VTT PUBLICATIONS 308

Model-based explanation of plant knowledge

Pertti J. Huuskonen

VTT Electronics

Academic dissertation for the degree of Doctor of Technology to be presented, with the permission of the Department of Electrical Engineering, University of Oulu, for public discussion in the Auditorium L10, Linnanmaa, on May 30th, 1997, at 12 o'clock noon.

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 1997

ISBN 951–38–5053–6 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–5054–4 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Leena Ukskoski

VTT OFFSETPAINO, ESPOO 1997

Huuskonen, Pertti J. Model-based explanation of plant knowledge. Espoo 1997, Technical Research Centre of Finland, VTT Publications 308. 173 p. + app. 59 p.

UDC 681.372.12:159.95
Keywords knowledge based systems, expert systems, industrial automation, support systems, explanation mechanisms, hypertext, context-based interaction, HCI

ABSTRACT

This thesis deals with computer explanation of knowledge related to the design and operation of industrial plants. The needs for explanation are motivated through case studies and literature reviews. A general framework for analysing plant explanations is presented. Prototypes demonstrate key mechanisms for implementing parts of the framework.

Power plants, steel mills, paper factories, and high energy physics control systems are studied to set requirements for explanation. The main problems are seen to be either a lack or an abundance of information. Design knowledge in particular is found to be missing in plants. Support systems and automation should be enhanced with ways of explaining plant knowledge to the plant staff.

A framework is formulated for analysing explanations of plant knowledge. It consists of three parts: 1. a typology of explanation, organised by the class of knowledge (factual, functional, or strategic) and by the target of explanation (processes, automation, or support systems), 2. an identification of explanation tasks generic for the plant domain, and 3. an identification of essential model types for explanation (structural, behavioural, functional, and teleological). The tasks use the models to create the explanations of the given classes.

Key mechanisms are discussed to implement the generic explanation tasks. Knowledge representations based on objects and their relations form a vocabulary to model and present plant knowledge. A particular class of models, means-end models, is used to explain plant knowledge. Explanations are generated through searches in the models. Hypertext is adopted to communicate explanations over dialogue based on context.

The results are demonstrated in prototypes. The VICE prototype explains the reasoning of an expert system for diagnosis of rotating machines in power plants. The Justifier prototype explains design knowledge obtained from an object-oriented plant design tool. Enhanced access mechanisms into on-line documentation are studied with examples from high-energy physics experiments. The Lepo prototype explains the behaviour of automation logic in various kinds of plants.

PREFACE

The research for this thesis was carried out during the years 1990 to 1996 at VTT Electronics (previously called the Computer Technology Laboratory) and at the European Center for Nuclear Reseach (C.E.R.N.), in the SELME, TIESU, CICERO and LEPO projects. The projects were funded by the Technology Development Centre of Finland (TEKES), VTT Electronics, the University of Oulu, C.E.R.N., and the participating companies: Enso Fine Papers, Imatran Voima, Konecranes Components, Pohto, Rautaruukki, Tampella Power, and Valmet.

I have also accepted significant support for this thesis from the following Finnish foundations: Suomen kulttuurirahasto, Jenny ja Antti Wihurin rahasto, Tauno Tönningin säätiö, and Tekniikan edistämissäätiö. I am grateful to all the people and organisations who have contributed to these efforts.

I wish to thank my supervisor at the University of Oulu, Prof. Matti Pietikäinen, for guiding my dissertation efforts. I am most grateful to Prof. Martti Mäntylä (Helsinki University of Technology) and Dr. Raimo Korhonen (Valmet Automation) for providing encouraging critique as the nominated reviewers this thesis. I would like to express thanks in advance to Prof. Kari Kuutti (University of Oulu) and Dr. Raimo Korhonen, who have been appointed as the opponents for the public defense of this thesis.

Dr. Kari “Pastis” Kaarela has coauthored several of the papers included in this thesis. Without his encouragement, sophisticated humour, and taste for sparkling wine this work would never have been completed. I owe similar thanks to Dr. Pekka Isomursu and Dr. Matti Kurki for supporting this thesis. Besides commenting on draft versions, these three friends and colleagues have given me many a good advice from the time of their own dissertation efforts.

Prof. Veikko Seppänen has provided comments on several versions of this thesis. A devoted reviewer, he gave suggestions that were usually full of insight, sometimes weird, but at all times thought-provoking. His ideas have been invaluable in formulating the structure of this thesis.

Anneli Korteniemi deserves the credit for attracting me into the field of artificial intelligence. Her work with the VICE system gave me a good basis on which to build my research. She graciously guided my efforts during my M.Sc. work and coauthored two of the fundamental papers in this thesis.

Johan Plomp and Esko-Juhani Malm deserve special thanks for their painstaking realisation of the Lepo C++ prototype, which has linked together many of the results documented in this thesis. Dr. Hannu Heusala from VTT and Jari Paanasalo, Janos Kovacs, and Hannu Paunonen from Valmet, together with our other industrial partners, have given us invaluable

support and comments for this work. A number of people from the plants and experiments deserve thanks for sharing with us their knowledge, elements of which have found their way in this thesis.

I also wish to thank the following people for their efforts: Juha Takalo for carrying out the on-line documentation studies in CICERO, Jaakko Oksanen for the P&ID work and Dr. Juha Jaako for modelling in TIESU, John Meech and Eric Wagner for the HCI analyses in CICERO, and Martti Meri, Jyrki Okkonen, Prof. Kauko Leiviskä, and Anne Väisänen for co-authoring the papers.

I wish to thank Prof. Hannu Hakalahti and Dr. Jean-Marie Le Goff for making it possible for me to work within the unbelievable research environment at CERN. This thesis was largely formulated during my stay in Geneva, which was in many respects the high time of my life. Special thanks go to Esa Salonen for the two wedding rides, not to mention the hikes, the downhill, and the raclette.

Thanks to Gordon Roberts, not only for checking the language of this thesis, but also for the most inspiring lessons during my academic career, and for good humour. I am also in debt to Douglas Foxvog and Belinda Ikeda, who have proofread and commented on some of the material. Merci à Mme. Huguette Cabel, who has been of great assistance in my work. Somewhat unexpected thanks must go to Mika Matturi, who donated me the guitar that has greatly helped in the process of writing this thesis.

I will always remain in debt to my dear mother, who has inspired me to look at this world with curious eyes. She has showed me, among other things, how one should be able to get really excited about something. To date, I have been really excited about several things, and plan to continue that way.

Finally, a kiss to Marja, my dear angel.

Saariselkä, Finland, April 1997

Pertti J. Huuskonen

CONTENTS

LIST OF ORIGINAL PUBLICATIONS

This dissertation includes the following eight original publications (Papers I through VIII):

I Korteniemi, A. & Huuskonen, P. 1991. *An Expert System with Explanations Tailored to Each User.* In: Proceedings or the 3rd International Symposium on Expert Systems Application to Power Systems (ESAP'91), Tokyo, Japan, April 1 - 5, 1991. Pp. 381 - 384.

II Huuskonen, P. & Korteniemi, A. 1992. *Explanation Based on Contexts*. In: Proceedings of the Eighth Conference on Artificial Intelligence for Applications (CAIA'92), Monterey, CA, March 2 - 6, 1992. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992. Pp. 179 - 185. ISBN 0-8186-2690-9

III Kaarela, K., Huuskonen, P. & Leiviskä, K. 1993. *The Role of Design Knowledge in Industrial Plant Projects.* In: Proceedings of the 4th International Conference on Cognitive and Computer Sciences for Organizations (ICO '93), Montreal, Canada, May 4 - 7, 1993. Montreal: Girico. Pp. 173 - 183. ISBN 2-7624-0555-6

IV Kaarela, K., Huuskonen, P. & Jaako, J. 1993. *Providing Plant Design Knowledge to the Operators.* In: Smith, M. J. & Salvendy, G. (eds.) Human-Computer Interaction: Applications and Case Studies. Proceedings of the Fifth International Conference on Human Computer Interaction (HCI '93), Orlando, Florida, August 8 - 13, 1993. Vol. 19A. Amsterdam: Elsevier Science. Pp. 546 - 551. ISBN 0-444-89540-X, ISSN 0921-2647

V Huuskonen, P. & Kaarela, K. 1995. *Explaining Plant Design Knowledge through Means-End Modelling.* In: Anzai, Y., Ogawa, K. & Mori, H. (eds.) Symbiosis of Human and Artifact: Human and Social Aspects of Human-Computer Interaction. Proceedings of the 6th International Conference on Human-Computer Interaction (HCI '95), Tokyo, Japan, July 9 - 14, 1995. Vol. 20B. Amsterdam: Elsevier Science. Pp. 417 - 422. ISBN 0-444-817956, ISSN 0921-2647

VI Meech, J. F., Huuskonen, P., Le Goff, J.-M. & Wagner, E. 1995. *An Analysis of the Human-Computer Interfaces to High-Energy Physics Control Systems at CERN.* In: Anzai, Y., Ogawa, K. & Mori, H. (eds.) Symbiosis of Human and Artifact: Human and Social Aspects of Human-Computer Interaction. Proceedings of the 6th International Conference on Human-Computer Interaction (HCI '95), Tokyo,

Japan, July 9 - 14, 1995. Vol. 20B. Amsterdam: Elsevier Science. Pp. 291 - 296. ISBN 0-444-817956, ISSN 0921-2647

VII Huuskonen, P., Kaarela, K., Meri, M. & Le Goff, J.-M. 1994. *On Knowledge Representation for High Energy Physics Control Systems.* In: Dasgupta, S., De, S.K. & Roy, A. (eds.) Accelerators: Control & Data Acquisition. Proceedings of the International Conference on Current Trends in Data Acquisition & Control of Accelerators (CTDCA ’94), Calcutta, India, December 6 - 8, 1994. Calcutta: Variable Energy Cyclotron Center. Pp. 100 - 109.

VIII Huuskonen, P., Kaarela, K., Okkonen, J. & Väisänen, A. 1995. *Explaining Control Logic to Process Operators.* In: Forsyth, G. F. & Moonis, A. (eds.) Proceedings of the 8th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE '95), Melbourne, Australia, June 5 - 9, 1995. Amsterdam: Gordon and Breach Publishers. Pp. 203 - 211. ISBN 2-88449-198-8

The papers will be referred to in the text by the corresponding Roman numerals (I - VIII).

The author of this thesis is the main author of Papers II, V, VII and VIII. The research for these papers has been done by the author. The co-authors have provided comments and material for these papers. They have also helped in the formulation of the research ideas and in the practical work.

Papers I, III, IV, and VI have been written by Ms. Anneli Korteniemi, Dr. Kari Kaarela, and Mr. John Meech. The author of this thesis has contributed considerably to the research reported in these papers in terms of providing ideas and material, carrying out experimental studies, implementing prototypes, and co-authoring the papers.

GLOSSARY

The following terms, definitions, and abbreviations will be used in this thesis. Some of the definitions are specific to this thesis; other, broader meanings may be assumed elsewhere. Unless otherwise specified, the definitions have been formulated by the author.

Artificial intelligence	The study of principles and techniques that enable computers to tackle problems that have previously been thought possible only for humans to solve.
Automation	1. The control and information systems that are used to control plants. 2. The act of automating plant operations.
Automation system	A kind of *control system*, here used as a synonym for a *DCS*.
CERN	The European Centre for Nuclear Research, in Geneva, Switzerland. The birthplace of the *WWW*.
Control system	A system that controls the operation of a plant or parts of it.
DCS	A distributed *control system*.
Designer	A human who designs plant systems.
Expert system	A *KBS* that uses knowledge acquired from human experts to achieve automated reasoning tasks with performance comparable to that of the experts.
Explanation	"The act of explaining; a making clear or understood; exposition; interpretation; the clearing up of matters between parties who have been at variance" [Thatcher 1965]
Explanation mechanism	A software module that constructs *explanations*.
HEP	High Energy Physics; here refers to the physics research that uses large experimental facilities, such as particle accelerators.
Hypermedia	An interlinked collection of information chunks; usually text, graphics, audio, and video connected through links that can be followed with a browser.
Hypertext	*Hypermedia* that mostly consists of text material. In this thesis, this term is used interchangeably with "hypermedia".

KBS	Knowledge-based system, a software program that uses captured human *knowledge*. Here the term is seen in its wider meaning, including *model-based systems* that do not necessary contain human knowledge.
Knowledge	The higher level information with semantic content that humans frequently apply to reason about systems, as opposed to lower level data that most computerised systems handle.
Knowledge engineering	A branch of *artificial intelligence* that seeks to design knowledge-based systems (*KBS*).
Maintainer	A human who maintains plant systems to keep them in operable conditions.
Mental model	The image of a system that humans form in their minds when dealing with the system.
Model	A (computer) representation of a target system that tries to make certain aspects of the system more easily analysable.
Model-based system	A *KBS* that is based on explicit *models* of the domain and/or problem-solving, rather than having them inseparably intertwined with the program code or rules.
Modelling	The act of creating a *model*; usually by a knowledge engineer, or automatically by software.
Operator	A human who controls the systems of a plant.
Plant	An (industrial) organisational unit in a physical location that has identifiable *processes*, *automation*, and people operating them.
PLC	A programmable logic controller, a *control system* that is usually used in more limited applications than a *DCS*.
Process	1. The (physical) parts of a plant that are controlled by *automation*. 2. The various phases that lead from a goal to an implementation, for instance in a design process.
Support system	The part of *automation* that is more concerned with supporting human tasks than controlling plant systems.
Task	1. A set of activities to be carried out by a *user*. 2. A unit of problem-solving knowledge in a knowledge-based system. 3. A (generic) method of constructing explanations for a particular type of needs.
Teleology	The theory or study of purposiveness in nature, on explanations of ends, aims, goals, intentions or

	purposes. (Adapted from Flew [1979]). “The science or doctrine of final causes; the science treating of the end or design for which things were created”. [Thatcher 1965]
User	A human using a system (here: *operators* and *maintainers*)
WWW	The World-Wide Web global *hypermedia* network.

1 INTRODUCTION

Two fundamental ingredients of human intelligence are *introspection*, the ability to examine one's own thoughts, and *verbalisation*, the ability to convey these thoughts to other people. Much of our daily lives seems to consist of these two activities.

Artificial intelligence seeks to develop computer systems that are intelligent. If we take human intelligence as a target for these developments, then an intelligent system should be capable of both introspection and verbalisation – in short, *explanation*. The system should be able to analyse the knowledge it uses and to communicate this knowledge to the outside world.

In computer software, the subsystem that is responsible for these activities can be called the *explanation mechanism*. This thesis concentrates on computerised explanation applied to the domain of plants[1]. We review requirements for explanation, propose explanatory concepts and mechanisms, and demonstrate results with cases drawn from plants.

1.1 THE NEED FOR EXPLANATION

People often have problems with the use of technical systems. They may not fully understand the intended functions of a system, which makes it difficult to effectively operate it. This is evidenced particularly well with the classical example of human-machine interface research: the controls for video-cassette recorders (VCR).

For many users, the majority of the functions in a VCR are unknown [Norman 1988]. Normally they only apply a handful of functions that are needed for basic operations. The rest of the functions are only used in exceptional cases (such as time zone adjustments), and instruction manuals are often needed to activate these functions. The internal operation of the VCR is most often not clear to the users, and they need external advice to use the machine. The advice can come from the manual, or perhaps more often, from a more experienced user.

[1] In this thesis the term "plant" covers industrial plants (for example, power plants) as well as non-industrial plants, such as particle accelerators.

1.1.1 Plant supervision

Difficulties very similar to the VCR case are found in industry. Modern plants are arguably among the most complex human-operated systems, which makes them particularly rich in operational problems. Many of the problems that plant supervisors (termed “operators” later in this thesis) face are information-related: either there is too much or too little information available [Buck 1989], or it is available in the wrong context.

Many operators lack a thorough understanding of the plants they operate. The daily functions quickly become familiar to them, but less frequent events may be unclear, leading to problems, and possibly to danger. One aim of this thesis is to point out practical difficulties in plant supervision and to offer concepts and mechanisms that can be used to solve some of these difficulties.

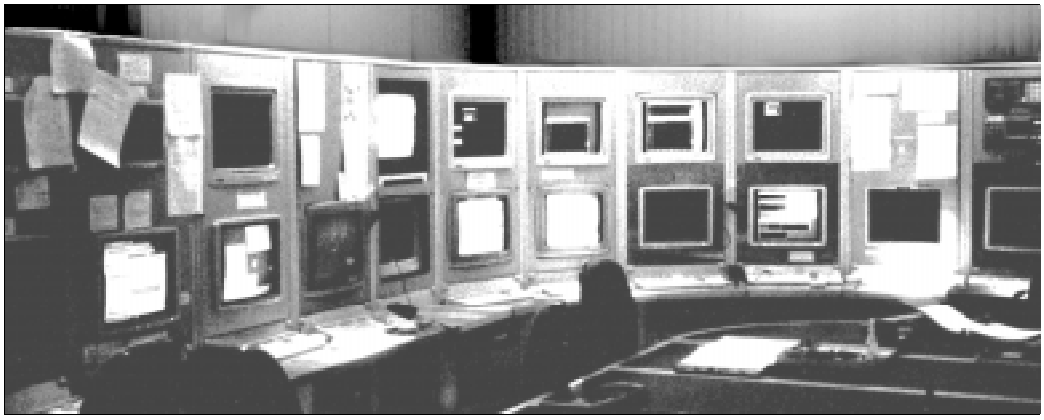

Figure 1. Part of the control room of the L3 experiment at CERN.

Figure 1 shows a particular case we have studied: the underground control room of the L3 experiment at CERN, the European Centre for Nuclear Research, in Geneva, Switzerland. This experiment is operated by visiting physicists, who spend a week or two there collecting physics data. Supervision is an auxiliary duty for them, and therefore they are not familiar with all the subsystems after the brief training period. Problem cases cause much confusion, and the operators frequently have to call in the control experts to solve the situation. Even though the operators may have doctorates in nuclear physics, they are not confident with systems with which they are unfamiliar. This is evident in the following episode from CERN [Le Goff 1993]:

> At four a.m., the telephone wakes up from a deep sleep the control expert who is on call. A nervous physics student on the phone explains that alarm number 15 has gone off. After stopping the alarm buzzer, he had quickly read the operating procedure for that particular alarm.

> He was rather worried, as the instructions seemed to suggest that the magnet cooling system could be shut down, and that was a serious situation for the valuable physics data collection and possibly for the machinery as well. Unsure how to proceed, he had decided to call the expert.
>
> After getting dressed and rushing the fifteen minute drive to the scene, the expert takes a look at the main sensor readings and decides that the situation is less serious than it sounded. A few minutes of troubleshooting reveal that the alarm was caused by a plotter running out of ink – a problem that hardly justified waking up the expert in the middle of the night!

In this case, the operator had received misleading information from the alarm system and could not evaluate the seriousness of the situation. In panic, he was unable to find more detailed information because he did not have a clear idea of the systems.

Although this false alarm case can be largely blamed on the design of the alarm system, other cases could have been more serious. If the control systems had been equipped with better diagnostics, the operator's distress might have been avoided. He could have used more (or better) information on the original problem, on the alarm system's functions, and on the proper action to take.

Similar problems are evidenced in almost all settings where people deal with technical systems. Users of computers, copying machines, elevators, kitchen appliances, and cellular phones all struggle with systems they need to use but do not fully understand. According to Norman [1983], Rasmussen [1986], Carroll & Olson [1988], Sheridan [1988], Stassen et al. [1988], Buck [1989] and Wexelblat [1989], humans form a *mental model* of a technical system they interact with. This model is formed unconsciously while learning to use the system. Learning is supported through instruction manuals, courses, and experience. Since proper instruction is often difficult to comprehend or is not available, people resort to trial and error. They form incomplete mental models of the system, revising the models little by little through experience with the system.

Sometimes the mental models can be faulty [Norman 1983]. This happens usually when proper instruction is not given. In such cases people seem to form their own theories on how systems work that do not necessarily correspond to the reality [Carroll & Olson 1988]. The users of systems then rely on their incorrect theories, until they become confused by the unexpected behaviour. At that point, computer support would be welcome.

1.1.2 Avoiding mental mismatches

The mismatch between the mental models and the reality could be a major source of the frustration that people often experience with technical systems [Norman 1988]. The frustration can be lessened by designing systems more according to the way that people work [Schneiderman 1987]. This approach is clearly visible in modern office automation software that tries to mimic ways that people worked in a traditional office. However, this is not always possible, especially in the case of such complex systems as industrial plants. Some computerised controls, for example automated start-up sequences, are based on principles that have changed fundamentally since the days of manual control [Sheridan & Ferrell 1974].

Another approach is to *educate* the users offering them correct mental models. This approach, based on training, tends to fail with time. The deteriorating memory of humans does not support extensive training beforehand. Normally, general guidelines can be given in advance, but regular training will be needed during use to refresh memories.

We want to promote a third approach: to offer *on-the-spot help* where the users have problems in understanding the systems [Rettig 1993]. Traditional help systems are not sufficient, since they cannot adapt to the situation or to the users' needs. More dynamic help systems are needed for the complex situations found at plants.

Explanation is a way to give dynamic help, to clarify the way a system works, and to educate people about the properties, functions, and limitations of a system. Explanations can be offered either upon the user's request or automatically whenever a system detects that a user has problems with the system. Properly designed control and diagnostic software would allow many systems to explain themselves.

1.2 MODELS OF PLANT KNOWLEDGE

Various kinds of support systems have been realised in industry to help people in plants with their tasks [Hayes-Roth & Jacobstein 1994]. Some of these systems apply techniques from *knowledge engineering*. An important subclass of these systems is based on *models of knowledge* related to the plant.

In this thesis, we distinguish two kinds of models related to plant knowledge, depicted in Figure 2: (A) the mental models that humans form and (B) the models that plant systems contain. We seek to *enhance the mental models* indirectly by *explaining the models of the plant systems*.

The figure shows a view to the interplay between a plant operator, an automation system, and the plant itself. The operator unconsciously forms mental models both of the plant (A_1) and of the automation system (A_2).

These models are initially formed during training and then get revised with the actual use of the plant and the automation.

Traditionally the automation serves to carry out the operator's commands and dispatch them to individual devices. An equally important function is to visualise the behaviour of the plant – to store, filter, and clarify the data that is received from the sensors. Automation also serves as the user interface to support systems.

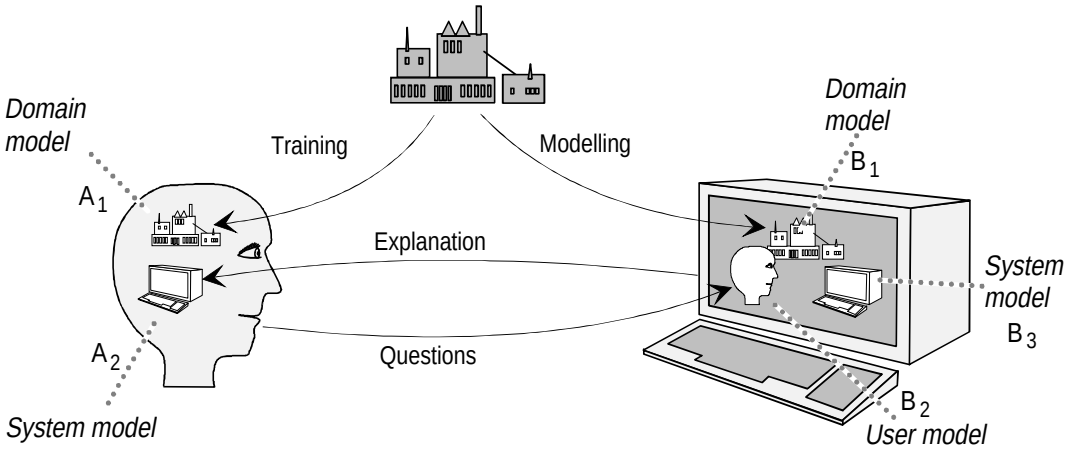

Figure 2. The models of the domain, the automation, and the user.

We propose a new function for the automation[2]: *to explain the plant*; to act as a documentation and knowledge repository; to clarify the plant's and automation's own behaviour, their underlying design choices, and their capabilities and limitations. The automation could participate in a *dialogue* with the users, becoming an assistant in their duties.

For this aim, the automation must acquire a model of the plant (B_1). It should also maintain a model of the user (B_2) for natural communication and should contain a model of itself (B_3) to be able to explain its own behaviour [Tanner & Keuneke 1991].

In this thesis, we suggest ways to model knowledge about plants and automation, and to explain this knowledge to the users. Our main focus is in domain and system models (B_1 and B_3).

[2] Here we assume that support systems are part of the automation, although presently they are often separate systems. We believe that, in the future, automation systems will carry out many tasks that are now done by separate support systems.

1.3 PROBLEM STATEMENT

We have studied the practical problems that people have in controlling and understanding plants with automation. These empirical studies are presented in the papers and discussed further in Chapter 2. To summarise, many problems are related either to a lack of information, or an excess of information concerning the plant systems. The information would have to be represented in such a form that it becomes easier to understand. In particular, ways must be found *to model this information* and to *communicate it to humans* in a suitable manner.

1.3.1 Research problem

Johannsen et al. [1983] stress that for successful interaction between human and machine, the following questions should be considered in the control of technical systems:

- What *kind of information* is needed?
- How should this information be *acquired*?
- How should the information be *organised* and *structured*?
- How should the information be *analysed*?
- How should information be *processed* before being *displayed*?
- How should information based *judgments* be formulated?
- How should the information be *transmitted* in decentralised situations?

In our view these are the questions that should be answered for information systems used in plants. Since the full scope of the questions encompasses a very large area of research, it is not possible in one piece of work to answer them completely. We have therefore chosen the most relevant questions as the topic of this thesis, as will be detailed shortly.

We direct our research to concern itself with an important class of information: *knowledge*. With "knowledge" we mean the high level information that humans frequently apply to reason about systems, as opposed to lower level information that most automation systems handle. For the purpose of this study, we largely ignore many kinds of the tacit knowledge [Suitiala 1993] that humans seem to use, such as common sense understanding of the world, or general factual knowledge found in lexicons. We restrict ourselves to deal with technical knowledge related to plants, that is, knowledge of structures, behaviour, functions, and purposes of plant systems. Much of this knowledge is abstractions of lower level information.

Based on this discussion, we define the research problem for this work as follows:

> How can plant knowledge be communicated to the plant staff?

This general problem can be stated as more specific subproblems:

> P_1: What difficulties do people have with technical systems in plants?
> P_2: What kinds of knowledge would be needed to solve these problems?
> P_3: How can this knowledge be captured and modelled?
> P_4: How can this knowledge be communicated to people?

The problem analysis and conceptual solution presented in this thesis will answer subproblems P_1 and P_2. Most of the practical results documented in the thesis will deal with subproblems P_3 and P_4.

1.3.2 Research hypothesis

We hypothesise that the research problem (more specifically, subproblems P_3 and P_4) should be approached through viewing the *communication of knowledge as explanation*. The problem is then reduced to finding such *modelling and interaction techniques* that facilitate explanation.

We further hypothesise that different kinds of explanations should be classified according to the target system to be explained and according to the classes of knowledge to be explained. This classification helps to analyse the various uses that explanation has in plants. We propose that explanations should be produced by generic explanation tasks that may employ similar techniques in different domains, facilitating reuse.

For a pragmatic side of the research hypothesis we propose that the research problem be solved by using specific explanation techniques for the tasks, in particular knowledge-based models and interaction mechanisms based on hypertext.

There are other possible solutions to the problem, for instance education of the plant staff, visualisation of plant information, and the development of the plant organisation. We believe that the knowledge-based solutions we propose are the most suitable ones for problems that originate from a lack of knowledge, which seems to be the case in plants. Moreover, when new information systems are introduced to plants for better conveying information to the staff, the systems may create yet another source of

confusion. Explanatory techniques are needed to clarify the information where traditional information systems are no longer sufficient.

The main hypothesis can be summarised:

> The research problems can be answered by viewing the communication of knowledge as explanation.

1.3.3 Research assumptions

In engineering, systems are often characterised only by their input-output relationships without regard to how these relationships are formed inside the systems. This "*black box*" principle is routinely used in engineering for analysing systems without a need to know their internal structure or behaviour.

The principle is useful for breaking down complex systems into smaller parts, or dividing up responsibilities between designers. Unfortunately, the principle also makes the users' life harder. They may spend much of their time trying to understand how the system works, even though the designers never anticipated such a need. The system's internals are designed to be invisible to the users; they are indeed black boxes.

We believe that the black box tradition actually shows up in the human-machine interfaces of many technical systems, and is one cause of many problems that people experience with such systems. The systems' internals are deliberately masked from people who would have a need to know them, especially in such abnormal conditions as process failures. [Sheridan 1988, Norman 1988]

We assume that *explanation helps to alleviate the black box problem*. Through explanation, the users of technical systems gradually gain a better understanding of the systems. As a mission of this thesis, we seek to find ways of allowing a technical system to incorporate sufficient knowledge to explain itself to its users.

Help systems have become an important part of commercial software. Unfortunately they are targeted towards general advice and thus are not easily customisable to actual use situations. We deal with such applications where it is generally not possible to predict all behaviours of a system, and thus not possible beforehand to construct explanations that cover all necessary cases. Dynamic explanation generation is needed in those applications. Model-based explanation is our approach to achieve such generation.

A central assumption, underlying most topics discussed in this thesis, is that knowledge-based techniques can help to construct explanations. Knowledge-based techniques require the existence of knowledge in some

form. Our approach is not to automatically generate or derive new solutions from observations, but rather to adapt existing knowledge into new forms that are useful to the plant staff.

In some cases, existing knowledge can be made available for the explanation mechanisms with relatively little effort, as is the case with many diagnostic expert systems. Such systems often store knowledge in their knowledge bases in a form that can be explained, although the knowledge may have to be restructured for explanation. In other cases, for instance with plant design databases, some of the knowledge has to be adapted to be explainable. This adaptation, termed *modelling*, is usually performed by knowledge engineers.

We make the optimistic assumption that the models in knowledge-based systems are valid enough to be explained to users. In practice, models will always be incomplete and/or contain inaccurate or contradictory information. We have not explored the implications that giving incorrect information may have to humans' mental models. However, explanation can help detect some of the limitations or contradictions in knowledge bases.

We assume that many problems in plant design and subsequently in the use of the plants are due to a lack of access to higher level design knowledge [Kaarela 1996]. We believe that users are able to understand both concrete and abstract design information, once it is placed in the right context through explanations.

The various systems that we will present all work in co-operation with the user, which is a necessity for support systems. Our systems are intentionally designed to be *assistants* to human problem solvers, with the associated trade-off of becoming less capable of autonomous operation.

1.3.4 Research methods

We seek to solve the research problem mainly with a constructive approach. We first analyse empirically information-related problems that people have with automation. We then present a framework that typifies the kinds of explanation that are needed to solve the problems. We propose model-based concepts and techniques for explanation as the solutions, and validate the key mechanisms in prototypes.

Chapter 2 of this thesis and Papers IV, VI, VII, and VIII cover the problem analysis. Together with Chapter 3, Papers I, IV, VII, and VIII propose the solutions on a conceptual level, while Chapters 4 - 7 and Papers II, V, VII, and VIII include solution mechanisms with prototypes.

1.4 SCOPE OF THE RESEARCH

1.4.1 Application areas

The research presented in this thesis covers several interrelated application areas (Figure 3): power plants, paper machines, high energy physics (HEP) systems, and steel mills. In each area we focus on specific topics.

The technical subsystems in HEP experiments are so similar to those used in industrial plants that we regard HEP systems as kinds of plants. In fact, HEP systems surpass most industrial systems in their complexity, making them a good target for our studies. The next generation of experiments, to be constructed at the start of the next millennium, will be even more complex by an order of magnitude.

Plant design is the first topic. There the focus is to study the information-related problems that people have with plant systems, and how this information could be obtained from designers. We suggest ways of capturing design knowledge and communicating it to the users in the plants. The results are demonstrated with cases drawn from power plants and high energy physics systems.

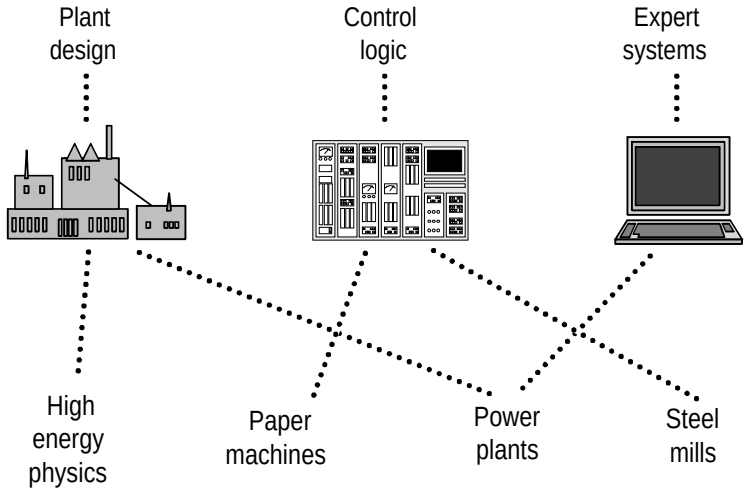

Figure 3. The main topics and application areas of the thesis.

The second topic concerns *control logic*, an important subset of automation. The aim is to study the specific problems that control logic solutions cause in industry, and we propose software solutions to these problems. Here the viewpoints range from behavioural analysis of the users to implementations of explanation algorithms. The industrial cases cover paper machines and steel mills.

The third topic is diagnostic *expert systems* that need to communicate their results to users. We focus on the task of adding explanation facilities into an existing expert system for power plants. With the case, we propose a number of conceptual level solutions and concrete mechanisms.

1.4.2 Research areas

Explanation falls within the interdisciplinary zone between several fields of science and technology. To achieve demonstrable results, we emphasise an engineering approach. We combine ideas from several sciences, especially human-related sciences, into engineering solutions.

In all application areas, we limit ourselves to information-related topics, and more specifically into ones that deal with knowledge-based techniques. As a consequence, we suggest results obtainable through the use of information technology. This restricts our research to exclude other possible solutions, such as organisational or educational aspects of communicating technical knowledge.

Within the large area of information technology, it has been necessary to select aspects of modelling and presenting knowledge. Our approach combines ideas from the fields of knowledge engineering and human-machine interface research. For knowledge representation, we apply network-oriented methods implemented in object-oriented software. Such knowledge representation methods as formal logics and connectionist systems are not targeted in this thesis.

Even though we discuss how our solution mechanisms can be implemented and maintained in practice, we largely ignore the professional software engineering perspective. The transfer of design knowledge from designers to users is, however, discussed.

We consider modelling of technical systems mostly from a semantic viewpoint, leaving such modelling constructs as differential equations and qualitative models outside the scope of this work. Our representations of design rationale are limited to those aspects that are needed to study explanations. They do not try to cover the full range of constructs necessary to represent the lifecycle of a plant, as this is a complex research issue in its own right.

Our discussion of human-machine interaction is focused on explanatory dialogue. We touch on the relevant topics within cognitive and behavioural sciences briefly. We have deliberately chosen lightweight approaches to user modelling, complexity management, navigation and dialogue. We base our approach on contexted hypertext, fully realising that it is but one possible medium for interaction. This approach facilitates integration with on-line documentation systems, which is highly beneficial for explanations in the plant domain.

1.5 OUTLINE OF THE DISSERTATION

Chapter 2 analyses the information-related problems that people have in design and use of plants. These problems are taken as the requirements for the explanation framework presented in Chapter 3. It defines generic explanation tasks for the plant domain. The framework is used in the succeeding Chapters 4 to 7 to give a point of reference for the discussion.

Ways of implementing parts of the framework are studied in several chapters, from a modelling viewpoint (Chapter 4), from a design recovery viewpoint (Chapter 5), from a diagnostic viewpoint (Chapter 6), and from a human-computer interaction viewpoint (Chapter 7). For each viewpoint, the proposed concepts for explanation are presented and demonstrated by a number of prototypes.

Related work is reviewed and compared to the results of this thesis in Chapter 8. An introduction to the included papers is the topic of Chapter 9. Chapter 10 draws the conclusions of the thesis.

Papers I to VIII in the appendices contain the original papers.

2 PROBLEM ANALYSIS

This chapter studies the difficulties that appear in the various parts of the design process and the use of the plant, with the aim of estimating how explanation could help in solving the problems. We use this study as the requirement analysis for an explanation framework that will be presented in the next chapter. Our views are based on empirical analyses in projects related to power plants and high energy physics control systems, as documented in the included papers. To support our views, we highlight some findings from the large body of research into industrial control and support systems.

2.1 PLANT DESIGN AND USE

Figure 4 shows a simplified model of plant design and use (the implementation phase has been omitted for clarity). We use the model in this and the following chapters to discuss information-related problems in plants.

The top left part of the figure depicts how the designers of various disciplines participate in the process. As end results, the design process produces the plant processes and the automation, with the associated documentation. They are used by the people that form the plant's organisation. Our focus is in operators and maintainers, although some of the results will apply to other staff, for example managers[3].

The area inside the dotted line emphasises the role that knowledge-based systems can have in the design process. Much valuable knowledge concerning the plant and the design process are captured in databases. To create support systems, a knowledge engineer adapts the knowledge into models. Support systems use the models to help the plant staff in operating the plant. Ideally, the systems gather the experiences of the staff to be reused in the design to build other support systems and plants.

[3] Buck [1989] uses the term "operator" very liberally to refer to anyone using the systems or processes in plants. In a way, all the people in a plant operate it to some degree.

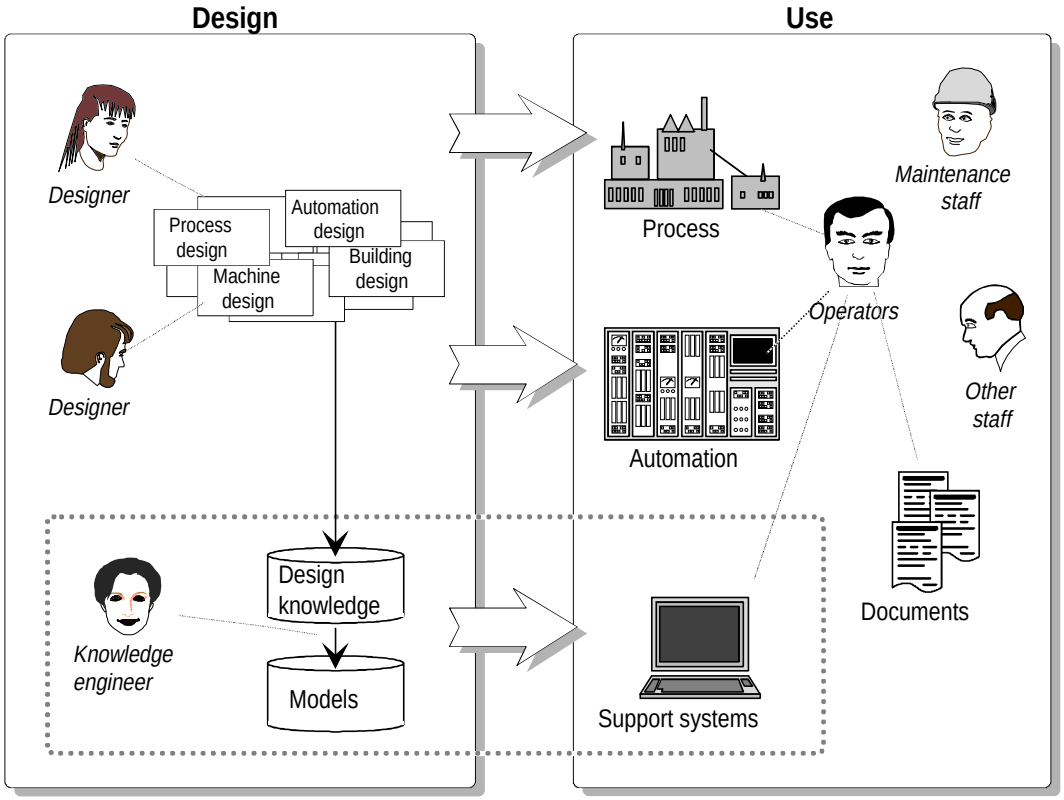

Figure 4. Plant design and use.

The model could be elaborated. For instance, Korhonen [1991] distinguishes separate predesign, specification, implementation, testing, and startup phases in automation projects. He also deals with reuse, which is not shown in the figure. However, with our scant model we want to emphasise that plant use is largely dependent on the results of the design phase, as will become apparent later in this chapter.

2.2 PROBLEMS IN PLANT DESIGN

Some of the numerous subtasks that make up a plant design are shown in Figure 5. These subtasks are normally divided up among several organisations, often involving hundreds or even thousands of people. The tasks overlap both in time, going on simultaneously, and in content, dealing with the same parts of the design from different viewpoints.

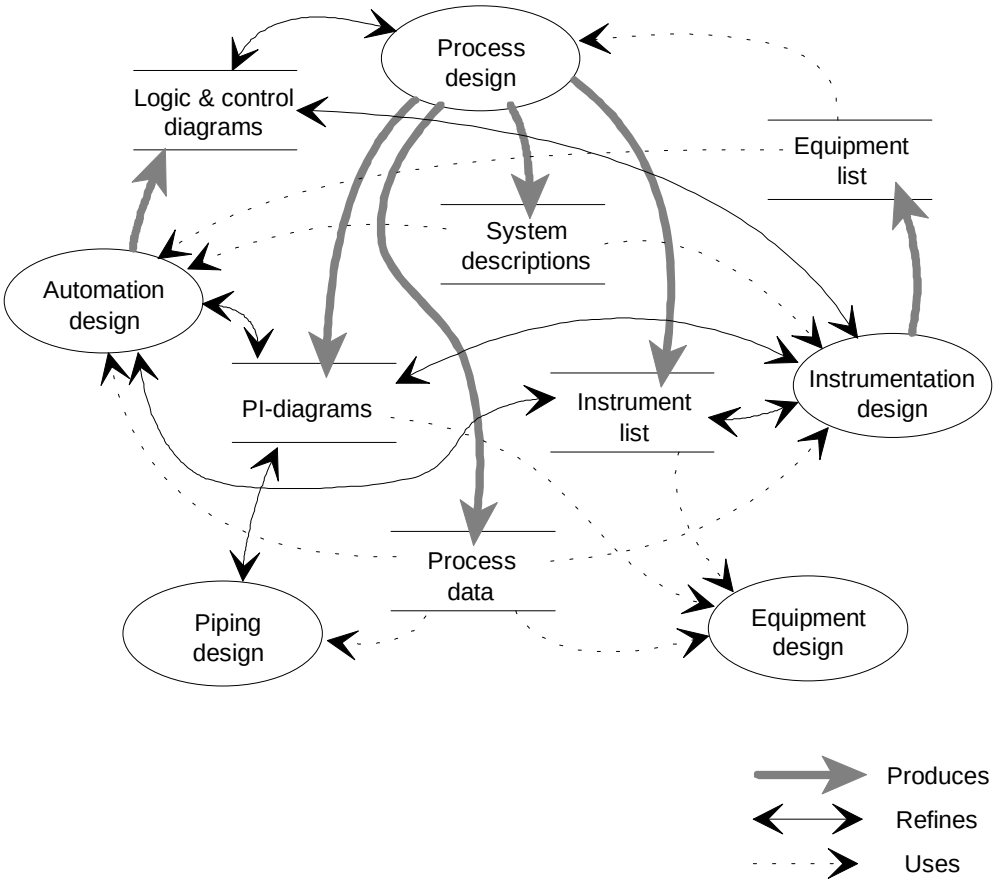

Figure 5. Dependencies between design areas.

2.2.1 Communication

Plant design, and design in general, is iterative by nature. Many design subtasks can only commence after receiving input from some other subtasks. These in turn depend recursively on further subtasks (Figure 5), mandating an incremental way of solving the design problems.

The complexity of plants and the nature of the design process together imply that communication is fundamentally important in plant design [Rytoft et al. 1990]. Unfortunately, in actual design projects much knowledge is never transferred. This was evidenced by our empirical studies of power plants, documented in Papers III and VIII, and high energy physics control systems, documented in Papers V and VII. Much of the knowledge is mutually understood and does not need explicit transfer, but some knowledge seems to get lost due to a shortage of suitable transfer methods [Klein 1993, Kaarela 1996].

Design work often stalls due to a lack of required data. Inputs from other designers arrive late, if ever. Therefore designers sometimes have to

base their work on assumptions and estimates of the other designer's work[4]. Moreover, designers are often reluctant to freeze their designs, tending to modify them up to the last minute. Inconcistency problems therefore plague plant design, caused by outdating replication of design documents in several organisations [Rytoft et al. 1990]. Current design tools' support for concurrent design work should be improved [Klein 1993].

2.2.2 Compatibility

There is an increasing trend towards electronic data transfer in plant projects. However, the formats for transferring design data are even today largely incompatible between organisations, despite the existing and emerging standards (for instance STEP (ISO 10303-1) [1994], IGES [1988], and CALS [Weich 1992]). The tool support for standards emerges slowly, and legacy designs remain in proprietary formats.

Even similar data formats are not in themselves sufficient to guarantee the usability of the data. For instance, an instrumentation diagram remains just on a drawing level if the transfer format cannot group the geometric primitives – lines, polygons, and text – into meaningful entities, such as pumps and valves. The semantic content of design documents is often lost in the transfers causing manual re-entry of information. [Kaarela et al. 1992]

2.2.3 Low level descriptions

Design descriptions tend to concentrate on device level details [Stephanopoulos 1990], largely ignoring higher level information such as justifications of the design choices. As an example, let us consider the process and instrumentation diagrams that are important in communicating process designs. They depict the various devices and instruments of the process, connected through pipes and signal lines. They serve well to transfer data on the interconnections of devices. Yet, somehow these diagrams are also believed to convey higher level knowledge on the design choices and tradeoffs, purposes of designs, and other tacit knowledge. [Kaarela 1996]

In reality, the knowledge is shared between designers who have used or seen similar solutions in the past. They unconsciously lend the higher level knowledge from these solutions. This convenient way of communication may fail in the case of exceptional designs, and will certainly fail if the designer does not have enough background knowledge in the specific

[4] Ideally, they would apply the black box principle by agreeing only on common interfaces of interrelated parts, rather than on shared functionalities of these parts. In practice, however, interdependencies seem unavoidable with physical systems.

domain. Design documents would have to be enhanced to capture as much as possible of this knowledge.

PI-diagrams, even with control diagrams, fail to show the interactions between different parts of the plant. Physical devices, automation, and the users collaborate to achieve operations. Unfortunately, it is not clear from many existing designs how the parts function as a whole [Rasmussen 1985].

2.2.4 Tool support

A large part of industrial design involves reuse [Korhonen 1991]. The most common way to design a plant is to use an existing plant design as a basis. Reuse first implies understanding an existing design and then making modifications to it [Fischer et al. 1991], a task which needs to be aided by higher level knowledge. Another activity that requires redesign is the maintenance of a plant. Modifications are frequent at some plants, causing much reverse engineering to understand the existing solutions before making changes to them. As Årzen [1991] says, "lack of knowledge makes it difficult and dangerous to modify the control system".

Designers tend to concentrate on the normal behaviour of the plant. Their task is usually to design the intended functionality, which may obscure the possibilities of undesired behaviour. As a result, design and documentation emphasise the normal state [Korhonen 1991, Ishack 1993]. In many cases, studies and simulations of problems are carried out during design, but it is difficult to predict all possible mishaps in a plant. In practical projects, simulation is usually too costly or too difficult, and some problem cases are never anticipated. Plant users are therefore not always well supported in handling a problem.

Most design tools are not able to capture conceptual level knowledge [Johannsen & Alty 1991, Chandra 1992]. They cannot normally include justifications, purposes, alternatives, or rationale behind the designs [Lee & Lai 1991]. This is unfortunate, since this knowledge would be best available at the time of design.[5]

2.2.5 Design knowledge

We conclude that current design documents, tools, and practices suffer from poor knowledge level content. Lower level data are abundant; higher level knowledge is often missing [Olsson 1993]. As we suggest in Papers III and IV, many problems in plant design and subsequently in the use of the plants seem to be due to this lack of knowledge. The lack has become apparent

[5] Yet even with tool support it may be difficult to motivate designers to record for other people the design knowledge that they do not themselves directly need. The plant staff are not seen as their direct customers during the design process.

only recently, when computerised tools and formats have gained acceptance in industry, and designs have been formalised.

At this point, we would like to to observe that despite the difficulties we have discussed, plants are successfully built and run in industry. We believe, however, that much of this success is owing to the humans' capacity for making sense of vague design knowledge. As more and more design tasks are becoming computer-aided, problems that we have described start to surface. The increasing complexity of plants with ever more strict production, safety, and environmental requirements cause further problems.

Conklin & Yakemovic [1991] and Fischer et al. [1991] suggest several practical reasons why design knowledge is not currently recorded or transferred: no time is allocated for it in projects; documenting work may disturb creativeness; designers are reluctant to document the "wrong turns"; the amount of data can be unmanageable with the available tools; and much of the successful expertise is tacit, to mention but a few. It may well be that the greatest difficulties may lie in the organisation and culture (and perhaps even in the nature) of plant design. However, we believe that proper tool support could lessen these difficulties.

We now look at how problems caused by design combine with problems caused by the plant's operational processes and systems, leading to difficulties for the plant staff.

2.3 PROBLEMS IN PLANT USE

The complexity of modern plants is evident in the work of operators, who may have to supervise hundreds of subsystems consisting of tens of thousands of components [Olsson 1993]. Most of these details are hidden from the operator, who sees only a distanced view through the automation [Ishack 1993]. However, the details surface as soon as there are malfunctions that the automation cannot handle. To clear the trouble, the operator must solve high level problems as well as carry out low level manipulation.

The maintenance staff face similar requirements. A plant evolves continuously; after the start-up phase improvements will be needed in efficiency and quality [Paunonen 1995]. Not only do the maintainers need to keep the plant in operation, but they also need to understand it well enough to improve it. Although often seemingly local in scope, maintainer's duties nevertheless require an understanding of the plant as a whole.

The discussion in this chapter concentrates on the work of the operators and maintainers, somewhat ignoring other groups, such as managers and laboratory staff.

2.3.1 Tasks and tools

To understand the problems of the plant staff, we need to look at the tasks they face and the tools they use.

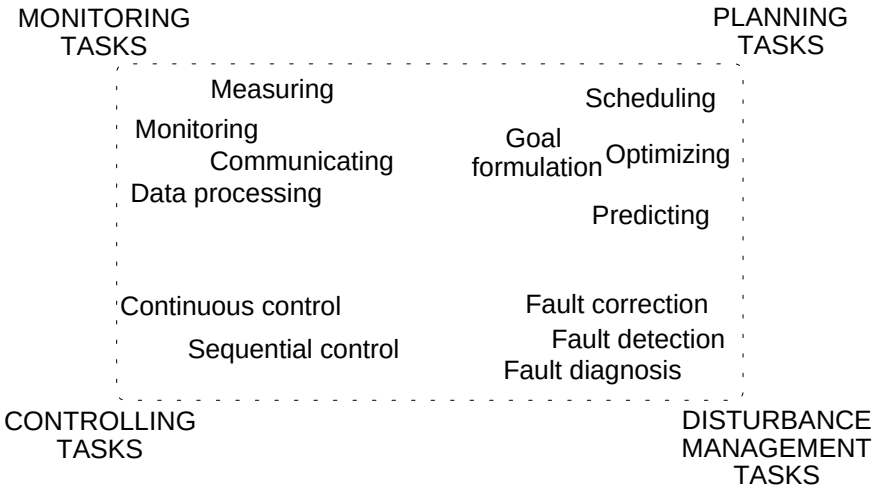

Figure 6. Categorisation of an operator's main tasks in plants.

Lees [1974] has listed some of the tasks that operators carry out in plants. Figure 6 classifies the tasks into four main categories. Traditionally, the work of the operators is concentrated on the left hand side, in control and data processing. Modern automation has moved the focus towards the right side – they have become planners and disturbance managers [Sheridan et al. 1983].

Maintainers are mostly occupied with repair, fault avoidance, and fault prediction. Their tasks tend to emphasise the right half of the figure. Two of their primary occupations are not included in Lees' list, however: preventive maintenance and information seeking. They spend much of their time in keeping the machinery in the plant in working condition, which in turn mandates diagnosis and search for relevant information.

The concrete tasks can be seen as instances of more general information processing tasks. The following list, gathered from several sources [Sheridan 1988, Johannsen 1990, Stephanopoulos 1990, Årzen 1991, Leitch & Gallanti 1992, Olsson 1993, Van de Ree 1994, Paunonen 1995], identifies some of these general tasks:

- Information gathering
- Interpretation
- Analysis and evaluation
- Planning and prediction

- Optimisation
- Decision making
- Execution of control actions
- Monitoring
- Communication
- Learning
- Teaching

As we see, much of the work done in plants consists of information processing. Plant automation is developing towards decision making support for the organisation [Paunonen 1995]. The tools in plants should support this development. In this thesis, we propose to apply explanations as an important feature of decision support systems in plants.

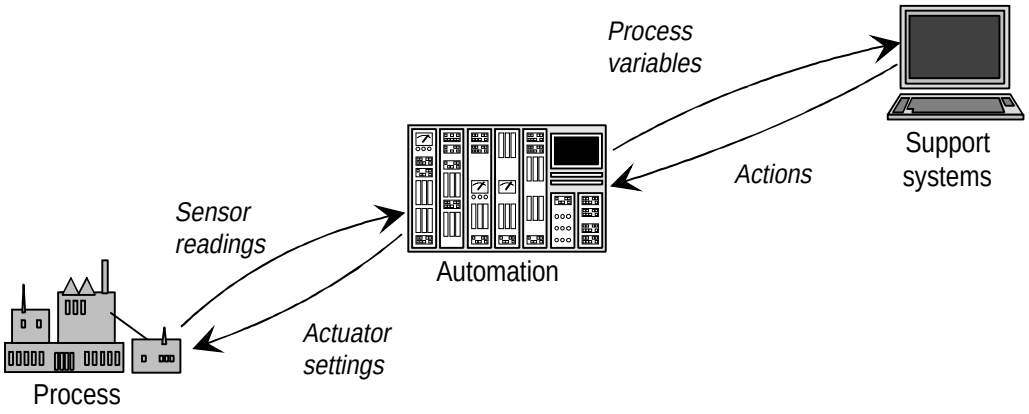

Figure 7. The three levels of systems in plant supervision.

Figure 7 shows a three-level model of the process supervision systems in plants: the *process*, the *automation*, and the *support systems* [Stephens 1992]. Each level builds on the level below. The automation abstracts the sensor readings it receives from the process into indicators that describe the process variables. It also dispatches the operator's commands into lower level actions that directly affect the process devices. Support systems further abstract the process variables into a diagnostic understanding of the state of the process, and carry out optimising or recovery plans by suggesting actions for the operator and the automation.

In the automation industry, the support systems are normally referred to as "information" systems, as opposed to automation systems (although the difference is diminishing). This emphasises their informational role. Support systems are often seen as more active systems, with capabilities for

problem solving. In this thesis, the term "support system" is taken to comprise both notions.

We now survey some of the problems found at each of the three levels: processes, automation, and support systems.

2.3.2 Processes

The work of the plant staff is to a large extent driven by the events of the process [Johannsen et al. 1983, Rytoft et al. 1990]. When the process is running normally, the people spend much of their time in routine monitoring and maintenance. When there are disturbances in the process, the operators must bring it back under control, often in a hurry[6]. The source of the disturbance needs to be quickly diagnosed and the maintainers may need to do some repair before normal operation can be restored. Stassen et al. [1988] point out that the operators are frequently underloaded, and occasionally catastrophically overloaded.

The process-paced mode of operation requires humans to quickly grasp the essence of complex situations and plan a set of correct actions to take. To avoid overload, the people need clear mental models of the plant, and must be able to search effectively for supporting information when needed [Buck 1989]. With current systems, especially so with plant documentation, this may be difficult. There is so much information and so many details that help would be needed in navigating through the references.

The complexity of the plants would require that the relations, both physical and logical, between process elements be made clear [Paunonen 1995]. To choose effective control strategies and plan actions people must understand how systems interact [Broadbent et al. 1986, Norman 1988]. The dependencies are often hidden in the process, which causes much work in plants to if one is to understand the process events. Processes should be made more transparent [Olsson 1993].

Figure 8 illustrates the lack of transparency. It shows a simplified process and instrumentation diagram of a gas system we studied in the Cicero project at CERN. This graphical description does not make it clear that the gas pressure in the main loop is to a large degree controlled by the vent valve at the right side of the diagram. This dependency is explained in the associated documentation [Peach 1994], but cannot be understood from the diagram alone. Additional information is needed.

[6] Perhaps the most lively description of the nature of this work was given by airline pilots and nuclear plant operators, who are said to refer to their work as '*hours of boredom punctuated by moments of terror'* [Sheridan et al. 1983].

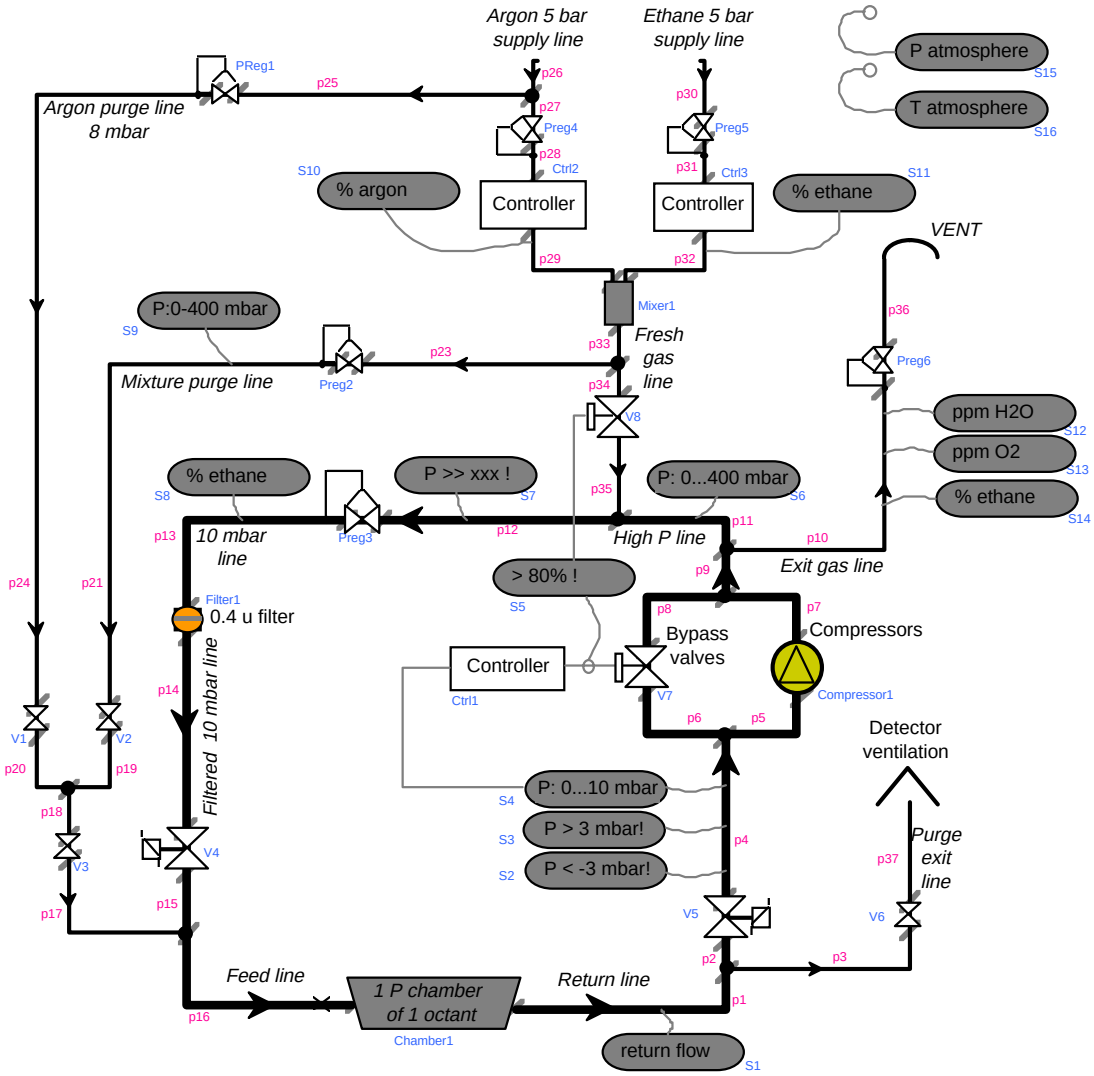

Figure 8. A simplified version of a gas system used at CERN.

The occasional malfunctions of the sensors that describe the processes add a further source of complexity [Rowan 1989]. In addition to dealing with the huge volume of data and internal dependencies, people must all the time suspect the information they receive to detect possible sensor faults.

There is also complexity in the behaviour of the processes. For instance, batch processes are always in a transient state [Keränen et al. 1988]. Many processes exhibit time delays to the order of hours, or even days [Hoc 1989]. To understand the dynamics of such processes, an operator should follow the process over long periods. This may not be possible in usual shift work, where new people take over the supervision after every eight hours or so. Process history databases contain valuable information of the past process states, but this information is not always made properly available to the operators [Årzen 1991].

Ideally, work would be directed by the goals of the plant [Kaarela 1996]. Each plant has a set of ideals that the operations aim at, such as

production, economy, environment, and safety goals. The processes are to be driven in such a way as to achieve these goals. In reality, not all goals may be reached simultaneously, and the staff have to find a suitable compromise between the goals.

Unfortunately, the goals of the plant are seldom made explicit to the people running it [Olsson 1993]. Consider the following passage from Mould's [1988] reconstruction of events in the control room of Unit Number 4 at the Chernobyl nuclear power plant, on the night of 25 April 1986:

> "01:22.30 - The operator looks at a printout of the parameters of the reactor system. These are such that the operator is required in the written rules to immediately shut down the reactor, since there is no automatic shutdown linked to this forbidden situation. The operator continues with the experiment (major fault no. 5 and the most serious one). Computer modelling has shown that the number of control rods in the reactor core were now only six, seven or eight, which represents less than half the design safety minimum of fifteen, and less than one-quarter the minimum number of thirty control rods given in the operator's instruction manual."

Within a minute this violation of safety goals, combined with several other mistakes, led to a power surge in the nuclear reactor and a thermal explosion in its fuel channels. The results are well remembered.

Could this accident have been prevented with better information systems? Medvedev [1991] suspects that the main reason for the accident was in fact mischievous management that forced the operators (despite their doubts) to ignore all safety regulations. On the basis of Medvedev's account, it could be reasoned that in this controversial situation the operators could not fully comprehend the state of the reactor, and acted upon faulty mental models. Since all the safety systems had been disabled, the situation got out of hand.

One can speculate that a better understanding of the safety goals and the state of the reactor could have helped. This information was offered through the automation and documents, but was ignored. In the rather fantastic case, the support systems should have detected the staff's faulty thinking and persuaded them to abandon their plan by explaining the possible consequences.

2.3.3 Automation

Modern automation is well able to manage the normal operation of plants. The operator is usually only needed in abnormal situations, when the automation can no longer cope with a disturbance. The operator has become a supervisor, who follows the process through the automation. The

manipulation of the process devices is largely becoming trusted to the automated sequences. However, there is currently less help to analyse the abundance of sensor data [Årzen 1991, Olsson 1993]. The data storage capabilities of modern plants are mainly used for low level displays and routine trends [Van de Ree 1994].

When problems arise, the operator's supervisory task changes into crisis management [Stassen et al. 1988]. This is a fundamental change, requiring different kinds of tools from those used in normal supervision. The quickening pace of production changes in modern plants has a similar effect, creating frequent little crises to be resolved. As a whole, it would be desirable that automation gave better help in managing changes.

A fundamental irony of automation [Bainbridge 1983] is that it tries to free the operator from knowing the exact low-level details of the process. Unfortunately, when there is a malfunction, the automation fails to operate as designed, and the operator must investigate precisely those low-level details that were obscured by the automation[7]. It is no surprise that in such cases there will be difficulties, since the details are not known to the operators. Normally the maintenance staff take over detailed problem tracing, but their situation is not much different. They often have several process areas to maintain, and many individual systems are likely to remain unknown until problems appear.

At any large plant there are bound to be several errors in the implementation (and design) of the automation. These errors show up in the use of the plant, often only in exceptional situations, adding new complications to be solved. The following example shows a minor but irritating problem in a power plant: an alarm that cannot be dismissed.

> Alarm "Flue gas fan bearing fan started by protection" results when the flue gas fan is stopped by a sequence program. The program also stops the bearing fan, which causes the bearing temperature to rise over the safety limit. This in turn forces the bearing fan to restart and raises the above alarm. When the flue gas fan is restarted, everything is normal, except that *the alarm stays on. It cannot be dismissed*, because that would require stopping the bearing fan, which (for some mysterious reason) is impossible when the flue gas fan is running.

The operators we interviewed at the plant considered such minor design errors to be very irritating. Even though most errors presented no real problem to the operation of the plant, they were said to cause much additional work. Moreover, the operators were frustrated by the functions they knew to be defective, but yet had to live with.

[7] A prime example is automation logic, as studied in Paper VIII and Chapter 6.

Sometimes the automation may mask device problems [Bainbridge 1983]. For instance, a drifting sensor may be hidden behind a control loop that compensates for the drift, until it starts to affect the process seriously.

When a critical event takes place, it can often only be inspected afterwards, when the original situation may have already passed. Process histories are then needed to reconstruct events. This is sometimes difficult due to the immense amount of logged data (or quite often, the lack of it). Relevant data for the situation should be brought forward and events replayed, suppressing unnecessary details in the histories.

As with the processes, the documentation concerning the automation tends to be spread out in several places and systems. There is still little support for cross-navigation between the systems, even though today the documentation is becoming available in an electronic form. The plant staff must therefore resort to their memory and manual search for finding information. At times, this is too difficult and the information remains unfound[8]. This may lead to assumptions and even wild guesses, which will lead to non-optimal performance in operating the plant.

The interactions of the various parts of the automation are sometimes unclear. The same problem is evident at the process level. Even though the connections between the systems may be documented in control or logic diagrams, the behaviour of the systems remains opaque. Support would be needed to explain these interactions [Paunonen 1995, Broadbent et al. 1986]. This is particularly evident in maintenance, where modifications may introduce undesired side effects, if the interactions are not understood [Fischer et al. 1991]. Maintainers spend much of their time reverse engineering existing solutions to be able to make modifications [Barman 1992]. This task should be helped.

2.3.4 Support systems

Support systems enhance the capabilities of automation by offering higher-level information to humans. They are used for help in monitoring, diagnosis, planning, interpretation, prediction, control, repair, instruction, and information management tasks [Buck 1989, Johannsen & Alty 1991, Årzen 1991, Oxman 1993, Durkin 1996]. Some of those systems apply knowledge-based techniques to achieve performance comparable to human experts in some limited areas. Knowledge-based support systems are the main focus of our treatment, although useful systems are created with other techniques as well.

The Tiger system [Milne et al. 1996] is a good example of a knowledge-based plant support system. It is used in a chemical plant for condition

[8] A power plant operator we interviewed explained that there were three ways to deal with an unknown alarm: either to ignore it (!), to ask for advice from other people, or (as a last resort) to inspect the manuals.

monitoring of a gas turbine that is the most vital component in an ethylene process. The system gives model-based advice on the current and predicted performance of the turbine in terms of diagnostic displays, trends, and on-line documentation. In problem cases, it can isolate faults in the process to the component that caused the original failure. A rich set of techniques have been applied, including alarm filtering, logic explanation, root cause diagnosis, causal graphs, qualitative simulation, and hypertext. The key benefit of Tiger is said to be its availability: "It acts as a gas turbine engineer monitoring system every second, 24 hours a day, every day of the year" [Milne et al. 1996].

A support system oversees the processes and automation, much like the operator, although from a more limited scope. It should be able to explain these two levels (Figure 9). In fact, a support system becomes another potential source of confusion, so it should also be able to explain itself.

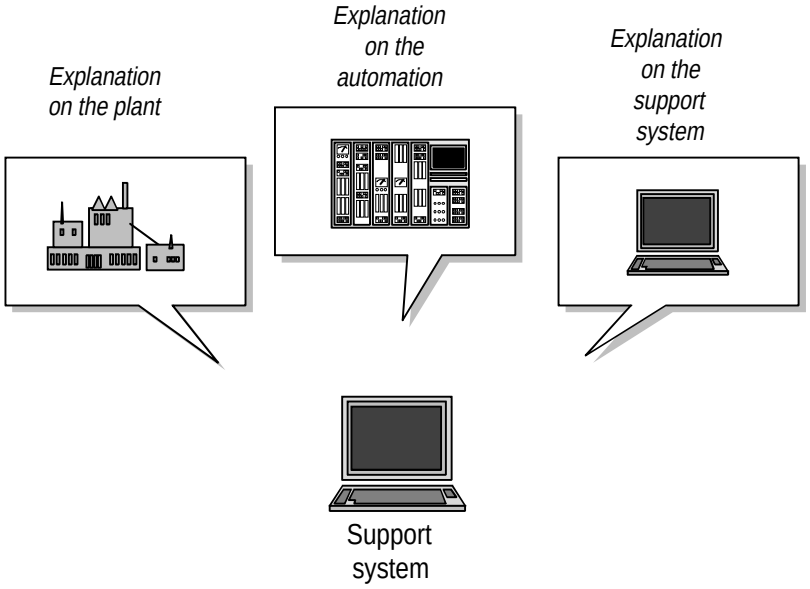

Figure 9. A support system should explain the other levels and even itself.

A central issue with support systems is their *scope*. Most successful knowledge-based systems function in very limited domains [Winston 1993]. As a result, most systems fail to treat topics that are outside the scope of their knowledge. Unfortunately, it is hard for humans to know when a question is within the scope of a system. The system should be able to describe its own limits [Wexelblat 1989, Buck 1989]. This would help humans estimate the kinds of results that can realistically be expected from a system.

Trust is another important issue with support systems. Industrial users are eager to abandon a support system the first time it produces an erroneous answer. Curiously, if a human expert makes a mistake, it is tolerated much better. Humans appear to trust other humans more easily

than machines[9]. A support system is useless if humans do not trust it, and therefore it must try to earn their trust [Berry 1995]. However, excessive trust and computer mysticism can be dangerous [Sheridan et al. 1983].

One way a system could earn trust is to justify its own results. Humans may want to know how the results were obtained; what data they were based on; where the knowledge for the derivation of results came from. These justifications can be used to evaluate the utility of the answers, or to discover deficiencies in the system's knowledge bases. In certain cases the system may give correct results, but the human using the system has different (possibly wrong) opinions about the results' validity. In such cases, the system could try to persuade the human to reconsider the problem.

Every knowledge-based system has some built-in strategy for reasoning that may be of interest to humans. When a system poses questions to the users, they generally want to know the reason why the question was asked. Knowing the system's strategy for finding and deriving information can much relieve the frustration of a human subject. Likewise, the actions of a fully autonomous system become understandable when it explains the strategy behind its actions. [Clancey 1983, Hasling et al. 1984]

The internals of the support system should be made visible to the users to a certain degree, to lessen the black box problem. A model of the support systems' functions should be offered, making connections, data and knowledge flows visible. [Buck 1989]

The different backgrounds of the different users should also be acknowledged. Paris [1987] has pointed out that the type of support given may have to vary in the amount of detail and in the type of content depending on the user. The system should maintain a user model, to understand the user's knowledge and intentions. This would help the system to give more directed answers and reduce irrelevant detail. Kobsa & Wahlster [1989] have covered many aspects of user modelling in dialogue systems.

The interaction should be made as natural as possible. Many systems that deal with knowledge-based concepts tend to use a question-answer kind of dialogue. However, a question-answer dialogue may not be optimal in plants. The medium of the dialogue may have to vary from natural language text to three-dimensional graphics, depending on the users' background. The maintenance of dialogue should be given special care in the case of advice-giving systems [Cawsey 1992].

---

[9] This is evidenced, for instance, in the common fear of autopilots. Most people (including the author of this thesis) would not board an aeroplane that flies without a human pilot, although it is well known that humans make mistakes easily. However, humans are clever in recovering from their own faults and in finding explanations for their errors, but machines are not. Perhaps this adds to the mistrust towards machines?

2.4 REQUIREMENTS FOR EXPLANATION

Explanation in general has many possible uses [Buchanan & Shortliffe 1984, Buck 1989, Wick & Slagle 1989, Wexelblat 1989, Berry 1995]. We summarise the potential uses as follows:

- *Clarification*. The actions taken by a system can be mysterious if the users do not know its internals. If the system can explain its actions, the users can more easily see the connection between the system's inputs and outputs. For instance, an alarm at the output of a control logic is due to some events in the inputs that the logic should be able to explain.
- *Justification*. Users of a technical system may be able to trust the system better if they know the kinds of design decisions that underlie the system's technical solutions. For instance, a quality indicator in a paper plant is more understandable if the details of the calculation for the indicator are known. This adds to the users' confidence, which may help them to accept the system.
- *Persuasion*. When there is a mismatch between the users' expectations and the results given by a system, the users do not know whether to trust the system or their own judgement. If the system can be shown to function correctly, explanation can convince the users to question their own views.
- *Validation*. If a system misbehaves, explanations can act as a debugging facility to find the source of problems.
- *Education*. An explanation facility can tutor the users about a system, offering on-the-spot help, definitions for terms, and detection of misunderstandings.
- *Visualisation*. Explanations can transform complex structures into more understandable ones, for instance by demonstrating the functions of a system through graphical means. The dynamic behaviour of many systems is best explained by animation.
- *Abstraction*. The complex behaviour of a system is easier to comprehend through proper data abstraction techniques. Explanation can show the flows of data in a system to reduce the apparent complexity.
- *Demystifying*. Often people do not know the limitations of systems, and may expect too much from them. Explanations can make these limitations clear, helping to lower the expectations.

Comparing the listed possibilities against the previously discussed difficulties in plant design and use, we conclude that explanation has the potential to help eliminate the difficulties.

From the discussion in this chapter we abstract the following key requirements for explanation in plants (Table 1):

Table 1. Key requirements for explanation in plants.

Requirement	***Justification***	***Discussed in:***
Make design knowledge (design choices, alternatives, tradeoffs, and justifications) available at plants.	Knowing the rationale behind designs is necessary for operation and maintenance.	Section 2.2, Chapter 5, Paper IV
Enhance design tools and documents to capture the tacit design knowledge.	Design knowledge is best available at design time. Without tool support it will not be collected.	Section 2.1, Chapter 5, Papers III and V
Clarify complex things by visualising interdependencies between items and by hiding excess details where they are not needed.	Two fundamental strategies to manage complexity are showing connections between things and abstracting away details.[10]	Sections 2.2, 4.3 - 4.6, Chapter 6, Papers VII and VIII
Offer proper mental models.	If mental models are not offered, users may invent their own, faulty models.	Section 1.2, 2.2, Papers IV and VI
Make goals and functions of plant items clear.	Knowing the role of things as a part of a whole gives them a meaning.	Section 2.2, Chapters 4 and 5, Papers III, IV, and V
Help search for relevant documentation.	Finding the right information can be very difficult with manual search, especially in critical situations.	Section 2.2 and Chapter 7, Papers II, III, and IV
Explain the behaviour of things by diagnosing and replaying events.	It is hard to analyse time-dependent events afterwards, when the causal chains are no longer apparent.	Section 2.2 and Chapter 6, Paper VIII
In support systems, explain the reasoning and the strategies behind the reasoning.	Explanation helps to earn the trust of the users and to reveal the scope of the systems.	Section 2.2, Papers I and II
Facilitate natural interaction with support systems.	Cumbersome systems will not be used in industrial settings where people have little computer experience.	Section 2.2 and Chapter 7, Papers I, II, IV, VI and VIII

[10] Rasmussen [1985] notes that "The only way to cope with control systems that are complex in terms of large numbers of information sources and devices for basic control actions is to structure the situation and thereby transfer the problem to a level with less resolution".

The “Justification” column of the table forms a conclusion to this chapter. It lists the essential problems that we have found in plants. The listed key requirements have guided the development of the explanatory concepts and mechanisms presented in this thesis.

The following chapters will show how the listed requirements have been met.

3 A FRAMEWORK FOR EXPLAINING PLANT KNOWLEDGE

In the previous chapter, we pointed out a number of problems in plant design and use, and suggested ways that explanation could be used to help in eliminating the problems. On the basis of this study we now define a framework for explanations in the plant domain.

The framework has been devised as a synthesis of the problem studies in Chapter 2, of the explanatory concepts and mechanisms in Chapters 4 to 7, and of the related research in Chapter 8. It is a tool that structures the various ideas and prototypes of this thesis into a common representation. Since our research has produced a wide variety of explanation types by using multiple modelling techniques, an intermediate concept is helpful to understand the relations between the types and the models. We therefore introduce the notion of generic explanation tasks for this aim.

The framework will be used as a map in each of the Chapters 4 to 7 to put the material in these chapters into a larger context. However, the framework is sufficiently general to find uses beyond this thesis. It could be used, for instance, as a guide in support system construction to see what kinds of explanations would be potentially needed.

Depicted in Figure 10, the framework consists of:

- A *typology* of explanations
- *Generic tasks* to produce the explanations
- *Models* that act as the input for the generic tasks

In addition, *explanation mechanisms* are needed to implement the tasks. They take the models as their input and produce various types of explanations as their outputs. A task may use several models and produce many kinds of explanations. In the figure, these multiple possibilities are illustrated with the arrows.

This chapter introduces the typology and the generic explanation tasks, while Chapters 4 - 7 and the included papers detail the modelling methods and the mechanisms that are used to implement the generic explanation tasks in a number of prototype systems.

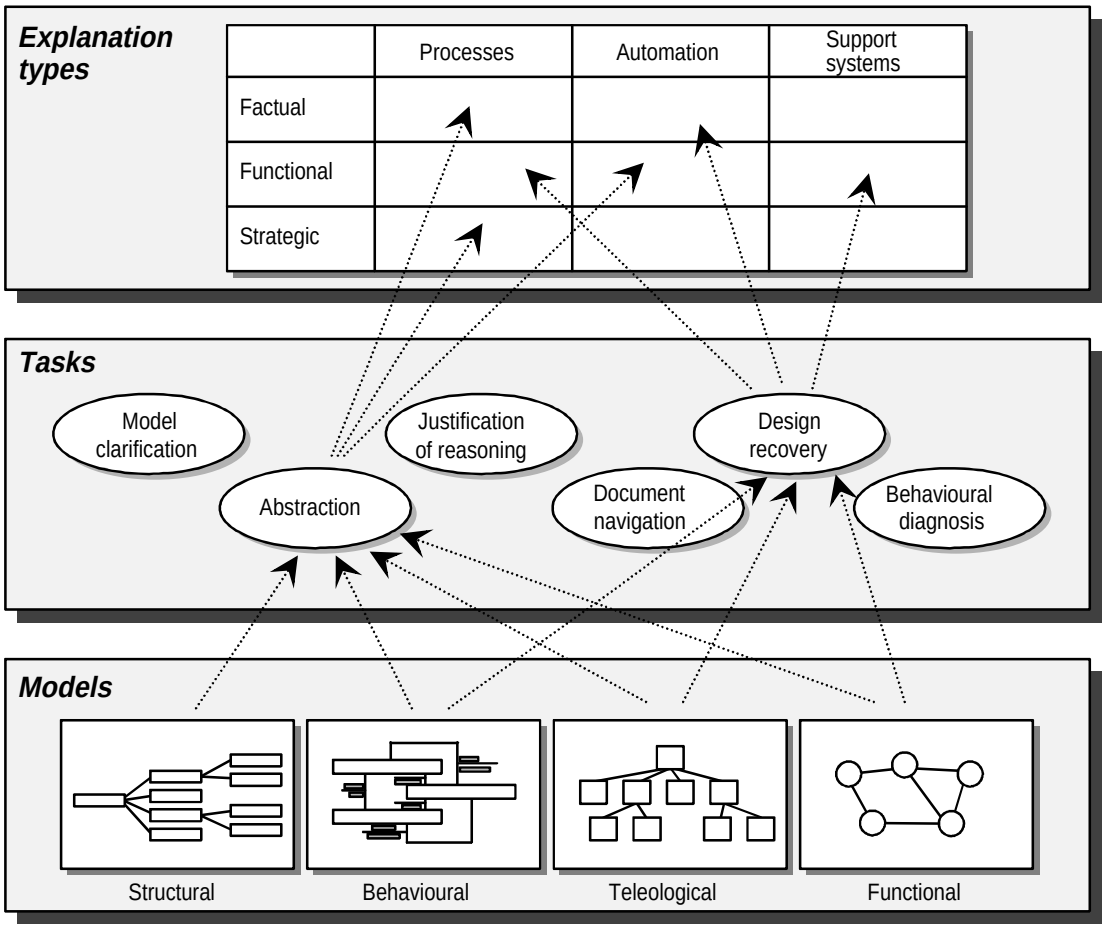

Figure 10. A framework for model-based explanation.

3.1 A TYPOLOGY OF EXPLANATION

The key requirements for explanation that were proposed in Table 1 can be detailed by defining the different types of explanations that are encountered in the plant domain.

We choose to classify explanations in two ways: according to the *class of knowledge* they explain and the *target* of the explanation (plant processes, automation, or support systems). For this study, we integrate the various phases of the design process into its end results. The classification hence assumes an end-user-oriented view. This is a meaningful assumption, because in plants explanations will be offered to end users through the support systems. Those systems are the medium for talking about the other targets, in addition to talking about themselves.

The other dimension is the class of the knowledge explained. As Chandrasekaran and Swartout [1991] note, “Each type of explicit knowledge makes specific kinds of explanation possible”. In this thesis, we recognise the following types: factual knowledge (what is known about

things), functional knowledge (how things work), and strategic knowledge (why things work the way they do).

Table 2. A typology of explanations for the plant domain.

Explanation target → *Knowledge class* ↓	**Processes** (Device level)	**Automation** (Information level)	**Support systems** (Knowledge level)
Factual knowledge (What is known about things?)	♦ Devices, systems ♦ Properties ♦ Concepts ♦ Connections ♦ Relationships	♦ Displays ♦ Databases ♦ Controls ♦ Symbols ♦ Diagrams ♦ I/O mappings	♦ Objects ♦ Properties ♦ Relations ♦ Sources of knowledge
Functional knowledge (How do things work?)	♦ Energy/matter flows ♦ Balances ♦ Physical functions ♦ Trends ♦ Process dynamics	♦ Information flows ♦ Loops, alarms, interlocks ♦ Interactions ♦ State behaviour ♦ Histories	♦ Chains of reasoning ♦ Decisions ♦ Justifications
Strategic knowledge (Why do things work the way they do?)	♦ Operation modes ♦ Process goals ♦ Process design ♦ Operation scenarios	♦ Control schemes ♦ Automation goals ♦ Automation design	♦ Control of reasoning ♦ Reasons for questions or inference

The matrix presented in Table 2 illustrates the classification. The first row includes explanations that describe the logical and physical things that are found in plants. The descriptions may include information about the properties of things, and the dependencies and relationships between them. The existence and meaning of things, as well as the scope of the models, are potential candidates for explanation.

Table 3 shows some examples of possible questions that might arise in plants as regards factual knowledge.

Table 3. Example questions concerning factual knowledge.

Processes	Automation	Support systems
♦ What is this box? ♦ What is the oil temperature? ♦ What does entropy really mean? ♦ Where does this pipe go to? ♦ What are the main parts of a generator? ♦ What kinds of bearings are there?	♦ Who uses this limit switch? ♦ Where is this power unit documented? ♦ Are there any alarms for these measurements? ♦ How many PID controllers are there? ♦ What's the meaning of this signal?	♦ What faults are related to overheating? ♦ Are there any unfinished analyses? ♦ What agents use this measurement? ♦ Who wrote this rule? ♦ What is known about bearings?

The explanations for the second row cover the flows of matter, energy or information in systems. The transformations that affect these flows are explained, together with the behaviour of the systems in past and present states. The ways in which knowledge is processed into decisions are clarified. Some possible questions of this kind are listed in Table 4.

Table 4. Example questions concerning functional knowledge.

Processes	Automation	Support systems
♦ How does the gas pressure affect its temperature? ♦ Why does this flow decrease when I turn on the pump? ♦ Tell me how the water goes round in the process. ♦ Have there been any significant changes in the ambient temperature?	♦ Where is this set-point given? ♦ How does this control loop stabilise the tank level? ♦ Where does this alarm come from? ♦ What would happen if I turned this switch? ♦ Why does this motor not start?	♦ How was it decided that the bearing is faulty? ♦ Why cannot the output power be calculated? ♦ Where did these input data come from? ♦ What rules were triggered for this pressure sensor? ♦ Tell me more about this result.

The third row makes clear the reasons the systems behave as they do. The control, operation, and reasoning schemes are laid out. The teleological purposes, goals, and reasons for the existence of things are clarified. The rationale for design choices as well as the history of the design process are explained. Possible questions include (Table 5):

Table 5. Example questions concerning strategic knowledge.

Processes	Automation	Support systems
♦ Why does the process stop after five seconds if there's no water? ♦ Why are there two pumps in parallel? ♦ Why should I purge the chambers? ♦ How is water removed from the circuit?	♦ Why is this pressure control set to 10 mbars? ♦ What is the purpose of this interlock? ♦ How did the system get into this situation? ♦ How does one make the system ready for the transition? ♦ Why are these controls separated?	♦ Why am I asked this question? ♦ What happens after frequency analysis? ♦ What is this computer doing? ♦ Can I do spectral analysis now? ♦ How does the system find the original fault?

This classification encompasses most explanation types that are given by support systems used in the industry and some potentially new types. In Chapters 4 to 7 we demonstrate specific solutions for realising explanations of many of these types.

3.2 GENERIC EXPLANATION TASKS

The proposed explanation types can be realised in a number of ways. We call them *generic explanation tasks*, which is an application of the generic task concept defined by Chandrasekaran [1986]. We have studied these tasks within the plant domain, although the tasks could be adapted into other domains.

Each task assumes a different approach to explanation. The approach depends on the problem to be solved by explanation and the kinds of knowledge to be explained.

Figure 11 illustrates the generic explanation tasks with relation to the plant design and use phases discussed in the previous chapter (cf. Figure 4).

Design recovery

Explanation techniques can be used to communicate the design knowledge that is currently lost in many design projects. By raising the abstraction level of design documents to better cover the semantics of the designs, it becomes possible to construct knowledge-based models of the designed artifact as well as of the design process. Support systems can then explain these models to humans. Design choices, rationale, alternatives, and criteria can

all be clarified, provided that the models are designed to capture this knowledge. Design tools would have to be enhanced to support this knowledge acquisition, since with current tools and design practices much of this knowledge is lost. Plant operators, maintainers, and designers would all benefit, as much of their work involves reverse engineering of existing solutions in plants.

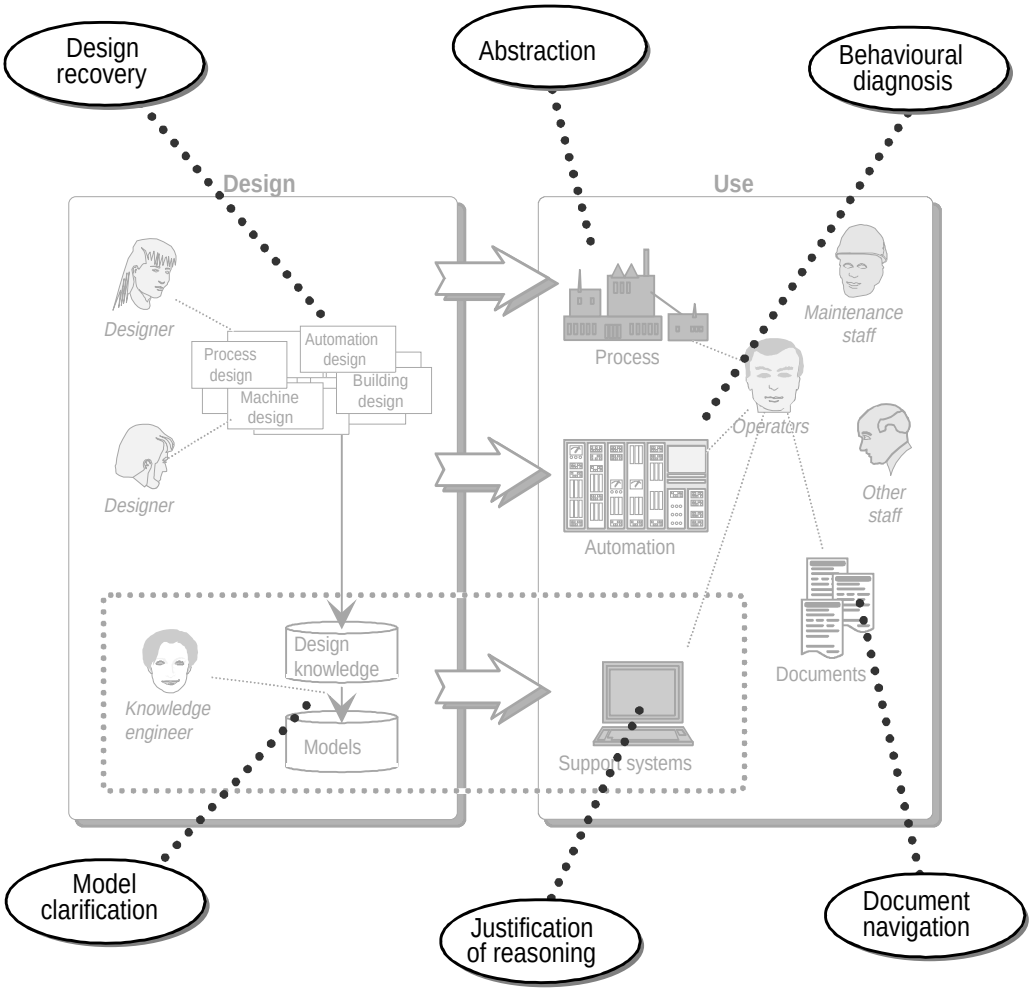

Figure 11. Generic explanation tasks in plant design and use.

Abstraction

Explanations can be used to reduce the complexity apparent to plant users. The various interdependencies in the systems can be demonstrated through explaining how individual parts are combined to function as a whole. Both physical decomposition (part/subpart) and functional decomposition (goal/function/device) dimensions are valuable to the users. In critical situations, details can be suppressed and only data relevant to the situation offered. The users can choose the level of detail that they desire.

Model clarification

Explanation techniques can be used to make the support systems' capabilities clear. The various items in the models can be explained: objects, properties, taxonomies, relations, constraints, and rules, among others. The model contents can be made active, capable of explaining themselves. This helps the users to understand the scope of the systems, demystifying them. This will also help knowledge engineers to analyse the contents of the knowledge bases.

Document navigation

Explanations can be used as an active index for the plant knowledge. They can be dynamically shaped into hyperdocuments, complementing the more static documents that come from the design process. The users' search for relevant documentation can be aided by systems that have a model of the documentation contents and knowledge of the search context. The users can be guided more directly towards relevant documents in the huge mass of information present in plants. Models can aid navigation between the documents.

Behavioural diagnosis

Explanations can demonstrate how processes behave and how controls work. The unfolding of events over time can be illustrated by selective replay, if proper data histories are available. The details of the automation can be abstracted or made clear to the smallest bit, depending on the desired level of detail. The strategies of control sequences and plant designs can be clarified to the users. All this helps them to understand why things happen. This understanding will help the plant staff to diagnose events and to plan the correct actions to take.

Justification of reasoning

Explanations can show how an expert system created its results. Even support systems implemented in traditional techniques can be analysed in terms of their problem solving methods. This helps to clarify both the knowledge contained in the system and the ways that this knowledge is used to process inputs to outputs. By understanding how a system creates its outputs, its users become more convinced that the results are correct. Alternatively, they may disagree with the system with greater confidence.

3.3 TASK FEATURES

Chandrasekaran [1986] states that a generic task can be defined in terms of its main features: the *task specification*, the *form of knowledge*, the *organisation of knowledge*, and the *control regime*. Table 6 details the

features of the generic explanation tasks that have been explored in this thesis.

Table 6. Features of the generic explanation tasks in this thesis.

Task	**Specification**	**Form of knowledge**	**Organisation of knowledge**	**Control regime**
Design recovery	Communicate design knowledge to users & designers	Design knowledge about the artifact, the design process & design history	Product structures, design procedures & decision rationale	Replay of design processes, search for design rationale
Abstraction	Help people to cope with complex information spaces	Multilevel structural & teleological decomposition of domain items	Part-whole & means-end networks	Navigation & knowledge derivation over levels of abstraction
Model clarification	Make system contents and capabilities more transparent	Objects, properties, relations, rules	Semantic networks of explainable objects	Queries & search over the networks
Document navigation	Retrieval of information relevant to the context	Static and dynamically created pieces of information and links connecting them	Contexted hypertext with indexes, semantic networks	Browsing, queries & search over hyperlinks
Behavioural diagnosis	Diagnose how things work	Causal, structural & functional dependencies of domain items	Causal networks, finite state machines	Backward chaining from events to causes & forward simulation
Justification of reasoning	Give rationale behind the expert system's results	Problem solving methods & strategies & domain models	Tasks/agents, traces, explainable objects, context	Backtracking through tasks from results to inputs

The list of the tasks is not exhaustive. Other approaches to explanation have been discussed in Chapter 8. At least the following aspects found in explanation research could potentially be regarded as generic explanation tasks:

- Explanatory tutoring
- Critiquing systems
- Explanation as storytelling
- Explanation as automated program writing
- Visualisation
- User understanding
- Agent-based help systems
- Commonsense understanding of the world
- Conversational repair of misconceptions

Other tasks could be found. However, the tasks listed in Table 6 are the most relevant as regards the work reported in this thesis.

Many of the listed tasks can be implemented in terms of other generic tasks. For instance, “behavioural diagnosis” could use the “hierarchical classification” and “abductive assembly” tasks defined by Chandrasekaran [1986].

3.4 MODELS FOR EXPLANATION

We advocate the use of models throughout the whole explanation creation process. To create the explanations of the listed types, the generic explanation tasks take the models as input. The techniques we propose for explanation are *model-based*.

We presume that many problems in explanation creation are reducible to the problem of finding suitable modelling techniques. The models produced should be usable by the listed generic tasks to cover the explanation types of Table 2, that is, the various classes of knowledge for each explanation target. The contents of the models should be obtained from design, where possible.

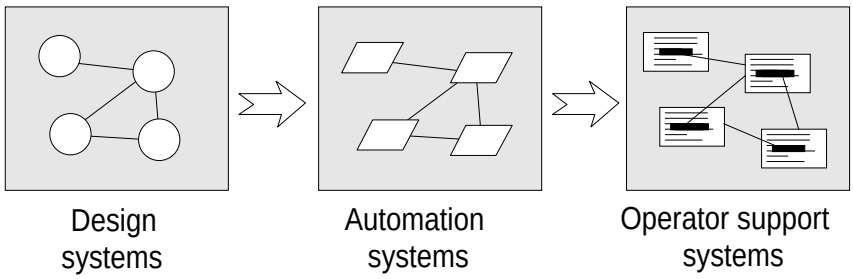

Figure 12. The use of the same conceptual model for all design phases.

The same conceptual model should be ideally shared in all phases of the plant lifecycle – from design to implementation to use and maintenance. Figure 12 illustrates this point. A semantic network that serves as the conceptual model is first created in the design, then stored in object databases in the automation, and finally communicated to the users through hypermedia. The similar interlinked nature of the conceptual model, the object databases, and the hypermedia nodes is believed to facilitate knowledge transfer from the designers to the users [Norman 1983].

In this thesis, we recognise four different kinds of knowledge-based models [Kaindl 1994, Franke 1991]:

- *Structural*, describing how things are connected physically or topologically;
- *Behavioural*, describing how things evolve through a series of states;
- *Functional*, describing how things are transformed into other things; and
- *Teleological*, describing the intended purposes of things.

A wide variety of different models have been reported in the literature. Kaindl [1994] surveys several different kinds of models used in artificial intelligence and software engineering. A range of different user-oriented models can be found in the field of human-computer interface studies [Helander 1988]. We view the four models above as the most relevant for this thesis.

There is general unclarity in the field of artificial intelligence about the definition of functional models. Many researchers refer to knowledge of purpose as “functional” [Tanner & Keuneke 1991], while others [Franke 1991] use both terms “teleological” and “functional” for such knowledge. We view functional models to express knowledge of flows or causality, and teleological models to express knowledge of purpose.

In model-based explanation, three points must be considered:

(1) how the models are created (*modelling*),

(2) what models are needed (*representation*), and

(3) how the models are shown (*presentation*).

The following four chapters are devoted to illustrating various mechanisms for implement generic tasks. In Chapters 4 to 6, we take a knowledge representation view to elaborate points **(1)** and **(2)**, while point **(3)** is targeted in Chapter 7 from an interaction viewpoint.

Each of the four chapters covers parts of the framework. The relation of the chapter to the framework is illustrated in the beginning of each chapter. Chapter 4 introduces the essential modelling concepts: explainable objects and means-end models. Chapter 5 builds on these concepts by explaining design knowledge stored in the models. Chapter 6 applies explainable objects into explaining the behaviour of automation. Chapter 7 studies more closely the necessary interaction techniques based on hypertext.

4 EXPLANATION AS OBJECT MODELLING

Generic explanation tasks need models as inputs for creating explanations. As depicted in Figure 13, this chapter concentrates on modelling methods that are useful for the “Model clarification”, “Justification of reasoning”, and “Abstraction” tasks of the framework presented in the previous chapter.

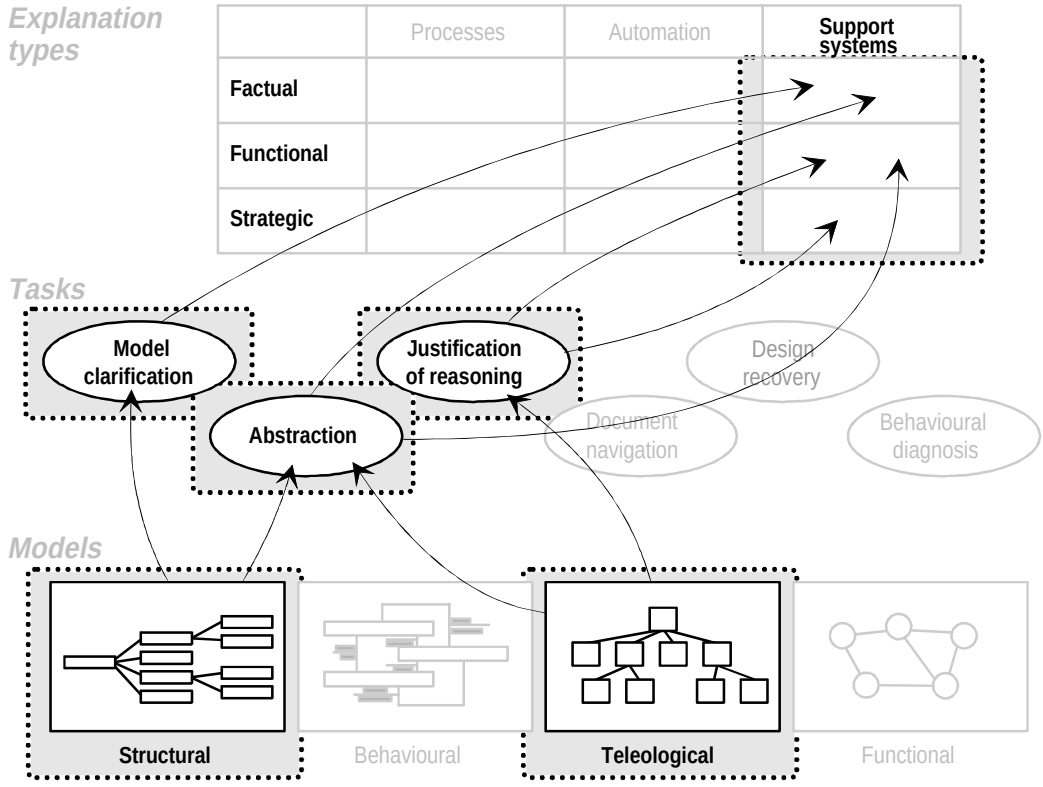

Figure 13. This chapter studies ways to implement the three tasks: Model clarification, Abstraction, and Justification of reasoning.

This chapter first introduces the concept of explainable objects for model clarification in Section 4.1. The VICE system makes the concept concrete in Section 4.2, using explainable tasks. They are a subclass of explainable objects capable of explaining their reasoning. Section 4.3 discusses relations between objects as an important part of explanatory models. A particular kind of relational model, the means-end model, is introduced in Section 4.4. Section 4.5 describes the extensions of means-end modelling that we propose for abstraction, resulting in the notion of contribution models.

4.1 EXPLAINABLE OBJECTS

Of the possible representation methods available in knowledge engineering [Garcia & Chien 1991], we have chosen object-oriented representations [Meyer 1988, Rumbaugh et al. 1991] as the basis for explanations. We use objects to implement models that capture semantic knowledge at an epistemological level. The models are, in essence, semantic networks [Nilsson 1982] that are implemented with object languages. There are sufficient grounds to this choice:

- Much of the available data about plants comes naturally in the form of objects (both physical and logical) and the relations between them [Rowan 1989, Gilbert & Wilhelm 1993, Korhonen 1991].
- It seems natural for humans to deal with computer representations of objects, as evidenced by the popularity of direct manipulation in human-computer interfaces [Schneiderman 1987] and object-based languages in programming [Rumbaugh et al. 1991].
- Objects in knowledge bases are close to human thinking [Hughes 1991], which should decrease the number of transformations from data to explanations.
- Objects can be made active, containing not only data but also procedures that reason on and explain the data.
- Often an object is the best expert concerning itself.[11]

Other representations, for instance predicate logic [Kowalski 1979], could be justified. From the implementation and maintenance points of view, however, it makes sense to select a representation that is well known in the domain. Most software systems in today's plants have been designed by conventional programming techniques, but there is a tendency towards object orientation [Gilbert & Wilhelm 1993]. Objects will therefore be natural to many people who design support systems [Årzen 1991], much more so than predicate logic or rule-based systems.

Many knowledge engineering tools today are *hybrid systems*, incorporating several paradigms [Hayes-Roth & Jacobstein 1994]. Most of these tools offer objects as the central way of organising knowledge. It is safe to base representations on objects, since it is or will be available in

[11] Often but not always. An external observer is sometimes necessary. In the real world, people resort to the help of experts (doctors, psychiatrists, and spouses) to understand themselves. An external observer can see beyond the local scope of the object, which is necessary for understanding [Card 1992].

many tools[12]. The emergence of object-oriented databases further emphasises this point [Kim 1990, Hughes 1991].

We propose to base explanations on the notion of *explainable objects*. They are objects that can offer explanations about themselves. Each object that is visible to users should be able to construct a local explanation of its own properties, of the relations that the object has with other objects, or maybe even of the design process that lead to its birth. The explanations should be accessible through a number of predefined methods. The methods should either return the explanations in a structured form to allow further processing or to directly dump the explanations in output devices (Figure 14).

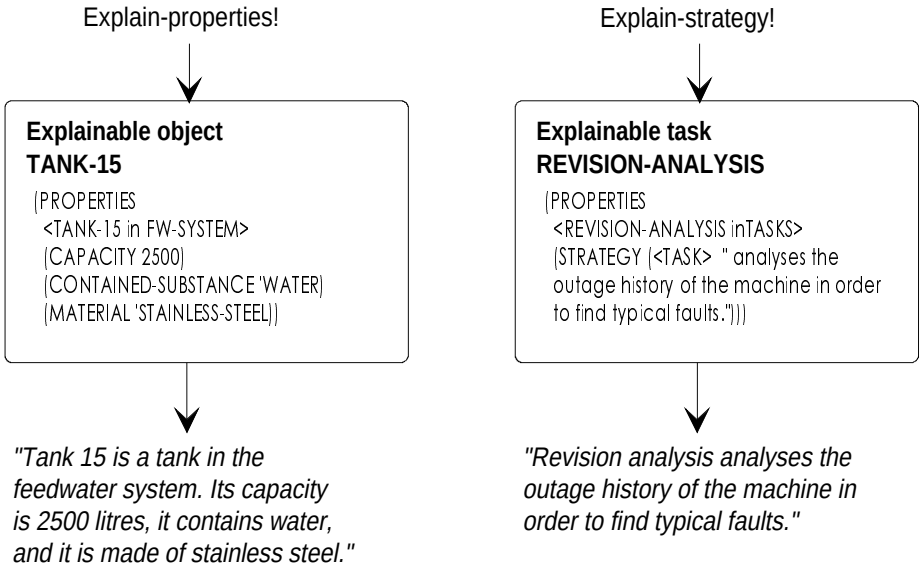

Figure 14. Two explainable objects.

The notion of explainable objects facilitates the distribution of explanatory duties. It obeys the principle of *locality*: each object knows only itself and its neighbours. This principle is known in the software engineering community as one of the main benefits of object orientation [Meyer 1988]. It increases the modularity of systems, minimising the surface area between the systems' parts. We invite this benefit to help in organising explanations in knowledge bases.

Another important feature of the approach is the *inheritance* of properties and methods through class relations. This feature can be used to advantage in explanation specification: a subclass may reuse the explanations from its superclass. In case the semantics of a class have been

[12] Some of the KBS tool providers even de-emphasise the AI techniques but stress the object capabilities [Durkin 1996].

altered through modifying its visible methods or properties, the explanations may have to be revised. Inheritance allows an economy of representation, which helps both explanation creation and maintenance.

4.2 EXPLAINABLE TASKS

We have used the notion of explainable objects in several prototype systems. Here we briefly describe the VICE system that bases its explanations on a particular kind of explainable object called an *explainable task*. VICE has been more thoroughly documented by Korteniemi [1989] and in Papers I and II. Other prototypes that use explainable objects are described in Chapters 5 and 6 of this thesis.

4.2.1 The VICE system

VICE (shorthand for VIbration Cause Expert) is a diagnostic expert system. It was developed by Imatran Voima Oy, a Finnish power company, to be used for analysing the condition of rotating machinery, e.g. turbine generators. Figure 15 shows an example of the system with explanations.

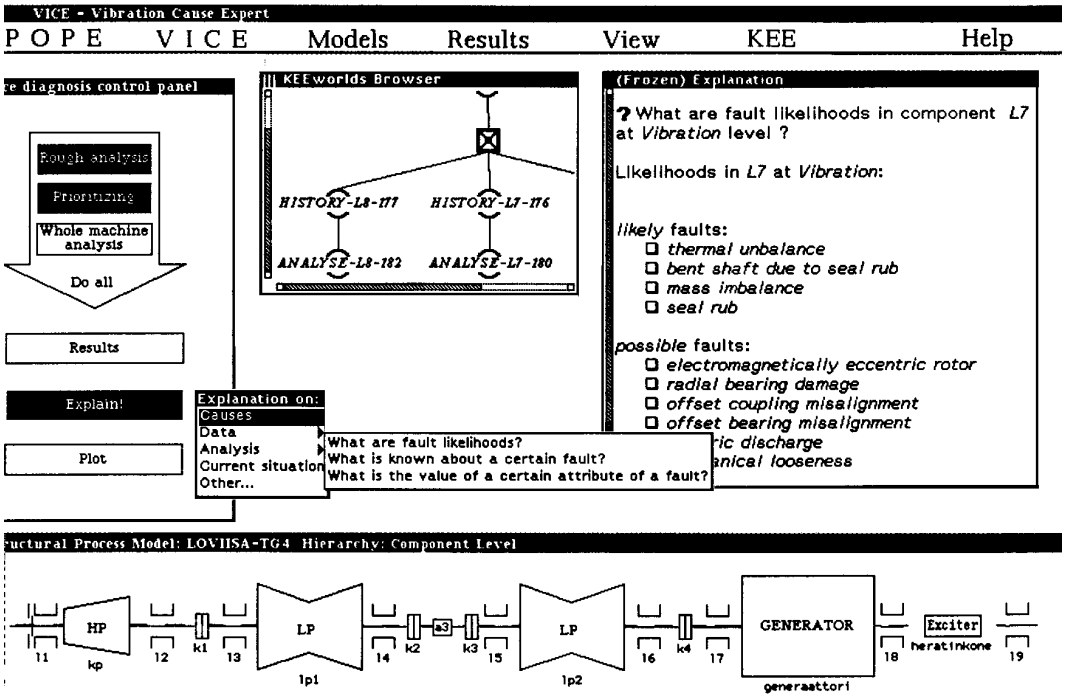

Figure 15. The VICE expert system with explanations.

VICE is a decision support tool mainly for vibration experts. They measure the vibration of rotating machine parts, and by interpreting measured signals they can analyse the condition of the machine. The goal is to predict failures and to avoid downtime of expensive power production facilities. In addition

to vibration experts, plant operators and system developers are also recognised as potential user types.

VICE has structural and behavioural domain models. The structural model contains the properties of turbine generator parts and the relations between them. The behavioural model contains the known fault types with their properties and interdependencies. The contents of these models have been partially made available to the users. Particularly the attributes of machine parts and known faults are important to the vibration analyst. The models are built using the KEE environment [Fikes & Kehler 1985].

The concept of explainable objects turned out to be useful in VICE. Part of the utility came directly from the convenience of explanation generation through active objects. Perhaps an even greater benefit was indirect: the explanations forced the system developers to reconsider the structure of knowledge bases to make them more easily explainable. The effect was most apparent in the problem solving model, where we refined the reasoning processes to a more fine-grained level. This restructuring made the system more understandable.

4.2.2 Explaining reasoning

Reasoning in VICE has been modelled as a set of problem-solving *tasks*. Several researchers have found explicit task models useful for reasoning, including Clancey [1983], Chandrasekaran [1986], David & Krivine [1989], Seppänen [1990], Steels [1990], and Tanner & Keuneke [1991]. In our view, tasks are active agents, *explainable tasks*, capable of generating conclusions based on some input knowledge. Normally these conclusions consist of making changes to the knowledge contained in the system's models. The reasoning within the tasks is performed by Lisp routines or rules attached to the tasks. The tasks are akin to the concept of generic tasks [Chandrasekaran 1986]. VICE tasks are generic within the domain of vibration analysis.

Every time a task draws a conclusion, a note is made in a task trace (stage 1 in Figure 16). When the user asks for justification (2) for a conclusion, the responsible task is located through the trace (3) and asked to explain its line of reasoning (4). The task responds by offering a template (5) that describes how the task uses the input knowledge to arrive at the result. This template is then filled in with local variable values (6) and shown as hypertext (7). This interaction principle is elaborated in Chapter 7 of this thesis.

This approach to explanation generation is an informed compromise in terms of detail. Instead of completely prefabricated answers, we allow the activation context to influence the answer generation (the mechanisms for this are discussed in Chapter 7). Instead of detailed rule derivation, we explain a group of rules or reasoning procedures together. Where the tasks contain formal rules or logic, they could be explained in the minute detail

using classical rule-based explanation [Buchanan & Shortliffe 1984], if so desired. We have not followed this route. In our experience, task-based explanation achieves a reasonable balance between extreme detail and extreme generality. Template-based text generation does suffer from some problems though that will be discussed in Chapter 7.

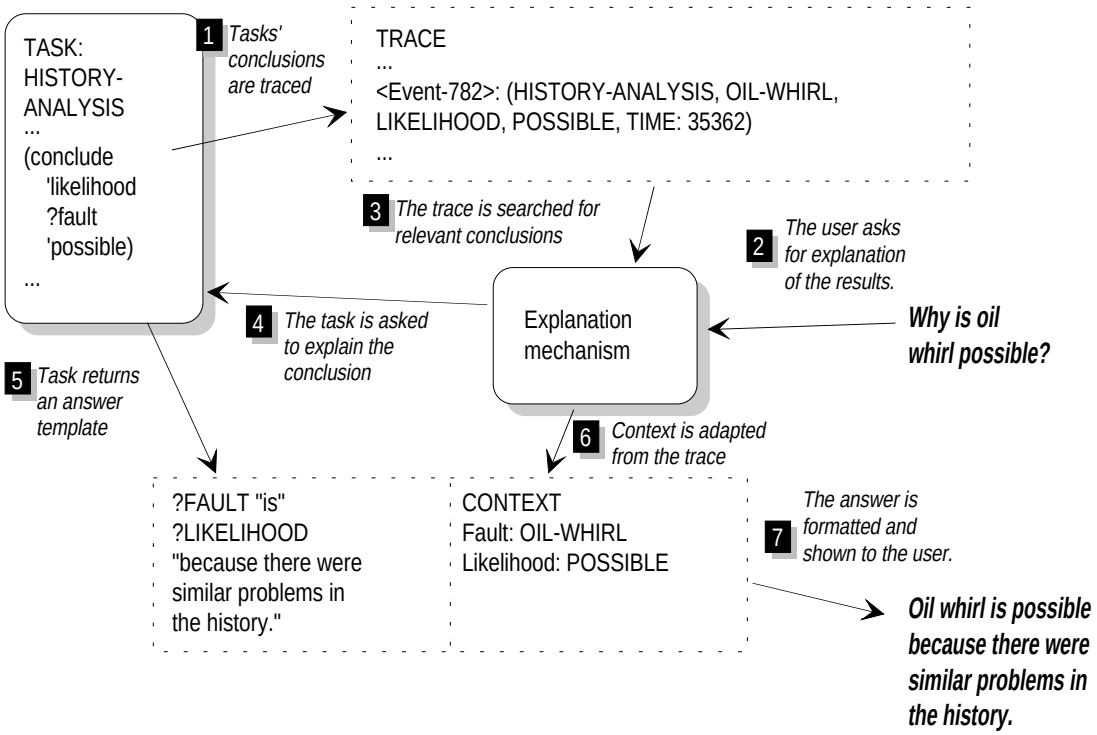

Figure 16. Tasks can explain their reasoning and strategies.

4.2.3 Explaining strategies

The tasks can also explain their purpose, or strategy, to help the user understand the meaning of a certain operation [Clancey 1983, Hasling et al. 1984, Chandrasekaran et al. 1987]. The tasks form a hierarchy: the topmost tasks correspond to major phases of vibration analysis, while the tasks below correspond to individual analyses inside these phases. The task division goes further until the bottom level consists of tasks that only make singular decisions on individual analysis results.

The control follows this hierarchy. When a task is invoked, it invokes its subordinate tasks in a depth-first manner. At any point of analysis, users can ask what is happening. The task that is executing at the time of the question can then offer an explanation of its strategy for preforming the analysis, again through templates and hypertext. The last figures of Paper II give examples of these explanations.

Note that we only explain the strategic purpose of a task [Chandrasekaran et al. 1987, Tanner & Keuneke 1991]. The execution control of the tasks is left unexplained, although it is visualised in a coarse manner through the user interface (the large downward arrow at the left of Figure 15 shows the major diagnosis phases).

This task division is useful not only for tracing conclusions, but also for clarifying the reasoning process to the users. The main users of VICE were normally vibration analysis experts, who could easily visualise the system's reasoning as a hierarchical set of analyses. Therefore the way of representing knowledge through tasks matched well with the users' mental models (the analyses). As a result, the reasoning was easier to explain.

4.2.4 Development support

The explanation mechanism was an essential aid for the development of the VICE system. The developers could debug the system's operation through the explanations. Validation became easier as well. The developers could walk through the analyses with the experts. If there were doubtful results, with explanations they could trace back to the relevant task and modify its knowledge. The explanations could be stored in the form of reports. Together the explanations and reports greatly enhanced the development and validation of the system.

On the problematic side, the increased number of tasks required by explanations lead to greatly increased memory requirements for the tasks and the traces. This was a problem for environments with limited resources. Tracing in general may become burdensome with systems that have intense reasoning activity.

The development of VICE was continued for five years. During its lifetime, VICE was used by four developers and half a dozen experts. The explanations were seen as an essential feature of the system and judged to be useful, although no comprehensive evaluation was done on the quality and utility of the explanation mechanism.

We have later used the principles developed for VICE in other projects, as will be detailed in Chapters 5 and 6.

4.3 EXPLAINING RELATIONS

In object models, only part of the knowledge is stored in the objects' properties. An equally important part are the relations between the objects. We have extensively used object networks in the implementation of our prototypes. They can be regarded as semantic networks or taxonomies [Woods 1986].

We have analysed several technical systems at industrial plants and high energy physics experiments. The results of these analyses have been

documented in the included papers. Judging by these analyses, a great deal of the knowledge contained in manuals, drawings, and databases, is readily representable as networks of related objects. The following digest lists some of the relations extracted from a gas system manual at one of the CERN experiments [Peach 1994]:

> The compressor is part of the recirculation system. The controller is located in the SG2 building. The P chambers contain gas. The fresh gas system connects to the recirculation loop. The loss of power stops the gas system. The alarm switches off the power supply cabinet. The mercury switch will close the fresh gas supply valve. The operator locks alarms when starting to purge. The Eurotherms control the bypass valves. The back pressure regulator maintains the pressure. Increasing the recirculation pressure causes increased vent flow. The main electrical cabinet powers the local cabinet. The bottles supply argon to the system. The filter removes particles. The bubbler prevents chamber overpressure. Oxygen is removed from the gas with the purifier. The sensor measures the gas flow. The documentation describes the system. The alarm informs the user about the overpressure. A magnetic valve is a kind of valve.

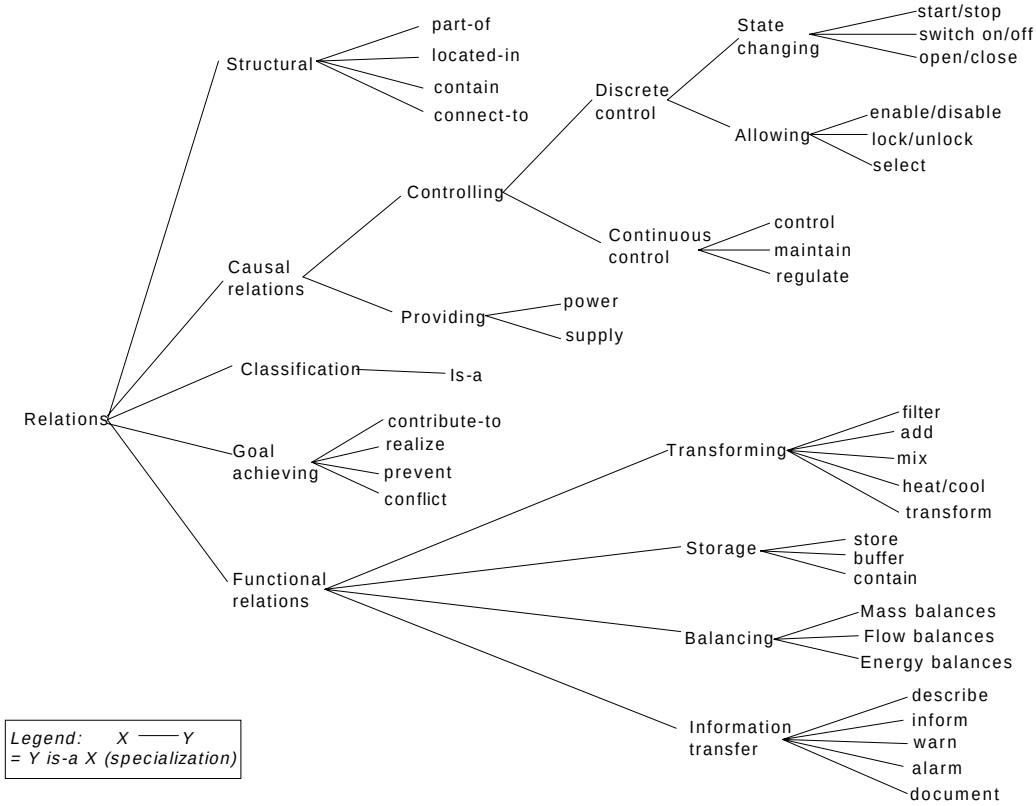

Figure 17. Relations for describing high energy physics gas systems.

Figure 17 shows a taxonomy of some of the relations that we have found useful for talking about high energy physics systems. We limit ourselves to binary relations[13]. According to Broadbent et al. [1986], binary relations are the most useful ones to be explained to humans. More complex relations are harder to comprehend and thus to explain, and we have left them aside as a topic for further research.

We noted in Chapter 2 that explanations should make relations visible. This can be readily accomplished in many knowledge-based tools (in the KEE system, for example), where relation graphs can present the relations in a visual form. Unfortunately, the graphs may become convoluted with many objects. A more local possibility is to use explainable objects as a starting point: when explaining an object, the references to other objects could appear as hypertext links. This principle has been used in our explanations, as detailed in Chapter 7.

A way to make knowledge bases more explainable is to design the objects and relations as a domain-specific vocabulary for talking about systems [Hayes-Roth & Jacobstein 1994]. More concretely put, there should be hierarchies of both objects and relations, where concepts could be explained on a general or detailed level as needed. Not all items in knowledge bases should be explained to all users.

The relations can be implemented as another class of objects. In a way they can be considered as metaobjects. The taxonomy of Figure 17 forms the class structure, where the properties of relations can be inherited. This helps us to manage a large number of possible relations. Knowledge representations can use the very specific relations, at the bottom of the hierarchy, while most of the properties of these relations can be defined at the topmost levels. In this way, one can talk about the domain in natural terms while the effort to model those relations is minimised [Woods 1986].

4.4 MEANS-END MODELLING

Rasmussen [1986] suggested a particular type of relational model, known as the *means-end model*, to model the decomposition of industrial plants at multiple levels of abstraction. We have based many of the developments presented in this thesis on variants of means-end models.

There are two dimensions to the decomposition, as depicted in Figure 18. The vertical axis is the means-end dimension. It shows how a system can be considered at multiple levels of abstraction, with concrete physical devices at the bottom and the abstract goals of the system at the top. Each device has a function that it implements. These functions in turn are needed

[13] Many researchers in the plant domain seem to have assumed the same limitation. For instance, Korhonen [1991] has outlined a rich set of (general and detailed) relations that are binary.

to realise certain goals. The relations can be many-to-many, since a device may have several functions, and a single function is usually implemented with several devices. Likewise the goals are often related to several functions and vice versa.

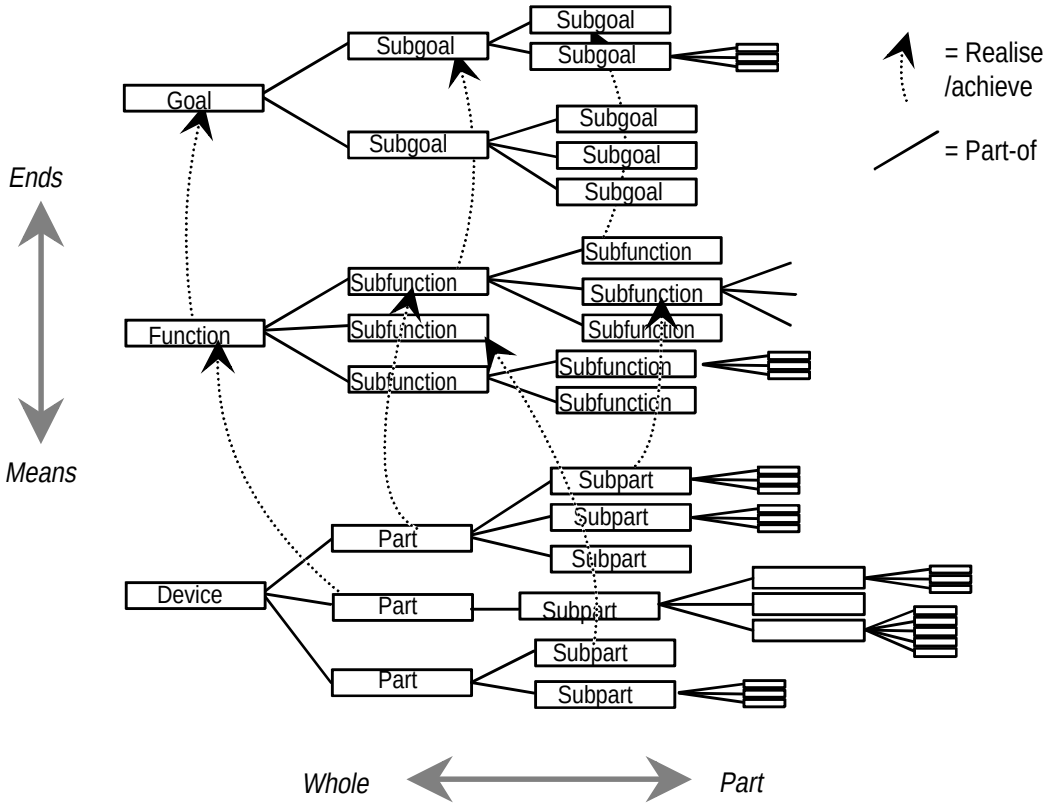

Figure 18. Decomposition in the part-subpart and means-end dimensions.

The horisontal axis extends along the whole-part dimension that is the traditional way of decomposing technical systems. A plant is made of subsystems that are in turn made of smaller systems. This successive decomposition forms a tree whose leaves form the concrete objects found in plants. Not only are the physical devices part of the hierarchy, but also logical elements, such as alarms or software modules. Even the humans at a plant could be modelled to carry out parts of the hierarchy. Decomposition is applied at all levels: functions are made of subfunctions and goals are made of subgoals.

Rasmussen [1986] identifies five levels of means-end abstraction for modelling industrial plants (Figure 19). Each level is an abstracted view of the level below. The three intermediate levels together form what we call the function level in this thesis.

The means-end model forms a framework for representing knowledge related to plants. It makes the goals and functions of the systems explicit and shows how they are related to the devices. This model can be used in various ways: to explain teleological knowledge (purposes of things), to

guide diagnosis of malfunctions, and to guide navigation and presentation of information in user interfaces, to mention but a few [Rasmussen 1986]. Generic tasks can be defined for these aims, each task using the same model for different purposes.

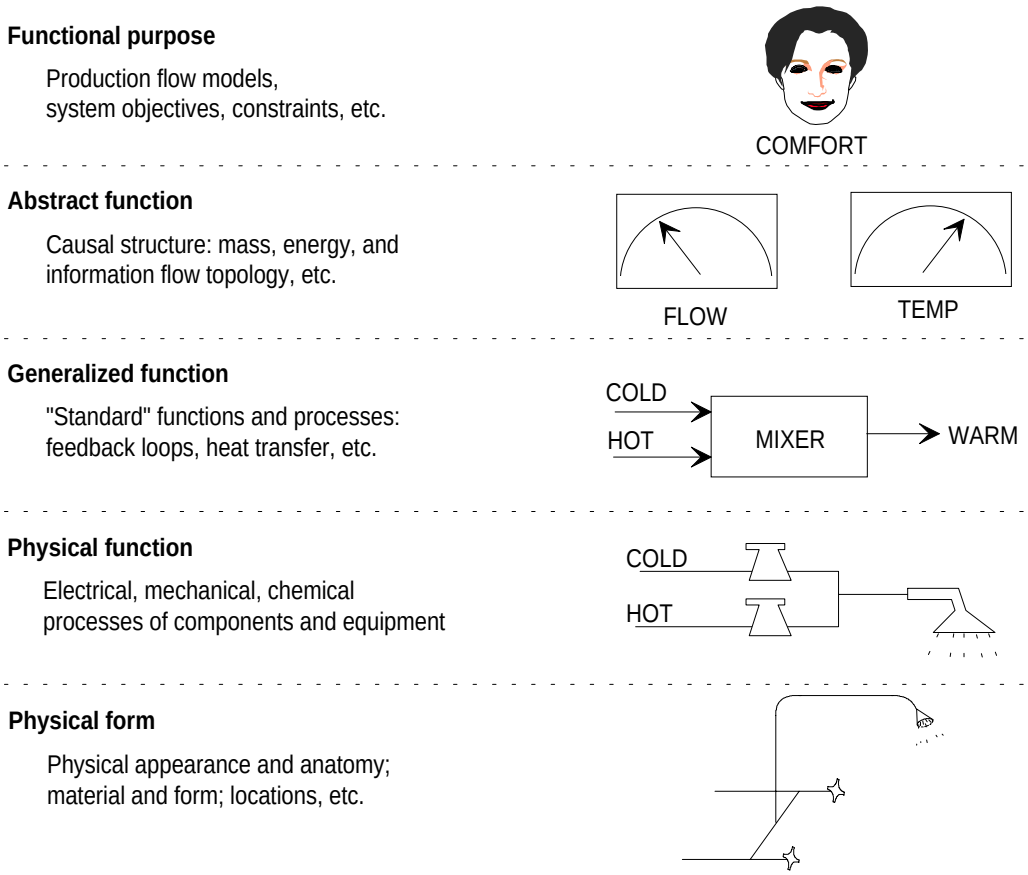

Figure 19. The five levels of abstraction for a very simple process (adapted from Rasmussen [1986] and Rytoft et al. [1990]).

The model attacks the complexity that is apparent to the users. As Bisantz & Vicente [1994] note, the model structures the domain in a psychologically meaningful way. It allows the users to study systems at multiple levels of resolution, shifting levels to “see the forest through the trees”.

It should be noted though that means-end models can be hard to build and maintain [Bisantz & Vicente 1994]. The difficulties with the models are discussed in Section 4.5.

A means-end model in itself is just a representation. It must be surrounded by systems that use that information. We have built prototypes of such systems, described in Sections 4.5.3, 5.3, and 7.3.

4.5 CONTRIBUTION MODELS

In our studies of the power plants and high energy physics control systems, as documented in Papers III, IV, and VI, we found that it is at times difficult to decide whether an item should be modelled as a function or goal. For instance, the goal "Maintain pressure" can be seen both as goal and as a function, depending on the viewpoint. We suspect that this is a major source of confusion for knowledge engineers who craft means-end models or its derivatives.

We have used means-end techniques to study two other industrial cases, besides the power plant and gas system documented in the included papers. In all these cases there has been debate as to where to draw the line between functions and goals.

Some of this unclarity may be due to our merging of the three function levels – abstract, generalised, and physical – into a single level. With the five separate abstraction levels it might be possible to find more satisfactory divisions. However, this benefit would come with an increased load in modelling, as five levels would have to be dealt with, instead of three.

There seems to be similar unclarity between the device and function levels as well, when non-physical devices are dealt with. Moreover, at the function level it is sometimes difficult to decide whether to model a dependency as a means-end or a whole-part relation. As a consequence, different people tend to create very different models.

4.5.1 Relaxing the levels

We believe that it is artificial to force the representation to strictly divide things into levels. Instead, we propose to *relax the notion of levels* in means-end models. We let all items in models to be considered as concrete or abstract functions, goals, or even devices. The levels are only defined implicitly through the mutual ordering between items. In object-based means-end models, an object may freely inherit the properties and behaviour of the desired level.

The mutual ordering is accomplished by the single "contribute" means-end relation type in the model. This directed relation could equally well be called "realise", "achieve" or "depends-on". It signifies the dependency of an item on another item. The network of these relations forms, in essence, a directed acyclic graph [Sedgewick 1989] that we call the *contribution model*.

The representational power of the contribution model is necessarily less than that of means-end models. It requires the use of context to interpret the role of a single model item. However, this is not a problem, since in support systems the navigational paths often lead to model items from external situations, where the context is given. Moreover, the users do not seem to really care whether an item is a goal or a function. This distinction is

therefore only significant to knowledge engineers building the models. Relaxing the levels helps them but does not affect the users.

4.5.2 Gas system example

We applied contribution modelling in the CICERO project at CERN [Barillère et al. 1993]. There a gas system was modelled that supplies an inflammable argon-ethane gas mixture to one of the particle detectors of the L3 experiment at CERN. The gas mixture and pressures must be precisely controlled to achieve suitable conditions for particle detection within the detector chamber. The gas is being constantly refreshed, with old gas being vented out and new gas brought in. [Peach 1994]

Figure 20 shows the portion of the gas system model that concerns the goal of maintaining the system's availability for physics measurements. The model was created through interviewing the designers of the system and by analysing the associated documentation. The whole model consists of about a dozen goals, forty functions and two hundred devices. The device level of the model was not completely defined; many subsystems were considered as black boxes. This level of detail was sufficient for the purposes of the project.

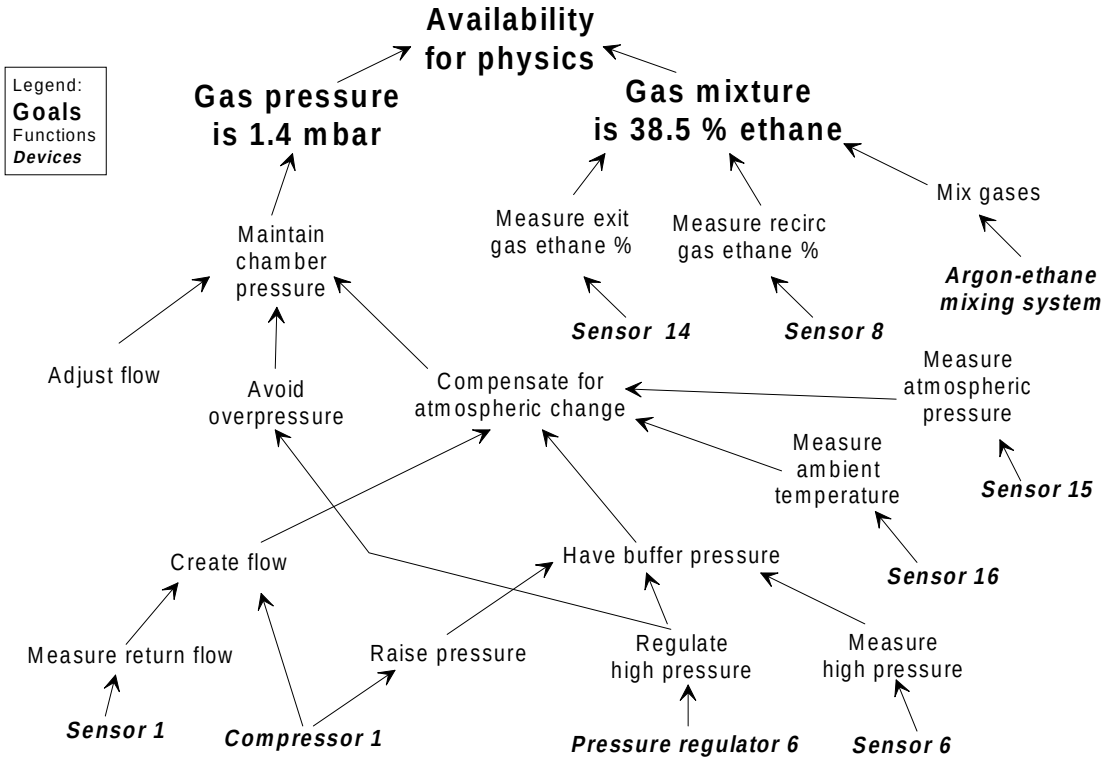

Figure 20. Portion of the contribution model of a gas system at CERN.

4.5.3 Weighted relations

Means-end models tend to suffer from a large number of transitive dependencies (this topic is elaborated in Section 5.4). The resulting multiple paths through the models may cause problems to algorithms that are based on search. To lessen this problem, we propose to encode *weights* into the contribute-relations. These numbers signify the strength of the contribution of an item towards another item. For instance, it could be said that the function “Regulate high pressure” contributes to the function “Have buffer pressure” by a factor of 0.7, but only by a factor of 0.3 to the function “Avoid overpressure”.

The weights can be used to control the search over the contribution network. Existing heuristically informed search methods, such as beam search, can be used to limit the number of paths that are taken in the search for purposes. The cumulative product of the weights can be used as the distance metric. Those paths can freely be ignored where the cumulative weight falls under a predefined limit, signifying that the overall transitive contribution over that path is too small to be of interest. Moreover, the weights can be used to select the order in which multiple results get shown to the users. The items with the largest cumulative weight are explained first.

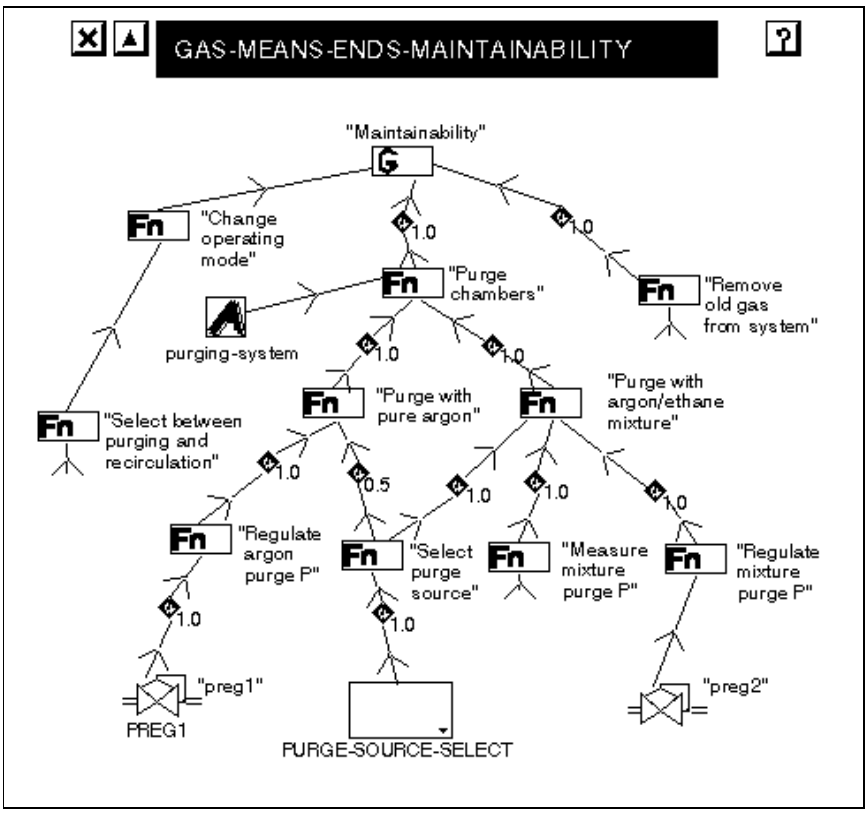

Figure 21. Portion of a contribution model prototyped on the G2 expert system shell.

Paper VII discusses contribution models in the case of HEP systems. Since the publication of the paper, we have investigated the use of weights in contribution networks with a small prototype implemented with the G2 expert system tool. Figure 21 shows a portion of a gas system's contribution model with the weights between the model items. The relations are drafted with the standard G2 facilities of drawing connections between objects. These connections are then translated into contribution relations.

As a result from the prototype, we have noted that the weights give valuable information that is missing from the means-end class of models. Unfortunately it is not clear how the weights should be encoded. There are a number of ways to manage uncertain knowledge, summarised in Garcia & Chien [1991]. We have used a straightforward fractional weight that ranges from 0 to 1, much like the certainty factors of Mycin [Buchanan & Shortliffe 1984]. Clear semantics are yet to be defined for the weights.

Humans tend to have problems in giving numerical estimates, whereas the linguistic terms used in fuzzy logic are more readily accepted [Zadeh 1984]. An interesting possibility would therefore be to define the weights using fuzzy factors, with fuzzy reasoning applied to calculating the cumulative weights over the network.

4.5.4 Dynamic relations

The dependencies in means-end models are not static. They often depend on the operational states of the plant components [Bradshaw & Young 1991]. For instance, the oxygen sensor's reading in the L3 argon-ethane gas system is irrelevant when the experiment is not collecting collision data. In normal use the same information is crucial for calibration of the measurements. This state information is often available from the control systems, where sequences or state machines drive the operations. The contribution weight can be made dependent on the state (Figure 22).

Van de Ree [1994] and Jaako [1996] propose to construct different models for the different states of a plant. In our view, this would considerably increase the effort in maintenance of the models. We depart from their approaches by merging the different models into a single state-dependent model, decreasing the need for model maintenance.

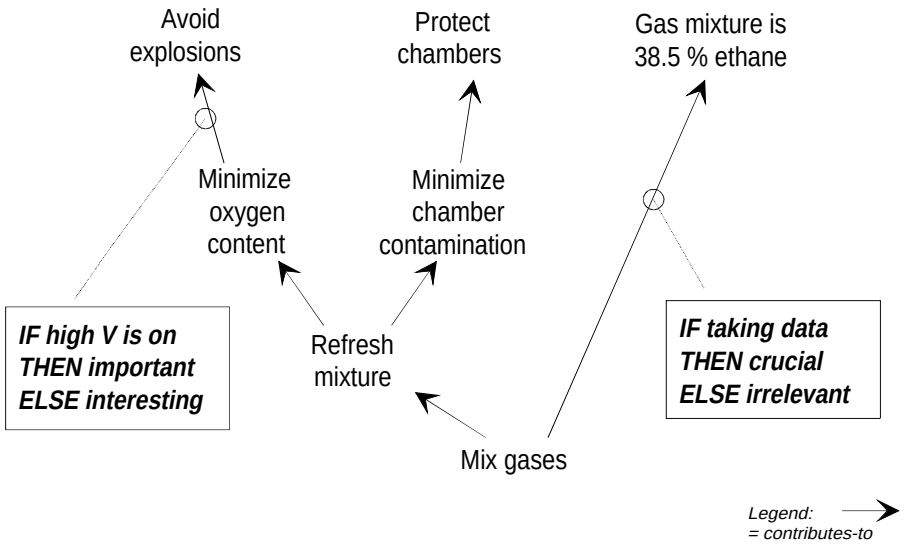

Figure 22. The relation weights can be made dependent on the system's state.

In conclusion, we propose to extend means-end models into contribution models enhanced with state-dependent fuzzy weights. We suggest this extension as an interesting research topic.

5 EXPLANATION AS DESIGN RECOVERY

In this chapter, we concentrate on explanations that could help the maintenance staff in plants. Much of their work deals with reverse engineering of existing designs, as we noted in Chapter 2. Explanations of design knowledge could decrease the effort in plant maintenance. The explanations can be crafted from models that capture the knowledge at the time of design.

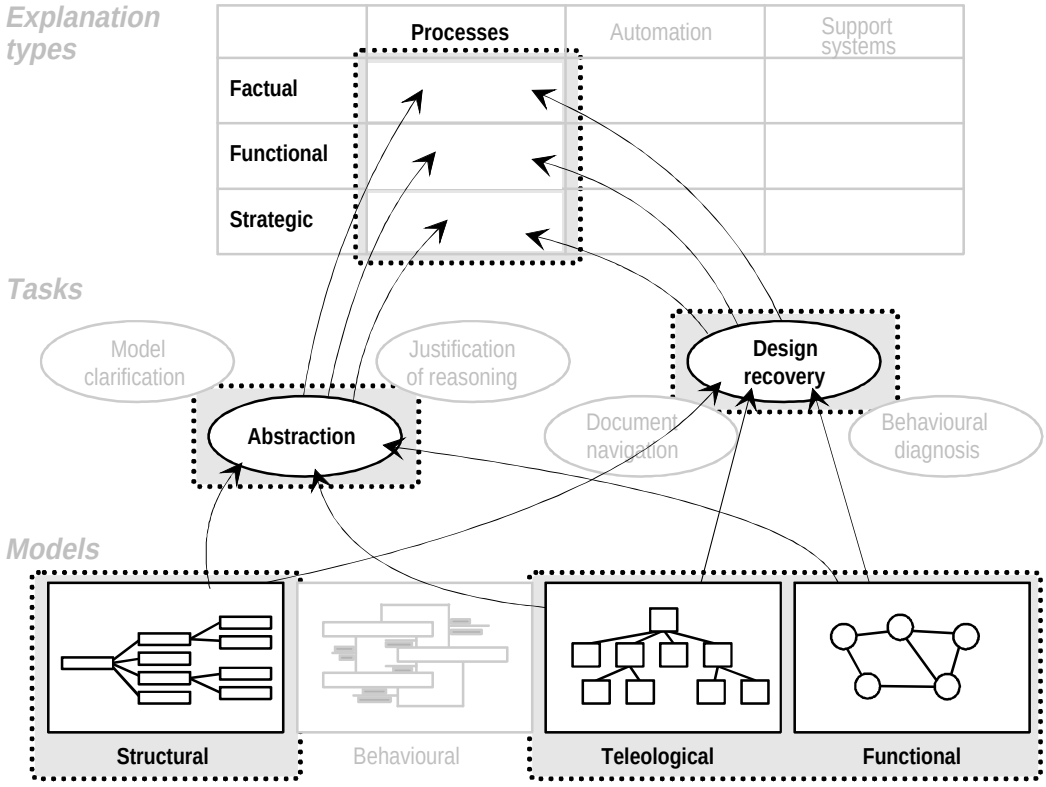

Figure 23. This chapter studies recovery of design knowledge through model abstraction.

Explanation calls for a common conceptual model throughout the design process, as formulated in Section 3.4. We propose Lind's [1990] Multilevel flow modelling (MFM) to function as the conceptual model. MFM models can be used to capture design knowledge, to enhance reuse of the knowledge, to model the knowledge in automation systems, and finally to construct explanations in automation and support systems. Thus MFM

facilitates the transfer of explanatory knowledge from design to use. In terms of the framework of Chapter 3, it forms the input to the Design recovery and Abstraction tasks (Figure 23).

Section 5.1 introduces MFM modelling as an extension to the means-end models of the previous chapter. Section 5.2 shows how we extended a design environment to capture explanatory design knowledge into object models at the time of design. Section 5.3 formulates the mechanisms for design recovery with methods of explaining the knowledge in the models at multiple levels of abstraction. Section 5.4 briefly discusses the utility of the approach.

5.1 MULTILEVEL FLOW MODELS

Lind [1990] has extended means-end modelling to cover the causal dependencies in the plant. His *Multilevel Flow Modelling* (MFM) method adds the concept of *flows* to the representations. This extension strengthens the models by making the causal interaction paths between process parts explicit. In physical systems, the interchange of energy and mass between the parts is responsible for the observable interactions [Lind 1990]. The patterns of these interactions would be valuable information for the plant users.

In MFM, the energy, material, and information flows are recognised at the various levels of abstraction. The flows through the parts of the plant are modelled as flow structures, over which the energy or matter balances are known. Inside the flow structures, each device is modelled to carry one or several primitive flow functions: sink, source, balance, barrier, transport, or storage. Combinations of these primitives form the functional structures that are found in plants.

The means-end dimension, as defined by Lind, consists of three levels: devices, functions, and goals. Mappings between the levels come in the form of four relations. The *realise* relation couples functions to devices. The goals are *achieved* by functions. The third relation, *achieve-by-control*, represents the need for a manager (usually a control loop or the operator) to achieve certain goals. The fourth relation shows the *condition* that a goal must be achieved in order for a function to work as expected.

Figure 24 shows an example of these relations for a feedwater system, depicted with the symbols that Larsson [1992] suggests for MFM diagrams. The device level is not shown in the diagram, as is usual with MFM. However, the relations to devices form an essential part of MFM models.

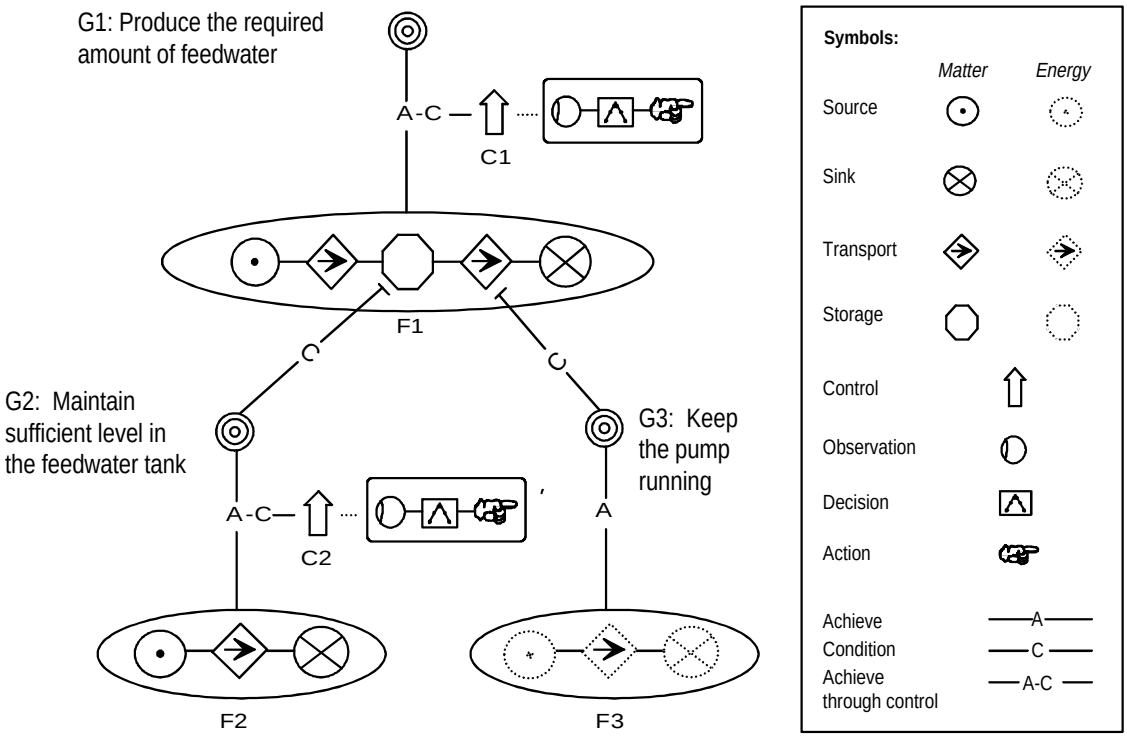

Figure 24. An example of a multilevel flow model (adapted from Kaarela [1996]).

Simple laws of matter and energy conservation are used in MFM to describe the normal state of the process. The model is thus a normative one, describing the process as it is meant to function. It can be used to detect variations from normal behaviour, though it is not efficient for simulating future behaviour. Abnormal process conditions will show up as variations in the matter or energy balances, which facilitates diagnosis. Other uses for MFM include alarm analysis, plant state abstraction, measurement validation, and user interface building [Lind 1990, Larsson 1992, Sassen 1993, Van de Ree 1994, Kaarela 1996].

Vicente [1992] and Kaarela & Oksanen [1994] have investigated the application of MFM in the user interfaces of automation. MFM forms a meaningful way of structuring the display hierarchies, and facilitating information abstraction and alarm processing. Normally the users would only see the abstracted state of the overall goals of the plant. When a goal ceases to be achieved, the displays then guide the users to the functions and devices that no longer work as expected. [Kaarela 1996]

5.2 THE P&ID ENVIRONMENT

Our industrial partner, Tampella Power Oy, developed an application called P&ID for supporting initial phases of boiler design [Riitahuhta 1988]. The P&ID tool provided us with a knowledge-based platform for testing our ideas about modelling design knowledge for explanations.

The goal of the system is to encode knowledge about preliminary boiler design into object structures, and thus to reduce time spent on routine calculations and labeling. Tampella Power Oy built a model of the feedwater system of the power plant that we studied.

P&ID was constructed on top of Design++, a general engineering tool marketed by Design Power Inc. It can be used for maintenance of product knowledge and design knowledge through automated design features. The system has been developed since 1987, through various generations. The version used in our work was 1.4, running under Unix on Sun Sparcstations.

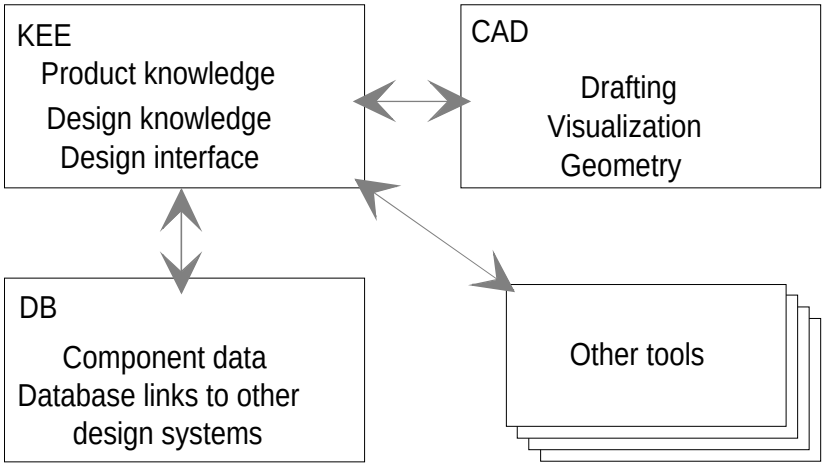

Figure 25. The Architecture of Design++.

Figure 25 portrays the Design++ architecture. The central element in the system is the KEE environment[14] by Intellicorp, which is used to maintain the knowledge about the product and the design. A CAD system assists in drafting and three-dimensional visualization. There is an interface to a relational database, which contains component data and allows communication with other design environments. Finally, there are links to other software, allowing transfer of text and CAD diagrams.

The components of the product structure are organised by two kinds of hierarchical relations: (1) the *is-a* relation, creating an inheritance hierarchy, and (2) the *part-of* relation, classifying the objects according to their role in the product's structural decomposition. The knowledge about these components' properties is stored in the objects' attributes. Each attribute may

[14] KEE has later been replaced in Design++ by a more modern object store.

have an associated design rule that can be used to infer the value of the attribute. The rules may retrieve knowledge from other objects' attributes.

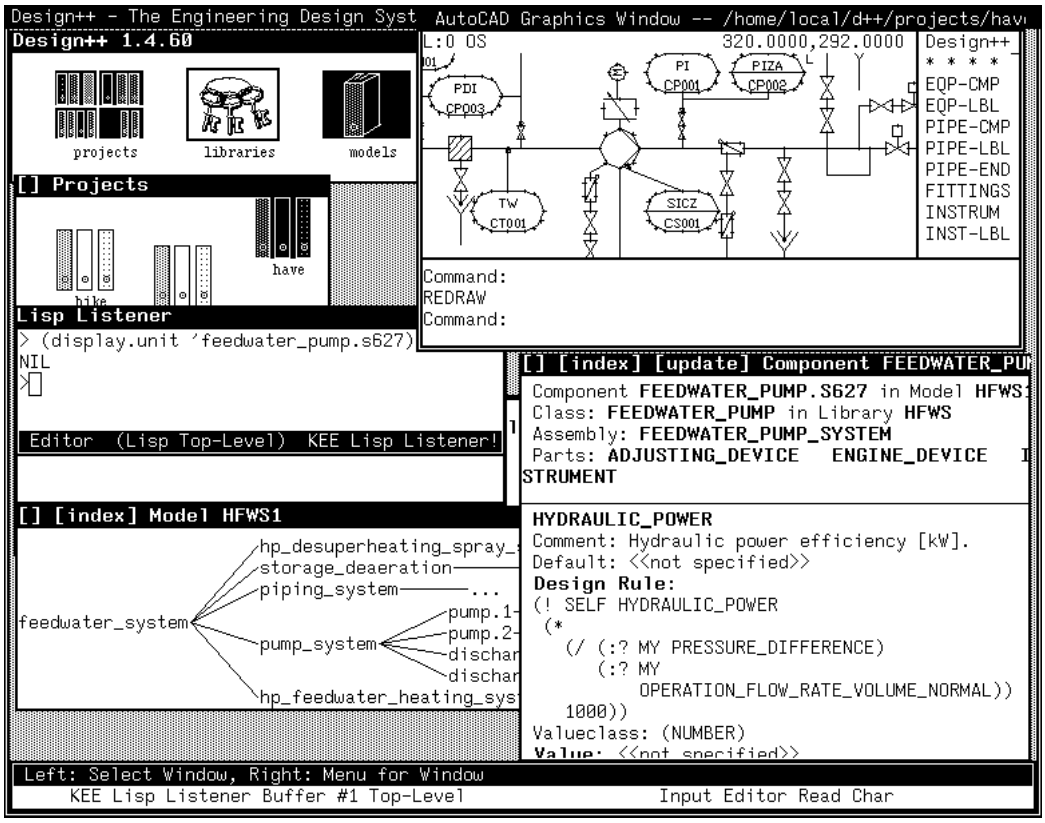

Figure 26. The P&ID application on the Design++ tool.

Figure 26 shows a screenshot of the P&ID application. The user interacts with the system through the facilities provided by Design++. The browser window on the lower left shows the top levels of the part-subpart-hierarchy in the feedwater model. The concrete devices are on the right, and the hierarchy shows their composition into larger assemblies. The Autocad window (upper right) displays the graphical view of the model. There is a one-to-one mapping of each concrete component in the model into its graphical appearance.

The object's properties can be accessed through separate windows (lower right). Each property has an associated design rule. The majority of the rules contain knowledge of sizing calculations, labeling and diagram layout. The window shows a rule that calculates the hydraulic power of the feedwater pump based on other data.

We extended the P&ID tool adding facilities to model the function and goal levels of means-end models. The Design++ tool already offered facilities to model the whole-part decomposition, and the P&ID application formed the device level of the models. Figure 27 portrays the extensions.

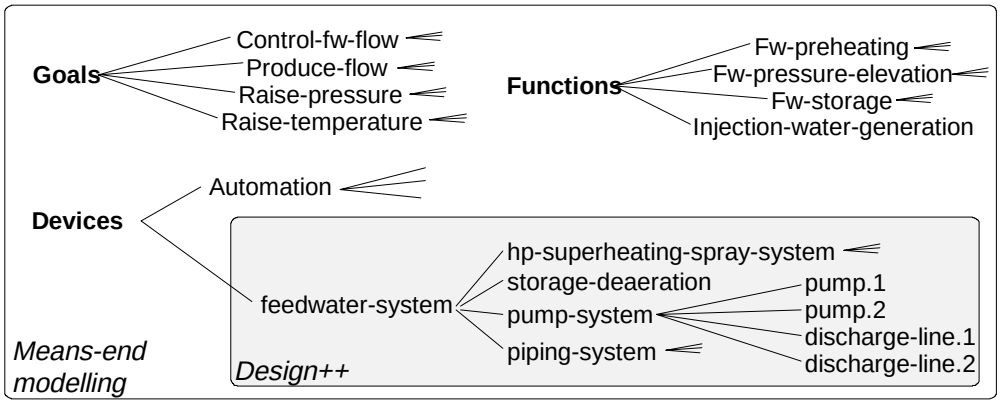

Figure 27. Means-end extensions to the Design++ tool.

We built simple menu-based mechanisms to define the relations of the objects at various levels. The resulting means-end networks could be visualised by the graphical browsers offered by KEE. An Autocad-based editor with MFM templates was also constructed to allow graphical modelling. Mechanisms were implemented into the model items to record design rationale at design time: design issues, alternatives, design choices, and justifications. Facilities were created for exporting the models in various forms: relational tables, C++ objects, structured text (SGML), and Prolog facts. [Kaarela 1996]

The key point of the approach is in recording the design knowledge. The MFM models should be constructed during design, when the background knowledge for the decisions is most readily available. Through suitable tools, the designers themselves could construct large portions of the MFM models. Ideally, designers could document their solutions in the models when they create the designs. This, of course, requires extensive support from the design tools to minimise the additional documenting work. The above mentioned extensions to P&ID show the way for such tools.

The approach could be used to reuse design knowledge [Kaarela 1996]. Existing designs are often taken as the starting point for new designs. If the semantic knowledge behind these designs were available with the design, reuse would be enhanced. The design processes themselves would become reusable, in addition to the end results. This would necessitate explanation of the knowledge contained in design models.

In our view, the most important dimension of the MFM models is their means-end knowledge, as far as reuse is concerned. We have therefore sought to convey such knowledge with the explanations. Explaining the interactions between plant parts through energy and matter flows remains an open research topic.

MFM forms a model for structuring the documentation concerning the plant [Kaarela et al. 1995]. The relations of the model form the hypertext connections between parts of the text. For instance, the online

documentation system that will be described in Chapter 7 could be enhanced through the explicit definition of the various gas and energy flows in the gas system. These flows would then serve as convenient guides when navigating through the web formed by the interactions in the system.

5.3 EXPLAINING DESIGN KNOWLEDGE

Rasmussen [1985, 1986] suggested that means-end models could be used for explanation. Any item in the model is given a purpose by its relations to upper levels. Similarly, the implementation of a function or a goal can be understood by tracing the relations towards lower levels. Figure 28 illustrates this point.

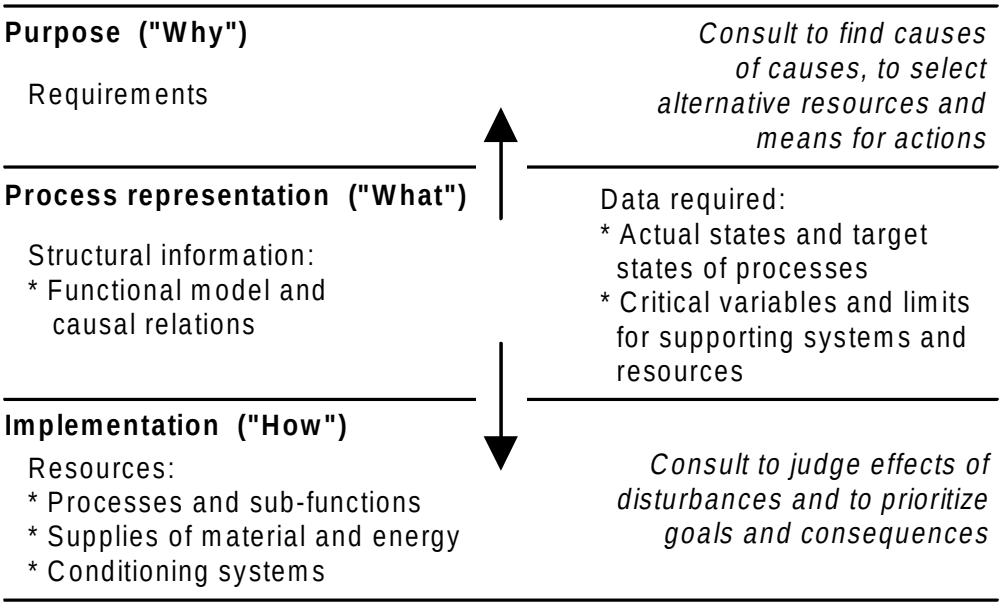

Figure 28. Neighboring levels can be used to answer questions about an item [Rasmussen 1986].

We used the P&ID environment with our extensions to prototype the explanatory possibilities. The work is described in detail in Paper V, and we summarise the main points here. The prototype called “Justifier” can answer the following kinds of questions:

- What is the purpose of this object?
- How is the value for this attribute obtained?
- Justify this design rule.

♦ How is this object implemented?

5.3.1 Deriving explanations through relations

We view design knowledge explanation as navigation in means-ends models, following Rasmussen's suggestions. We distinguish two kinds of explanations concerning a model item: (1) *explicit* explanations that are given by referring to design choices, and (2) *derived* explanations that are inherited through relations between items. [15]

Explicit explanations concern design rationale, the account that a designer might give when asked to justify a design choice [Lee & Lai 1991]. We have taken a lightweight approach to modelling design rationale: justifications are data structures in the objects they talk about. They can consist of keywords, such as "authority", "standard", "safety", "cost", "preference" that are used to denote the criteria for choosing a certain kind of design. Such keywords can be supplemented by references to documents. The structures may also contain freeform text.

```
[] [index] [update] Component FEEDWATER_PUMP.1
Component FEEDWATER_PUMP.S627 in Model HFWS1
Class: FEEDWATER_PUMP in Library HFWS
Assembly: FEEDWATER_PUMP_SYSTEM
Parts: ADJUSTING_DEVICE   ENGINE_DEVICE   INSTRUMENT

HYDRAULIC_POWER
Comment: Hydraulic power efficiency [kW].
Default: <<not specified>>
Design Rule:
(! SELF HYDRAULIC_POWER
 (DOCS
   (RATIONALE
     (TEXT
       "The hydraulic power of the pump is the pressure differ
ence over the pump divided by the operation flow rate volume.")
      (TEXTBOOK "Fundamentals of hydraulic engineering, p.999")
)
   (PURPOSE (TEXT "To calculate the hydraulic power of the pum
p."))))
  (*
   (/ (:? MY PRESSURE_DIFFERENCE)
      (:? MY
          OPERATION_FLOW_RATE_VOLUME_NORMAL))
   1000))
Valueclass: (NUMBER)
Value: <<not specified>>
```

Figure 29. Justifications in a design rule.

[15] "Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information upon it."– Samuel Johnson, in Boswell's *Life of Johnson.*

These justifications are implemented as Lisp structures stored in the devices, goals, and functions. Depending on their position they refer to the whole object or to its attributes, or to the rule that was used to compute a value to the attribute (see Figure 29 for an example). The justifications are generic structures in the sense that their meaning changes with the position but their form remains the same.

If an item is not explicitly justified, a justification can be *derived* by traversing the network formed by the relations. Figure 30 illustrates this process. The search follows the whole-part and means-end relations, with the latter having precedence. When a superpart is found that is related to a function, the function's purpose is recursively searched. The search ends when an explicit justification is found. Bisantz and Vicente (1994) produce explanations from means-end models with similar mechanisms.

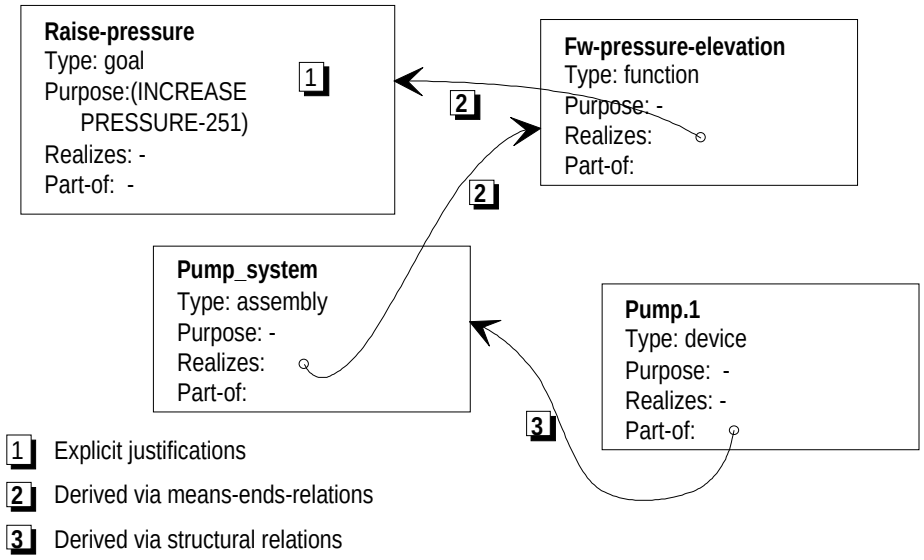

Figure 30. Derivation of explanations through relations.

An example of the explanations with Justifier is seen at the bottom of the window in Figure 31. The user interface is modest, using the basic windowing facilities available in KEE and Design++. The queries for justification can be made either via object menus or Lisp commands. The justifications are displayed as narrative text or as relation graphs. Further queries can be made by pointing to the objects displayed in the graph.

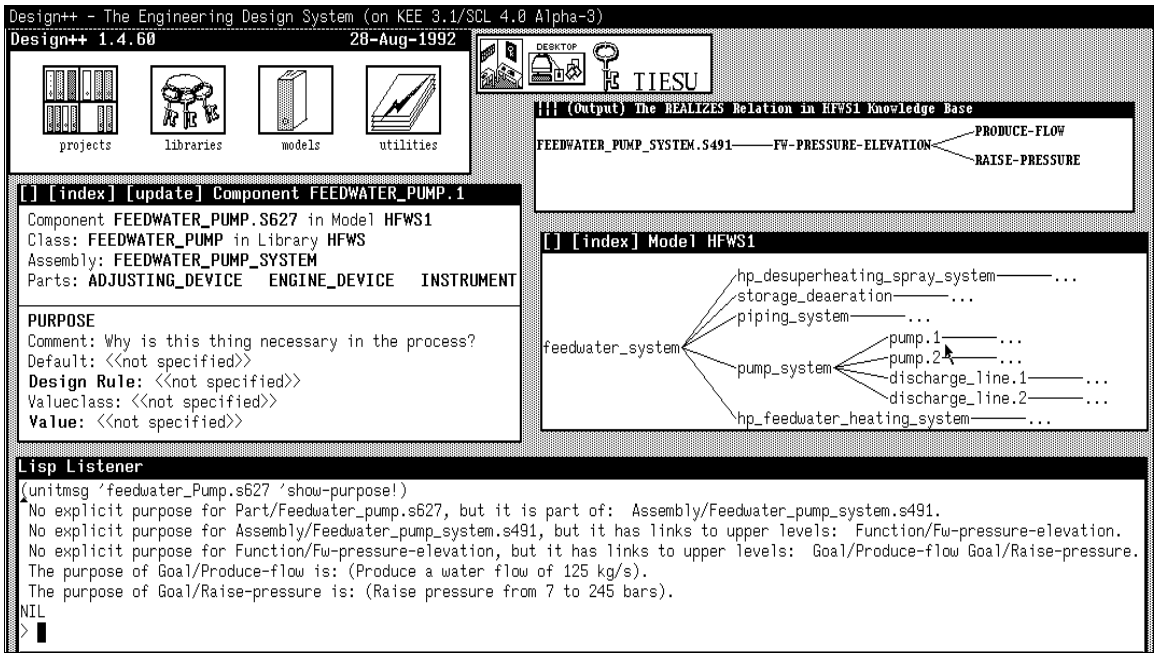

Figure 31. An example of the explanations with the Justifier prototype.

5.3.2 Justifiable objects

In addition to modelling facilities, we have extended the Design++ tool with the notion of *justifiable objects*. They are objects, classes, or instances which can reside at any level of abstraction or decomposition, and can offer explanations of themselves in the manner described above. They are a class of explainable objects defined in Chapter 4.1.

Questions are asked by sending the message JUSTIFY! to the object. The object answers by returning a structured representation of the justification. It can also print this structure in English-like text. The justifiable objects are defined with a superclass JUSTIFIABLES that defines the functionality. The defined methods and properties are then inherited by all of the Design++ model objects at all abstraction levels.

5.4 EXPERIENCES WITH THE JUSTIFIER PROTOTYPE

The Justifier prototype shows how design knowledge can be explained from MFM models that have been constructed at the time of design. Although the prototype has only been tested in a design tool, the mechanisms will work in automation systems that can store object models in their databases. For the greatest benefit, the maintenance staff should receive explanations from the databases they use for maintenance.

Our simple model of design rationale, the keywords, is not sufficient to model real design processes. More elaborate models would be needed to capture the iterative dependencies inherent in design [Lee & Lai 1991, Gruber & Russell 1992]. However, to study explanations based on means-end models, the keywords scheme was sufficient.

In Justifier we presume that an object shares its semantics with the superparts, which is often the case. A component normally assumes a purpose from the system it belongs to. In the example of Figure 30, a pump gets its purpose from the pump system.

When a single device participates in multiple systems or functions – as is often the case – this assumption causes trouble. There are several possible paths for deriving a purpose. A proportional valve, for example, may serve to adjust gas flow or to isolate gas circuits, depending on the way it is used. As a part of a gas line, the former purpose dominates; as a part of the protection circuit, the latter. It is not clear which justification should be chosen.

The problem gets worse when the derivation traverses several levels. At each level, the paths multiply, potentially leading to myriads of possible purposes for a single device. While storing large numbers of objects and relations is quite feasible with today's systems, the explanatory search in the networks would certainly create computational troubles. Moreover, the order in which the various links are traversed affects the explanatory results and the complexity of the search.

These problems caused us to seek ways of managing this complexity. We propose to use the contribution models described in Section 4.5 for this aim. However, with the model we studied, the problem of multiple paths was not a significant one. Only top level devices and subsystems had defined functions and in our model multiple purposes were largely absent. Many functions were explicitly justified, so the searches ended early.

The problem can be further lessened through the interactive nature of explanation. In many cases it is sufficient to proceed a few links up the hierarchy and then stop. At this point the partial paths can be shown to the user, who then selects the most promising path to follow for further explanations. This becomes convenient if the user interface is based on hypertext, as we will propose in Chapter 7.

The results we got from the prototype can be summarised:

- The way of deriving explanations over the means-end relations can be effectively used for explanation.
- More elaborate models of design rationale could be used to give a more meaningful understanding of the design choices.
- The inherent complexity of the search over many-to-many relations needs to be tackled. Contribution models are one way of directing the search.

- The interactive nature of explanations allows us to show intermediate results as pointers to more detail. This helps to lessen the complexity of the search.

6 EXPLANATION AS DIAGNOSIS

This chapter is devoted to realising the Behavioural diagnosis task of the framework to cover an important subclass of automation: the logic. Figure 32 shows how behavioural diagnosis offers explanations about automation. The Document navigation task is also touched upon, as the explanations are shown in the form of hypermedia. A more thorough treatment of the mechanisms to realise document navigation will be found in Chapter 7.

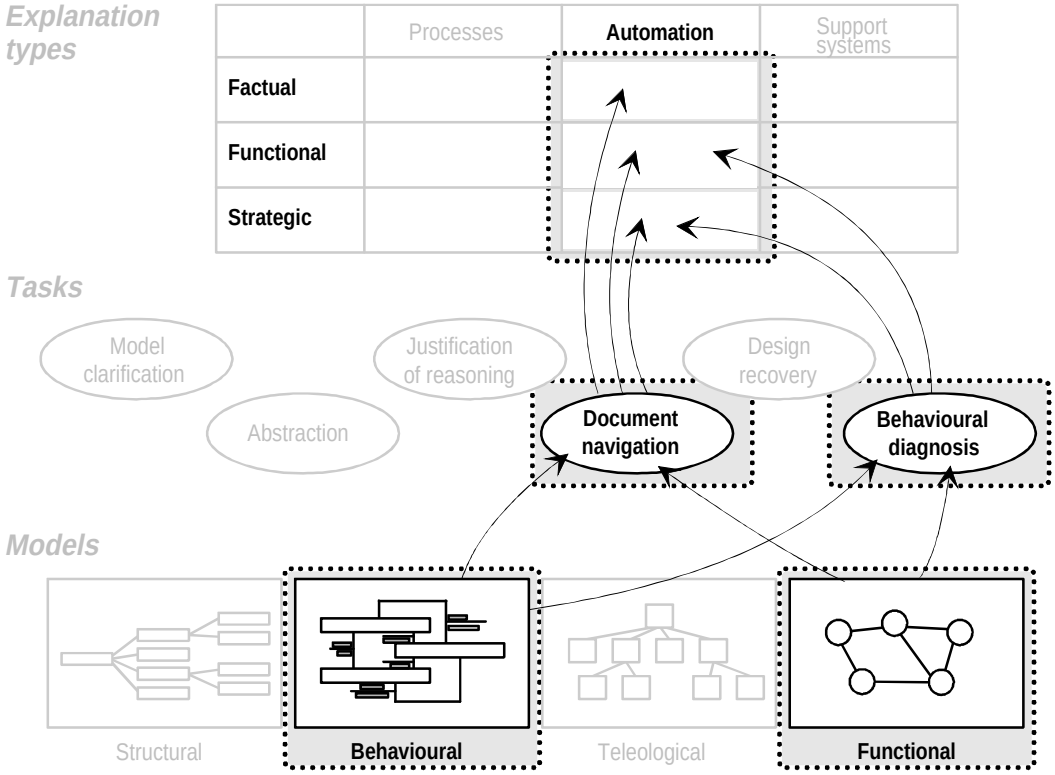

Figure 32. This chapter applies the behavioural diagnosis and document navigation tasks into explaining automation logic.

Section 6.1 gives a brief summary of the project background. In Section 6.2, we outline a four-level model of logic explanation in plants. The model is based on multilevel information abstraction. We then study in Section 6.3 the main features of the Lepo C++ prototype. The concept of explainable objects from Chapter 4 is a central feature in the prototype. Management of feedback and complexity issues are covered in Section 6.4. Section 6.5 draws some conclusions on the work, and some future research possibilities are pointed out in Section 6.6.

6.1 BACKGROUND

Paper VIII documents our work in diagnosing the behaviour of automation logic. The problems that the plant staff have with logic have been outlined in Chapter 2 of this thesis. The main problems can be summarised as follows:

- Automation logic is designed as black boxes (invisible to users).
- Yet the users need to decode that logic by hand in the case of problems.
- This decoding is mentally demanding and too slow in critical situations.
- The design choices behind the logic are invisible, impeding modifications.
- The supporting documentation must be searched manually.

We propose to attack these problems through offering knowledge-based help to the users. Our approach concentrates on explaining the behaviour of logic.

Paper VIII describes our experiences with the Lisp-based prototype called Lepo (shorthand for Logic Explanation for Process Operators) that we constructed to demonstrate the possibilities of explanation. Since the paper was written, we have made advances in the research to cover more industrial cases, management of feedback, and coupling to automation. We have built a new prototype with C++, also called Lepo, and analysed industrial cases with it. The prototype extends the work described in the paper and joins together many of the explanatory techniques presented in this thesis.

The research for the Lepo C++ prototype has been carried out in a joint research project with industrial partners. Examples drawn from the partner's systems were used as the reference cases. Paper winders, paper coaters, pallet conveyors, and hot steel rolling drives have been studied. Logic code, documentation, examples of problems, and users' opinions were collected together with data that was logged from the PLC's. The logic in the cases has been implemented with Siemens S5 PLC's and the Damatic XD automation system from Valmet.

6.2 LEVELS OF LOGIC EXPLANATION

We identify three main classes of explanations on automation logic:

A. Explaining the meaning of logic
B. Explaining the behaviour of logic
C. Speculative help

These classes correspond to the three classes of explanatory knowledge identified in the framework of Chapter 3. Figure 33 shows examples of these question classes with the case of a feedwater pump interlock circuit. The ways of answering the questions have been discussed in more detail in Paper VIII.

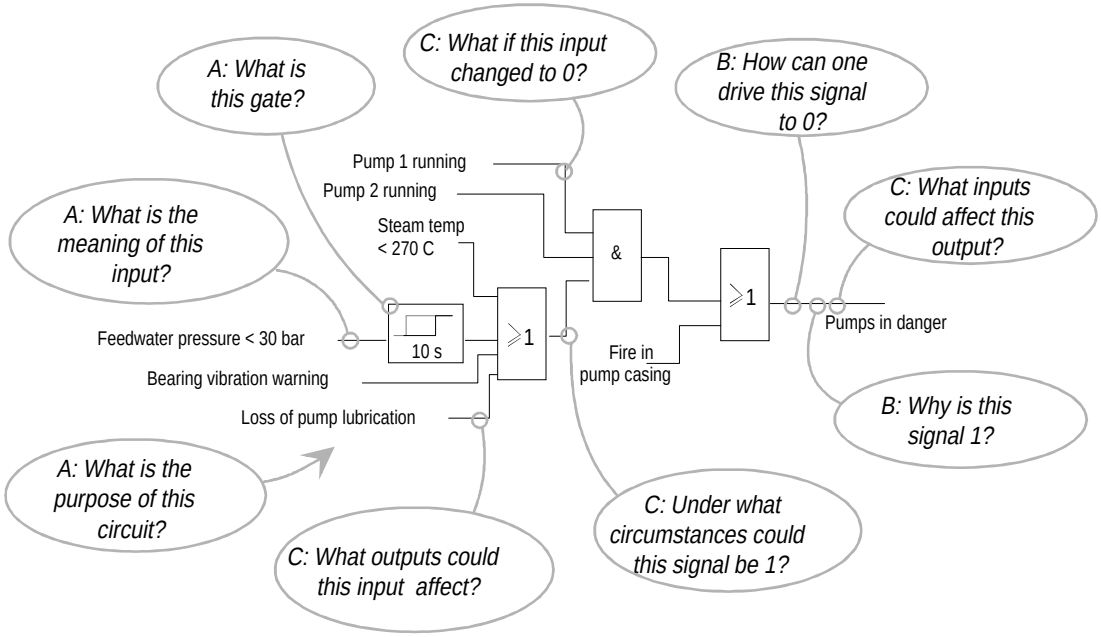

Figure 33. Possible questions on a logic circuit.

The explanations that Lepo offers can have multiple uses in plants. The various users – operators, maintainers, and designers – all have needs at different levels due to their backgrounds and the nature of their assignments. Figure 34 shows the four levels of logic explanation that can be identified in plants. The three classes of questions may be answered at any of these levels. In practice, the lowest levels are available through PLC programming environments, whereas the highest levels are offered through support systems.

The abstraction level rises from the machines towards the users. The *program level* is closest to the actual operation of a PLC or a DCS. On this level the logic is seen as a series of assembly code instructions executed on a serial CPU.

The *functional level* views the logic as a network of nodes and gates. The execution of the logic is quasi-parallel, functional mapping of inputs to outputs. The state machines are formed implicitly by elements that have memory (for instance set-reset gates) and by feedback loops in the network.

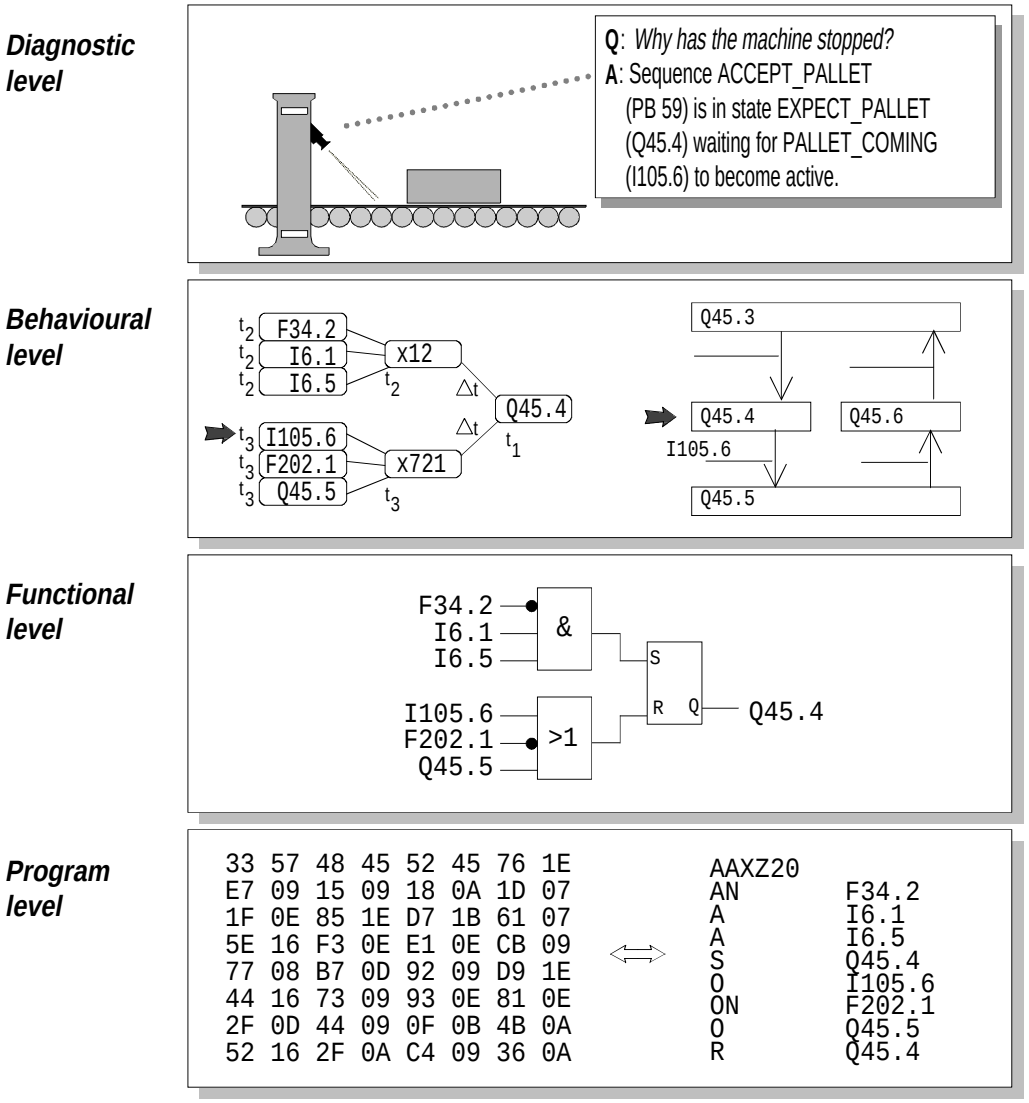

Figure 34. The four levels of logic explanation.

The *behavioural level* views combinatorial logic as a time interval-based dependency network, where the changes in inputs propagate to the outputs after the time delays inherent in the network. The state machines formed by the network are abstracted into state-transition diagrams, with identified states and conditions.

The *diagnostic level* is more concerned with the logic's environment and operational context. The logic itself has been abstracted into causal explanations between inputs and outputs. The explanations give direct advice on the reasons for events and states, and can suggest possible actions to take. The advice is restricted to the inputs and outputs of the logic, since the analysis of the logic can not directly diagnose the plant around it. However, navigation from the logic into related documentation can help us to understand the explanations in a wider context.

Operators in plants are usually not concerned with the internal operation of the logic. When problems appear, they are eager to call the maintainers to trace the reasons. This causes much work for the maintenance staff. What is worse, they are not available during night shifts. With Lepo, the operators could do more themselves. They can remain on the diagnostic level for their disturbance management tasks, where prompt actions are often needed.

The maintenance staff in plants may need to work on all levels simultaneously. Their tasks frequently involve diagnosing faults in or through the logic and making changes to it. For this they have to reconstruct chains of events. With Lepo, they can look at the logic at all levels of abstraction, and study the past events. They may even simulate the effects of possible actions on the logic.

The designers who create the logic traditionally work either on the program level or on the logic level. With Lepo coupled to their design tools, the level of the design would rise. Designers could analyse the behaviour and the dependencies of the logic to see if their design works in the way it was meant to. Moreover, the design knowledge on the purposes and design choices could be communicated to other users and designers through Lepo, if the design tools allowed recording the knowledge.

6.3 THE LEPO C++ PROTOTYPE

Due to the industrial orientation, one of the central requirements for the second Lepo prototype has been its embeddability into the control systems. We chose to use C++ as the main implementation language, running under Windows NT or Unix. Algorithms and user interfaces have been built for both of these operating systems. With the increase of industrial personal computers and Unix-based solutions in plants, these choices minimise the embedding effort.

Figure 35 describes the architecture of the Lepo C++ prototype. A filter decodes the logic code used in the automation from binary or source forms into an intermediate file format. This format is then transformed into a generic object-based model that is used to simulate and explain the logic. Events are explained with the help of data histories. They store logged data from the automation, which is then read into the objects. References to supporting documentation are included in the objects, when available from design systems. The resulting explanations are shown to the users as context-sensitive hypertext, linked with logic diagrams and other documentation.

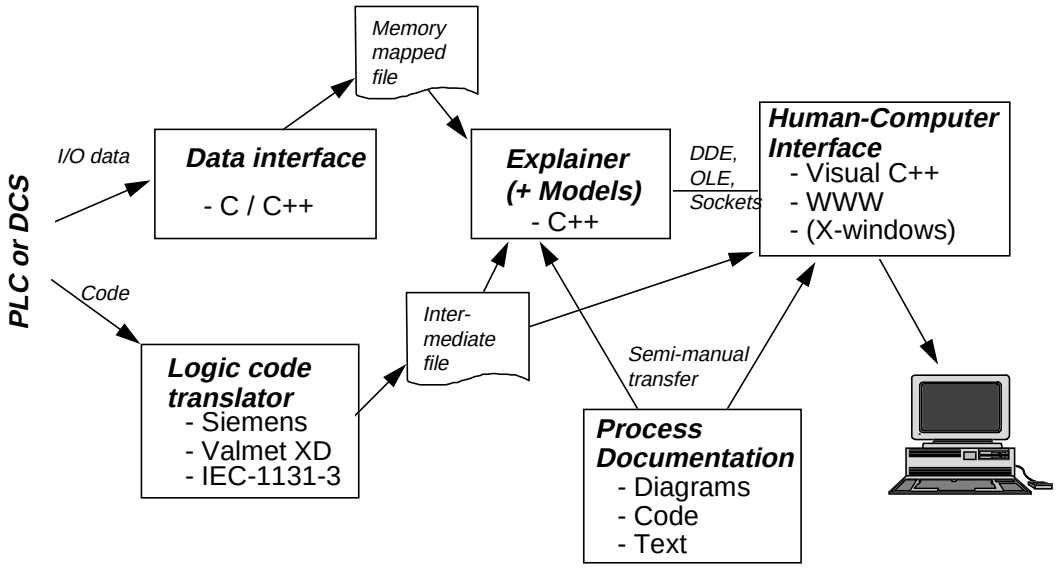

Figure 35. The architecture of the Lepo C++ prototype.

Lepo covers the four levels of explanation. The filters recover code structures from the program level for use at the upper levels. The simulator works at the functional level. The explanation algorithms work by selecting pieces from the logic level to the behavioural level. The hypertext interface of Lepo, together with the various diagram displays and documentation, gives some help to the users at the diagnostic level (although diagnosis of the environment is not within the scope of Lepo).

6.3.1 Filters

The filters assume the main responsibility for processing the raw automation logic code into more easily analysable object structures. A partial filter has been built for the Unix environment as an extension to Valmet's design tools. It extracts logic gates from Damatic XD automation diagrams, leaving out such non-logic constructs as motor control blocks, for example. These diversified constructs must be explained using other means, since they are not meaningfully representable as binary logic networks. Symbolic boolean logic expressions, in which logic functions are often defined in XD applications, are converted into equivalent nodes and gates.

A more ambitious filter has been implemented by Valmet for the Step5 language used in Siemens S5 PLC's. The filter directly decodes the binary S5 code, in principle allowing the changes to the code to be automatically updated in the object structures of Lepo. With today's tools, the binary code is the only representation that is guaranteed to correspond to the program the PLC is executing. If the source code is not stored in the PLC itself, there is always a chance for version mismatches. This problem is avoided by directly decoding the binary code.

The IEC 1131 standard [IEC 1993, Lewis 1996] may become an important format for the transfer of logic definitions between design systems and PLC's in the next decade. It defines five different languages for representing logic: FB/function blocks, LD/ladder diagrams, SFC/structured function charts (state machines), IL/instruction lists, and ST/structured text (a Pascal-like language). Of these, FB, LD, SFC and IL source codes will map relatively easily into our representations. The ST language will require more extensions to the current formats and object representations. Future work will include implementation of filters for these IEC languages.

6.3.2 Data histories

Lepo uses history data available from the process to analyse past events. If the state of a logic variable in the automation is known, it can be directly used for explanation. This speeds up simulation of the past events. However, some simulation will remain necessary, since in most cases it will not be possible to keep complete traces of the execution of a PLC. The execution cycles may be too fast or memory sizes insufficient for exhaustive logging. In any case, some of the internal variables will remain hidden and must be recovered through simulation.

In our case, Valmet has implemented a data logging system for Lepo. A personal computer records the inputs, outputs and internal variables of an S5 PLC through an Ethernet connection. The achieved logging rate with the current system is to the order of 50 ms for a total of 1500 bits. The data histories are kept for half an hour and then discarded or summarised. These histories are available for Lepo through a memory-mapped file scheme.

6.3.3 Object models

The logic is modelled within Lepo as directed cyclic graphs that are implemented as C++ objects. There are currently seven classes for modelling logic nodes and eleven classes for the different gates. More classes can be conveniently added as needed for new kinds of logic, since the interfaces for each object have been defined with extensions in mind. The object model is generic; it can accommodate new logic solutions by only defining new subclasses. It is thus usable for any system for which a filter can be constructed.

The incoming code from the filters is analysed and translated into a network of instances of the node and gate classes. The execution order and the module hierarchies of the PLC are also modelled, since they define the block interconnections in the network. In other respects, the network is a purely abstract representation of the logic, readily analysable through graph algorithms.

Some aspects of the original code (for instance such instructions as conditional backward jumps) are not easily representable with the object

structures. Our current object model covers the most important binary S5 functions (AND, OR, NOT, registers, and timers).

The data histories obtained from the PLCs are stored in the nodes when needed for the explanation. An incremental representation is used for the data that records only the changes to logic variables. This is convenient for logic where changes are relatively infrequent compared to the logging rate, as is the case with much of the automation logic in plants today.

All objects in the network have been modelled as explainable objects in the manner presented in Chapter 4. Every node and gate can offer explanations concerning its values or functions. For instance, a node can explain its meaning, and a gate can explain how it propagates logic values from inputs to outputs.

6.3.4 Simulation

Lepo has a simulator that mimics the binary operations of a serial PLC. It guides the generation of explanations by inferring signal states at time points under study. The simulation is based on three-valued logic, with *true*, *false*, and *unknown* states. An unknown signal state may result in a network where there has not been logged data for a given time period and the state cannot be derived from other signals.

In principle, the simulation starts from a known input state vector and propagates the changes towards the outputs of the network. The serial scanning nature of a PLC can be accurately mirrored through the interconnections of the object model. In the case of S5 code, the evaluation order of code blocks and segments are known through the program organisation. The time dependencies caused by delays and gates with memory are covered. However, the simulator does not currently cope with non-binary logic constructs (such as PID controllers). The simulation therefore only reflects the internal operation of a PLC to a certain degree.

For explanation, the simulation does not need to cover the whole logic network but only the part that affects the output. In those cases the evaluation works recursively backwards from the output towards the inputs. When explicit values are found for the inputs, the values are propagated forward to the outputs. Thus Lepo's simulator is mostly used in a backtracking mode.

The gates have an active role in the simulation. At each stage of the forward propagation, the gates are asked to calculate their output values based on the given inputs. Propagation then backtracks the recursive route until the value in the output can be determined.

6.3.5 Explanation

Explanations implemented so far cover several of the question types in Figure 33:

Q1: “What is this gate?”
Q2: “What is the meaning of this node?”
Q3: “What inputs caused this output state?”
Q4: “What inputs caused this output event?”
Q5: “What inputs may affect this output?”

Questions Q1 and Q2 are answered through explainable objects by listing comment data obtained from the logic code or other design systems, or by referring to supporting documentation.

The analysis of behavioural questions Q3 and Q4 is carried out by first simulating the logic and then asking each object in the circuit to explain its output values. The explanation proceeds recursively from outputs toward inputs. When direct input variables are encountered, they are output as hypertext objects to the users. When other parts of the logic are referenced, they are listed as possible continuation questions in the form of hypertext links. Figure 36 shows an example of an explanation for question type Q3.

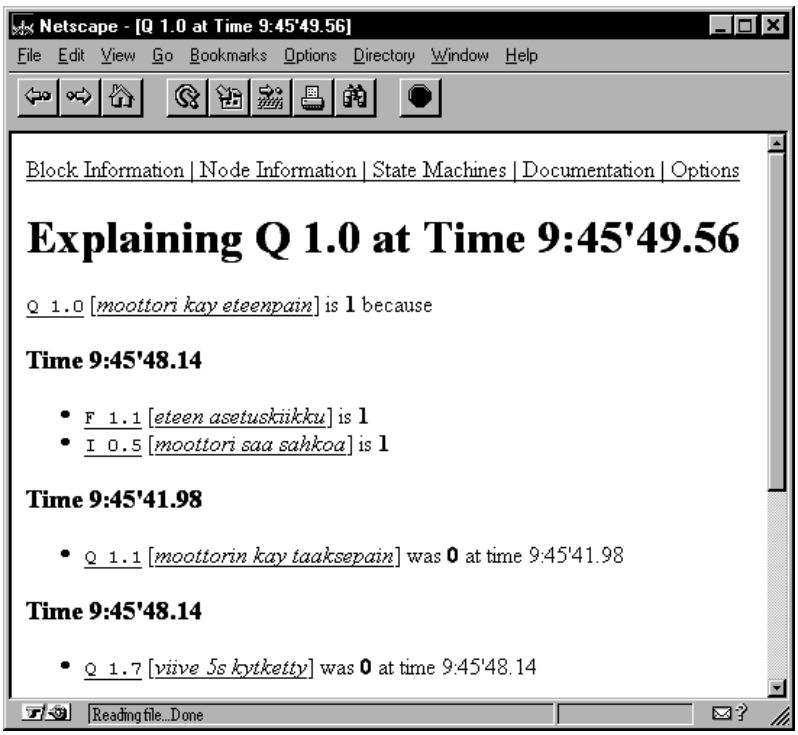

Figure 36. The hypertext interface of the Lepo C++ prototype with an explanation on the state of an output (type Q3).

Question Q5 clarifies the complex interdependencies in the logic. The resulting listing acts as a static output-input cross-reference that helps to understanding dependencies. This point is elaborated in Section 6.4.

6.3.6 Human-computer interfaces

Figure 36 shows an example of the human-computer interfaces of the prototype. Explanations are shown as hypertext through a WWW [Berners-Lee et al. 1994] browser connected to Lepo. The explainable objects produce hypertext based on their instance variable values. The irrelevant details can be diminished and important features emphasised by typographic means. The level of detail may be adjusted depending on the user's preferences.

Continuation questions can be asked through pointing at the hyperlinks. When a link is pointed at, Lepo generates possible continuation questions based on the context. Each link stores the context under which it was created, thus allowing context-based interaction. The context representation here is straightforward, consisting of the object name, question type, and the relevant time interval. Context refinement is performed through a WWW page, where the user may modify the question parameters.

We use the standard navigational devices of WWW. With the browser's "back" and "history" functions it is always possible to come back to a previous explanation for further questions.

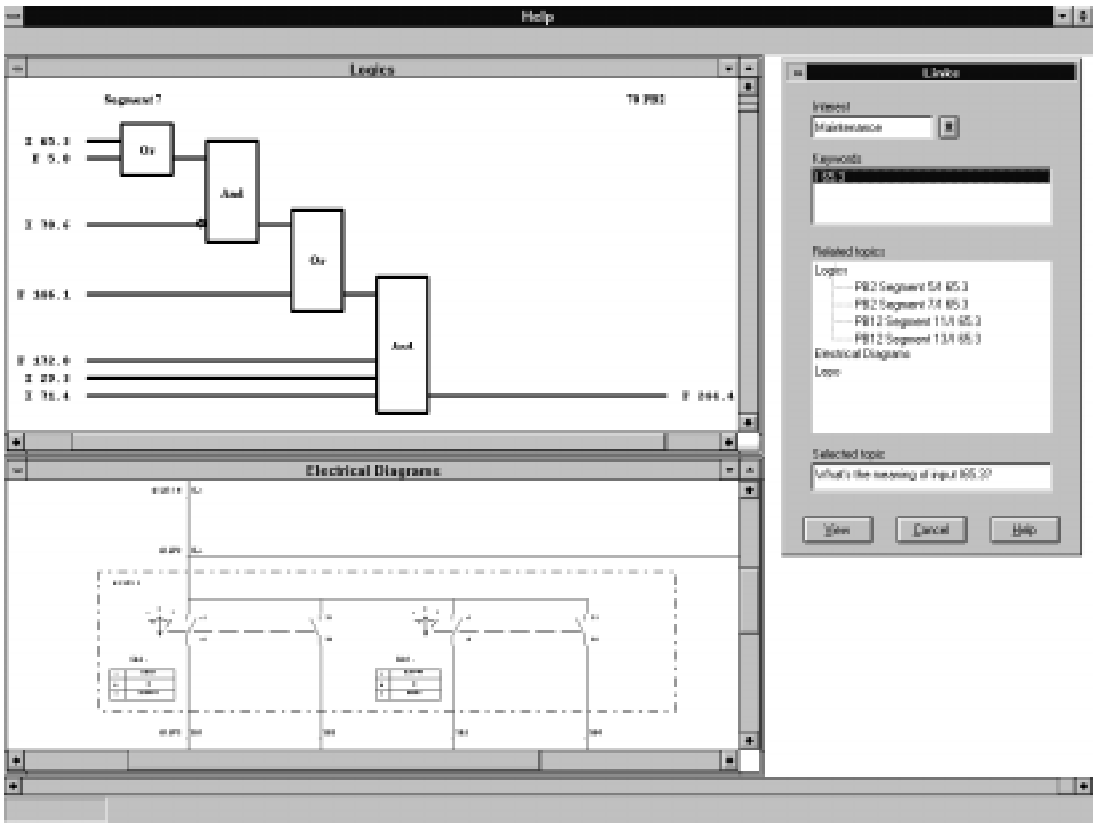

Figure 37. A diagram viewer and navigator built by Valmet.

In industry, graphical explanations would normally be favoured to textual ones. A viewer implemented by Valmet shows the logic diagrams with

signal values changing in real or simulated time (Figure 37). Lepo may show a diagram with the viewer as an answer to a question. The objects in the diagrams are active in the same way as the hyperlinks in the WWW pages: pointing to the diagram gives access to further explanations through Lepo.

CAD viewers may be used to show related information, such as wiring or electrical diagrams. In the figure, there is a navigation panel that shows the possible ways that a single item, for instance a signal, can be visualised with Lepo and the viewers. In addition, there are facilities to invoke Lepo via DDE (Dynamic Data Exchange) from external applications, for example supervisory software packages.

These interfaces can be made available over networks. We have constructed a WWW server version of Lepo that allows explanations to be accessed over the Internet with a simple WWW browser. Diagrams are made available through a viewer implemented in the Java language that allows showing dynamic logic displays on the WWW pages created by Lepo.

WWW access to Lepo can facilitate remote diagnostics. PLC programmers or plant experts can study the operation of the plant from remote locations, complementing the experience of the plant staff. Plant documentation can be linked in the explanations, once it is converted into electronic formats. With its context-sensitive dialogue, this interface to explanations demonstrates a flexible way to access plant information.

6.4 COMPLEXITY ISSUES

The search in the graph from outputs of a PLC to its inputs has potential for a combinatorial explosion. This implies that an output can potentially depend on a large number of nodes, maybe all of the I/O space. When these nodes in turn depend on others, the space becomes even larger. Explanation may therefore fail to find a satisfactory answer before the search space becomes too large to manage. This also directly affects the speed of the explanations, which can be crucial for operator assistance.

On the basis of the initial analyses of existing logic codes, we believe that this combinatorial complexity is not a problem in the majority of situations. We have also found ways of decreasing the effects of the complexity. The discussion in the rest of this section offers some insight into the significance of the problem, although it is not meant to be a rigorous proof.

Question Q5 can be used to analyse the complexity of logic explanation. The question is answered through a statical analysis of the logic graph we call the *fan-in-analysis*. Effectively, it is a breadth-first traversal of the graph. The fan-in analysis forms the *fan-in tree* of the graph, where the concerned output is the root of the tree and the inputs affecting it are the

leaves. A very large tree may be created, where a significant fraction of all the systems' inputs is included.

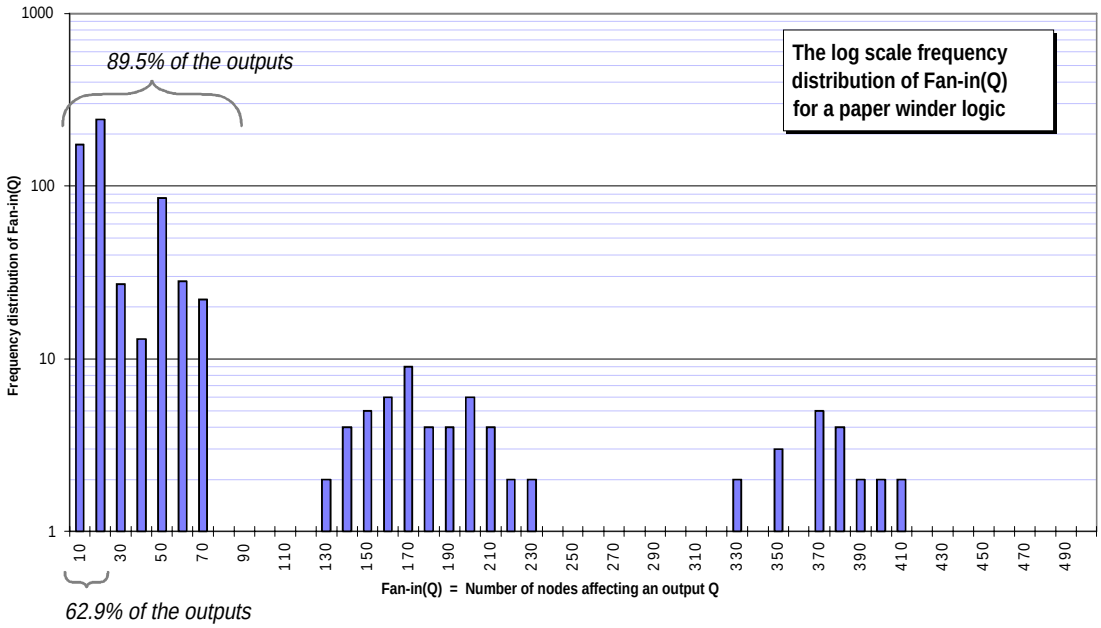

Figure 38. The frequency distribution of the number of nodes that potentially affect an output node in a paper winder logic.

Figure 38 shows the case of the paper winder PLC software where the total number of logic nodes exceeds one thousand. Here the fan-in analysis has backtracked a single execution cycle of PLC logic from each node that has been used as an output. The figure shows the frequency distribution of the number of nodes that the analysis finds per output. The nodes that are included in the number form the potential search space for the explanations.

In this case, the majority (nearly 90%) of the outputs are only dependent on less than 80 nodes, which is 5% of the total of 1500 nodes. Approximately 2/3 of the outputs depend on less than 20 nodes (3% of the total). These numbers correspond to the number of individual nodes that the user would see as an answer for a single output, if all potential nodes are included. In practice many of these nodes (such as intermediate variables) will not be shown, cutting the average number (44 nodes) roughly in half.

This study shows that, even with industrial examples of realistic scale, only a fraction of the logic outputs will require large searches. Thus the typical size of the search spaces will not be prohibitive in terms of computational complexity. However, the space will grow due to recursive feedback when several PLC execution cycles are considered. The rate of growth is a topic for further study.

There are ways to limit the search for the more complex cases. For questions Q3 and Q4, the search space becomes pruned during the traversal. With many gates, some inputs can be considered as "don't-cares" for the situation at hand. The sub-networks behind those inputs are then ignored in the search. This heuristic pruning of the search space can be necessary, since not all the potential inputs can be included in the explanation. We currently use three kinds of heuristics:

1. Such inputs for a gate that cannot possibly affect the output state of the gate are ignored. Classic examples of this are "0" inputs of an OR gate with a "1" in the output.
2. Such inputs for a gate are investigated first that changed their state just before a change in the outputs. This heuristic assumes that simultaneity implies a causal relation (which will not hold in all cases).
3. Outputs of a state machine are considered terminal nodes for the search (assuming that the state offers sufficient explanatory information to the users).

Other possible heuristics can be defined. If semantic information about nodes is available, it can be used to rank the relative importance of each node in the course of search. Currently this ranking is implicit in the algorithms, but could be made explicit.

The use of heuristic pruning may affect the reliability of Lepo's results. By excluding portions of the network from analysis we may ignore important inputs that nevertheless have affected the output in some way. Therefore one cannot guarantee the completeness of Lepo's explanations where heuristics are used. Some way of limiting the search will be needed however, as the whole logic cannot be exhaustively analysed without it.

We consider an approximate solution to be more valuable than no solution at all. Moreover, according to our studies documented in Paper VIII, humans appear to use heuristic pruning effectively when they analyse logic. We are trying to mimic their way of finding approximate solutions with the heuristics of Lepo.

One non-heuristic way to limit the complexity is to control the depth of the search. In Lepo, explanations do not traverse segment borders unless told to do so. This serves to limit the amount of explanatory detail. Hyperlinks allow continuation questions where more detail is desired. In this way, the problem of excess detail can be partially solved by trusting some of the explanation planning tasks to the user. However, with this limitation it would no longer be possible to give fully automatic explanations that go all the way to the relevant inputs.

Minimisation [Brayton et al. 1987, Biswas 1986, Bostick et al. 1987, Malik et al. 1988, Yang & Ciesielski 1991, Hong & Muroga 1991] of the logic network limits the size and thus the complexity of the analyses. We

have implemented some straightforward minimisation algorithms that can reduce the network into a factored form [Brayton 1987]. The utility of this approach has not yet, however, been formally analysed in the case of automation logic.

In Lepo, none of the strategies for managing complexity – heuristic pruning, importance ranking, minimisation, and user-driven search – is sufficient alone. Several strategies must be combined for solving industrial problems. Moreover, their relative utility depends on the preferences of the users. We plan to allow the users to control the strategies by means of priority settings. In this way, a variety of problem-solving styles with automation logic could be accommodated.

6.5 EXPERIENCES WITH THE PROTOTYPE

While the development of the Lepo C++ prototype is still going on, industrial problems on a realistic scale can already be analysed with it. We have deliberately concentrated on an important subclass of industrial automation, the logic. As we point out in Paper VIII, a major part of the problems on plants is related to logic. We therefore expect that knowledge-based help, as offered by the prototype, will considerably help plant operation once it is available through the automation.

The concept of logic explanation has raised industrial interest, as evidenced by the number of companies participating in the research. One company will extend the prototype into a commercial product, and others are keen on having the solutions available in their plants. We can therefore judge that the research has produced results that have practical value.

The basic architectural choices of the first Lepo prototype presented in Paper VIII have been shown to work in practice. The object representations, the graph algorithms, and hypertext interfaces have since been much extended, but the concepts behind them have remained the same.

The serial nature of the PLC creates a number of problems. We simulate the behaviour at a parallel network level, but the actual PLC works as a serial CPU. Certain PLC constructs, such as conditional jumps and function calls, are not always easy to map to a graph representation. In those cases, the same physical PLC code may be entered from multiple locations – something that logic graphs do not describe conveniently.

The problem is complicated by the fact that single memory locations may be overwritten in multiple places in the code. Unfortunately, data logging only captures the last result, not the intermediate ones. If these memory locations are used as inputs in other parts of the code, it becomes necessary to know the execution order of the program. This order may be dynamic. It is not possible in all cases to construct a static graph of the program that captures the execution dynamics. Instead, other ways of modelling the dynamics will be needed.

We have modelled some of the dynamics through finite state machines. Typical sequence implementations in the logic can be semi-automatically analysed and converted to state machines. Currently Lepo can reverse engineer state machines from PLC code that has been written according to a predefined coding style. The conditions of the state machines are implemented in combinatorial logic, falling back to graphable constructs. Likewise the actions associated with states become inputs to combinatorial networks, returning to the realm of the current Lepo. Such questions as "What is going on?", "What is the purpose of this step/transition?", "How did we reach this step?", and "How do we reach this step from the current one?" are being implemented.

Another possible approach to managing dynamics is to simulate the execution of a PLC at the lowest possible CPU level and keeping a trace of the events. Behaviour can then unambiguously be explained from the trace. Unfortunately, keeping such a trace will require much memory and processing capacity. This possibility should be investigated, though.

In most logic implementation languages there are some constructs that cannot be analysed through binary logic. Much of the automation used in plants, for instance control loops, is non-binary by its nature and thus not within the scope of Lepo. This implies that in many logic programs there will be parts whose behaviour cannot be analysed. However, major parts of the code that is causing problems in plants are analysable by Lepo, as the discussion in Section 6.4 shows.

The very existence of Lepo could be questioned. It could be argued that the languages used in logic programming today are simply too primitive. Instead of explaining such low level details, the programs should be written at higher level languages that are more understandable by humans. In fact, with suitable design systems, programs could be constructed automatically based on given constraints [Bonfatti et al. 1995]. Models for explanations would result as a side product. The historical developments in software engineering suggest the same conclusion: the low level assembly languages are very little used today, and higher-level languages dominate. Application generators automatically produce important portions of software. Similar trends can be seen in all engineering, and we expect the same developments to happen with PLC programming tools.

The argument of raising the level of languages is challenged by the large body of low level PLC software in existence. Whole populations of PLC programmers are not very familiar with higher level languages, not to mention the issue of tool support. The programs that are in use today are likely to be used well into the next century. Accepting this situation, it makes sense to invest in automatic ways of improving the understandability of today's automation. Lepo shows some ways of achieving this aim.

A language in itself is seldom self-explicable, however. We believe that most PLC constructs will remain on relatively low levels for some time to come. Even in other areas of software engineering there are few high level

languages that approach the (imprecise, vague) human way of expressing problems. It could be argued that since computer programs, by definition, are meant to interpreted by computers, they will always contain implementation artifacts that make it harder for humans to understand their intended function. It seems to us that once higher level languages start being used, systems similar to Lepo may still be needed to explain them. Such systems could function as very high level debuggers for the languages.

To summarise the discussion in this section, we conclude that Lepo offers clear advantages as compared to manual diagnosis:

- The user avoids the laborious, error-prone mental "simulation" of the logic.
- Lepo improves navigation in the logic code.
- Interdependencies within the code become more visible.
- Lepo can find a relatively small set of "important" dependencies.
- The whole diagnosis becomes faster, when not fully automatic.

There are possible disadvantages as well:

- Heuristic search methods are not fully reliable, although humans may assume so. They may trust Lepo too much.
- An assisting mode is not enough; fully automated answers may be needed that require no human intervention.
- The operators want to know what happened in the *process*, not which input was the cause of the output. With its current data sources, Lepo necessarily goes no further than the I/O limit of a PLC.
- Data logging may not be always possible. The explanations would then become much more speculative in their nature.
- PLC codes invariably contain non-binary constructs. Therefore Lepo can never explain all of the PLC through binary diagnoses.
- The plant staff may not accept Lepo.
- The staff could lose some of their problem solving skills if they rely on Lepo.

Some answers to these problems are suggested in the next section as pending research topics. It should be noted, though, that many of these disadvantages apply to human diagnosers as well.

6.6 FUTURE RESEARCH

Lepo should be accessible from the automation displays that the plant staff use in their everyday work. Some users may prefer to view the explanations visually in the mimic diagrams, while others prefer text displays. Graphical interfaces can be constructed for explanation, or existing interfaces enhanced to support explanations. Via its connections Lepo can be linked to the automation in existing plants.

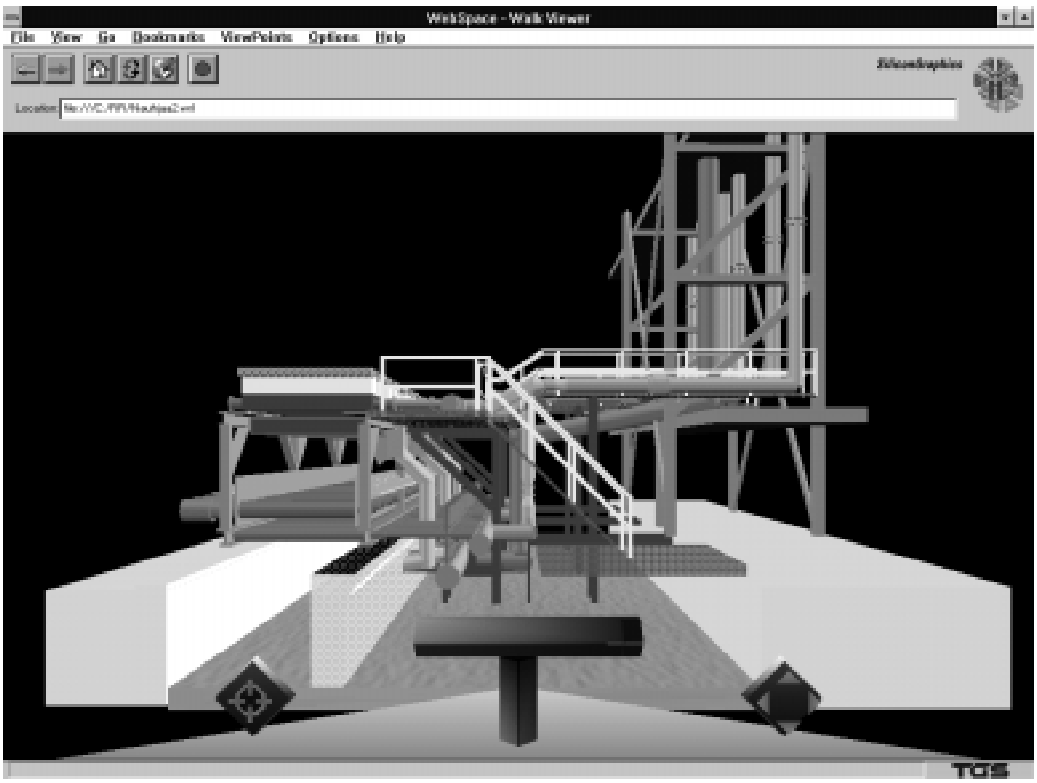

Figure 39. A three-dimensional interface to Lepo.

As the capabilities of the display devices continue to grow, it will become possible to find new interaction mechanisms. Animation and three-dimensional graphics will be increasingly used. We have built a virtual reality model of a steel plant, through which explanations concerning the plant items can be accessed (Figure 39). The model will be used to study alternative navigation methods for plant information. Since Lepo has been integrated to the WWW, it can be accessed through browsers just like any other form of on-line documentation. This line of development may ultimately lead to a point where all applications appear as electronic documents.

In the domain of digital electronics design, there is a lot of research into analysis of logic circuits [Brayton et al. 1987, Biswas 1986, Bostick et al.

1987, Malik et al. 1988, Yang & Ciesielski 1991, Hong & Muroga 1991]. More extensive minimising techniques could be used to cut the explanatory search, replacing the network under study with an equivalent, but smaller network. Such enhancements should be studied in more depth. The experiences of modelling and analysing digital circuits with VHDL [IEEE 1076 1987] could likewise be applied to Lepo.

Minimisation is a way of transforming the logic into a new form for easier analysis. It could also be used for presentation. Transforming the logic into another visual form (for instance, ladder logic into function blocks) could make certain things easier to see for humans. However, it is not clear whether the plant staff would find multiple representations of the same logic more disturbing than illuminating.

The explanation algorithms of Lepo carry out a heuristic search in the network created by the logic nodes and gates. Currently the search procedures are implicit in the explanation subroutines. They could be formalised as a parametrised search. The same applies to the simulation algorithms in Lepo. In essence, they perform time-based constraint propagation in dependency networks, although the approach is not explicitly formulated as such. The formal methods of temporal logic [Perkins & Austin 1990], interval arithmetic, and algorithm analysis should all be studied in relation to the problem space set by Lepo.

As we noted, the users may not understand the imprecision of the heuristic search. They could trust Lepo too much. Therefore Lepo should also be able to *give warnings and rationale of its own reasoning processes*, to disillusion the users. This is an important topic for further research. Many of the explanatory techniques detailed elsewhere in this thesis would be directly applicable. For instance, the heuristic search algorithms could be formulated as explainable tasks.

What-if questions are not yet available to the users, although Lepo internally uses simulation for similar purposes. Simulation essentially answers the question “What would happen with these input values?”. This question can be implemented as soon as suitable interfaces are available for defining the input states. The more involved question, “How can we drive this output to a given state?” would require more exhaustive simulation with minimisations.

The explanations of the purposes of logic have not yet been implemented. The object structures of Lepo allow straightforward implementation of means-end models, once such models are available concerning automation logic. Logic design environments should be extended to allow for recording such knowledge. The future PLC programming tools are likely to be based on the IEC 1131 standard. Common languages will then facilitate better information transfer between the tools [Lewis 1996]. In such an environment it would be possible to enhance the tools with knowledge caption.

Logic explanation stops at the limits of the automation. The connections of the inputs and outputs to the process are not modelled. However, changes that the logic makes in its outputs affect the process and, eventually, show up as changes in the inputs. This feedback via the process (or people) has been outside the scope of Lepo. To achieve full understanding of the behaviour of the plant, all its parts should be analysable together. This implies that to fully explain logic, the related processes and people should be explained as well, which leads us to a large area of diagnostic problems. Here we have focused on a more manageable subset of this problem domain.

One possibility for extending Lepo outside the borders of a PLC would be to model the processes and the people as discrete (binary) machines. This simplification would then render them analysable by Lepo. Although such explanations would be coarse at best, they could be of help in some cases. [Paanasalo 1996]

It is possible that the plant staff would actually resist the use of Lepo. It could be seen to taking away a mentally laborious yet rewarding task. The issue of trust could also work against the system. Moreover, relying on Lepo the staff may lose some skills in problem solving that would be badly needed where Lepo is no longer usable. Studies need to be carried out on these aspects in plants with the actual use of the systems.

More filters could be built for popular logic languages. The IEC 1131 standard, if successful, could lead to a widely used interchange format. It would then suffice to create a filter in Lepo for the five languages of this standard[16], and a wide range of different logic programs would become available for Lepo. However, the high-level languages of the standard (particularly ST, Structured Text) are no longer easily analysable by the binary logic networks of Lepo. New modelling and analysis methods will be needed to explain high-level code.

It is very awkward to recover the operation of the PLC from the outside. Instead, the PLC itself should contain Lepo's functionality. Since many PLCs today are implemented with powerful microprocessors, this would be quite feasible. The PLC would then not only execute the logic but also explain it when required. Likewise the automation systems could explain themselves. In essence, this would mean expansion of the concept of *explainable objects* to cover *explainable systems*.

These topics form interesting possibilities for future research. Our main aim in this thesis has been in studying practical approaches to explanation that can be implemented in industrial settings. Judging from the reception of the Lepo C++ prototype, we have succeeded.

---

[16] Or just to the IL (Instruction List) language that is very close to the intermediate language used in Lepo.

7 EXPLANATION AS INTERACTION

We propose to use *hypertext*[17] as the main presentational medium for explanations. It also conveniently serves as the presentational metaphor and the dialogue model. In this light, explanations can be considered to be *active documentation* that can present itself in ways suitable to the situation. In our systems, the situations have been explicitly modelled as objects, termed "*contexts*".

We have applied this principle in the VICE and Lepo prototypes and studied the effects of hypertext with on-line documentation. As shown in Figure 40, the emphasis in this chapter is on the "Document navigation" task of the framework.

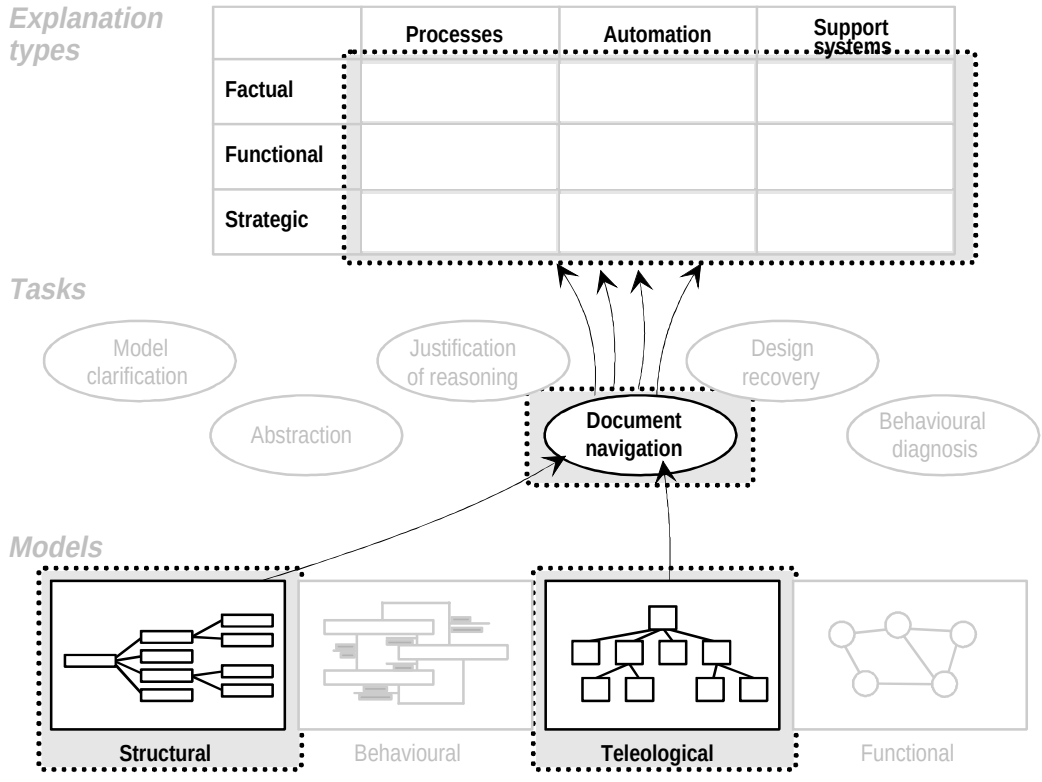

Figure 40. This chapter deals with the Document navigation task.

[17] We do not distinguish *hypermedia* from hypertext in this treatment; we consider hypertext from a general viewpoint that encompasses technical documents containing other media as well as text. Our explanations are mostly text-based, although some graphics are used as well.

In this chapter, Section 7.1 first introduces the concept of hypertext applied in explanation. Section 7.2 then expands the concept with advanced access mechanisms and evaluates the results with a prototype. Section 7.3 expands the palette further with the notion of context. Section 7.4 brings these concepts together into a dialogue model. Section 7.5 discusses the results.

7.1 HYPERTEXT

Hypertext [Conklin 1987] has become a widely accepted way of presenting information even in industry [Årzen 1991]. During the time that the research for this thesis was being carried out, hypertext has advanced from isolated systems to a seemingly ubiquitous presence of the World-Wide Web (WWW) [Berners-Lee et al. 1994]. At the same time, the creation of documents has more or less made the transition from paper to an electronic form, which helps the content creation of hypertext. The progress towards structured documentation, most notably SGML [ISO 8879 1986, Smith 1992], has further increased the potential of hypertext for presenting information.

7.1.1 Dynamic documents

We distinguish two kinds of electronic documentation: *static* and *dynamic*. Static documentation, for instance a book on a CD-ROM, is constructed with the publisher's machines well before its use. A dynamic document, for instance a database report, may be created only just before showing it to users [Gruber et al. 1995]. In reality, there is no clear division between the two kinds, and many electronic documents are a combination of both. In fact there is an infinite range of possibilities between these two extremes. They only differ in the creation time that can be well in advance or just before publishing.[18]

Explanations are dynamically created documents. They are constructed in a system after the user asks a question. The outline and possibly some of the contents of the explanation may have been written in advance by the system's designer, but the rest is crafted from the knowledge contained in the system.

The degree of prefabrication depends on the system. Usually it is not feasible to predict all different kinds of explanatory situations beforehand

[18] This makes an interesting analogy with computer programs, where subroutine calls may be linked beforehand at compile time (early binding), or just at the time of the call (late binding). Moreover, automated "compilation" of documents from smaller units is becoming possible in today's tools. In many respects, electronic documentation is becoming just another form of software, with similar problems.

[Swartout 1983]. This may be one of the reasons why help systems for traditional software are not very satisfactory [Schneiderman 1987]: they fail to understand the user's situation and can only present prefabricated material that may not be what the user is looking for. Explanation could enhance, if not replace, the current help systems.

7.1.2 Hypertext for explanation

Hypertext is a convenient medium for explanation in knowledge-based systems. Many knowledge representations, particularly those put forward in this thesis, can be considered as items of data with links between these items – in other words, hypertext. There is a clear analogy between semantic networks and hypertext [Conklin 1987, Tyrväinen 1994].

Norman [1983] notes that a common conceptual model should be shared among interface designers and users. Hypertext facilitates this sharing, because it is sufficiently close to the representations we propose for design systems and knowledge bases in plants.

Hypertext is highly suitable for the question-answer dialogue mode that is common in explanation [Cawsey 1992]. Questions can be specified by pointing at hypertext links, and answers can be shown as new hypertext [Moore & Swartout 1990]. Unnecessary explanatory details can be hidden, yet remain accessible through links at any time they are needed. Supporting documentation can also be referenced from within the explanations as links.

Explanation subsystems can be viewed of as sophisticated interactive report generators. Some of them (including the one in VICE) may function in a batch mode where standardised reports are created through the explanation mechanisms. These reports can then be imported to text processing systems or databases for inclusion into other documentation, or for storage. The stored reports could also be reused as a basis for further explanations at a later time.

7.1.3 Content generation

We have applied hypertext for explanations in the VICE and Lepo prototypes, described in more detail in Chapters 4 and 6. In both prototypes, the explanations are assembled from the system's models and shown in hypertext windows. For VICE, we had to construct our own dynamic hypertext languages and browsers in Lisp, as these were not available in the environment we used. In the case of Lepo, we could benefit from the existing infrastructure of the WWW. Explanations in Lepo are created in HTML (Hypertext Markup Language) code [Berners-Lee et al. 1994] and shown on Netscape, a commercial WWW browser.

The VICE explainer uses a *template* approach [Rubinoff 1985] for content generation. The templates contain the text and graphics outlines of the explanations with placeholders for values that are dynamically retrieved

from the objects. Conforming to the explainable object principle, the models' objects contain the templates and fill them in. For Lepo, the same mechanism has been foreseen but not yet implemented. Currently Lepo's templates are implicit in the object's methods.

A novel feature of the VICE hypertext language was the *procedures*. The templates could contain Lisp code that was evaluated in the context of the explanation. This construct allowed us to dynamically generate the contents of the fields, a feature which proved valuable with VICE. Recent WWW developments, for instance Java [Campione & Walrath 1997], serve similar purposes by allowing code execution in the document viewer's environment.

7.2 ENHANCED HYPERTEXT

Plain hypertext suffers from problems that originate from its web-like structure. A paper document can be read or browsed from cover to cover, whereas hypertext is more suited to random paths guided by curiosity. With a traditional document the reader retains a sense of orientation. It is easy to tell from the physical form of a book whether a passage belongs to the beginning or the end of the book. With hypertext it is easy to get lost in the web of links [Conklin 1987]. It is difficult to anticipate the size of the document, since it is hidden behind the links. Therefore various indexes, guides, and search facilities are needed with complex hyperdocuments [Rivlin et al. 1994].

A related problem is that a reader, faced with multiple links, will have difficulty in deciding which link to follow [Conklin 1987]. Links in traditional hypertext documents offer little information about the target they point to. Equipping the links with more semantic information about the target could help the users navigate in a document [Halasz 1988].

Dynamically constructed hypertext suffers from the same problems as static hypertext. Explanations based on hypertext should therefore take into account those problems. We now briefly look into solutions that can enhance static hypertext. The techniques, once they become usable, can then begin to be used for explanations in the manner that will be described in Section 7.4.1.

7.2.1 Model-based hypertext

We have constructed enhanced access mechanisms in hypertext in the Cicero project [Kaarela et al. 1995]. The central idea is to use an *information model* to structure the documentation. The model contains semantic information about the elements of the document, which can help navigation. We have built a hypertext linker that merges the information model with the hypertext documents. The documents have been marked up

with identifiers that correspond to the items of the model. The resulting document contains hypertext links that match the semantic links in the model.

The other main feature of the approach is to use the screens of an automation system as a *graphical map* to the documentation. Every item on a screen (for instance, a pump) can potentially have a link to the documentation related to that item. In plants, it is essential to have direct access from the automation to the electronic documents, especially in hurried situations.

We used the gas system discussed in Chapter 4 as a prototype. The existing means-end model was reused as the information model. The documentation of the gas system totalled some 150 pages [Peach 1994]. We used publicly available HTML tools to translate the formatted document into hypertext, organised into a hierarchically and linearly linked set of WWW pages [Takalo 1995].

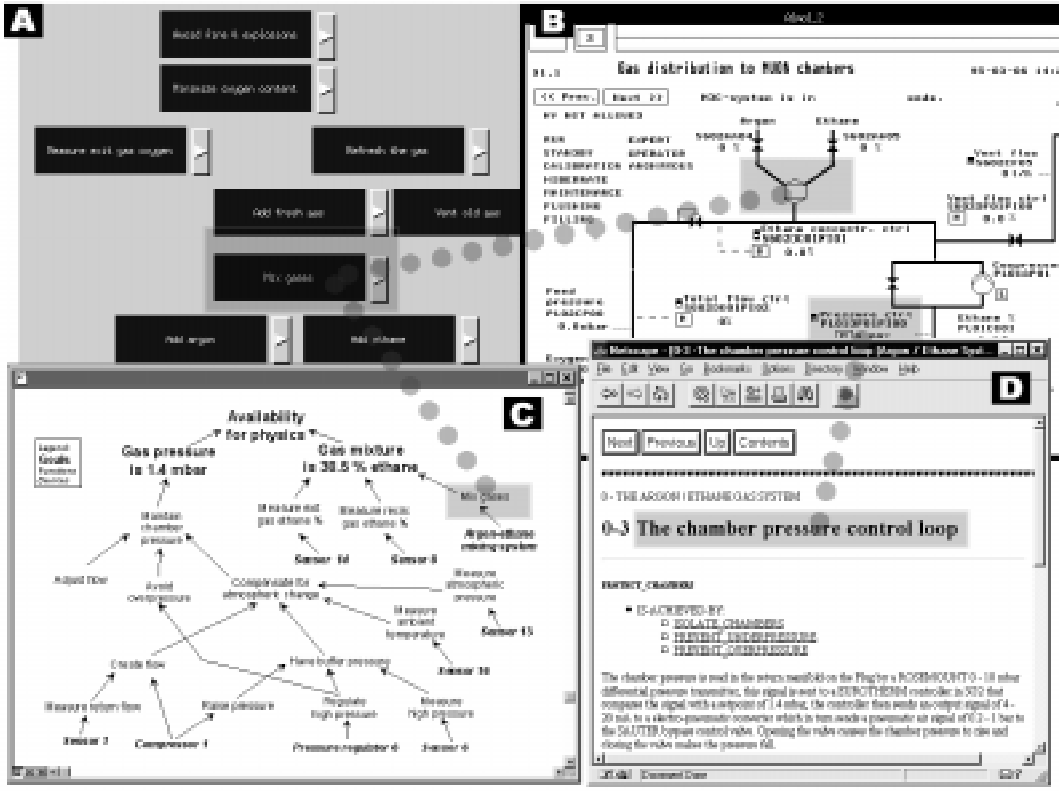

Figure 41. The enhanced on-line version of the gas system documentation.

Figure 41 shows an example screen from the documentation system prototype. We enhanced the resulting hypertext with the means-end information. Parts of the document and the corresponding model items (C) were linked through unique identifiers. The result was a document with semantic means-end links corresponding to the model. The documentation was accessed through a publicly available WWW browser (D).

We connected the browser to the XD automation system (B) and the XIS process information system (A), both from Valmet Automation, Inc. Our partners in the Cicero project had built displays for the gas system with these tools. The connection enabled navigation from the automation system or process information system displays into the relevant points in the documentation, as depicted with the dotted lines in the figure. In addition, the WAIS indexing engine [Stein 1991] was used to allow keyword-based searches in the documentation.

7.2.2 A usability study

We evaluated the on-line documentation prototype in a usability test, where twelve subjects carried out a set of problem solving tasks related to the gas system. We collected data on the time required to complete the tasks and evaluated the correctness of the results.

The aim of the test was to find out how model-based enhancements could benefit in browsing the documents. The point was to reuse the same models that had been constructed earlier for explanation, although explanation generation was not used for this limited test. The test has been documented in depth by Takalo [1995], with the main features summarised here.

The subjects were divided into three groups of four people. Group A used a paper version of the documentation, group B used a plain hypertext version, and group C used hypertext with the enhanced access mechanisms[19]. All had access to process flow diagrams. The tasks were crafted to resemble actual problem solving situations. Four of the nine tasks required the use of means-end knowledge, for instance: *"To make the conditions suitable for physics, the gas pressure in the chambers is maintained at a steady 1.4 mbar over the ambient pressure. How is this control carried out?"*. This knowledge was available indirectly in the texts and directly in the models accessible through the process information system screens.

We hypothesised that it would be easier to find answers to these questions through the enhanced access mechanisms. We received mostly satisfactory answers to the questions. Upon subjective inspection, it appeared that the subjects who had access to the enhanced on-line documentation did indeed find the best answers in the shortest time. The numeric estimates in Table 7 show that group C had the best performance. This corresponds with the increase in the power of the available tools. The details for the calculations are available in [Takalo 1995].

[19] The subjects in were already familiar with the WWW and had some prior experience of the various search facilities available there.

Table 7. The performance estimates of each group.

Performance per group		
A. Paper	*B. Hypertext*	*C. Enhanced*
6.9	***6.8***	***8.2***

Our observations of the subjects made us conclude that, in this case, the boolean string search facility was the most significant utility of enhanced on-line documentation. It was fluently used by all subjects who had access to it. In addition, the navigation from automation screens to relevant points in the document seemed beneficial for the questions that required means-end information. Likewise the links inside the document that reflected the means-end model did give some guidance to find purposes and implementations.

Many of the subjects complained of being lost in the hypertext, unaware of the context, and thus not knowing whether there would be more valuable information elsewhere. Group C also complained that the search facility overwhelmed them with a multitude of possible choices. Nevertheless, they considered the facility indispensable. An active table of contents that shows the reader's position was felt to be desirable. The static index in our document seemed insufficient to guide the users through the information.

We take these complaints as evidence of the disorientation that people suffer with hypertext [Conklin 1987]. We suspect that people in group A could employ the familiar browsing styles, instant access, and location memorising that paper documents allow. However, tasks that required finding dependencies did lead more often to failures with paper users.

It appears that static on-line documents, even enhanced with models, are not sufficient. The systems should guide the users more through participating in a *dialogue* with the users. The following sections show ways of maintaining explanatory dialogue.

7.3 CONTEXT

Knowledge is little;
to know the right context is much;
to know the right spot is everything.

– Hugo von Hofmannsthal, The Book of Friends, 1922

Explanation based on hypertext has similar shortcomings to hypertext documents. Users must retain their orientation; they must know what is being discussed. In VICE, we alleviate these problems by keeping an *explanation history* that allows the user to come back to previous

explanations. All explanations are objects that are retained during diagnosis sessions. They could be browsed afterwards, and could result in continuation questions just like newly generated explanations. In Lepo the same facilities are available in the WWW browser.

A more fundamental solution to the problems is to maintain a *dialogue* with the user [Kobsa & Wahlster 1989, Cawsey 1992]. When explanations are viewed as conversations, many orientation problems are solved through dialogue maintenance techniques.

We base dialogue on a simple model of *context*. In our definition, context is the subset of the knowledge contained in a support system that is the topic of an ongoing conversation. The knowledge is viewed as a context space that is traversed in a conversation. In explanatory dialogue, the topic and the context are constantly changing (Figure 42). This implies that in all phases of a conversation, the context must be maintained to give understandable explanations.

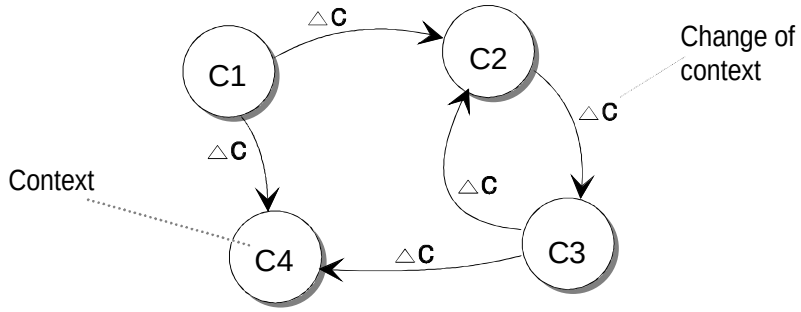

Figure 42. In explanatory dialogue, the context is always changing.

Figure 43 illustrates our context model. All classes of knowledge in a support system – objects, tasks, their properties, and relations, among others – are potential degrees of freedom that form a multidimensional context space. Contexts in VICE are defined as tuples of (object, property, value, machine component, task, abstraction level). An example of a context instance could then be (SEAL-RUB, LIKELIHOOD, ******, BEARING-L1, HISTORY-ANALYSIS, ******). Some dimensions of the context can be unspecified (here shown with ******), which reflects the specificity of the discussion. Initially the topic is not well defined and the context is unspecified. During a conversation the topic gets gradually refined and thus the context gets more clearly specified.

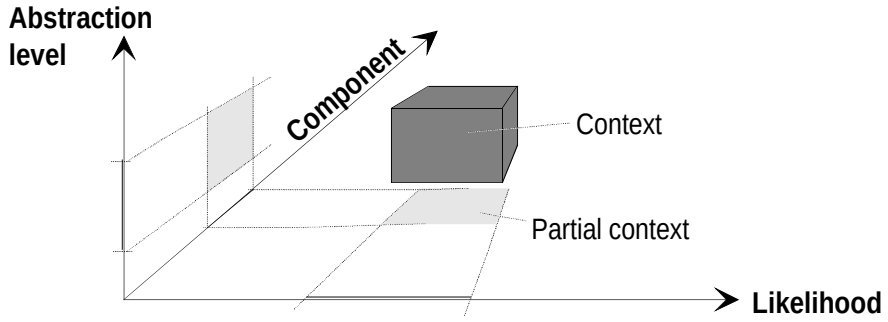

Figure 43. Context is the subset of all the possible knowledge in a system.

7.4 DIALOGUE

We have used the notion of context to help explanatory dialogue in several ways: to understand questions, to search for answers, and to generate the answers' contents. Figure 44 shows the dialogue model we propose for explanation (the model shown here is an expanded version of the one outlined in Paper II). It shows the various phases in the process of creating explanations and the flow of data through the process. The bold arrows show the main explanatory cycle, while the other parts bring additional data to the process.

7.4.1 The explanation creation process

The user triggers the process by asking a question. In hypertext-based explanation, this is accomplished by pointing to the question that is shown as a hyperlink, or by choosing a suitable question from a menu. First the question must be understood (1). For instance, the question "Why" may concern either strategy "Why do you ask this question?" or reasoning "Why do give this answer?". In our case this phase is straightforward, since we model all questions as predefined classes ("types" in the figure). The suitable classes are selected by the menu choice. If no specific question is selected – which is the case with such continuation questions as "More" – all classes are selected. This phase is more involved in systems that use natural language for interaction, for instance in Mycin [Buchanan & Shortliffe 1984], but becomes easy with hypertext [Moore & Swartout 1990].

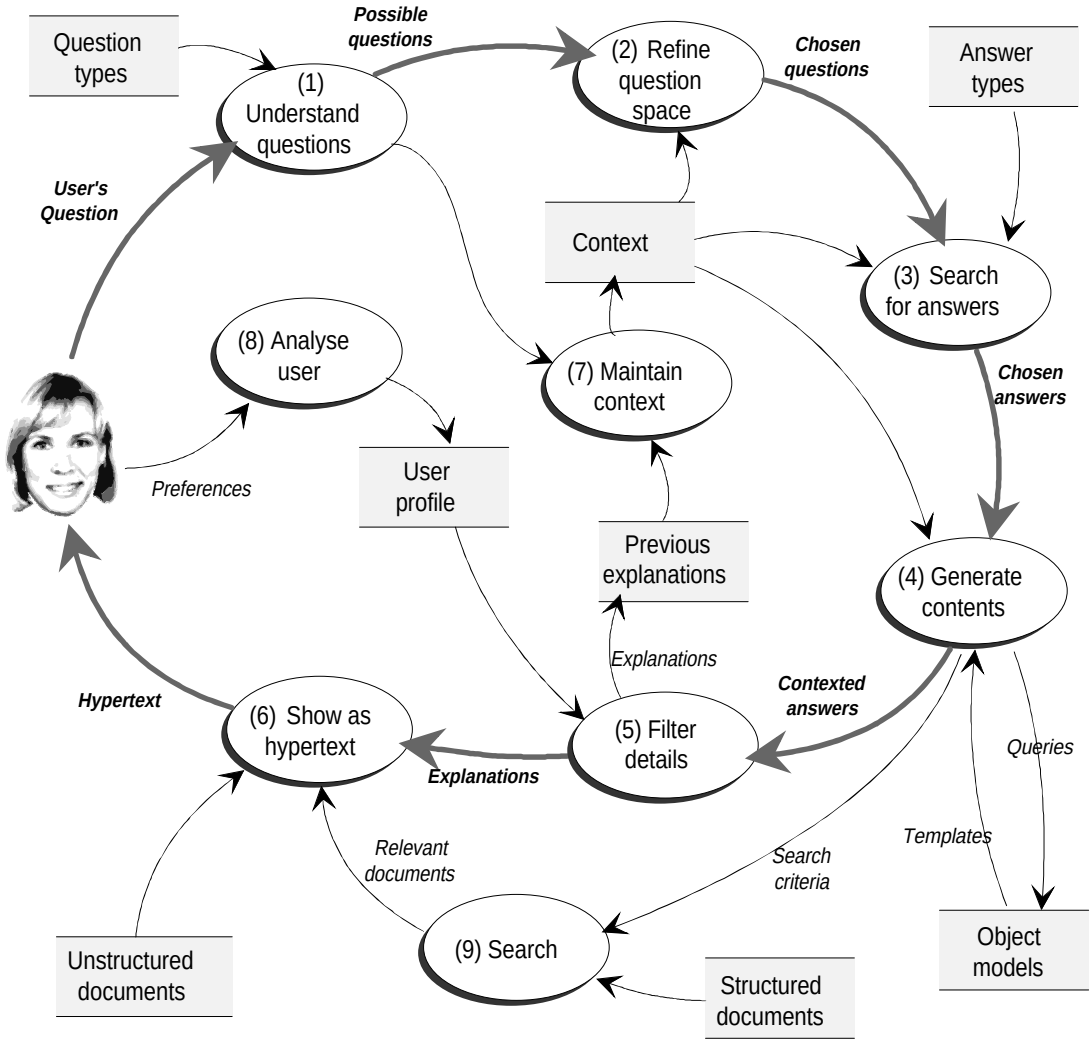

Figure 44. A dialogue model for explanation creation.

Next, the question space must be narrowed down (2), if there are several possible questions. This refinement involves defining the desired context. Each question class has a defined context where it is applicable. If a previous context is available (7), it is used as a starting point. If some dimensions are not specified, the user is asked to refine the context. In practice this in done by listing the possible values for the unspecified dimension in a multiple choice menu. For instance, if the component dimension were unspecified, the user would choose a machine component or a whole subsystem from the menu. Various object hierarchies could be used instead of menus to help the refinement.

Ultimately the refinement results in a single question. The predefined answers are then matched with the question (3). Only those answer classes are chosen whose context corresponds to that of the question. This matching is based on the *match roles* for the dimensions. The roles are attributes of

each dimension, as specified in the questions and answers. A dimension can be GIVEN (must be explicitly listed in the question), EMPTY (must be empty), FIND (must be filled in during answering), or IGNORE (irrelevant – can be anything). These match roles are compared between the questions and answers. The answer classes whose match roles satisfy the conditions are considered to be in the proper context. Those classes are then instantiated as suitable answers.

Next the answer's contents are generated (4). The hypertext templates contained in the answers are filled in with values of the context. Portions of the templates are received from the explainable object models in response to explanation queries. The procedures contained in the templates are then executed for additional contents. The resulting raw explanations are then filtered (5) according to the user profile. This profile is created through analysing (8) the user's experience, background knowledge, beliefs, goals, or other factors. In our case, the profile consists of levels for each knowledge type. A suitable level is selected by the user. If there are details in the answers that are not compatible with the chosen level, they are left out of the explanations.

The resulting explanations are stored with their contexts (7) for later retrieval and shown as hypertext (6). The hypertext templates are translated from an internal representation into a visible form. The objects listed in the templates are visualised as hypertext links. The various typographic details are added. References to other documents can be shown as hypertext links. Structured documents are offered in the order of relevance to the answers, if suitable search (9) criteria have been defined. Unstructured electronic documents are simply referred to through links that have been defined in the templates.

The user completes the cycle by reading the explanation that now consists of hypertext questions and answers. If he/she wants more detail on one of the mentioned topics, pointing to one of the hypertext links (objects, questions, answers) gives a contexted trigger, and the cycle continues.

Not all of the dialogue phases need to be included in all systems. Portions of this general model can be adopted as necessary. For instance, the VICE explainer does not have an explicit user analysis and filtering, or search mechanisms for external documents. In the Lepo explainer, the question-answer pairs are more hardwired, and phases (2) and (3) are straightforward.

7.4.2 The use of context

Context is essential for our dialogue model. It is used for several purposes in the process:

- *Question disambiguation*. The context information stored in links serves as an implicit mechanism for understanding the context of the questions.
- *Explanation planning.* The mapping of questions to answers is based on context. This scheme allows an heuristic search for finding the answers.
- *Content generation.* The context defines the knowledge space that the answers must talk about. Explainable objects get some of their case-specific attributes from context.
- *Search criteria*. An explicit context can be taken as the keyword set for finding documentation that is relevant to the given explanations.

The principle of storing context information in hypertext links, used in the VICE and Lepo prototypes, has recently become popular in the WWW. The dynamic HTML documents created by databases or search engines now routinely represent context information as encoded fields in the links. A good example is the Alta Vista search engine that returns a list of document links in response to queries [Alta Vista 1996]. We consider the popularity of hypertext to be empirical proof of the concept.

The notion of contexted hypertext could be taken beyond links. The hierarchical structure of the hypertext could be used to derive the context from higher levels of the hierarchy towards the bottom levels, and vice versa. Figure 45 illustrates this point. Not only do individual links have a context but all text items can have one, either explicitly given or derived through its position in the tree. Different contexts could be assumed in different parts of the hypertext explanations. The user could then more naturally point to any parts of the text, not just to links. In the VICE system, the graphical bullets before paragraphs each had a context that referred to that paragraph as a whole, whereas links inside the paragraphs had a more limited context. The derivation of context through the hierarchy was not, however, implemented in the system.

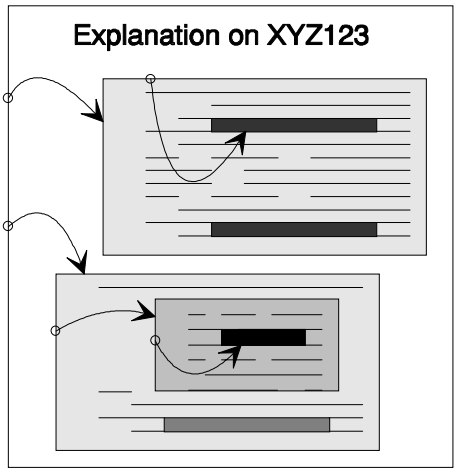

Figure 45. Deriving context in the hierarchical hypertext structure.

7.5 EXPERIENCES WITH HYPERTEXT

In our experience, hypertext has been a useful choice for communicating explanations. The computer literate users of our systems have accepted the notion of hypertext with little hesitation. In the future, as people get more and more familiar with hypertext, it will become an even more valuable medium for explanations.

By way of conclusion about the usability test, we saw that means-end based navigation in online documentation worked as expected. The evaluation of the small data set suggests that enhanced access mechanisms help the operators to find more accurate information in less time. The utility of making means-end knowledge explicit was apparent in questions that used the means-end knowledge contained in the model. The means-end displays were not seen to be very helpful[20], but the means-end links in the document were used.

The realisation of the VICE and Lepo prototypes show that our dialogue model works satisfactorily, although it could be improved. Our subjective evaluation suggests that the users adopt the question-answer interaction style easily. This may be due to the inherent nature of diagnostic explanation: it falls naturally into a expert-novice mode where the novice poses questions to the expert. A tutoring system [Sleeman & Brown 1982] would have to assume a different mode, where the expert presents

[20] One the subjects wanted to dispense with the means-end displays altogether and just have links from automation screens to corresponding documents. It appeared that in general the subjects could have needed more training in the use of the means-end displays, which does not speak for the utility of the concept for casual users.

information and the novice indicates the understanding gained. Even there the same dialogue model would work, but the answer planning and generation would differ.

The template approach to explanation allows some dynamic generation. Instead of completely prefabricated answers, we allow the activation context to influence the answer generation. The active procedures and the assembly of templates from several explainable objects gives a sufficient degree of freedom to achieve dynamic answers. Unfortunately, template-based text generation suffers from consistency and inflexibility problems. As Tanner & Keuneke [1991] note, template-based explanation generation cannot be as flexible as approaches that generate text from deep models [Swartout 1983, Moore & Swartout 1990]. In any case, the templates are directly implementable with today's tools and therefore have a high pragmatic value.

Our context model is not complete. Among other things, it does not account for temporal ordering of dialogue events, nor is it based on any linguistic theory. However, it was directly implementable with the object models we had, and sufficed for the need of referring to parts of the system's knowledge.

The notion of dialogue in our system is somewhat bare. We do not explicitly plan the dialogue; it occurs naturally through the heuristic question/answer matching and the user-driven navigation through contexted hypertext links. Stronger models for dialogue do exist [Cawsey 1992, Moore & Swartout 1990, Swartout et al. 1991, Weisang & Zinser 1992], giving supposedly better results. Also the use of levels for user modelling in our approach is not sufficient. Better user models could further enhance the utility of the explanations [McKeown et al. 1985, Paris 1993]. Filtering of excess details should also be given more study.

Applying these principles into the prototypes described in this thesis gives good targets for further research. It would be particularly interesting to conduct usability studies of the various approaches to explanatory interaction. At this point we must content ourselves to say that our dialogue model was a workable solution for the prototypes presented in this thesis.

8 RELATED RESEARCH

The topic of this thesis, explanation, is on the intermediate ground between several fields of science and technology. There are large amounts of research that are related to our work, particularly in the areas of artificial intelligence, computer science, process engineering, human-machine studies, and cognitive sciences. We survey here the work that either bears similarity to our results or is otherwise parallel to the treatment of explanation in this thesis. With each topic we discuss how the work is related to that presented in this thesis.

Of the sections in this chapter, 8.1 reviews the development of reasoning and modelling mechanisms for explanation. Section 8.2 studies some explanatory support systems for the plant domain, their effect to the user's mental models, and the design knowledge needed. Section 8.3 reflects on the research on knowledge-based analysis of logic. Section 8.4 discusses the research on human-machine interaction from hypertext, dialogue, and user modelling viewpoints.

8.1 APPROACHES TO EXPLANATION

Various approaches to explanation in knowledge-based systems exist in the literature. The emphasis in explanation research has changed over time from reasoning to representation to tailoring to interaction (Figure 46).

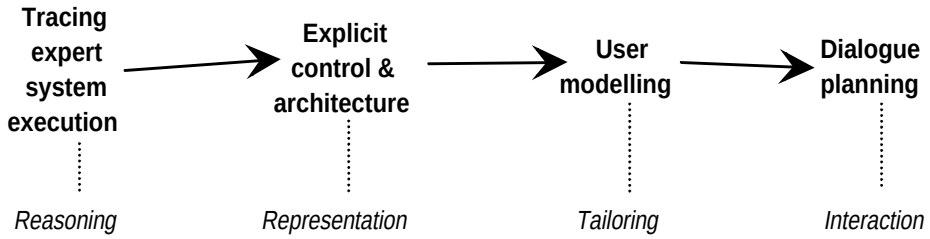

Figure 46. The shifting of emphasis in explanation research.

Historically there has been much interest devoted to the problem of explaining the reasoning of an expert system. The rule-based Mycin system [Buchanan & Shortliffe 1984] allowed two kinds of questions concerning reasoning: "Why" and "How". The questions were answered through tracing the execution of rules and explaining the derivation of results through the rules. These two forms of questions have been very popular in

commercial expert system shells [Wick & Slagle 1989]. However, it has since become apparent that the rules are too detailed to be shown to the users [Jackson 1986]. Other formalisms were needed to raise the level of explanations.

Explicit models of control and the architecture of the expert system have been proposed to provide better explanations [Clancey 1983, Swartout 1983, Neches et al. et al. 1985]. More recently, it has been hoped that user models would tailor the level of detail in explanations so that users could follow them more easily [McKeown et al. 1985, Cawsey 1992, Paris 1993]. During the 1990's it has been realised that effective explanations must take into account the dialogue context [Moore & Swartout 1990]. Explicit dialogue planning can provide better interactive explanations [Cawsey 1992, Moore & Mittal 1996].

In the following sections, we examine some of the reasoning and modelling techniques and, in Section 8.4, the interaction techniques that have been proposed for explanation.

8.1.1 Explaining tasks

The Neomycin system extended the explanations of Mycin to cover problem-solving strategies [Clancey 1983]. The problem solving in Neomycin was modelled as tasks that derived conclusions and as metarules that controlled the reasoning. The execution of the tasks was traced and conclusions were explained by the task that produced them. Explanations of the strategies were obtained by examining the metarules. In this way, the explanations were raised to a higher conceptual level, closer to the user's mental mindset [Hasling et al. 1984]. The concept of tasks was found to be convenient for showing to the users [Clancey 1983].

Similar task-based approaches have since been developed further. Keravnou & Johnson [1986] emphasise the importance of explaining the control strategy to the users. Chandrasekaran [1986] formulates the concept of generic tasks, where each task is specialised to solve a specific problem and to explain its reasoning [Tanner & Keuneke 1991]. We have based our explanations in the VICE system on the same concept. We use the term "explainable tasks" to stress the active role of tasks in explanation creation. David & Krivine [1989] have constructed a similar task-based system called Diva, from which we have adopted ideas into VICE. Mittal & Paris [1995] also stress that for good explanations, the problem solving models should contain explicit representation of the tasks.

The notion of explainable tasks bears a certain similarity to the knowledge sources in blackboard systems [Engelmore & Morgan 1988]. This similarity could be exploited to provide blackboard systems with explanations for conclusions. However, in blackboard systems the control is nearly non-deterministic, complicating the strategic explanations.

Inversely, the derivation of explanations from models by generic explanation tasks resembles the problem solving in blackboard systems, where multiple levels of data are gradually transformed to higher level results by active agents. Could explanations be constructed by blackboard systems, abstracting low level details into coherent high level accounts? This question forms a possible topic for further research. The four levels of logic explanation (Chapter 6) are a step in this direction.

8.1.2 Explaining models

Swartout's [1983] XPLAIN system introduced explanation of models. This influential system contained explicit models both for the domain knowledge and for problem solving. The expert system and its explanations were automatically derived from these models, helping explanation creation and maintenance. The domain terminology was explicitly defined in the models, easing text creation. XPLAIN's ideas were developed further in the EES (Explainable Expert Systems) framework by Neches et al. [1985] and Swartout et al. [1991]. The key point there was that explanations' contents are formed when the systems are designed. If design choices are recorded, explanations can be generated through this design knowledge. Explanation can cover the design of the expert system as well as its execution.

Chandrasekaran & Swartout [1991] point out that the more explicit the knowledge representations in a system are, the better its explanations. Each kind of explicit knowledge makes specific kinds of explanation possible. This is equivalent to Winston's [1993] remark on how finding the right representation is the key to intelligence. The way of representing a problem can dramatically affect how easy or difficult it is to solve.

The notion of *explainable objects* has not been explicitly formulated in the literature, although it is used implicitly in most explanatory systems. We want to emphasise the benefits that this concept offers: modularisation of explanatory duties and inheritance of explanations. Both of these principles are well known in object-oriented analysis [Meyer 1988, Rumbaugh et al. 1991]. This approach is valuable in settings where major parts of the software surrounding explanations are constructed with objects. For instance in HEP control systems this is desirable [Arnault 1992, Barillere et al. 1993, Butler et al. 1993, Flasinski 1994]. The emergence of object-oriented databases [Kim 1990, Hughes 1991] gives a convenient implementation platform for the proposed explanations.

The representations we proposed in Chapter 4 essentially form a domain-specific semantic network language that offers a vocabulary for talking about plants. Hayes-Roth & Jacobstein [1994] advocate the use of such languages to make the knowledge bases more understandable. In fact, the mere classification of objects into inheritance hierarchies already gives much valuable information [Fikes & Kehler 1985]. Our models are limited to binary relations, which makes them easier to explain and to understand

[Broadbent et al. 1986]. Stephanopoulos has studied a wider set of process-specific relations in the Design Kit engineering system [Stephanopoulos 1990]. Kowalski & Lebensold [1989] have designed a diagnostic system for pulp production that uses a number of relations to organise knowledge into taxonomies. They point out that much of the descriptive domain information in contained in the relations themselves.

Gruber & Gautier [1993] and Gautier & Gruber [1993] report promising work on generating causal explanations from qualitative models. Their method is implemented in the DME environment developed at Stanford [Fikes et al. 1994] that integrates domain models and mathematical constraint expressions into explanation. The aim of their work is to explain events in the simulations and the contents of the models. The reported explanation mechanism uses similar approaches to those that we have adopted. The contents for the explanations are first generated locally from the model components, then combined together according to the causal chain of an event simulation, then filtered to remove detail, and finally shown as formatted English hypertext over the WWW. Some items in the generated hypertext are links pointing to model objects. Mouse clicks to these links are interpreted as continuation questions.

Compared to our work, Gruber and Gautier are more directed towards explaining events in qualitative models. They demonstrate the results with a space shuttle thruster process. We have been more interested in the behaviour of binary automation, although similar graph-based techniques are applied in both systems to understand causality. Their mechanisms employ natural language generation techniques, which we have not adopted in the prototypes. We put more emphasis on analysing the requirements for explanations in several domains.

8.1.3 Functional knowledge

Functional knowledge is central to our approach to explaining systems in plants. Several researchers have studied ways of modelling and presenting functional knowledge: Rasmussen [1986], Franke [1991], Keuneke [1991], Lind [1990], Rytoft et al. [1990], Larsson [1992], Vicente [1992], Sassen [1993], Van de Ree [1994], Bisantz & Vicente [1994].

We distinguish a specific form of functional knowledge: the teleological meaning behind the functions of plant systems [Franke 1991]. This contrasts the processing-oriented views where the functional input-output mappings in systems are considered to be functional knowledge.

Rasmussen's influential book "Information Processing and Human-Machine Interaction" [1986] is a rich repository of ideas on plant analysis and modelling. His means-end modelling forms the underlying principles of much of our work. Rasmussen suggests that means-end models could be used to capture design knowledge, manage complexity, and justify designs. We have developed these ideas further and realised them in our prototypes.

Rasmussen [1986], Lind [1990], Tanner & Keuneke [1991], Franke [1991] and Bisantz & Vicente [1994] have all suggested the use of means-end relations to give explanation on purposes of plant items. We have formulated an algorithm for this aim and tested the concept with a prototype. In Paper VIII, we have also suggested a novel application of the concept into automation logic. Bisantz and Vicente [1994] have created similar algorithms for generating explanatory operator support over means-end abstraction hierarchies. They extend the model with topological knowledge, which facilitates better diagnosis and advice on possible control options.

Franke [1991] derives teleological explanations from qualitative models (Qsim) that are enriched with descriptions of purpose. He uses three basic primitives “guarantee”, “prevent”, and “conditionally” to derive a set of relations that are potentially richer than the ones we used. His primitives open up an opportunity for us to extend our contribution models, although the added representation effort should be balanced against the expressive benefits.

Lind [1990] extended Rasmussen’s ideas with his Multilevel Flow Modelling (MFM) method. It captures knowledge about causal interdependencies between plant items in the form of flow structures. We have applied this methodology in Papers III and IV. MFM modelling has greater representational power than means-end models, and is therefore potentially more useful in analysing plants. As far as explanation is concerned, we have mostly applied the means-end-oriented subset of MFM.

Several researchers have studied MFM. Larsson [1992] has constructed a toolbox for MFM modelling with algorithms for alarm analysis, fault diagnosis and sensor validation. Vicente [1992] has reviewed the use of MFM in control room interfaces. Sassen [1993] has used the method to construct diagnostic support systems for generic nuclear plants. Van de Ree [1994] discusses the application of MFM in plant state abstraction. Kaarela [1996] proposes MFM for structuring design-related knowledge, user interfaces, alarm processing, and on-line documentation.

We have extended means-end modelling with ways to reduce its computational complexity and modelling effort. Sassen [1993] has also extended the means-end relations, with the slightly different aim of modelling diagnostic uncertainty. In our contribution models, we suggest including other modelling constructs, such as fuzzy logic and state information. State machine representations are particularly attractive, since the controls in plants are often based on recognisable sequences. This is especially true in the case of high energy physics control systems [Adye et al. 1992].

It is not clear that the domain experts would be able to create contribution models. Johannsen & Alty [1991] suggest that a knowledge engineer may be needed to identify and to encode the knowledge for support systems, even if the language is close to the domain. Still, shown through

suitable interfaces, the constructs should be understandable both for experts and end users, which in our view is a major asset.

8.1.4 Targets of explanation

Most researchers have assumed that explanations must closely reflect the knowledge in the systems. Wick et al. [1988] and Chandrasekaran et al. [1989] attack this assumption. They point out that in the real world decisions are often justified by referring to more general knowledge, unrelated to the actual decision making mechanisms. Explanation mechanisms that use only knowledge external to the expert system are *loosely coupled*, and ones that only use the knowledge in the models are *tightly coupled*. Our systems favor tight coupling, since loosely coupled explanations would be doubtful in some domains, such as safety systems in power plants. However, the template method we use for text generation can suffer from inconsistency problems [Swartout et al. 1991]. It can be seen to loosen the tight coupling.

Chandrasekaran et al. [1989] have defined three kinds knowledge that should be explained in expert systems: 1. Reasoning, 2. Models, and 3. Strategies. We have adopted this classification: these levels correspond to the functional, factual, and strategic classes of knowledge defined in our framework in Chapter 3. However, we extend Chandrasekaran's classification along a new dimension: the *target* of explanation.

Much explanation research is confined to the limits of expert systems, or knowledge-based systems in general. We stress that at industrial plants, a support system should be able to explain the plant, in addition to explaining itself. The targets of explanation in our framework therefore comprise the processes, the automation, and the support system itself, as detailed in Chapter 3.

Stephens [1992] proposes similar dimensions for advanced support systems. Johannsen [1990] mentions two classes of computer use in advanced automation systems: (1) *computer control* (supervision and control systems), and (2) *computer support* (decision support systems). These two classes correspond to the automation and support systems levels of our framework. Enterline [1988] goes further and remarks that in addition to data, information, and knowledge levels that correspond to our framework, plant supervision calls for operational *wisdom*. We can only hope that explanation given on the mentioned levels results in increased wisdom.

8.2 EXPLANATION IN PLANTS

Process supervision has been a popular research topic from the early days of computerized automation to modern systems. The literature emphasises the

difficulty of supervision: the complexity of the plants, the process-driven operation mode, and the shortcomings of the tools. We could confirm many of these problems in our case studies in plants. We have expanded the treatment to cover the problems of the maintenance staff, who have been studied less in the literature than have the operators. To augment the problem study of Chapter 2, we now briefly recapitulate the research on support systems, mental models, and design knowledge.

8.2.1 Support systems

Problems in processes and automation have been circumvented by designing various kinds of knowledge-based support systems in plants [Rowan 1989, Johannsen & Alty 1991, Leitch & Gallanti 1992, Oxman 1993, Hayes-Roth & Jacobstein 1994]. Durkin [1996] has analysed the application types of expert systems in general from a massive survey of 2500 systems. Of those systems, the vast majority have been used for diagnosis. This is natural, since many of the problems we found in our problem studies could be solved through better diagnostic aids. Moreover, as Durkin points out, in industry there is a large amount of expertise for this class of problems, and diagnostic systems are relatively easier to develop than many other kinds of systems.

Our developments are along the same lines: most of the results in this thesis deal with either diagnostic systems or systems that try to indirectly enhance diagnostics through offering assisting information [Kurki 1995]. Other popular application types include interpretation, prescription, design, and planning, according to Durkin [1996]. Although support systems for plants have been widely reported, much fewer systems seem to apply explanation. Some illustrative systems that have used explanatory techniques are surveyed in the following.

Sachs et al. [1986] have studied support systems with a simulation of an oil production platform. Their knowledge-based Escort system, written in Lisp, seeks to avoid costly shutdowns due to process disturbances. It provides advice on plant alarms, their relative importance, possible consequences, and the underlying reasons. A “Why” button on the operator’s alarm screen gives elementary explanation of the diagnostic reasoning.

The Tiger system for turbine condition monitoring, with its advanced knowledge-based support, was discussed in Chapter 2. The Diva system [David & Krivine 1989] was built for equivalent purposes. Diva is in many respects similar to the VICE system used in this thesis: both systems are designed as tools for vibration analysis of rotating machines, both organise problem solving knowledge in a hierarchy of tasks and explicit fault models, and both base explanations on traces of task execution. Diva is reported to offer explanations on strategy and reasoning, and even some negative “Why-not” explanations. The authors conclude that structuring

Diva through a task formalism undoubtedly made its maintenance easier. We have come to the same conclusion with VICE.

The very design of the D.M.I. interface for a process control support system [Le Strugeon et al. 1992] seems to imply explanation. The system's designers asked themselves questions "what, when, how" to guide the design of the system to be able to give satisfactory information to the users. The system itself appears to be a typical forward-chaining rule-based alarm analyser. It does, however, contain facilities for justifying the reasoning for several levels of users. A trace of the conclusions gives the proper context for explanations.

Jovanovic & Maile [1992] report on the ESR system that offers knowledge-based support for lifetime assessment of high-temperature pressurised components (mainly pipes) on power plants. The system applies hybrid techniques, including rules, objects, hypermedia, numerical calculus, and data bases. The explanations in the system seem to consist mostly of prefabricated reference material and dynamic database reports shown as hypermedia, although the system does give some dynamic context-sensitive support for navigation and management of several levels of detail. The interaction can be either system-driven or user-driven.

Årzen [1993] reports on a support system concept that encompasses a broad range of plant information. He proposes a plant database that includes knowledge currently found in:

- Control systems (e.g. logic, interfaces, history & real-time data),
- CAD systems (e.g. drawings, 3-D object descriptions),
- Information management systems (e.g. supervisory and planning functions),
- On-line documentation systems (e.g. manuals, data sheets),
- Design databases (e.g. process component and product information),
- Plant simulators (e.g. dynamic models), and
- Real-time knowledge-based systems (e.g. qualitative models and heuristics).

Årzen discusses the various ways that data could be represented in the database and presented to users. A demonstrator system has been developed in collaboration with several industrial partners. Built with the G2 expert system shell, the demonstrator contains process simulators, user interfaces for process engineers and end users, and a variety of modelling methods and diagnostic algorithms. Although the concept does not explicitly target explanations, it strives for the common purpose of making plant information more comprehensible and accessible to humans from several perspectives.

Therefore we believe that Årzen's concept could give excellent grounds for constructing explanations for plants.

The Gradient system (Graphical Dialogue Environment) for operator support features alarm analysis, consequence prediction, operator error detection, fault diagnosis, and recovery procedures [Weisang & Zinser 1992]. The system contains an explicit dialogue subsystem. The dialogue is designed with an interactive environment and various user interface modules (termed "dialogue assistants") are then automatically generated. In a power plant demonstration, dynamically generated graphics and explanations are shown to users through multiple coordinated displays. No details are given on how the explanations are generated, however.

Zinser & Frischenschlager [1994] further report on a prototype of a power plant control room that uses a number of modern interaction techniques: visualisation, on-line documentation, multimedia, groupwork, and support systems. Again, even though there are not explicit mechanisms for explanation, the design of the control room applies many of the same principles that we have seen useful for presenting explanations. Of particular interest in the article are novel "mass" displays, where large amounts of data are visualised through patterns that are reported to be easy for the humans to follow. It would be interesting to see how explanation could take better advantage of human pattern recognition capabilities, instead of concentrating on reducing the amount of explanatory detail. Another interesting dimension that the article refers to is audiovisual information. Could explanations in some cases be formulated into, for instance, auditory patterns?

Slotnick & Moore [1995] point out that a large number of problem solving systems in industry are still based on quantitative knowledge. In many cases "mathematical" solutions are preferable to expert systems, particularly in terms of provability, stability, and easier knowledge acquisition. Besides symbolic knowledge, quantitative knowledge should be explained as well. Slotnick & Moore have developed a representation that translates mathematical relationships into domain-level concepts and strategies that are typically more understandable to humans. Their explanation subsystem finds a match between hierarchical models of human scheduling knowledge and mathematical knowledge of bottleneck dynamics. With common terms in both, the subsystem then plans an explanation to answer questions of the "Why" and "Why not" varieties. Slotnick & Moore have applied the results into an automated scheduling system for a specialty metal plant.

The surveyed systems give information corresponding to many of the aspects defined by our framework in Chapter 3, although no single system offers everything. Explanation has not usually been the main focus of these systems. In contrast, the present thesis studies essentially all plant support systems' functions from an explanation perspective.

The prototypes shown in this thesis together cover a wider range of explanations than the support systems found in the literature. However, we still have work to do to extend the prototypes into working systems in plants.

8.2.2 Mental models of plant systems

Much of the research on mental models concerns interaction with computer software systems [Carroll & Olson 1988, Wexelblat 1989, Bannon 1990]. Since an increasing part of the automation is computerised, many observations from human-computer interaction research apply to industrial systems.

Carroll & Olson [1988] state that a user's mental model should contain knowledge of the target system's functions, components and their relations, internal processes, and how they affect the components. A user will need several models at different levels of abstraction. Rasmussen's work on human problem solving [1986] supports this claim. We have constructed explanation techniques that clarify such knowledge at multiple levels of abstraction.

Crossman & Cooke [1974], Stephanopoulos [1987], Hoc [1989], and Bainbridge [1983] all emphasise that mental models are harder to form with plant processes of a time-dependent nature. Mental models of the states only emerge with time [Bainbridge 1983]. Many processes with their long time delays can be hard to understand, because short-term mental models tend to degrade before they can be refreshed with new findings [Stephanopoulos 1987, Hoc 1989]. We help this understanding by offering behavioural explanations that can recover the past states of the automation.

Norman [1988] observed that there are in fact several mental models in plant design processes. The *conceptual model* is what a designer has in mind when designing a system. The *system image* is what a user sees of the system, and the *mental model* is what people form in their heads. Usability is highest when these three models are consistent with each other. The more the models differ, the more there will be misunderstandings that show up as difficulties in use. If no conceptual model is offered at all, the users quickly invent their theories of how the system works [Bannon 1990, Wexelblat 1989]. In such cases the user's performance will decrease [Stassen et al. 1988], increasing fatique and stress.

We seek to transfer design knowledge from the designers to develop the mental models of the users. We base the necessary conceptual model on the multilevel relationships of the plant's objects. The system image is offered as a set of interlinked hypertext documents that reflect these relationships. The designer's intentions behind the design choices are explained through models. This should enforce the users' mental models to resemble those of the designer, since the same conceptual model is employed in all phases [Norman 1983].

Stassen et al. [1988] warn that overtraining may reduce the mental models from a knowledge-based level to rules of thumb. Users would then apply overlearned procedures in all cases, even when they are not appropriate. Stassen et al. conclude that training should concentrate on an understanding of the plant rather than on procedures. We want to improve this understanding with explanations.

Some of the training could be trusted to on-the-spot explanations. As Rettig [1993] notes, "…the best time to provide training is at the very moment the learner needs to apply the knowledge. So the best place for computer-based training is *inside* the tool need to accomplish the work." However, for training, the explanations should be crafted into a tutoring mode, which has been outside the scope of this thesis.

Norman [1988] remarks that mental models are usually incomplete, and even faulty. Yet even incomplete models can be useful [Lees 1974]. In practice it is impossible to force "correct" mental models. We take this to imply that even partial explanations of a system can help the users. Explanations do not need to cover all aspects of the whole plant, as long as they help in many cases enough to be useful.

8.2.3 Design knowledge

Kaarela [1996] maintains that many problems in plants are due to the hardware-oriented level of today's plant designs. Higher level knowledge gets lost in the design process [Klein 1993]. This necessarily affects the level of automation and therefore the success of coupled knowledge-based systems. It also creates a mismatch between the abstract models of a knowledge-based system and the operator who is forced to think in terms of hardware. Korhonen [1991] and Rautila [1992] have sought ways to raise the level of design with better representations. This has also been our focus with the P&ID and Justifier prototypes.

Stephanopoulos [1990] has outlined the typical and potential uses of knowledge-based design tools in process and chemical engineering. He argues that current design tools are geared towards numerical computation and offer little help for knowledge-based tasks in design. Ideally, the tools should know about the goal structure of the design process, record the process, and have rationale for design decisions. Such a system could reduce the cost and improve the reliability of the design. Our prototype based on P&ID achieves some of these ideals through its models and explanations. The work behind the Design++ platform has been documented by Riitahuhta [1988], Harmon [1990], and Katajamäki [1991]. Klein [1993] reports on the DRCS system that captures knowledge concerning both the designed artifact and the design process. The system also uses a language for design representation that essentially consists of semantic networks.

We view plant design essentially as navigation in a functional abstraction space. In other words, design consists of making decisions on

how abstract goals are realised by concrete systems. As regards design rationale, this approach is close to that of Chandrasekaran et al. [1993]. We cover a specific facet of design rationale from an explanation viewpoint: the teleological justifications of items in plants. Design rationale has been said to imply "an explanation that answers a question about why an artifact is designed as it is" [Lee 1992]. This has been our main focus as regards design. Other researchers have covered many aspects of design and design rationale [Seppänen 1990, Stephanopoulos 1990, Conklin & Yakemovic 1991, Fischer et al. 1991, Keuneke 1991, Lee & Lai 1991, Gruber & Russell 1992, Chandrasekaran et al. 1993, Klein 1993].

It should be noted that design knowledge does not cover all knowledge needed in plants. Barentsen [1991] has found evidence that a considerable amount of knowledge is transferred in the everyday discussions of the plant staff. This experiential knowledge is often unavailable during design phases. If facilities for gathering this operational knowledge could be provided, information transfer from the users to the designers would be enhanced [Kramer 1987, Gilmore et al. 1989, Johannsen & Alty 1991]. We have concentrated on design knowledge, since it is more readily available with today's tools (although even there the situation could be improved).

Barentsen [1991] further argues that useful knowledge for diagnostic applications surfaces in the form of stories that the staff tell in plants. People seem to prefer entertaining stories of past problems compared to dry problem analyses. This could form an interesting research topic: how could experiential knowledge in plants be collected into case bases [Lenz et al. 1996] and then explained to users through story-telling mechanisms that generate explanations from the cases.

8.3 LOGIC EXPLANATION

There is plenty of reported research on analysing logic in the domain of digital electronics design. Much less research is applied to the techniques in the field of automation, and attempts to explain automation logic are quite rare. In this section we review some of the relevant publications in these fields.

8.3.1 The diagnosis background

The large body of work on diagnosis and automated reasoning on discrete devices [Barrow 1984, Genesereth 1984, Davis 1984, de Kleer 1984, Davis & Hamscher 1988, Hamscher 1991, Malhotra & Seviora 1992] is relevant to our work. Logic explanation is, in essence, a form of diagnosis in which the aim is not to locate faults in the logic network but to explain its behaviour. In electronics, detecting physical faults in integrated circuits is essential for testing, whereas in automation, logic is normally assumed to work correctly

(provided that the design is correct). Of particular interest regarding the automation domain is to find "important" events at the network inputs. These tasks, diagnosis and explanations, call for similar techniques. We have adopted several of the techniques from diagnosis research, including representations and search algorithms.

In their survey of logic synthesis techniques, Brayton et al. [1990] give an introduction to many of the techniques used for automatic analysis of digital logic circuits. Artificial intelligence techniques have been used extensively in this domain: for simulation [Filer & Marshall 1989, Brown et al. 1993], testing [Bending 1984, Abadir & Breuer 1985, Singh 1987], minimising [Hong et al. 1974, De Geus & Cohen 1985], design understanding [Filer et al. 1991, Mir 1994], and layout design [Filer & Spink 1987]. Many of these applications involve graph-based representations and both heuristic and algorithmic solutions, in the same way as Lepo does. This could give grounds to a two-way cross-fertilisation: that Lepo be used to explain digital circuits, and the research on digital circuits be adapted into the automation domain.

De Kleer [1984] has formulated a theory for commonsense understanding of electronic circuits. This fundamental paper seems to have later been an inspiration for many other researchers who have tackled problems of knowledge-based circuit analysis. Even though the paper concerns analogue electronics, many of the techniques are applicable to the more tractable domain of discrete devices. De Kleer observes that the function of a circuit (or its purpose) is related to its structure (for instance, its schematic). The intermediate notion between function and structure is the behaviour (or, what the device does). With causal analysis of qualitative models, his Equal system can predict a circuit's behaviour and explain it in terms of *mechanism graphs*. It recognises the purposes of the circuit in a number of known *configurations*. De Kleer promotes the power of teleological knowledge, saying that it helps to see the role of a component as a part of the whole, and to recognise similar circuit configurations.

Malhotra & Seviora [1992] have devised an object-oriented framework for representing and reasoning on digital logic. Their Digital System Understander models logic through multilevel object representations, with explicit mappings between concrete and abstract levels, and interdependencies and constraints between objects. A blackboard-based framework reasons over the representations through a form of constraint satisfaction. The system supports two logic-related activities: redesign and design fault localisation. The paper once again brings up the idea of viewing Lepo as a constraint-based blackboard system.

Genesereth [1993] reviews the work of the Stanford Logic Group on knowledge-based analysis of discrete systems. He notes that originally their research started on explanations, although it quickly moved towards rule- and model-based diagnosis of discrete devices. He emphasises the need for design information, for which purpose the Helios CAD system was built. It

allowed automated simulation, debugging, testability analysis, test generation, and diagnosis. Helios helped to describe devices at multiple levels of abstraction, from block diagrams to detailed schematics. Moreover, it facilitated recording design alternatives and rationale. Slowness of the general model-based reasoning implemented in first-order predicate calculus is mentioned as a major problem in the system.

Hamscher [1991] has modelled complex digital circuits for diagnosis. In the XDE system, circuits are represented as a union of both functional and physical decomposition hierarchies. He observes that abstraction allows efficient reasoning on coarser levels of detail, which is more efficient. Binary signals can be described in terms of such coarse concepts as *frequency, cycles, counting, sequence, duration, sampling, and change*. These temporal abstractions make it possible to reason about large numbers of events without needing to specify them all. As Hamscher remarks, temporal precision can be sacrificed without losing too much predictive force. He postulates that humans appear to use similar abstractions when analysing electronics.

Hamscher's work could benefit Lepo. Currently it represents most of the dynamic behaviour of PLC logic as time-based dependency nets. Perhaps the complexity of the logic could be better managed by finding suitable abstractions that explicitly translate the logic networks into higher level representations. The reasoning would then be carried out on these higher levels. The recovery of state machines that Lepo currently does is already a step in this direction. Other higher level formalisms would be needed. Further work should seek abstractions that are valuable in the domain of automation logic.

Perkins & Austin [1990] have added a temporal reasoning facility into the frames of an expert system. The time representation in Lepo objects has used two of their primitives (exact, valid_until_changed), although we have not explicitly applied temporal relations in reasoning. Clarke et al. [1986] have used temporal logic for automatically verifying specifications of state machines. Moon et al. [1992] have extended their work to safety analysis of process control systems. In their work the temporal logic constructs are used to model not only the controls but also the process. Lepo could be extended in a similar manner for analysing discrete-event processes, if they are modelled through logic.

8.3.2 Explanations of industrial logic

Milne & Guasch [1994] and Guasch et al. [1995] have studied logic explanation with an approach that is close to ours. They have analysed the alarm logic of industrial gas turbines implemented on ladder logic. Their software reads the ladder logic source code, performs a series of reductions to optimise the code, and then uses rule-based reasoning to generate diagnostic messages on the reasons of alarms. The software has been tested

with an actual PLC application that is comparable in size to the paper winder example we have studied. They emphasise, as we do, the acquisition of the logic code directly from existing design descriptions. They have not studied the use of the software with other PLC languages, although they could be added with suitable translators. The algorithms appear generic enough to be usable with other languages.

Zinser and Frischenschlager [1994] have applied electronic documentation techniques in power plants. Their treatment of the various application possibilities includes color-coded animation of signal states in logic diagrams. They note the advantage “far less time is needed to look up either the diagram of the actual signal states”. Their main focus is on innovative interaction technologies, though, and the article does not give further evidence of any reasoning support for the task, besides better presentation techniques.

Falcione & Krogh [1993] report on a way to transform ladder logic code into a state machine representation. Their motivation is design recovery, with the convincing hypothesis that the resulting state machines are easier to understand and maintain than the ladder logic. Their algorithms achieve this conversion through a series a graph transformations. They group nodes into parallel and sequential structures, exploiting the internal dependencies of the PLC code. The authors do not detail the many potential uses of the algorithms, although a neutralisation tank example is used to clarify the algorithm. From the paper it appears that the object structures of Lepo could accommodate the algorithms relatively easily. Such extensions could potentially give better analyses of the dynamic behaviour of logic, although it has not been investigated how the results should be explained to the plant staff.

Moon [1994] has analysed PLC ladder logic languages to verify the safety and operability of PLC’s. His method uses temporal logic to analyse the time-dependent behaviour of a PLC. A model checker constructs a state transition graph of the ladder logic code that is then used to find errors in the code. This approach allows simulation of the logic to answer speculative questions. Moon reports that the method works well, although it has been studied with small cases (less than a hundred alarms). The system does not yet cover timers, counters, and other non-binary constructs either, according to the author. Lepo has been similarly constrained.

The commercial GDA package (G2 Diagnostic Assistant) [Finch et al. 1991] offers some built-in explanations of the propagation of logic states through circuits. The explanations and simulations are immediately available on screen once the circuit topology has been drafted. Besides logic, GDA can simulate several other data types and functions. The package has good facilities to create support systems that are understandable to users but can be constructed by plant experts. However, the system is rather closed. It is difficult to import the logic definitions from outside the tool. The Lepo C++ prototype has been designed to receive the

logic code from the external world, even directly from the logic controller. In addition, our prototype answers a wider set of questions.

8.3.3 Improved logic languages

Bonfatti et al. [1995] have devised ways of constructing logic automatically with their specification language called Easier that allows one to express constraints ("laws") for defining the logic code functionality on a high level. The high level representation can then be translated to PLC code with a set of well-defined transformations. State machines and safety laws are among the high-level abstractions that they have applied to industrial controls. Their language also suits domain modelling and formal analysis, and could even form a basis for explanations.

Laitinen [1995] has studied natural naming in software. He has observed that the choice of names for program constructs (variables or subroutines, for instance) can dramatically affect the understandability of the program. A name can have much semantic meaning to humans. Explanations should try to establish a common meaning for a program element in all the places it is mentioned. In PLC software, I/O references should be named according to the techniques that Laitinen proposes. This would help the understandability of PLC software, and would make explanation easier.

Better specification methods and tools for PLC software could enhance logic explanations considerably. We believe that the trend towards higher-level programming languages for PLC's will ultimately realise this concept. With an advanced tool, the designer would simply express formally the purpose of the circuit and the tool would generate the corresponding logic automatically, documenting the purpose at the same time. Application generators and library components are likely to be popular among future PLC programmers. If PLC software components are stored in libraries, the explanatory models corresponding to the components could be a part of the library. This would facilitate reuse of explanations.

Lewis [1996] proposes a number of functions that a PLC software programming station based on the IEC 1131 standard should comprise in the future (classification in Table 8 modified from the source publication):

Table 8. Functions for future PLC programming tools (adapted from Lewis [1996]).

Interface issues	***Analysis tools***	***Project management***
A graphical user interface	Ability to pick the right language for a given task	Library management for software components
Multiple views to the software	Support for hierarchical designs	Concurrent development and access management
More friendly graphical editors	Import/export of design information	Revision control
Navigation in the code (follow signal paths)	Program validation, syntax and consistency checking	Documentation generation
On-line help on the programming constructs	On-line diagnostics (visualising dynamics)	On-line modifications with audit trails

Our discussion is compatible with Lewis's list. Lepo already in its present form provides many of the listed features. We believe that the techniques used in Lepo will become necessary in the future programming environments that Lewis envisions. It should be noted that many of the listed functions are already included in the current programming tools for general (non-PLC) software. In the future, PLC software may be programmed with similar tools as other software.

Compared with the majority of the related work, our research has been more directed towards industrial automation. We have not attempted to develop new, more powerful logic languages. Rather we have tried to offer improved tools for the analysis of existing logic solutions in plants. In terms of algorithms, we have pursued more industrial relevance than theoretic significance. We have focused on function block logic, whereas ladder diagrams are used in most other studies of automation logic. This is just the surface: most researchers have used similar modelling constructs adapted from diagnostic research.

8.4 INTERACTION TECHNIQUES

The emphasis in the literature on explanation in the recent years has changed from content creation to studies of human-computer interfaces for explanation [Paris 1987, McKeown et al. 1985, Moore & Swartout 1990, Cawsey 1992, Swartout et al. 1991, Feiner & McKeown 1991, Moore & Mittal 1996]. This trend may be taken to reflect the changing focus of human-computer interface research [Grudin 1990]. Such topics as customising, dialogue maintenance, context, and content generation have

raised much interest. We briefly survey the research on these interaction techniques in the context of our prototype systems.

8.4.1 Hypertext

Hypertext has been studied extensively since the popularisation of the concept [Conklin 1987]. A great deal of the research is directed towards office documentation management. However, hypertext for technical documentation has also been treated in depth, for instance by Tyrväinen [1994]. Many issues concerning hypertext are also relevant to our work, since we employ it both as the metaphor and as the medium in our explanations. As Moore & Swartout [1990] observe, hypertext is a natural form for explanations, since it avoids many of the problems associated with question understanding. It also conveniently facilitates continuation questions.

It should be noted though that hypertext is not without its problems. The orientation problems, discussed in Chapter 7, may not be the most fundamental ones. As Paunonen [1993] formulates, *"…reading does not traditionally belong to the operator's assortment of problem solving methods"*. In many industrial settings other means of explanatory interaction would be preferable to hypertext, for instance embedding the explanations into graphical displays. Feiner & McKeown [1991] have approached the presentation of explanations through dynamically generated three-dimensional graphics, which should ease the acceptance in industry.

Paper I (for an earlier version, see [Huuskonen 1990]) was among the early publications that applied dynamically produced hypertext for explanation. Similar work had been carried out for some time by Moore [1989]. Since then, the popularity of hypertext has grown immensely due to the World-Wide Web (WWW) [Berners-Lee et al. 1994]. Hypertext in HTML format is today being dynamically generated by thousands of computers in response to mouse clicks in browser windows. Typical research applications are described in [Jeffery 1996].

Models that are based on semantic networks are particularly easy to turn into hypertext [Snaprud & Kaindl 1992]. An interesting demonstration of the concept can be found on the Internet [Gruber 1995]. Tyrväinen [1994] opposes this view, maintaining that text structure and knowledge-based structures be separated. Nevertheless, more formal structures for the text contents are desirable. SGML is the most important standard in this area [ISO 8879, 1986].

We have taken a template-based approach to text generation, following Rubinoff [1985] and Cawsey [1992]. Many commercial systems use fixed templates [Wick & Slagle 1989], whereas our templates are adapted to the situation through filling in values based on context. Some systems translate knowledge structures directly to English [Buchanan & Shortliffe 1984, Feiner & McKeown 1991], which helps consistency problems associated

with templates [Neches et al. 1985]. Our hypertext templates contain active procedures. This concept is now being popularised through Java [Campione & Walrath 1997].

The orientation problems with hypertext that Conklin [1987] and Nielsen [1990] complain about have not yet been solved in today's WWW. Graphical visualisation techniques could help [Nielsen 1990, Rivlin et al. 1994]. We have placed our trust in the standard navigational commands of WWW browsers and search facilities that can be built into WWW documents to help navigation. This approach proved workable for relatively small document masses, as detailed in Chapter 7. Text retrieval for larger technical documents has been studied in more depth by Tyrväinen [1994].

8.4.2 Dialogue

Advice-giving systems have been the traditional domain of explanations. The dialogue in those systems usually takes a question-answer form, often implemented through natural language interpretation. Our hypertext dialogue is similar, although questions are implicitly given by pointing with the mouse. We target user assisting applications, where much of the initiative comes from the user. The user's role in the dialogue is that of an active controller.

There is much discussion about dialogue in the field of computational linguistics and human-computer interfaces. A good compilation of the relevant research issues is found in [Kobsa & Wahlster 1989]. It is generally agreed that a system that participates in dialogue with humans should model the goals and intentions of the user for good dialogue. The role of planning is often emphasised in dialogue systems [Moore & Swartout 1990, Swartout et al. 1991, Cawsey 1992, Moore & Mittal 1996].

We depart from this approach and presume that elementary dialogue can be based on simple representations of *context*, without a need to know the goals of the user. Our explanatory mechanisms do not contain an explicit planning module. Instead, dialogue occurs in navigation through hypertext links, context is maintained within these links, and heuristic question-answer matching forms the implicit dialogue planning. However, the quality of the dialogue could be raised by explicit dialogue planning. A planner could be included in our dialogue model in a straightforward manner.

Gruber and Gautier [1993] support our approach with their DME system that uses contexted hypertext. They note that "...although DME does not do text planning, the technique allows the user some control over the level of detail and focus of the explanation. This lessens the need for a user or dialog model, especially when tutoring is not the explanation task." This is equivalent to our claim.

XPLAIN was one of the first explanatory systems to emphasise the notion of context. The explanations could be generated from several

viewpoints. To find a suitable viewpoint, the system employed knowledge of the execution state of the expert system, of the dialogue exercised so far, and of the possible intentions of the user [Swartout 1983]. McKeown et al. [1985] have also noted that a user model helps in finding the right context. If the user's intentions are known, the contexts for follow-up questions are easier to guess. Mittal & Paris [1995] define five elements of context:

1. The problem solving situation,
2. The participants involved,
3. The mode and medium of interaction in which communication is occurring,
4. The discourse taking place, and
5. The external world.

To craft context-dependent explanations, the system needs to be able to reason about these elements. They must therefore be represented declaratively. This has also been the intention behind our context models, although they do not comprise the whole range of possibilities set by Mittal & Paris.

Of the context elements defined by Mittal & Paris, our use of context covers: 1. The problem solving modelled as tasks, and knowledge representation as explainable objects; 2. User types or "levels"; 3. Hypertext as the medium and browsing as the mode of interaction; 4. The history of the discourse, implicitly represented in the hypertext pages that can be returned to (excluding, for instance, communicative goals, rhetorical relations, or the information conveyed). The external world, or, more precisely, the communicative situation type is fixed in our systems.

To explain things successfully the use of context seems necessary [Swartout et al. 1991, Mittal & Paris 1995]. We have taken a lightweight approach to context modelling and maintenance: context is stored in hypertext links and it gets refined through menus. This approach does not require a dialogue planner, but achieves sufficient dialogue capabilities for limited systems. Explicit planning would undoubtedly increase the quality of the explanations [Moore & Mittal 1996].

8.4.3 Rhetoric for explanation

Research into linguistics and human dialogue structures has revealed a number of different ways of explanation that humans appear to use when describing things [Moore 1989, Cawsey 1992]. For instance, Paris [1993] has used the following *rhetorical predicates* to model descriptions commonly found in encyclopedias and high school text books:

- Identification: Description of an object in terms of its superclass.
- Constituency: Description of subparts.
- Attributive: Illustration of the properties of things.
- Cause-effect: A causal relationship between things.
- Analogy: Demonstrating the similarity between things.
- Renaming: Giving another name for a thing.
- Comparison: Comparing a thing to another.

These or similar predicates have then been used to find suitable strategies for constructing explanations. For instance, Moore & Mittal [1996] add "Elaboration", "Background", "Concession", "Justification" and "Evidence" into the palette. These predicates can be seen as the possible moves in the explanatory planning process.

The identification of the relevant rhetorical predicates has been of less interest to us, since in our mechanisms the planning is not explicit. However, we have implicitly applied the predicates Identification, Constituency, Attributive, Cause-effect, and Justification in our explanations. Future work should include identification of more predicates relevant to the plant domain, and inclusion of the predicates into explanation planning.

8.4.4 User modelling

Every operator is an individual, and should receive explanations tailored to his/her personal needs [Paris 1993]. The operators have different backgrounds, motives, and methods. The automation system should support this customization. It should form a *user model*, guessing what the user knows and does not know. With this model, the systems can adjust the amount and kinds of information to suit the individual. There is a body of research available on user modelling [Carroll & Olson 1988, Kobsa & Wahlster 1989].

Rich [1989] suggests the notion of *stereotypes* to model the users. The users are modelled into classes, organised in a frame hierarchy. Each frame lists the user's typical attributes that can be inherited through hierarchies, or specialised for individuals. These attributes can then be used to customise the views of a knowledge base. McCoy [1989] proposes the use of *perspectives* for this purpose. In her view, any object could simultaneously be visible from several hierarchies, the choice of which depends on the context. The relevance of an object to a dialogue topic is determined by applying a weighting function over its attributes.

Our models of the user in the VICE system are based on predefined stereotypes ("user profiles") for each class of users and on predefined

perspectives (“levels”) for each class of objects. Each user can select the desired stereotype in his/her profile. Each object may explicitly list the kinds of users it is suitable for. The user’s view to the knowledge in the system only contains those elements that suit the user’s class.

Since our dialogue model does not explicitly contain a planner, it is not possible to reason about the user’s goals and intentions [McKeown et al. 1985]. For this reason we did not implement the dynamic changing and numeric weighting of the perspectives, in the manner suggested by McCoy. However, the user may change his/her level in the profile at any time to receive less or more details and to adjust the views to the knowledge. In the VICE and Lepo prototypes, more detailed help is always available through the hyperlinks.

Wexelblat [1989] argues that users cannot be classified into fixed categories. In fact a single user may require explanations at several levels during one session. This calls for a more dynamic notion of user levels, where each user selects the level of detail through hyperlinks as desired. This is the approach we have taken in the Lepo prototype.

Paris [1987] has found that different users need not only different levels of detail, but even entirely different kinds of explanation. Domain experts tend to seek structural knowledge, whereas novices are more interested in understanding the functions. Several different strategies should be devised for different classes of users and situations. A planner would then create the contents based on these strategies. In our systems strategies are implicit: we let the user’s curiosity guide the choice of explanations.

9 INTRODUCTION TO THE PAPERS

The papers included in this thesis were written over a period of four years, from 1990 to 1994. They demonstrate how ideas unfold over time – many of the principles used in the later papers were first outlined in the early papers. To illustrate this maturation process, we have included some comments on the development of the ideas.

Table 9. Summary of the views presented in the original papers.

Paper → *View* ↓	**I, II**	**III, IV, V**	**VI, VII**	**VIII**
Domain	Support systems at power plants	Power plant design & user interfaces	High energy physics controls & user interfaces	Industrial control logic (PLCs)
What is explained	Expert system's operation	Design knowledge	Dependencies & design knowledge in controls	Control logic behaviour
Dominating explanation tasks	Justification of reasoning, Model clarification, Document navigation	Design recovery, Abstraction	Document navigation, Abstraction & Model clarification	Behavioural diagnosis
Modelling techniques	Tasks, Context	MFM, means-end models	Task analysis, contribution models	Logic graphs
Interaction techniques	Contexted hypertext	MFM in design tools, displays & hypertext	Control & support systems	Logic diagrams & hypertext
Users	Vibration experts, developers	Plant operators, maintainers & designers	Physicists & maintainers	Maintainers, programmers & operators
Implementation environment	Kee/VICE	D++/P&ID	-	Xlisp, C++
Main contribution	Explainable objects, hypertext dialogue	MFM for capturing and explaining design knowledge	Requirement analysis, contribution modelling	Concepts for logic explanation

Table 9 summarises the various views presented in the papers.

The papers discuss various aspects of explanation in plants. Table 10 places the different views of the papers in the context of the framework presented in Chapter 3 of this thesis.

Table 10. The relation of the original papers to the explanation framework.

Explanation target → / *Knowledge class* ↓	**Processes**	**Automation**	**Support systems**
Factual	Papers I, IV & VII	Papers VI & VIII	Papers I, II & V
Functional	Papers III & IV	Papers VI & VIII	Papers I, II & IV
Strategic	Papers III, IV & V	Paper V	Papers I, II & V

9.1 EXPLAINING KNOWLEDGE IN A DIAGNOSTIC EXPERT SYSTEM

The first two papers deal with the diagnostic expert system VICE that was briefly described in Chapter 4. These papers lay out the essential mechanisms for explanatory modelling and interaction that the author added into VICE. In these early papers, the background requirements for explanation are not explicitly described, since they were inherent in the design task of creating an explanation facility for an existing expert system.

9.1.1 Paper I: Explainable tasks, user profiles, and hypertext

Paper I introduces the gradual development of VICE and its main architectural features, most notably the division of reasoning into tasks. The section “Explanation types” lists the three types of explanation provided by VICE: (1) explaining reasoning, (2) explaining strategies, and (3) explaining domain models. These three types are reflected in the classes of knowledge in the framework of Chapter 3. The most significant development, one that also proved effective in practice, was the concept of explainable tasks.

User profiles are discussed in the next section. A profile scheme was implemented that allows the different users to get different levels of detail in the produced explanations. After the publication of the paper, as the

system was developed further, it became evident that maintaining the three levels of detail much increases the effort in developing the explanations. This is a known feature of text generation mechanism that is based on templates. As a consequence, a single level (the experts') was used in practice. The title of the paper appears slightly too optimistic in its emphasis on tailoring the explanations to each user.

The subsequent sections introduce an idea that was rather new at the time of writing the paper (in 1990): the use of hypertext as the explanatory interaction medium. The two figures show early examples of hypertext explanations implemented in the Symbolics Lisp environment. The users could navigate in the explanations by pointing to highlighted words with the mouse.

The section "Reporting facility" views explanations as text and graphics fragments that are collected into reports. In essence, the explanation subsystem functions as a report generator that produces both interactive explanations and static reports to be included in the documentation concerning the diagnostic session. An automatic mode to collect standardised reports was also implemented in which the reporting facility of VICE was seen as just another kind of user of explanations. The implemented explanation mechanism was therefore quite general and usable in several modes.

9.1.2 Paper II: Context for explanation generation and navigation

Paper II emphasises the notion of *context*. It is used in two main ways: (1) as a basis for answer planning and (2) as a basis for hypertext navigation. While providing some theoretical results, the paper assumes a somewhat implementational view, discussing in depth the mechanisms to implement contexted hypertext. We have applied in other research projects many of the concepts and techniques presented in this paper.

Section 2 lays out the central ideas about explanation modelling and interaction in the case of expert systems. Explainable objects (termed "explicable" in the paper) are defined. Explainable tasks are now seen as a special case of explainable objects.

Section 2.3 presents the application of context in searching for possible question/answer pairs. This novel mechanism for content planning relies on matching the given question's context profile to predefined answers. Explanations are only generated for those answers that suit the context of the question.

Dialogue is viewed as navigation in context spaces in the following sections. Figure 2 shows our iterative dialogue model that has been further refined in Chapter 7 of this thesis. Explanations are seen as contexted nodes in hyperspaces, and navigation between the nodes is seen as context changes. This analogy is demonstrated with example explanations.

In Section 3, the explanations are seen to be most useful in program development: validating, debugging, and knowledge acquisition from users. This section judges the concepts of explainable objects and context-based answer planning to be succesful ones. Templates and keyword-based user levels are regarded as impractical techniques for large-scale systems. An explicit dialogue planner is seen to be necessary for larger systems. The hypertext explanations were seen to be very useful from an interaction point of view.

9.2 EXPLAINING DESIGN KNOWLEDGE OF INDUSTRIAL PLANTS

Papers III, IV and V all discuss the same topic from different perspectives: how to capture design knowledge into models and to present it to users. The papers offer successively more detailed views, with Paper III providing a problem setting and knowledge acquisition view, Paper IV a support system view, and Paper V an explanation mechanism view.

9.2.1 Paper III: Capturing design knowledge into models

Paper III introduces the use of Multilevel Flow Modelling (MFM) for capturing design knowledge. It motivates the need for modelling by pointing out that there is a lot of information exchanged between parties involved in plant design.

Section 2 regrets that in real projects, the communication is far from efficient, which became clear in a power plant case study. An important point is that design documents tend to concentrate on lower level details, ignoring higher level knowledge such as purposes for equipment. Especially process knowledge and design criteria seem to get lost on the way to the users. We emphasise the need to cover the higher level knowledge in addition to device-level details. Such knowledge is certainly created and used during the design process, but for various reasons is never recorded.

Section 3 advocates the use of a systematic approach to acquiring design knowledge (for instance the functional dependencies between devices and goals of a plant, or the criteria behind design decisions) during the design process, at the point where the knowledge is most easily obtainable. Section 4 presents the formalism used in our approach, the MFM method (discussed in Chapter 5 of the present thesis). Its use in building support systems, user interfaces, and on-line documentation systems is suggested.

Section 5 shows how we extended design tools for capturing design knowledge. The Design++ tool with the P&ID design application is described first. The extensions, the object-based and graphical environments

are used to capture design knowledge that is then exported to other applications in the form of objects and structured text.

Section 6 concludes the paper by judging the results to be very promising, although a certain degree of doubt is expressed whether designers would accept the increased workload and changes in design culture that MFM requires. In retrospect we see this as the major drawback of MFM for industrial applications. This shortcoming has lead us to concentrate on the less bulky means-end aspects of the technique in Paper V.

9.2.2 Paper IV: Bringing design knowledge to users

Paper IV elaborates the ideas of Paper III of bringing design knowledge to the users in plants. The central idea is to use a single conceptual model from all phases of plant design to its operation.

The paper illustrates the use of the conceptual model. It helps to capture design knowledge, communicate the knowledge between designers, and represent it in control systems. The model also facilitates construction of user interfaces, presentation, interaction, and gathering users' experiences. MFM is suggested for realising the model, since it captures interdependencies and interactions between the concrete and abstract parts of a plant. This is essential information for a conceptual model.

We observe that hypertext is convenient for presenting design knowledge to users, since its form (nodes of text with links to other nodes) is very similar to the form of the knowledge models (objects and relations) and to the nature of the domain knowledge (concepts and their interdependencies).

As concrete results, the paper refers to the on-line hypertext prototype coupled to control systems. This prototype was fully realised only later, but the advantages could be seen at the time of writing the paper. Other small prototypes demonstrating these ideas had already received positive comments from the industrial partners.

9.2.3 Paper V: Explaining design knowledge from means-end models

Paper V takes an explanation view to the topic of papers III and IV. The point is to find ways of explaining purposes and justify design choices related to plant items. A subset of MFM, the means-end analysis, was assumed as the modelling method for this paper.

Section 1 introduces the need for teleological explanations. There we focus our treatment of design knowledge to cover two specific aspects of design knowledge: purposes and justifications. Section 2 details how means-end models encode the knowledge about purposes in the relations between model items. We view explanation as navigation in this model,

offering explanations either explicitly from the objects' properties or by deriving explanations through the relations. Structured text in objects is adopted to encode design knowledge. The algorithm for deriving the explanations through the model is then described.

Section 3 describes how we extended the Design++ tool used in the previous papers to provide explanations of design knowledge. This research prototype convinced us that the principle works as expected in a design environment. As a secondary result, the notion of justifiable objects is defined as a specialisation of explainable objects in the case of design knowledge.

We point out in Section 4 that our representations of design are not sufficient to cover all aspects of design knowledge. Especially justifications of design choices would need more elaborate representations of design rationale. However, the approach works for explaining means-end knowledge. The large number of dependencies encountered at real plants poses few problems, since the search space is reduced during the explanation process. We did realise, however, that several alternative paths in the model and the excess of explanatory details need more work. This point is later developed further in Paper VII.

9.3 SUPPORT SYSTEMS IN HIGH ENERGY PHYSICS

Papers VI and VII deal with the domain of high-energy physics (abbreviated HEP) control systems. They were written as part of the Cicero project at CERN, Europe's central laboratory for particle physics. The different control systems in a physics experiment are similar to those used in industrial plants. Accordingly, control strategies and equipment employed at CERN are similar to those in industry. However, there are subtle differences that affect the operations from the support systems' point of view and set requirements to the modelling and interaction methods that can be used. These two papers examine this topic in more detail.

9.3.1 Paper VI: Human-machine interfaces in high energy physics control systems

Paper VI studies the control systems used at CERN from a user interface point of view. Section 1 introduces the domain of HEP experiments and describes the Cicero project in which context the work for papers VI and VII was carried out. The rest of the paper describes the results of a survey of human-computer interfaces in CERN control systems. The main findings of the survey are:

- Due to the organisation of the experiments, the operators do not really know the systems they should control.

- With their physics education, the users could understand abstract information more easily than their colleagues in industry. This ability should be used to some advantage.
- Shift experts would need help to decode the past and present behaviour of the control systems.
- There are few process models in use at the experiments, which we suspect to cloud the user's mental models. Explicit models should be employed to clarify the processes.
- The physical and functional relationships in the domain should appear in the user interfaces.
- Since the turnover of operators is high in HEP control, it is useless to invest into heavy training. Instead we suggest giving them help on the spot through advanced help systems.

One of the conclusions that the paper reaches is that context-sensitive knowledge-based support systems (in short, explanation) could help with many of the perceived problems. Paper VII discusses this topic in more depth.

9.3.2 Paper VII: Modelling high energy physics control systems

Paper VII aims at reducing the complexity apparent to users by devising knowledge representations that are suitable for HEP support systems. We claim that a representation based on entity-relationship models (semantic networks) is sufficient for many aspects of HEP control, especially for support systems that function in the user assisting mode.

The paper gradually works towards successively richer representations, at each stage motivating the need for further enhancement. As a solution we propose contribution modelling, a technique based on means-end modelling. In line with the Cicero project, we advocate object orientation, and promote the notion of explainable objects as a natural way to partition representations and explanation.

We then discuss a number of possible uses of contribution modelling in the operational context of HEP systems. The various benefits include:

- Familiarity to the HEP community used to object-orientation,
- Explicitness of dependencies,
- Ease of implementation with object databases,
- Capacity to capture design knowledge, and
- Conformance to ongoing object-based standardising work in HEP controls.

In our view, contribution models essentially form a domain-specific semantic network language for describing HEP controls. We also point out some drawbacks: the lack of behavioural knowledge, unsuitability for more detailed diagnosis and simulation, the potential connectedness of the models, the danger of disorientation in models, and the need for a knowledge engineer to construct useful models. Some of these problems are avoided through the loosely coupled nature of the domain knowledge and the assisting interaction mode.

9.4 EXPLAINING AUTOMATION LOGIC (PAPER VIII)

Of all papers included in this thesis, Paper VIII deals most closely with the automation systems' internal operation. We focus our study on an important part of the controls: the control logic that carries out the discrete functions in plants. We conclude that ways of improving the presentations and understandability – in short, explanation – of the logic are needed to alleviate the problems.

The paper proposes various ways of visualising logic solutions into more understandable displays, but stresses that it is also necessary to offer knowledge-based help with the logic. We recognise three kinds of explanations on logic circuits: explaining meaning, explaining behaviour, and speculative help. The paper details a number of examples and their rationale.

Section 3 outlines the architecture and implementation of Lepo, the Lisp-based prototype that was built to demonstrate the ideas. The main features of Lepo were:

- The logic representation was based on explainable objects that formed logic graphs.
- The graphs were traversed to simulate the propagation of logic values through the circuits and to generate explanations.
- Templates were used to generate the contents of explanations.

We anticipated in the paper that it would be difficult to acquire the knowledge to form the logic circuits, that the embedding of the explanations into automation would need more work, and that complexity issues would need to be studied. These three points have since been elaborated with the development of the Lepo C++ prototype to which Chapter 6 of this thesis is devoted.

We contently noted in the paper the posivite interest from the industry towards Lepo. This interest has since grown into a larger research project, also called Lepo, with several industrial partners. Many of the results in the

present thesis have been concretised in this project, guided by the vision given by Paper VIII.

10 CONCLUSIONS

We have studied explanation with relation to plants. We have approached the topic in a constructive manner: we have shown the need for explanations in plants, proposed a conceptual framework for explanation, and demonstrated several prototypes that realise key elements of the framework. Finally, we have positioned the results in a larger context by reviewing related work on explanation and plants. The approach is well visible in the structure of this thesis. This chapter reviews the contributions and summarises topics for further development.

10.1 RESULTS

The main results of this thesis can be summarised as follows:

Problem analysis

- Many problems in plants are related to the lack or abundance of information.
- Design knowledge in particular is missing in plants.
- Support systems and automation would have to be able to explain plant knowledge.

Framework

- Explanations should cover several classes of knowledge at various levels of plant systems.
- Generic explanation tasks can use models as the knowledge source for creating explanations.

Solutions

- Knowledge representations based on objects and their relations form a vocabulary for modelling and presenting plant knowledge.
- A particular class of models based on means-end modelling is useful for explaining design knowledge.
- Graph algorithms can be used to generate explanations through searches in the models.

- Hypertext serves to communicate explanations over dynamic dialogue based on context.

The results are detailed in demonstrations:

- A prototype for explaining the reasoning of an expert system
- A prototype for explaining design knowledge
- A prototype for model-based on-line documentation
- A prototype for explaining control logic

10.2 ANSWERS TO THE RESEARCH PROBLEMS

Referring back to the research problems set in Chapter 1, we can now formulate the answers:

> P_1: What difficulties do people have with technical systems in plants?

The requirements for explanation were obtained through studying the problems that people have in plants. Our studies concentrated on problems that are related either to a lack or an excess of information. Most of our work aims at helping process operators. The needs of the maintenance staff in plants, software developers, plant designers, and experts of specific application domains were also studied. A multitude of plant types were examined to define the problems: power plants, paper machines, steel mills, and high energy physics experiments. In the range of systems studied, we found certain common patterns:

- There is too much information available to be digested in a hurry.
- Many elements of crucial information are nevertheless missing.
- Interdependencies between things are unclear.
- Design criteria for systems are not understood.
- The relevant information in documents is hard to find.
- The functions of systems are difficult to grasp.

In short, people were poorly supported in many of their informational tasks.

P_2: What kinds of knowledge would be needed to solve these problems?

We chose to solve the problems through knowledge-based support systems. Model-based explanation is the particular approach we use to convey the knowledge contained in these systems. As a result of the problem analysis, we proposed to classify the necessary explanations with a two-dimensional framework, where one axis is the class of knowledge being explained (factual, functional, strategic), and the other is the target of explanations (processes, automation, support systems). For each of the nine fields of this model we suggested certain kinds of knowledge that would be potentially explainable.

P_3: How can this knowledge be captured and modelled?

We proposed that the knowledge should be captured into models at the time of design, covering each level of the explanation framework. These models could then be used to carry the knowledge to the users in plants. We stressed that a common conceptual model for all design phases was needed, and proposed object-oriented models based on means-end analysis for this aim.

Our case studies suggest that knowledge representations based on objects and their relations form a domain specific vocabulary for modelling and presenting plant knowledge. We recognised the need for at least structural, behavioural, functional, and teleological models. These models can capture important parts of the knowledge that was found to be missing in plants. We showed mechanisms for recording and explaining design knowledge in the P&ID and Justifier prototypes. The contribution models extend means-end modelling with ways of managing large amounts of dependencies, which helps to tackle an abundance of information.

Another solution to knowledge caption was to derive the knowledge directly from operational systems. This solution was adopted in the LEPO C++ prototype that gets its models directly from the interpreted PLC code.

In all cases the models were based on object networks. The algorithms that create explanations were based on various kinds of search over the graphs formed by these networks. The algorithms implement a number of generic explanation tasks. We used the tasks as a framework for describing the functions that the explanation mechanisms typically incorporate.

P_4: How can this knowledge be communicated to people?

We proposed several techniques for interaction. These techniques were outlined through a number of prototypes. The VICE prototype demonstrated many of the essential techniques. It showed that the notion of explainable objects can be effectively used to organise explanatory knowledge. Dynamic hypertext serves well to communicate explanations, as it corresponds with the structure of knowledge in models. Dialogue can be managed with simple representations of context. Both non-computer experts and knowledge engineers were content with the explanations based on these techniques.

We constructed enhanced access mechanisms for hypertext documentation and carried out a usability study with them. The results obtained from the study suggested that hypertext is more effective in communicating plant knowledge than traditional documentation. Means-end models embedded in the documents helped on understanding of the dependencies between the systems in plants.

Figure 47 summarises the essential techniques for explanation that have been developed in this thesis.

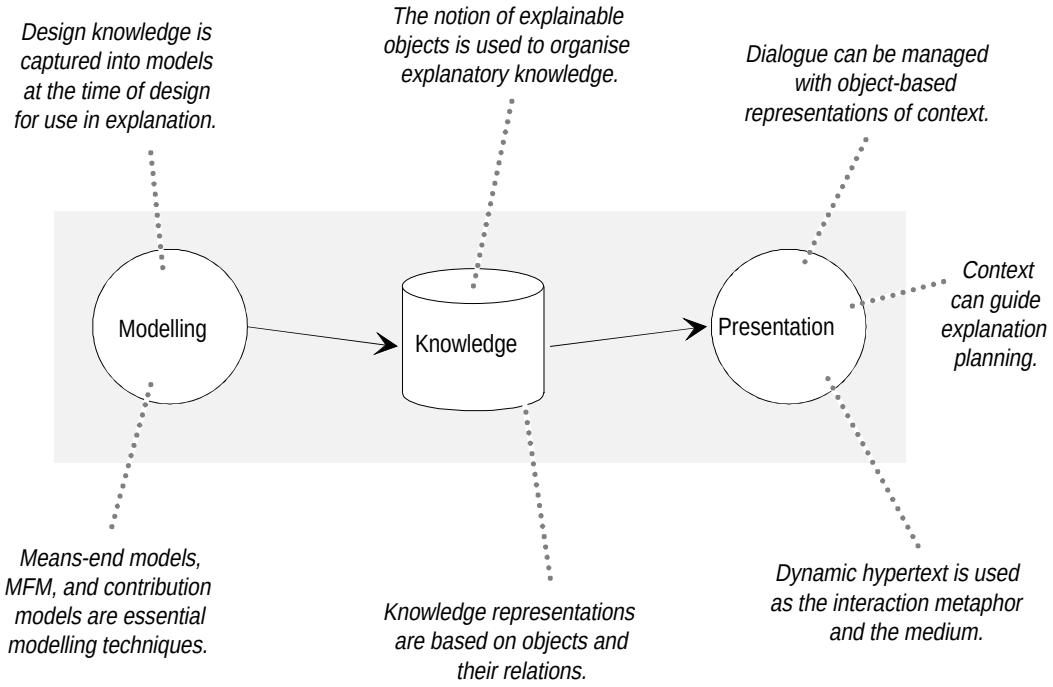

Figure 47. The essential explanation techniques developed in this thesis.

10.3 THE MAIN CONTRIBUTIONS

Many of the techniques presented in our papers and realised in the prototypes have independently appeared as standard techniques: hypertext as dynamically created explanation, the use of context in hypertext links, and the notion of explainable objects.

The industrial interest in explanations is most apparent in the Lepo C++ prototype that is being embedded into a commercial product. Both Lepo prototypes, implemented in Lisp and C++, showed novel ways of explaining the behaviour of automation logic. The two prototypes used many of the explanation techniques developed in this thesis.

The results – the various prototypes, the approval of the users, the results from the field studies, the industrial interest, and the appearance of the techniques in world-wide use – validate our research hypothesis. The main contributions of this thesis can then be listed:

- The problem study shows that explanation is relevant in solving the problems that people have in plants.
- The framework makes explicit the targets and classes of explanation that must be considered in the support systems in plants.
- The generic explanation tasks give a point of reference for analysing and constructing explanations.
- The prototypes demonstrate that knowledge caption, modelling, and communication can be based on object-oriented models that are shown as contexted hypertext.

The framework of explanation presented in this thesis extends the views of explanation and automation research. To our knowledge, this is the first study that covers the field of explanation in plants from all the presented viewpoints.

We believe that these results will help people understand technical systems in plants. We show ways of improving automation with knowledge-based systems. Once the concepts and techniques presented in this thesis become commercially available, the users receive more help for their tasks. With this support, the problems at the use of the plants can decrease. This can in turn increase the safety, quality, and economy of plants.

10.4 DIRECTIONS FOR FUTURE RESEARCH

Our discussion of explanation in plants has revealed a number of questions that have not been completely resolved. Topics that could benefit from further research include:

Knowledge representation and reasoning

- Identification of new generic explanation tasks
- Improved modelling methods for explanations
- Complexity & efficiency analyses of explanatory algorithms
- Inclusion of explicit planning in the explanation creation process
- Application of the blackboard paradigm for creating explanations
- Meta-level explanation on the explanation mechanisms
- Implementation of explainable objects through agent technologies

Interaction

- Identification of proper rhetoric for plant explanations
- Solutions for navigation problems with hypermedia
- Better customisation and dialogue through user modelling
- Display mechanisms in automation systems for plant users
- Investigation of the possibilities of multimedia

Deployment into design and use

- Design tools that better facilitate design rationale capture
- Explanation of experiential knowledge collected in plants
- Cross-fertilisation between electronics design and automation
- Usability studies of explanations in plants

10.4.1 Applications outside the plant domain

The results can be applied in other domains besides industrial plants. Many computer-controlled devices and appliances manifest problems that are very similar to those in plants. We believe that the concepts and techniques shown in this thesis are applicable in those fields. For instance, telecommunication networks can be viewed of as systems to be explained at three key levels (Figure 48).

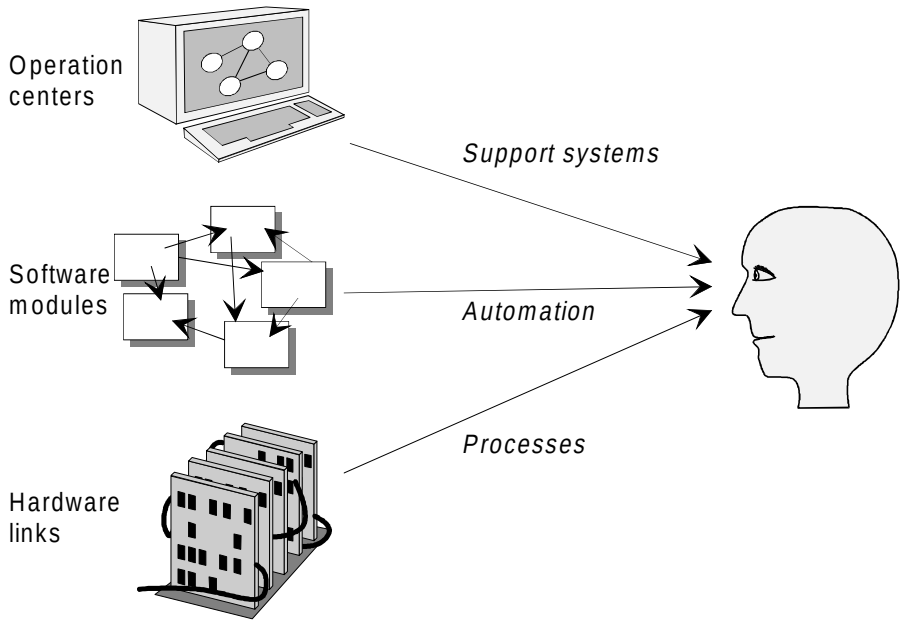

Figure 48. Targets of explanation in telecommunication networks.

We believe that the framework for explanations is applicable outside the plant domain. Its features – explanation types, tasks, and models – can be generalised to several fields. As an example, consider the domains in Table 11. These are some of the areas that are frequently mentioned in technical and scientific publications as candidates for increased computer automation in the near future. For each of these areas, processes, automation, and support systems can be identified that are similar to those found in plants.

Many professions are quickly becoming computer-aided, or even computer-based. The people in these professions will increasingly (sometimes exclusively) carry out their duties with the aid of computers. Therefore *they are becoming operators*. As a consequence, they are very likely to encounter problems that are most similar to those we have found in plants, for instance distancing, information overload, and lack of design knowledge. Explanation techniques, among other solutions, will be needed in these fields to overcome the problems.

The knowledge types of the framework – factual, functional, and strategic – are generic and not tied to the plant domain. In any of the areas mentioned in Table 11, all these types may be needed for the reasons detailed in Section 2.4. We believe that for any computerised system, its users will ultimately want to know how the system works and why it works the way it does.

Table 11. Examples of areas with increasing computer automation.

Domain	**Processes**	**Automation**	**Support systems**
Finance	Investments	Computerised trading	Market analysers & forecasters
Telecommu-nications	Transmission networks	Switches, routers, line equipment	Network management centers
Health care	Human life processes	Pacemakers, scanners, patient databases	Signal/image analysers, diagnosers, remote operations
Road traffic	Traffic flow	Computerised roads, smart cars, route information systems	Traffic analysers, congestion avoidance systems, hazard predictors
News agencies	World events	Databases of facts, stories, images	Translators, search engines, news analysers
Airspace	Flight management	Autopilots, approach systems, collision avoidance	Flight control centers, route planners, safety analysers
Habitation	Energy & safety management	Intelligent buildings with temperature & lighting control, alarm systems, and home networks	Remote surveillance, home control posts, theft detection, environmental optimisers

The explanation tasks that we have defined in the framework are generic enough to function in several domains. For instance, the Abstraction task will suit any domain where there is an abundance of information, yet where the problem space can be structured at several levels of detail. We believe that the central technique for abstraction, means-end analysis, holds much potential for analysing human-designed systems in general, not only plants. Another intriguing possibility would be to apply the Behavioural diagnosis task into control software outside the automation domain.

A central feature of the framework are the models. They are needed for the generic tasks to produce explanations. Therefore the applicability of the framework depends on the availability of models. Fortunately, in many fields a tendency for computer-supported design and maintenance can be seen. Application generators produce important amounts of software. Product and design knowledge is becoming available in design databases, and use time information is increasingly being logged in history databases. Such databases can then be used as the input models for the tasks. The models we propose (structural, functional, behavioural, and teleological) are

generic and, as reported in the artificial intelligence literature, applicable in many domains.

10.4.2 Ubiquitous explanations

Perhaps the most generic concept put forward in this thesis is that of *explainable objects*. We believe that this concept, although seemingly obvious, fundamentally affects the way that knowledge in systems is organised. It could be generalised to all kinds of software (explainable software) or computer-controlled systems in general (explainable systems). Negroponte [1995] seems to envisage this line of development:

> "The future of any appliance is likely to be a stripped-down or buffed-up personal computer. One reason to move in this direction is to make appliances more friendly, usable, and self-explicating. ... The best instructor on how to use a machine is the machine itself. It knows what you are doing, what you have just done, and can even guess at what you are about to do."

Such trends as the ever decreasing size of electronics, the explosion in networking, and the (slowly) rising intelligence level of software, may ultimately lead to the situation where networked controllers are ubiquitous in our environment. Such controllers could then contain explanatory capabilities to talk about themselves and the machines they control. The communication would take place through the information channels available in the home, offices, or plants.

Information appliances [Juliussen 1997] may become an important part of our everyday life in the near future. Such small embedded computers could help in many of the tasks that we now have to carry out more or less manually. The appliances will contain sufficient processing power, memory capacities, and network connectivity to facilitate explanations.

The mechanisms proposed in this thesis have been built for larger platforms, but the underlying techniques hold promise for being useful in portable devices. A fundamental concern for explanation research in the next century will be to embed explanatory capabilities into appliances. Such capabilities may be needed not only in mobile instruments and portable operator consoles found in plants, but also in cars, telephones, watches, and network computers that we wear on ourselves – and ultimately in ourselves.

REFERENCES

Abadir, M.S. & Breuer, M.A. 1985. *A Knowledge-Based System for Designing Testable VLSI Chips.* IEEE Design & Test, Vol. 2, No. 4, August 1985, pp. 56 - 68. ISSN 0740-7475

Adye, T. et al. 1992. *The DELPHI experiment control.* Proceedings of the International Conference on Computing in High Energy Physics ´92, Annecy, France, September 21 - 25, 1992. Geneva, Switzerland: European Organization for Nuclear Research. Pp. 269 - 274. (CERN 92-07).

Alta Vista. *AltaVista: Main Page.* 1996 [cited December 15, 1996] Available in the Internet: <URL:http://www.altavista.digital.com/>

Arnault, C. 1992. *Object Oriented Systems in High Energy Physics.* Proceedings of the International Conference on Computing in High Energy Physics ´92, Annecy, France, September 21 - 25, 1992. Geneva, Switzerland: European Organization for Nuclear Research. Pp. 75 - 80. (CERN 92-07).

Bainbridge, L. 1983. *Ironies of Automation.* Automatica, Vol. 19, No. 6, pp. 775 - 779. ISSN 0005-1098

Bannon, L. 1990. *From Human Factors to Human Actors.* In: Greenbaum, J., Kyng, M. (eds). Design at Work: Cooperative design of computer systems. Hillsdale, NJ: Erlbaum. 294 p. ISBN 0-8058-0611-3

Barentsen, K.B. 1991. *Knowledge and Shared Experience.* In: Proceedings of the Third European Conference on Cognitive Science Approaches to Process Control, Cardiff, U.K., September 2 - 6, 1991. Pp. 217 - 232.

Barillère, R. et al. 1993. *CICERO: Control Information system Concepts based on Encapsulated Real-time Objects – A study on Generic Control Systems for Large Scale LHC Experiments.* Geneva, Switzerland: European Organization for Nuclear Research, December 21, 1993. 22 p. (CERN / DRDC / 93-50).

Barman, D.K. 1992. *Capturing Design Rationale with Semi-Structured Hypertext.* Working notes of the AAAI'92 Workshop on Design Rationale Capture and Use, July 15, 1992, San Jose, CA. Pp. 15 - 21.

Barrow, H.G. 1984. *VERIFY: A Program for Proving Correctness of Digital Hardware Designs.* Artificial Intelligence, Vol. 24, pp. 437 - 491. ISSN 0004-3702

Bending, M.J. 1984. *Hitest: A Knowledge-Based Test Generation System.* IEEE Design & Test, May 1984, pp. 83 - 92. ISSN 0740-7475

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H.F. & Secret, A. 1994. *The World-Wide Web.* Communications of the ACM, Vol. 37, No. 8, August 1994, pp. 76 - 82. ISSN 0002-0782

Berry, D.C. 1995. *Explanation: The Way Forward.* Expert Systems with Applications, Vol. 8, No. 4, pp. 309 - 401. ISSN 0957-4174

Bisantz, A.M. & Vicente, K.J. 1994. *Making the Abstraction Hierarchy Concrete.* International Journal of Human-Computer Studies 40, pp. 83 - 117. ISSN 1071-5819

Biswas, N.N. 1986. *Computer-Aided Minimization Procedure for Boolean Functions.* IEEE Transactions on Computer-Aided Design, Vol. CAD-5, No. 2, April 1986. Pp. 303 - 304. ISSN 0278-0070

Bonfatti, F., Gadda, G. & Monari, P. D. 1995. *Re-usable Software Design for Programmable Logic Controllers.* ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems, La Jolla, CA, June 1995. In: ACM - SIGPLAN Notices 11, Vol. 30, November 1995, pp. 31-40.

Bostick, D., Hachtel, G.D., Jacoby, R., Lightner, M.R., Moceyunas, P., Morrison, C.R. & Ravenscroft, D. 1987. *The Boulder Optimal Logic Design System.* Proceedings of the IEEE international conference on computer-aided design, Santa Clara, CA, November 9 - 12: Digest of technical papers. Washington, DC: IEEE Computer Society Press, 1987. Pp. 62 - 65. ISBN 0-8186-0814-5

Bradshaw, J.A. & Young, R.M. 1991. *Evaluating Design Using Knowledge of Purpose and Knowledge of Structure.* IEEE Expert, Vol. 6, No. 2, April 1991, pp. 33 - 40. ISSN 0885-9000

Brayton, R.K., Rudell, R., Sangiovanni-Vincentelli, A. & Wang, A.R. 1987. *MIS: A Multiple-level Logic Optimization System.* IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 6, November 1987. Pp. 1062 - 1081. ISSN 0278-0070

Brayton, R.K. 1987. *Factoring logic functions.* IBM Journal of Research and Development, Vol. 31, No. 2, March 1987, pp. 187 - 198.

Brayton, R.K., Hachtel, G.D. & Sangiovanni-Vincentelli, A.L. 1990. *Multilevel logic synthesis.* Proceedings of the IEEE, Vol. 78, No. 2, February 1990. Pp. 264 - 300.

Broadbent, D.E., Fitzgerald, P. & Broadbent, M.H.P. 1986. *Implicit and explicit knowledge in the control of complex systems.* British Journal of Psychology 77, pp. 33 - 50.

Brown, M., Moosa, Z., Britton, P. & Filer, N.P. 1993. *An Express Model for Heuristic Classification.* Personal communication.

Buchanan, B.G. & Shortliffe, E.H. 1984. *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley. 748 p.

Buck, L. 1989. *Human operators and real-time expert systems.* Expert Systems, Vol. 6, No. 4, November 1989, pp. 227 - 237.

Butler, H., Myers D.R., Von Rüden, W. & Yang, J. 1993. *Beam-line Operation Using an Industrial Control System and Distributed Object-Oriented Hardware Access.* Geneva, Switzerland: European Organisation for Nuclear Research. 5 p. (CERN/ECP 93-22).

Campione, M. & Walrath, K. 1997. *The Java Tutorial: Object-Oriented Programming for the Internet [online]*. Feb 19, 1997 [cited February 22, 1997]. Mountain View, CA: Sun Microsystems. Available on the Internet: <URL:http://www.javasoft.com:80/nav/read/Tutorial/index.html>.

Card, O.S. 1992. *Speaker for the Dead.* London: Random House. 415 p. ISBN 0-09-950320-4

Carroll, J. M. & Olson, J. R. 1988. *Mental Models in Human-Computer Interaction.* In: Helander, M. (ed.). 1988. Handbook of Human-Computer Interaction. Amsterdam: Elsevier Science Publisher B.V. Pp. 45 - 65. ISBN 0-444-88673-7

Cawsey, A. 1992. *Explanation and interaction: the computer generation of explanatory dialogues.* Cambridge, MA: MIT Press. 232 p. ISBN 0-262-03202-3

Chandra, D.N. 1992. *Innovative design systems, where are we and where do we go from here? Part I: Design by association.* The Knowledge Engineering Review, Vol. 7, No. 3, pp. 183 - 213.

Chandrasekaran, B. 1986. *Generic Tasks in Knowledge-Based Reasoning: High-level Building Blocks for Expert System Design.* IEEE Expert, Vol. 1, No. 3, Fall 1986, pp. 23 - 30. ISSN 0885-9000

Chandrasekaran, B., Tanner, M.C. & Josephson, J.R. 1987. *Explanation: The Role of Control Strategies and Deep Models.* In: Hendler, A. (ed.). Expert Systems: The User Interface. Norwood, NJ: Ablex. Pp. 219 - 247.

Chandrasekaran, B. et al. 1989. *Explaining Control Strategies in Problem Solving.* IEEE Expert, Spring 1989, pp. 9 - 24.

Chandrasekaran, B. & Swartout, W. 1991. *Explanations in Knowledge Systems: The Role on Explicit Representation of Design Knowledge.* IEEE Expert, June 1991, pp. 47 - 49. ISSN 0885-9000

Chandrasekaran, B., Goel, A. & Iwasaki, Y. 1993. *Functional Representation as Design Rationale.* IEEE Computer, Vol. 26, No. 1, pp. 48 - 56.

Clancey, W.J. 1983. *The Epistemology of a Rule-Based Epert System - a Framework for Explanation.* Artificial Intelligence 20, May 1983, pp. 215 - 251. ISSN 0004-3702

Clarke, E.M., Emerson, E.A. & Sistla, A.P. 1986. *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.* ACM Transactions of Programming Languages and Systems, Vol. 8, No. 2, April 1986, pp. 244 - 263. ISSN 0164-0925

Conklin, E. J. 1987. *Hypertext: An Introduction and Survey*. IEEE Computer, Vol. 20, No. 9, pp. 17 - 41.

Conklin, E.J. & Yakemovic, KC B. 1991. *A Process-Oriented Approach to Design Rationale.* Human-Computer Interaction, Vol. 6, No. 3 & 4, pp. 357 - 391.

Crossman, E.R.F.W. & Cooke, J.E. 1974. *Manual Control of Slow-Response Systems*. In: Edwards, E. & Lees, F.P. (eds.). The Human Operator in Process Control. London: Taylor & Francis Ltd. Pp. 51 - 66.

David, J.-M. & Krivine, J.-P. 1989. *Designing Knowledge-Based Systems within Functional Architecture: the DIVA Experiment.* In: Proceedings of the Fifth Conference on Artificial Intelligence Applications, Miami, Florida, March 6 - 10, 1989. IEEE Computer Society Press. Pp. 174 - 180.

Davis, R. 1984. *Diagnostic Reasoning Based on Structure and Behaviour.* Artificial Intelligence, Vol. 24, pp. 347 - 410. ISSN 0004-3702

Davis, R. & Hamscher, W. 1988. *Model-based Reasoning: Troubleshooting.* In: Strobe, H.E. Shrobe (ed.). Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann. Pp. 297 - 346.

De Geus, A.J. & Cohen, W. 1985. *A Rule-based System for Optimizing Combinatorial Logic.* IEEE Design & Test, Vol. 2, No. 4, August 1985, pp. 22 - 32. ISSN 0740-7475

De Kleer, J. 1984. *How Circuits Work.* Artificial Intelligence, Vol. 24, pp. 205 - 280. ISSN 0004-3702

Durkin, J. 1996. *Expert Systems: A View of the Field.* IEEE Expert, Vol. 11, No. 2, April 1996, pp. 56 - 63. ISSN 0885-9000

Engelmore, R.S. & Morgan, A.J (eds.). 1988. *Blackboard Systems.* Wokingham, England: Addison-Wesley Publishing Company. 602 p. ISBN 0-201-17431-6

Enterline, L.L. 1988. *Strategic Requirements for Total Facility Automation.* Control Engineering, Vol. Two, September 1988, pp. 9 - 12.

Falcione, A. & Krogh, B.H. 1993. *Design Recovery for Ladder Logic.* IEEE Control Systems, Vol. 13, No. 2, April 1993, pp. 90 - 98.

Feiner, S.K. & McKeown, K.R. 1991. *Automating the Generation of Coordinated Multimedia Explanations.* IEEE Computer, Vol. 24, No. 10, October 1991, pp. 33 - 41. ISSN 0018-9162

Fikes, R. & Kehler, T. 1985. *The Role of Frame-Based Representation in Reasoning.* Communications of the ACM, Vol. 28, No. 9, September 1985, pp. 904 - 920.

Fikes, R., Gruber, T. & Iwasaki, Y. *The Stanford How Things Work Project* [online]. Palo Alto, CA: Stanford University, September 23, 1994 [cited November 2, 1997]. Available from Internet: <URL:http://www-ksl.stanford.edu/htw/htw-long-overview.html>.

Filer, N.P. & Spink, M.A. 1987. *Knowledge-based Control for VLSI Layout.* In: Proceedings of IEEE International Workshop on AI Applications to CAD Systems for Electronics, Münich, Germany, October 1987. Pp. 119 - 136.

Filer, N.P. & Marshall, R.A.J. 1989. *The Design of a Methodology for the Intelligent Control of Simulation Using the Manchester Simulation Engine.* In: Proceedings of the European Simulation Meeting, Rome, Italy, June 1989. Pp. 163 - 168.

Filer, N.P., Mir, S. & Wray, D. 1991. *Description of a Prototype Knowledge-based Tool Exploiting Design Semantics.* Technical report, Jessi-CAD-Frame deliverable D1.1, Esprit Special Project 5082. 7 p.

Finch, F.E., Stanley, G.M. & Fraleigh, S.P. 1991. *Using the G2 Diagnostic Assistant for Real-time Fault Diagnosis.* European Conference on Industrial Applications of Knowledge-Based Diagnosis, Segrate (Milan), Italy, October 17 - 18, 1991.

Fischer, G., Lemke, A.C. & McCall, R. 1991. *Making Argumentation Serve Design.* Human-Computer Interaction. Vol. 6, No. 3 & 4, pp. 393 - 419.

Flasinski, M. 1994. *Further Development of Zeus Expert System: Computer Science Foundations of Design.* Hamburg, Germany: Deutsches Elektronen-Synchrotron. (DESY 94-048).

Flew, A. 1979. *A dictionary of philosophy.* London: Pan Books. 351 p. ISBN 0-330-25610-6

Franke, D.W. 1991. *Deriving and Using Descriptions of Purpose.* IEEE Expert, Vol. 6, No. 2, April 1991, pp. 41 - 47. ISSN 0885-9000

Garcia, O.N. & Chien, Y.-T. 1991. *Knowledge-based Systems: Fundamentals and Tools.* Los Alamitos, CA, USA: IEEE Computer Society Press. 495 p. ISBN 0-8186-924-4

Gautier, P.O. & Gruber, T.R. 1993. *Generating Explanations of Device Behaviour Using Compositional Modeling and Causal Ordering.* Proceedings of the Eleventh National Conference on Artificial Intelligence. Cambridge, MA: MIT Press. Pp. 264-270. ISBN 0-262-51071-5

Genesereth, M.R. 1984. *The Use of Design Descriptions in Automated Diagnosis.* Artificial Intelligence, Vol. 24, pp. 411 - 436. ISSN 0004-3702

Genesereth, M.R. 1993. *From Dart to Designworld: a chronicle of research on automated engineering in the Stanford Logic Group.* Artificial Intelligence, Vol. 59, pp. 159 - 165.

Gilbert, J.W. & Wilhelm, R.G. 1993. *A Concurrent Object Model for an Industrial Process-Control Application.* Journal of Object-oriented Programming (JOOP), Vol. 6, No. 7, November-December 1993, pp. 35 - 44.

Gilmore, W.E., Gertman, D.I. & Blackman, H.S. 1989. *The User-Computer Interface in Process Control.* London: Academic Press. 313 p.

Gruber, T. R. & Russell, D. M. 1992. *Beyond the Record and Replay Paradigm for Design Rationale Support.* In: Working Notes of the AAAI '92 Workshop on Design Rationale Capture and Use, San Jose, CA, July 15 1992. American Association for Artificial Intelligence. Pp. 111 - 118.

Gruber, T.R. & Gautier, P.O. 1993. *Machine-generated explanations of engineering models: A compositional modelling approach.* Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, 1993. San Mateo, CA: Morgan-Kaufmann, 1993. Pp. 1502 - 1508.

Gruber, T. *HTW Demonstrations [online]*. Palo Alto, CA: Stanford University, March 8, 1995 [cited February 11, 1997]. Available from the Internet: <URL:http://www-ksl.stanford.edu/htw/htw-demos.html>.

Gruber, T.R., Vemuri, S. & Rice, J. *Virtual documents that explain How Things Work: Dynamically generated question-answering documents.* Palo Alto, CA: Stanford University, 1995 [cited February 11, 1997]. Available from the Internet: <URL:http://www-ksl.stanford.edu/people/gruber/virtual-documents-htw/>.

Grudin, J. 1990. *The Computer Reaches Out: The Historical Continuity of Interface Design.* In: CHI'90 Conference proceedings: Empowering People, Seattle, WA, USA, April 1 - 5, 1990. SIGCHI Bulletin Special Issue (April 1990), pp. 261-268. ISSN 0736-6906

Guasch, A., Milne, R. & Sarrate, R. 1995. *Relay Ladder Logic Diagnosis.* In: Artificial Intelligence in Real-time Control 1994. Elsevier Science Publishers. Pp. 275 - 280. ISBN 0-08-042236-3

Halasz, F. 1988. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. Communications of the ACM, Vol. 31 No. 7, pp. 836 - 852.

Hamscher, W.C. 1991. *Modeling Digital Circuits for Troubleshooting.* Artificial Intelligence, Vol 51, No. 1 - 3, October 1991, pp. 223 - 271.

Harmon, P. 1990. *DESIGN++: An Expert System for Automating the Engineering Design Process.* Expert System Strategies, Vol. 6, No. 6, pp. 9 - 12.

Hasling, D.W., Clancey, W.J. & Rennels, G. 1984. *Strategic explanations for a diagnostic consultation system.* International Journal of Man-Machine Studies 20, pp. 3 - 19.

Hayes-Roth, F. & Jacobstein, N. 1994. *The State of Knowledge-Based Sytems.* Communications of the ACM, Vol. 37, No. 3, March 1994, pp. 27 - 39. ISSN 0002-0782

Helander, M. (ed.). 1988. *Handbook of Human-Computer Interaction.* Amsterdam: Elsevier Science Publisher B.V. 1167 p. ISBN 0-444-88673-7

Hong, S.J., Cain, R.G. & Ostapko, D.L. 1974. *MINI: A Heuristic Approach for Logic Minimization.* IBM Journal of Research and Development, Vol. 18, No. 5, September 1974, pp. 443 - 458.

Hoc, J.-M. 1989. *Strategies in controlling a continous process with long response latencies: needs for computer support to diagnosis.* International Journal of Man-Machine Studies 30, pp. 47 - 67. ISSN 0020-7373

Hong, S.J. & Muroga, S. 1991. *Absolute Minimization of Completely Specified Switching Functions.* IEEE Transactions on Computers, Vol. 40, No. 1, January 1991. Pp. 53 - 65. ISSN 0018-9340.

Hughes, J.G. 1991. *Object-oriented Databases.* Hemel Hempstead, England: Prentice Hall International. 280 p. ISBN 0-13-629882-6

Huuskonen, P. 1990. *An Explanation Mechanism for a Diagnostic Expert System Based on Functional Architecture.* In: Djupsund, M., Salonen, P. & Syrjänen, M. (eds.). Proceedings of the Finnish Artificial Intelligence Symposium (STEP-90), Oulu, Finland, June 11 - 14, 1990. Oulu, Finland: Finnish Artificial Intelligence Society & Blanko ry. Pp. 333 - 342.

IEC 1131-3. 1993. *Programmable controllers - Programming languages.* International Electrotechnical Commission. 207 p.

IEEE 1076. 1987. *VHDL Language Reference Manual.* New York: IEEE.

IGES. 1988. *Initial Graphic Exchange Specification IGES Version 4.0.* Washington DC: US Department of Commerce, National Bureau of Standards. 515 p.

Ishack, G. 1993. *Are we making the most of advanced digital control systems?* Nuclear Engineering International, January 1993, pp. 48 - 49.

ISO 10303-1. 1994. *Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles.* Geneve: International Organization for Standardization. 17 p.

ISO 8879. 1986. *Information Processing - Text and Office Systems.* Standard Generalized Markup Language (SGML). Geneve: International Organization for Standardization. 155 p.

Jaako, J. 1996. *The Extension of Multilevel Flow Modelling.* Oulu, Finland: University of Oulu. (Acta Univ. Oul. C 87). ISBN 951-42-4277-7

Jackson, P. 1986. *Explaining Expert System Behaviour.* In: Workshop on Explanation. Alvey IKBS Expert Systems Theme, Guildford, England, March 20 - 21, 1986. London: Alvey Directorate. Pp. 83 - 95.

Jeffery, K. 1996. *Special Issue on the World Wide Web.* ERCIM News No. 25, April 1996. European Research Consortium for Informatics and Mathematics. Pp. 6 - 31.

Johannsen, G., Rijnsdorp, J.E. & Sage, A.P. 1983. *Human-System Interface Concerns in Support System Design.* Automatica, Vol. 19, No. 6, pp. 595 - 603. ISSN 0005-1098

Johannsen, G. 1990. *Towards a New Quality of Automation in Complex Man-Machine Systems.* In: Utkin, V. & Jaaksoo, Ü. (eds.). Proceedings of the IFAC 11th World Congress on Automatic Control in the Service of Mankind. 11. Tallinn, Estonia, Aug 13 - 17, 1990. Tallinn: IFAC. Pp. 435 - 441.

Johannsen, G. & Alty, J.L. 1991. *Knowledge Engineering for Industrial Expert Systems.* Automatica, Vol. 27, No. 1, pp. 97 - 114. ISSN 0005-1098

Jovanovic, A & Maile, K. 1992. *ESR - A Large Knowledge-Based System Project of European Power Generation Industry.* Expert Systems with Applications, Vol. 5, pp. 465 - 477. ISSN 0957-4174

Juliussen, E. 1997. *Computers (1997 Technology Analysis & Forecast).* IEEE Spectrum, January 1997, pp. 49 - 54. ISSN 0018-9235

Kaarela, K., Oksanen, J. & Lukkari, K. 1992. *Laitosprojektin toteuttamisen aikana ilmenevät tiedonsiirto-ongelmat.* Internal project report (not published). Oulu, Finland: VTT Electronics & University of Oulu. 36 p. (In Finnish.)

Kaarela, K. & Oksanen, J. 1994. *Operator Support System Based on an Information Model.* In: Proceedings of the 1994 Symposium on Human Interaction With Complex Systems (HICS-94), Greensboro, NC, USA, September 18 - 20, 1994. Greensboro, NC: NASA-CORE, North Carolina Agricultural & Technical State University, and SPIE. Pp. 156 - 165.

Kaarela, K., Oksanen, J. & Takalo, J. 1995. *An Information Model as a Basis for Hypermedia-Based Plant Documentation.* Computer Networks and ISDN Systems, Vol. 27, pp. 751 - 764.

Kaarela, K. 1996. *Enhancing Communication of Plant Design Knowledge.* Espoo, Finland: VTT. 110 p. + app. 81 p. VTT Publications 272. ISBN 951-38-4930-9, ISSN 1235-0621

Kaindl, H. 1994. *Object-Oriented Approaches in Software Engineering and Artificial Intelligence.* Journal of Object-oriented Programming, Vol. 6, No. 8, January 1994, pp. 38 - 45. ISSN 0896-8438

Katajamäki, M. 1991. *Knowledge-Based CAD.* Expert Systems with Applications, Vol. 3, pp. 277 - 287.

Keravnou, E. & Johnson, L. 1986. *Competent Expert Systems: a Case Study in Fault Diagnosis.* London: Kogan Page Ltd. 320 p.

Keränen, R., Tommila, T. & Heimbürger, H. 1988. *The Safety of Process Automation, Experiences and Methods.* In: Man-Machine Systems: Analysis, Design and Evaluation: Preprints of the IFAC/IFIP/IEA/IFORS Conference, Oulu, Finland, June 14 - 16, 1988, Vol. 1. Helsinki: Finnish Society of Automatic Control. Pp. 202 - 205.

Keuneke, A. 1991. *Device Representation: The Significance of Functional Knowledge.* IEEE Expert, Vol. 6, No. 2, pp. 22 - 25.

Kim, W. 1990. *Object-Oriented Databases: Definition and Research Directions.* IEEE Transactions of Knowledge and Data Engineering, Vol. 2, No. 3, pp. 327 - 341.

Klein, M. 1993. *Capturing Design Rationale in Concurrent Engineering Teams.* IEEE Computer, January 1993, pp. 39 - 47. ISSN 0018-9162

Kobsa, A. & Wahlster, W. 1989. *User Models in Dialog Systems.* Berlin: Springer-Verlag. 471 p. ISBN 3-540-18380-9

Korhonen, R. 1991. *Framework for Improving Quality and Efficiency in Automation Design.* Tampere, Finland: Tampere University of Technology, Publications 85. 113 p. ISBN 951-721-761-7. ISSN 0356-4940

Korteniemi, A. 1989. *An Expert System for Fault Diagnostics of Rotating Machines.* In: Proceedings of the Second Symposium on Expert Systems Application to Power Systems, Seattle, Washington, USA, July 17 - 20, 1989.

Kowalski, R. 1979. *Logic for problem solving.* New York: North-Holland. 287 p. (Artificial Intelligence Series 7). ISBN 0-444-00368-1

Kowalski, A. & Lebensold, J. 1989. *A Diagnostic Aid to Pulp Production.* In: Trivedi, M.M. Applications of Artificial Intelligence VII, Orlando, FL, March 28 - 30, 1989. Bellingham, WA: SPIE. Pp. 858 - 866. (SPIE Proceedings 1095). ISBN 0-8194-0131-5

Kramer, M.A. 1987. *Expert Systems for Process Fault Diagnosis: A General Framework.* In: Reklaitis, G.V. & Spriggs, H.D. (eds). Foundations of computer aided process operations: Proceedings of the First International Conference on Foundations of Computer Aided Process Operations, Part City, Utah, July 5 - 10, 1987. Amsterdam: Elsevier. Pp. 557 - 587. ISBN 0-444-98925-0

Kurki, M. 1995. *Model-based Fault Diagnosis for Mechatronic Systems.* Espoo, Finland: VTT. 116 p. VTT Publications 223. ISBN 951-38-4761-6, ISSN 1235-0621

Laitinen, K. 1995. *Natural Naming in Software Development and Maintenance*. Espoo, Finland: VTT. 99 p. + app. 70 p. VTT Publications 243. ISBN 951-38-4781-0, ISSN 1235-0621

Larsson, J.-E. 1992. *Knowledge-based Methods for Control Systems.* Doctoral thesis. Lund, Sweden: Lund Institute of Technology. 236 p. ISSN 0280-5316

Le Goff, J.-M. 1993. Personal communication.

Le Strugeon, E., Tendjaoui, M. & Kolski, C. 1992. *Knowledge Specification and Representation for an "Intelligent" Interface Devoted to Process Monitoring and Supervision.* In: Preprints of the IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control, Delft, The Netherlands, June 16 - 18, 1992. Delft: IFAC. Pp. 245 - 250.

Lee, J. & Lai, K.-Y. 1991. *What's in Design Rationale?* Human-Computer Interaction, Vol. 6, No. 3&4, pp. 251 - 280.

Lee, J. 1992. *Design Rationale Management Research.* The Knowledge Engineering Review, Vol. 7, No. 4, pp. 363 - 366.

Lees, F.P. 1974. *Research on the Process Operator.* In: Edwards, E. & Lees, F.P. (eds) *"The Human Operator in Process Control"*. London: Taylor & Francis Ltd. Pp. 387 - 425.

Leitch, R. & Gallanti, M. 1992. *Task Classification for Knowledge-Based Systems in Industrial Automation.* IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 1, January/February 1992. Pp. 142 - 152.

Lenz, M., Burkhard, H.-D., Pirk, P., Auriol, E. & Manago, M. 1996. *CBR for Diagnosis and Decision Support.* AI Communications 9 (1996), pp. 138 - 146. ISSN 0921-7126

Lewis, R.W. 1996. *Programming industrial control systems using IEC 1131-3.* London: The Institute of Electrical Engineers. 293 p. (IEE Control Engineering Series 50). ISBN 0-85296-827-2

Lind, M. 1990. *Representing Goals and Functions of Complex Systems - An Introduction to Multilevel Flow Modeling.* Institute of Automatic Control Systems, Technical University of Denmark, Report No. 90-D-38. 86 p. ISBN 87-87950-52-9

Malhotra, P. & Seviora, R.E. 1992. *Object Oriented Framework for Generating Machine Understanding of a Digital System Design.* In: Belli, F. & Radermacher, F.J. (eds.) Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: 5th International Conference (IEA/AIE-92), Paderborn, Germany, June 9 - 12, 1992. Berlin: Springer-Verlag. Pp. 690 - 700. ISBN 0-387-55601-X

Malik, A.A., Brayton, R.K., Newton, A.R. & Sangiovanni-Vincentelli, A.L. 1988. *A Modified Approach to Two-level Logic Minimization.* Proceedings of the IEEE international conference on computer-aided design (ICCAD-88), Santa Clara, CA, November 7 - 10, 1988. Digest of technical papers. New York: IEEE. Pp. 106 - 109. ISBN 0-8186-0869-2

McCoy, K.F. 1989. *Highlighting a User Model to Respond to Misconceptions.* In: Kobsa, A. & Wahlster, W. (eds.) 1989. User Models in Dialog Systems. Berlin: Springer-Verlag. Pp. 233 - 254. ISBN 3-540-18380-9

McKeown, K.R., Wish, M. & Matthews, K. 1985. *Tailoring Explanations for the User.* In: Aravind, J. (ed.). Proceedings of the Ninth International Joint Conference of Artificial Intelligence (IJCAI'85), August 18 - 23, 1985. Los Angeles, CA: Morgan Kaufmann Publishers. Pp. 794 - 798. ISBN 0-934613-02-8

Medvedev, G. 1991. *The Truth About Chernobyl.* London: I.B.Tauris & Co Ltd. 274 p. ISBN 1-85043-331-3

Meyer, B. 1988. *Object-oriented Software Construction.* New York, NY: Prentice Hall. 534 p. ISBN 0-13-629031-0

Milne, R. & Guasch, A. 1994. *Automatic Diagnostic Development Based on a Programmable Logic Controller.* In: Applications and Innovations in Expert Systems II. SGES Publications. Pp. 99 - 110. ISBN 1-899621-00-8

Milne, R., Nicol, C., Travé-Massuyès, L. & Quevedo, J. 1996. *TIGER: knowledge-based gas turbine condition monitoring.* AI Communications, Vol. 9, No. 3, 1996, pp. 92 - 108. ISSN 0921-7126

Mir, S. 1994. *Heuristic Reasoning for an Automatic Commonsense Understanding of Logic Electronic Design Specifications.* Technical report series UMCS-94-4-2, Department of Computer Science, University of Manchester, England. 246 p. Also available in anonymous ftp in ftp.cs.man.ac.uk, directory pub/TR.

Mittal, V.O. & Paris, C.L. 1995. *Generating Explanations in Context: The System Perspective.* Expert Systems With Applications, Vol. 8, No. 4, 1995, pp. 491 - 503. ISSN 0957-4174

Moon, I., Powers, G.J., Burch, J.R. & Clarke, E.M. 1992. *Automatic Verification of Sequential Control Systems Using Temporal Logic.* AIChE Journal, Vol. 38, No. 1, January 1992, pp. 67 - 75.

Moon, I. 1994. *Modeling Programmable Logic Controllers for Logic Verification.* IEEE Control Systems, Vol. 14, No. 2, April 1994, pp. 53 - 59. ISSN 0272-1708

Moore, J.D. 1989. *A Reactive Approach to Explanation in Expert and Advice-Giving Systems.* Doctoral thesis, University of California, Los Angeles, CA. 332 p.

Moore, J.D. & Swartout, W.R. 1990. *Pointing: A Way Toward Explanation Dialogue.* In: Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), Vol. 1, Boston, MA, USA, July 29 - August 3, 1990. Menlo Park: AAAI Press. Pp. 457 - 464. ISBN 0-242-51057-X

Moore, J.D. & Mittal, V.O. 1996. *Dynamically Generated Follow-up Questions.* IEEE Computer, Vol. 29, No. 7, July 1996, pp. 75 - 86. ISSN 0018-9162

Mould, R.F. 1988. *Chernobyl – The Real Story.* Oxford, U.K: Pergamon Press. 255 p. ISBN 0-08-035719-9

Neches, R., Swartout, W.R. & Moore, J.D. 1985. *Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development.* IEEE Transactions on Software engineering, Vol. SE-11, No. 11, November 1985. Pp. 1337 - 1351. ISSN 0098-5589

Negroponte, N.P. 1995. *Being Digital.* New York: Random House. 255 p. ISBN 0-679-43919-6

Nielsen, J. 1990. *The Art of Navigating Trough Hypertext.* Communications of the ACM, Vol. 33, No. 3, March 1990, pp. 296 - 310.

Nilsson, N.J. 1982. *Principles of Artificial Intelligence. Berlin: Springer-Verlag.* 476 p. ISBN 3-540-11340-1

Norman, D. A. 1983. *Some Observations on Mental Models.* In: Gentner, D. & Stevens, A. (eds.) Mental Models. Hillsdale, NJ: Lawrence Erlbaum Associates. Pp. 7 - 14. ISBN 0-89859-242-9

Norman, D.A. 1988. *The Psychology of Everyday Things.* New York: Basic Books. 257 p. ISBN 0-465-06709-3

Olsson, G. 1993. *Operator-Process Interaction is more than HCI.* In: Smith, M. J. & Salvendy, G. (eds.) Human-Computer Interaction: Applications and Case Studies. Proceedings of the Fifth International Conference on Human Computer Interaction (HCI '93), Orlando, Florida, August 8 - 13, 1993. Vol. 19A. Amsterdam: Elsevier Science. ISBN 0-444-89540-X, ISSN 0921-2647

Oxman, S. 1993. *Knowledge-Based Systems in Manufacturing.* Intelligent Software Strategies, Vol. 9, No. 12, December 1993, pp. 7 - 13. ISSN 1052-7214

Paanasalo, J. 1996. Private communication.

Paris, C.L. 1987. Combining Discourse Strategies to Generate Descriptions to Users along a Naive/Expert Spectrum. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI'87), Milan, Italy, 1987. Los Altos, CA: Morgan Kaufmann. Pp. 626 - 632.

Paris, C.L. 1993. *User Modelling in Text Generation.* London, U.K.: Pinter Publishers. 205 p. ISBN 0-86187-809-4

Paunonen, H. 1993. *Developing the Operator's Job Through Automation Products.* In: Proceedings of the 12th European Annual Conference on Human Decision Making and Manual Control, Kassel, Germany, June 1993. Pp. 1 - 6 of Session 9.

Paunonen, H. 1995. *Decision making tools for changing paper production organizations.* In: The First Ecopapertech. An international conference on papermaking and paper machine technology, Helsinki, Finland, June 6 - 9, 1995. Jyväskylä, Finland: Gummerus Kirjapaino Oy. Pp. 437 - 448. ISBN 952-90-6454-3

Peach, D. 1994. *The L3 Gas System Operators Manual.* CERN internal document. Geneva, Switzerland: CERN, ECP Division. 120 p.

Perkins, W.A. & Austin, A. 1990. *Adding Temporal Reasoning to Expert-System-Building Environments.* IEEE Expert, February 1990, pp. 23 - 29. ISSN 0885-9000

Rasmussen, J. 1985. *The Role of Hierarchical Knowledge Representation in Decisionmaking and System Management.* IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC - 15, No. 2, pp. 234 - 243.

Rasmussen, J. 1986. *Information Processing and Human-Machine Interaction.* Amsterdam: North-Holland. 215 p. ISBN 0-444-00987-6

Rautila, E. (ed). 1992. *Prosessin hallinta – Automaation tehtäväkuvaus.* (Process Management – Automation Task Description). Helsinki: Suomen Automaation Tuki Oy. 128 p. ISBN 951-96567-0-7. (In Finnish.)

Rettig, M. 1993. *Cooperative software.* Communications of the ACM, Vol. 36, No. 4, April 1993, pp. 23-28.

Rich, E. 1989. *Stereotypes and User Modeling.* In: Kobsa, A. & Wahlster, W. (eds) 1989. User Models in Dialog Systems. Berlin: Springer-Verlag. Pp. 35 - 51. ISBN 3-540-18380-9.

Riitahuhta, A. 1988. *Enhancement of the Boiler Design Process by the Use of Expert System Technology.* Acta Polytechnica Scandinavica, Mechanical Engineering Series No. 92. Helsinki: Finnish Academy of Technology. 122 p. ISBN-951-666-272-2. ISSN 001-687X

Rivlin, E., Botafogo, R. & Schneiderman, B. 1994. *Navigating in Hyperspace: Designing a Structure-based Toolbox.* Communications of the ACM, Vol. 37, No. 2, February 1994, pp. 87 - 96. ISSN 0001-0782

Rowan, D. 1989. *On-line Expert Systems in Process Industries.* AI Expert, August 1989, pp. 30 - 38.

Rubinoff, R. 1985. *Explaining Concepts in Expert Systems: the CLEAR System.* In: Proceedings of the 2nd conference on Artificial Intelligence Applications, Miami Beach, FL, December 11 - 13, 1985. Los Alamitos: IEEE Computer Society Press. Pp. 416-421.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorenser, W. 1991. *Object-oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice-Hall, Inc. 500 p. ISBN 0-13-630054-5

Rytoft, C., Johansson, B., Bladh, K. & Hoggard, N. 1990. *Aspects of Future Process Control Systems.* Control Engineering, Vol. 37, No. 9, pp. 135 - 138.

Sachs, P.A., Paterson, A.M. & Turner, M.H.M. 1986. *Escort - an expert system for complex operations in real time.* Expert Systems, Vol. 3, No.1, January 1986, pp. 22 - 29.

Sassen, J.A.M. 1993. *Design Issues of Human Operator Support Systems.* Doctoral thesis. Delft, The Netherlands: Delft University of Technology. 226 p. ISBN 90-370-0090-8

Schneiderman, B. 1987. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Reading, MA: Addison-Wesley. 448 p. ISBN 0-201-16505-8

Sedgewick, R. 1989. *Algorithms.* Second edition. Reading, MA: Addison-Wesley Publishing Company. 650 p. ISBN 0-201-06673-4

Seppänen, V. 1990. *Acquisition and Reuse of Knowledge to Design Embedded Software.* Espoo, Finland: VTT. 218 p. + app. 10 p. VTT Publications 66. ISBN 951-38-3579-0, ISSN 0358-5069

Sheridan, T.B. & Ferrell, W.R. 1974. *Man-Machine Systems: Information, Control, and Decision Models of Human Performance.* Cambridge, MA: MIT Press. 452 p. ISBN 0-262-19118-0

Sheridan, T.B., Vámos, T. & Aida, S. 1983. *Adapting Automation to Man, Culture, and Society.* Automatica, Vol. 19, No. 6, pp. 605 - 612. ISSN 0005-1098

Sheridan, T.B. 1988. *Task Allocation and Supervisory Control.* In: Helander, M. (ed.). Handbook of Human-Computer Interaction. Amsterdam: Elsevier Science Publisher B.V. Pp. 159 - 173. ISBN 0-444-88673-7

Singh, N. 1987. *An Artificial Intelligence Approach to Test Generation.* The Kluwer international series in engineering and computer science; 19. Norwell, MA: Kluwer Academic Publishers. 193 p. ISBN 0-89838-185-1

Sleeman, D. & Brown, J.S (eds.). 1982. *Intelligent Tutoring Systems.* London, U.K: Academic Press. 282 p.

Slotnick, S.A. & Moore, J.D. 1995. *Explaining Quantitative Systems to Uninitiated Users.* Expert Systems With Applications, Vol. 8, No. 4, pp. 475 - 490. ISSN 0957-4174

Smith, J.M. 1992. *SGML and Related Standards: Document Description and Processing Languages*. Chichester, England: Ellis Horwood Limited. 151 p. ISBN 0-13-806506-3

Snaprud, M. & Kaindl, H. 1992. *Knowledge Acquisition Using Hypertext.* Expert Systems With Applications, Vol. 5, No. 3 & 4, pp. 369 - 375.

Stassen, H. G., Johannsen, G. & Moray, N. 1988. *Internal Representation, Internal Model, Human Performance Model and Mental Workload.* In: Preprints of the IFAC/IFIP/IEA/IFORS Conference on Man-Machine Systems, Oulu, Finland, June 14 - 16, 1988. Vol. 1. Finnish Society of Automatic Control and International Federation of Automatic Control. Pp. 252 - 261.

Steels, L. 1990. *Components of Expertise.* AI Magazine, Summer 1990, pp. 28 - 49. ISSN 0738-4602

Stein, R. 1991. *Browsing Through Terabytes.* Byte Magazine, May 1991, pp. 157 - 164.

Stephanopoulos, G. 1987. *Intelligent Systems for Process Operations Overview.* In: Proceedings of the First International Conference on Foundations of Computer Aided Process Operations, Park City, Utah, July 5 - 10, 1987. Amsterdam: Elsevier. Pp. 503 - 555.

Stephanopoulos, G. 1990. *Artificial Intelligence in Process Engineering – Current State and Future Trends.* Computers in Chemical Engineering, Vol. 14, No. 11, pp. 1259 - 1270. ISSN 0098-1354.

Stephens, G.L. 1992. *Advanced process management – a vision for the not too distant future.* Tappi Journal, March 1992, pp. 127 - 131.

Suitiala, R. 1993. *Work-oriented Development of Interactive Software Tools. Understanding the Work of Software Maintainers and Making an Interactive Software Tool for Them.* Espoo, Finland: VTT. 176 p. + app. 10 p. VTT Publications 139. ISBN 951-38-4257-9, ISSN 1235-0621

Swartout, W.R. 1983. *XPLAIN, a System for Creating and Explaining Expert Consulting Programs.* Artificial Intelligence 21, September 1983, pp. 285 - 325.

Swartout, W., Paris, C. & Moore, J. 1991. *Explanations in Knowledge Systems: Design for Explainable Expert Systems.* IEEE Expert, June 1991, pp. 58 - 64. ISSN 0885-9000

Takalo, J. 1995. *Enhancing Access to Technical On-line Documentation Through Information Modelling and Retrieval Techniques.* Master's Thesis. Oulu, Finland: University of Oulu, Department of Information Processing Science. 81 p.

Tanner, M.C. & Keuneke, A.M. 1991. *The Roles of the Task Structure and Domain Functional Models.* IEEE Expert, June 1991, pp. 50 - 57. ISSN 0885-9000

Thatcher, V. S. (ed). 1965. *The New Webster Encyclopedic Dictionary of The English Language: International Edition.* New York: Grolier Inc. 972 p.

Tyrväinen, P. 1994. *Domain Modelling for Technical Documentation Retrieval.* Helsinki: Finnish Academy of Technology. 171 p. (Acta Polytechnica Scandinavica, Mathematics and Computing in Engineering Series No. 64). ISBN 951-666-406-7, ISSN 1237-2404.

Van de Ree, R. V. 1994. *SCWERE: Supervisory Control Systems.* Doctoral thesis. Delft, The Netherlands: Delft University of Technology. 214 p. ISBN 90-900-7028-1

Weich, R. E. 1992. *Utilizing CALS Standards to Achieve Effective Information Management.* CALS Journal, Vol. 1, No. 4 (Winter 1992), pp. 47 - 52.

Weisang, C. & Zinser, K. 1992. *GRADIENT - an intelligent, knowledge-based system for industrial process control.* ABB Review 5, 1992, pp. 11 - 16.

Wexelblat, R.L. 1989. *On Interface Requirements for Expert Systems.* AI Magazine, Vol. 10, No. 3, Fall 1989, pp. 66 - 78. ISSN 0738-4602

Vicente, K.J. 1992. *Multilevel Interfaces for Power Plant Control Rooms I: An Integrative Review.* Nuclear Safety, Vol. 33, No. 3., July-September 1992, pp. 381 - 397.

Wick, M.R., Thompson, W.B. & Slagle, J.R. 1988. *Knowledge-based Explanation.* Technical report 88-24, Computer Science Dept., Univ. of Minnesota, Minneapolis, USA. 21 p.

Wick, M.R. & Slagle, J.R. 1989. *An Explanation Facility for Today's Expert Systems.* IEEE Expert, Vol. 4, No. 1, Spring 1989, pp. 26 - 36. ISSN 0885-9000

Winston, P. H. 1993. *Artificial intelligence*. Third edition. Reading, MA: Addison-Wesley Publishing Company. 737 p. ISBN 0-201-53377-4

Woods, W.A. 1986. *Important Issues in Knowledge Representation.* Proceedings of the IEEE, Vol. 74, No. 10, October 1986. Pp. 1322 - 1334.

Yang, S. & Ciesielski, M.J. 1991. *Optimum and Suboptimum Algorithms for Input Encoding and Its Relationship to Logic Minimization.* IEEE Transactions on Computer-Aided Design, Vol. 10, No. 1, January 1991. Pp. 4 - 12. ISSN 0278-0070

Zadeh, L.A. 1984. *Making Computers Think Like People.* IEEE Spectrum, August 1984, pp. 26 - 32.

Zinser, K. & Frischenschlager, F. 1994. *Multimedia's Push Into Power.* IEEE Spectrum, July 1994. Pp. 44 - 48. ISSN 0018-9235

Årzen, K.-E. 1991. *Knowledge-based Applications in the Process Industry: Current State and Future Directions.* In: Proceedings IFAC Workshop on Computer Software Structures Integrating AI/KBS in Process Control, Bergen, May 29 - 30, 1991.

Årzen, K.-E. 1993. *Using multi-view objects for structuring plant databases.* Intelligent Systems Engineering, Autumn 1993, pp. 183 - 200.