

140 Juhani Hirvonen, Eija Kaasinen, Ville Kotovirta, Jussi Lahtinen, Leena Norros, Leena Salo, Mika Timonen, Teemu Tommila, Janne Valkonen, Mark van Gils & Olli Ventä. Intelligence engineering framework. 2010. 44 p. + app. 4 p.

ISBN 978-951-38-7480-3 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)