LiTH-IDA-R-88-17

ISSN-0281-4250

INSTITUTIONEN FÖR DATAVETENSKAP
UNIVERSITETET OCH TEKNISKA HÖGSKOLAN
LINKÖPING

Multi-Level Software Architectures - Industrial Experiences and Research Directions

by

Bengt Lennartsson

RESEARCH REPORT
PELAB, August 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

The Department of Computer and Information Science
Linköping University

PhD theses:

* Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

(*Linköping Studies in Science and Technology. Dissertations.*)

*No 77	**Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982.
No 94	**Hans Lunell:** Code Generator Writing Systems, 1983.
No 97	**Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983.
*No 109	**Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984.
*No 111	**Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984.
*No 155	**Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons. 1987.
No 165	**James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning. 1987.
No 170	**Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems. 1987.

Licentiate of engineering Theses:

(*Linköping Studies in Science and Technology. Theses.*)

No 48	**Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching. 1985.
No 52	**Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems. 1985.
No 60	**Johan Fagerström:** Simulation and Evaluation of an Architecture based on Asynchronous Processes. 1985.
No 71	**Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System. 1987.
No 72	**Tony Larsson:** On the Specification and Verification of VLSI Systems. 1986.
No 73	**Ola Strömfors:** A Structure Editor for Documents and Programs. 1986.
No 74	**Christos Levcopoulos:** New Results about the Approximation Behaviour of the Greedy Triangulation. 1986.
No 104	**Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology. 1987.
No 108	**Rober Bilos:** Incremental Scanning and Token-based Editing. 1987
No 111	**Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments. 1987.
No 113	**Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge. 1987.
No 118	**Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987
No 126	**Dan Strömberg:** Transfer and Distribution of Application Programs. 1987
No 127	**Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems. 1987
No 139	**Christer Bäckström:** Reasoning about Interdependent Actions. 1988.
No 140	**Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing. 1988.

LiTH-IDA-R-88-17 ISSN-0281-4250

Multi-Level Software Architectures - Industrial Experiences and Research Directions

by

Bengt Lennartsson

Abstract: This paper presents some experiences in Swedish industry from using multi-level software architectures in a few application areas. The idea behind the multi-level architecture is to start by developing an application modelling tool tuned to the specific application area. The design and implementation of this modelling tool is a substantial part of the total system development effort.
The objective is to find a model where the computer scientist and the application domain expert can contribute to system development without having to learn details from each other's domains. The experience, so far, is that the model allows the software specialists to concentrate on software design and development, and that the application specialists are able to tune and maintain the systems mainly by themselves. Some of our research efforts in the area are also presented.

This work has been supported by STU, The Swedish Board for Technical Development, and by Mekanförbundet, The Trade Association of the Swedish Mechanical and Electrical Engineering Industry.

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

1. Introduction

As the area *Computer-Aided Software Engineering* is very wide, there are many different views of central problems and methods. The view this paper is based on research in the programming environment community, and on the current situation in Swedish industry, in particular in branches like process control, telecommunication systems, and defense systems.

There are also different traditions in different countries. Swedish industry is in general concentrating on developing and marketing large systems. The basic methods for software project management has in general been inherited from what was used when the systems were mainly built in hardware. There is some demand for better support for software management, but not at all to the same degree as in US. The efforts on developing 'Integrated Project Support Environment' in the English style is not a hot issue in Swedish industry. In many countries in Europe there is a strong emphasis on formal methods. This trend is rather week in Sweden.

When reading current software engineering literature one gets the impression that there is always a strict separation of specification and design from the coding activity. In many Swedish companies it is considered normal that groups responsible for coding also designs subsystems, and even that they are designing when coding. My references are mainly from companies designing their systems themselves, that is, from companies deciding what they intend to develop and will try to sell on the market. The situation may be different in software houses contracted for development of pre-specified systems. There are not very many such software houses in Sweden, however.

This paper presents some observations, experiences, and results from interaction between our academic research at the Department of Computer and Information Science and activities in Swedish industry. We have basically two channels for this interaction. Firstly, the knowledge transfer program [1], [2], where interesting companies 'subscribe' on a ticket to our research groups. The participating companies can send persons, normally half time, to be 'observers' in our ordinary research groups. They get supervision and engineer support to about the same degree as our graduate students. The objective of the participating companies is normally to get early access to and to evaluate new methods or new tools.

The second channel for industry-academia interaction is via The Trade Association of the Swedish Mechanical and Electrical Industry, *Mekanförbundet*, or Mekan for short. We have been running as series of Mekan projects during the last years. The idea is to have one researcher as conductor for the study, and a support group of engineering managers from industry. A project can consist of analyzing whether current or future technology could solve specific identified problems in industry, or of reviewing selected industrial activities with potential new technology in mind. The result of such a project is twofold. Most important is the exchange of ideas and experiences during the work and, secondly, the work is also presented in a report.

This paper is an attempt to summarize the experiences from two such Mekan projects on *Future Techniques for Software Systems Design.* In the next

paragraph there is a short presentation of a model where we are trying to optimally exploit abilities of both software specialists and application domain experts. Then a sketch of a software architecture appropriate for this model follows. In the third paragraph there is a short presentation of some industrial experiences from partial use of this model and architecture. Finally there is a presentation of some of our research efforts inspired by the Mekan projects.

2. The Roles of the Software Specialist and of The Domain Expert

The view of software engineering in this paper is based on the following assumptions:

- a software system is to developed to meet some need in some application domain
- highly skilled software specialists are available for system design and implementation
- there are application domain experts, possibly the intended users, who knows the application domain and its needs better than the software specialists do

The situation can be illustrated as a two dimensional space with degree of formalism in the horizontal direction and degree of abstraction in the vertical direction.

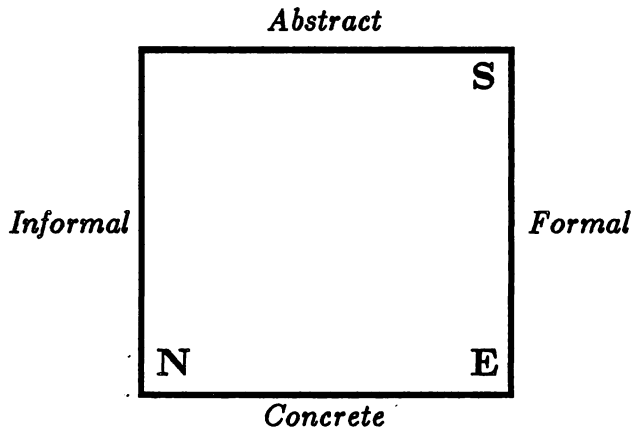


Figure 1. System Requirement-Specification-Implementation Space

A system is to be developed to match the need N in a concrete and often un-formalized situation. To understand the need N is often very difficult, in particular when software systems are introduced in areas where computers have not been used at all before.

The ultimate result of the system development is the executable code **E**. If the end user, when **E** is delivered, finds that the functionality and performance is exactly as expected, then the process has been successful. At least in the short run. In general the user's need will change, so a mechanism to update **E** according to changes in the need **N** would be helpful.

Our view is that:
Given **N** and available resources and constraints, Computer-Aided Software Engineering, CASE, should fully support specification, design, implementation and maintenance of **E**, and also support management of the whole process.

The classical model is to start by analyzing **N**, to develop a requirement specification and later a functional specification. It is often assumed that system design and implementation will be based on some kind of abstract formal specification **S**. Most tools and methods mainly support the process of transforming **S** to **E**, and this process will in general not involve the end-user group or domain experts. All relevant facts about the need **N** should, in this model, be captured by the specification **S**.

An alternative model, often called *experimental system development* or *prototyping*, is based on design iterations. Several versions, or partial implementations, of **E** are developed, and feedback from the application domain is used to modify **E**. This process is mainly aiming at better understanding of the need **N**. When, or if, this has been achieved it is common to to use the classical machinery and proceed via a specification **S**.

A third model, used very successfully in some special cases, is to build what is common to a class of applications into a modelling tool. Several techniques can be used:

- for compiler development scanner, parser, and pretty-printer generators, etc. are frequently used
- knowledge based systems are often developed by means of *shells* containing some inference machinery, user interfaces and the like
- standardized office applications can be generated by application generators
- commonly used subfunctions can be retrieved from a well documented and tested set of library functions
- special purpose languages tuned to certain application domains can be developed.

This paper presents some ideas and experiences on using this third model as a more general approach for some new application areas.

In our Programming Environments Laboratory we have also recently started research projects aiming at investigating a CASE environment tuned to what we call *Multi-Level Software Architectures*. Our approach is to base SE tools on the characteristics of the software process rather than vice versa. As the software process we are currently studying is quite different from that in the the classical life-cycle model, so is our view of appropriate tools, models, and procedures.

3. Multi-Level Software Architectures

By tradition development of operating systems, compilers and other utilities, data base management systems and the like has been considered to be difficult. The best software specialists available, system designers and programmers, have been used for this. Application development has been looked upon as a simpler task. Some training in the programming language used and maybe some course on a 'development method' has been a sufficient background for application developers. Programming languages like Fortran and Cobol initially intended for coding moderate size systems in specific domains have been widely used in other domains, also for very big systems.

Another approach is to use highly skilled computer scientists, not only for 'minor' development of compilers and other general purpose tools but also for the design and implementation of applications. The computer scientist is able to use higher levels of abstraction, think in terms of more powerful operations, etc. The traditional programming languages are too restricted and too close to the machine to be a suitable notation or to supply a suitable set of concepts to the software specialist's design activity.

There are at least three different ways to exploit the computer scientist's ability to work at a higher level of abstraction:

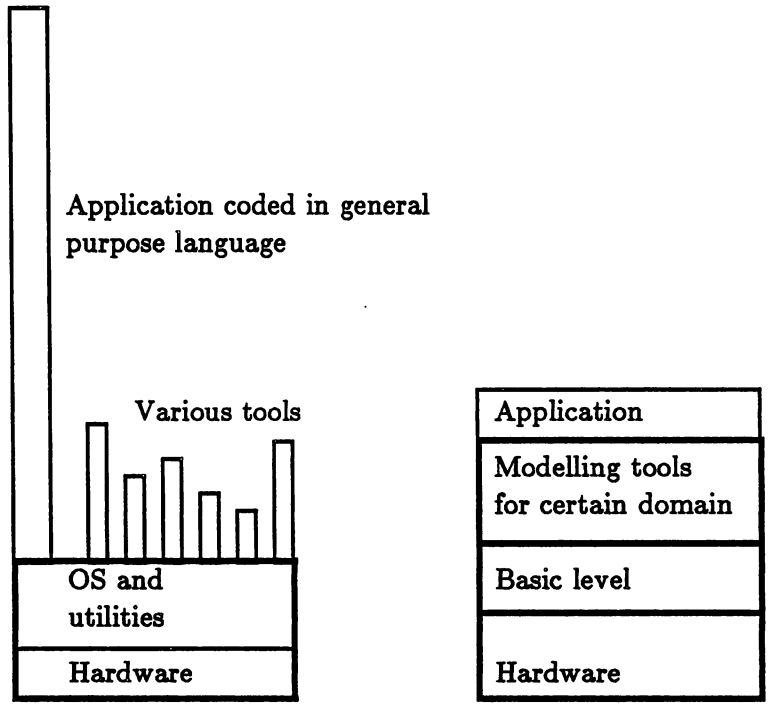
- **The formal specification approach:** somehow the computer scientist is able to understand the expected functional behavior of the system to be developed and produces an abstract formal specification, which is then used as the basis for design and implementation. The application domain knowledge is to be expressed through this formal specification.

- **The very high level language approach:** If the 'programming' is done by people with a background in modern computer science, it is possible to use more powerful and abstract executable languages for the implementation. SETL and REFINE are such examples. This approach supports prototyping, since there is no tedious manual work from the high level specification to execution.

- **The multi-level approach:** The computer science specialists develop tools such that the application domain specialists can supply their application domain knowledge in their own terms. These 'tools' interpret or compile this 'application specification'.

For the very large systems the problem of structuring and of understanding the system and the dependencies between its subcomponents become critical. The best software scientists and engineers available are needed for this. At the same time, large systems can be assumed to solve difficult problems in the application area. This means that the best possible experts from the application domain are needed too. Our basic hypothesis is that in many situations it will pay to spend a considerable part of the development resources on developing tailor-made application modelling tools. The objective is to find

a model where the software specialist builds a platform for the domain expert. The latter shall be able to communicate with the system in notions familiar to him. Objects, relations, and operations on this level should be oriented towards the application area.



Traditional System *Multi-Level Architecture*

Figure 2.

When using the multi-level approach, the specific application is small and easy to update and maintain. Software is re-used, not in terms of library modules but in terms of a re-usable tool. Application generators and 4GL systems have proved very productive in standard office applications. The success of expert system techniques may originate from the multi-level architecture of the systems developed rather than from some magic built-in knowledge.

The software specialist developing the tools can handle a variety of software views. It is reasonable to think of the basic level in the multi-level architecture as giving support for functional, procedural as well as object oriented programming and logic programming. These four paradigms will occur in new combinations.

4. The Mekan Study

Mekanförbundet, The Trade Association of the Swedish Mechanical and Electrical Engineering Industry, has supported two studies on *Future Techniques for System Design in Software* [3], [4]. Both projects have been conducted by the author, and the work has consisted of meetings at some companies and reviewing selected software engineering efforts in progress at each company. The review group has consisted of software managers from the companies involved. In the first project we made a survey of potentially available new software architectures, new development models and methods, and new support environments.

As a background to what we considered 'potentially available' there were two basic facts. Firstly, the occurence of powerful workstations communicating over local area networks considerably moves the optimal boarder between what could and should be done by a software engineering environment and what should be done by its user. Secondly, and this applies in particular to Sweden, the output of computer scientists having a modern undergraduate education from universities has increased dramatically. The key issue was how to exploit these basic facts in organizations heavily loaded by maintaining old traditional software systems. Our proposal in the first project was the Multi-Level Software Architecture model presented briefly in previous paragraph.

4.1 Experiences from use of the model

In the second project, that started one year after the first was completed, we reviewed a number of uses of the multi-level idea. This review has just beeen completed. The main results and conclusions are presented here.

In four companies, Asea Brown Boveri Automation, Ericsson Radar Electronics, SattControl, and Philips Elektronikindustrier, we found that the idea of separating the computer scientist's system building activities from the domain expert's tuning for specific deliveries and installations had been exploited to some degree. Quite different techniques had been used though.

Asea Brown Boveri has a system for process control and monitoring, where the process engineer can use some hundred predefined functions available in subroutine libraries. He has a graphical user interface which enables him to select and instantiate those functions and to connect signals between them. The experience from Asea Brown Boveri was that the software specialist could proceed and develop new tools, mainly for version control and configuration management. The application and marketing groups were able to handle the deliveries.

At Ericsson Radar Electronics we studied the software engineering environment for the development of display systems in military aircraft. Ericsson has developed both a special purpose high level language for the display processors and a programming environment for most of the software in the current Swedish military aircraft project. One motive for the tailor-made support for the display system was the requirement on prototyping. As

hardware develops very fast it was considered necessary to be able to demonstrate the current state of the system and possibly modify the requirement during the course of the project.

At SattControl we studied systems for process monitoring and control [5]. The situation at SattControl was to a very high degree a realization of what we had envisioned when presenting the multi-level software architecture model in the first Mekan project. In particular, we reviewed their facilities for giving the process operators a possibility to design their own images of the plant. On this level process displays can be built in a Macintosh-like fashion. Special standard building blocks for different branches of industry (breweries, dairies, paper mills, etc) can be built by experts from the application areas without any coding. The computer scientists in the system development groups were freed to use their skills for new steps in the same direction. The technique used within the system development group at SattControl was to work fully object oriented. In terms of languages they have developed their own pre-processors for ISO-Pascal, which in turn is used as a 'machine-code' to achieve portability.

The activities studied at Philips Elektronikindustrier have not reached a point where definite conclusions can be drawn. What we studied was their use of the Ada Environment from Rational Inc. [6]. Philips has twelve such systems in use in two major projects for the Swedish and for the Danish Navy. The selected method for the system structuring is object orientation. One interesting experience can be mentioned, however. It is a well-known fact that differences in productivity and quality between individuals in a software development team are often obvious. Philips claims that in the groups using the incremental highly interactive Rational environment, the differences between the individuals are significantly greater than they have ever experienced before. One reasonable hypothesis is that a traditional software engineering environment stressing strict top-down development, reviewing and approving all decisions taken, etc. may limit the creativity and productivity of the very best designers. A powerful incremental support system, on the other hand, may encourage this category. There is admittedly a weak base for this interpretation, but it is nevertheless exciting.

We also observed that in all four cases the system development groups had been defining their own objectives. The development had been driven by one or a few visionary persons being in the confidence of the top management. It was claimed from the companies that the object oriented approach used had enabled interleaved design and coding. Functionality had been added to objects as the system grew. Very little back-tracking had been used in the design.

5. Current Research Efforts in Linköping

In our Programming Environments laboratory we have during the years gathered experience in program analysis and transformation [7], in incremental compilation [8], and in support for distributed targets [9].

Inspired by the visions in the Mekan projects we are now investigating what kind of support environment a software specialist will need to contribute to system development in the frame-work of Multi-Level Software Architectures. Our approach is an amalgamation of what is normally called the transformational approach [10[and VHLL [11]. Our research vehicle, at least in the initial phase, will be Refine from Reasoning Systems, Inc. [12]. We will look at powerful and flexible tools for program transformations from higher to lower levels of abstraction, and also for state transformations from lower levels to higher. A general problem when using VHLL is to relate the systems (unintended) behavior to the high level program. Our aim is to be able to have a dialogue with the executing system on an arbitrary level of abstraction.

Acknowledgement

I have been continously encouraged by Mekan's very helpful support committee during the collection of material from Swedish industry. I will also thank all the inspiring colleagues at the Department of Computer and Information Science, Linköping University, for being exactly the way they are.

6. References

[1] **S. Hägglund:** The Linköping Approach to Technology Transfer in Knowledge Engineering. To appear in *The Knowledge Engineering Review.* Vol 2, No 3. Cambridge University Press, 1988.

[2] *Annual Research Report 1986.* Department of Computer and Information Science, Linköping University, Linköping, Sweden. ISBN 91-7870-168-6.

[3] **B. Lennartsson, A.M. Fransson, J. Wilander:** *Future Techniques for Software Systems Design.* (In Swedish). Mekanresultat 86004. Sveriges Mekanförbund, Box 5506, S-114 83 Stockholm, Sweden. 1986. ISBN 91-524-0861-2.

[4] **B. Lennartsson:** *Software Systems Design - Some Swedish Examples.* (In Swedish). Mekanresultat 88xxx. Sveriges Mekanförbund, Box 5506, S-114 83 Stockholm, Sweden. To appear during 1988.

[5] **H. Elmquist, S.O. Eriksson, S. Runeson:** New Concepts for Operator Interaction using Modern Graphic Techniques. *Proceedings of the Programmable Controllers'86 Conference.* Birmingham, UK. Nov. 1986.

[6] **R.T. Bond:** Productivity Impact of Advanced Programming Environments on Software Development throughout the Lifecycle. Presented at *Military Computing Conference.* Anaheim, CA. May 1987.

[7] **P. Emanuelson, A. Haraldsson:** On Compiling Embedded Languages in Lisp. *Proceedings of the 1980 Lisp Conference.* Stanford, CA. 1980.

[8] **P. Fritzson:** The Architecture of an Incremental Programming Environment and some Notions of Consistency. *Proceedings of the GTE Workshop on Software Engineering Environments for Programming-in-the-large.* Harwichport, MA. June 1985.

[9] **J. Fagerström:** Design and Test of Distributed Applications. *Proceedings of the 10th International Conference on Software Engineering.* Singapore. April 11-15. 1988.

[10] **C. Rich, R. Waters, eds.:** *Readings in Artificial Intelligence and Software Engineering.* Morgan Kaufmann Publishers, Inc. Los Altos, CA. 1986. ISBN 0-934613-12-5. pp. 97-200.

[11] Same as [10]. pp. 233-306.

[12] **Reasoning Systems:** *Refine 2.0 Users Guide.* Reasoning Systems Inc. 1801 Page Mill Road, Palo Alto, CA 94304. Sept. 1987.

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-17

Upplagans storlek:
Number of copies:

Datum:
Date: August 1988

Projekt:
Project:

Titel:
Title: Multi-Level Software Architectures -
Industrial Experiences and Research Directions

Författare:
Author: Bengt Lennartsson

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

Abstract: This paper presents some experiences in Swedish industry from using multi-level software architectures in a few application areas. The idea behind the multi-level architecture is to start by developing an application modelling tool tuned to the specific application area. The design and implementation of this modelling tool is a substantial part of the total system development effort.
The objective is to find a model where the computer scientist and the application domain expert can contribute to system development without having to learn details from each other's domains. The experience, so far, is that the model allows the software specialists to concentrate on software design and development, and that the application specialists are able to tune and maintain the systems mainly by themselves. Some of our research efforts in the area are also presented.

Nyckelord (högst 6):
Keywords (6):

Bibliotekets anteckningar

A Selection of Previous Research Reports.

LiTH-IDA-R-88-08 **Erik Sandewall:** Formal Semantics for Reasoning about Change with Ramified Causal Minimization.

LiTH-IDA-R-88-07 **Erik Sandewall et al:** IDA Annual Report 1987.

LiTH-IDA-R-88-06 **Christer Bäckström:** A Representation of Coordinated Actions Characterized by Interval Valued Conditions.

LiTH-IDA-R-88-05 **Christer Bäckström:** Keeping and Forcing: How to Represent Cooperating Actions.

LiTH-IDA-R-88-04 **Wlodek Drabent, Simin Nadjm-Tehrani, Jan Maluszynski:** Algorithmic Debugging with Assertions. Also in *Proc. of META 88 - Workshop on Meta Programming*, Bristol, June 21-24 1988.

LiTH-IDA-R-88-03 **Staffan Bonnier, Jan Maluszynski:** Towards a Clean Amalgamation of Logic Programs with External Procedures. Accepted to the *Joint Fifth Int. Logic Programming Conf. and Fifth Logic Programming Symp.*, Seattle Washington, August 15-19, 1988.

LiTH-IDA-R-88-02 **Ulf Nilsson:** Inferring Restricted AND-Parallelism in Logic Programs using Abstract Interpretation.

LiTH-IDA-R-88-01 **Jacek Leszczylowski, Staffan Bonnier, Jan Maluszynski:** Logic Programming with External Procedures: Introducing S-Unification. A shorter version is to appear in *Information Processing Letters*, North Holland.

LiTH-IDA-R-87-26 **Jonas Löwgren:** Applying a Rapid Prototyping System to Control Panel Dialgues.

LiTH-IDA-R-87-24 **Sven Moen:** Drawing Dynamic Trees.

LiTH-IDA-R-87-22 **Lars Ahrenberg, Arne Jönsson:** An Interactive System for Tagging Dialogues. Also in *Proc. of 14th Conference of the Association for Literary and Linguistic Computing*, Göteborg 1-5 June, 1987.

LiTH-IDA-R-87-21 **Harold W. Lawson, Jr.:** Challenges and Directions in Computers and Education.

LiTH-IDA-R-87-20 **Krzysztof Kuchcinski, Zebo Peng:** Parallelism Extraction from Sequential Programs for VLSI Applications. This paper is to appear in *Microprocessing and Microprogramming, the Euromicro Journal*, 1988.

LiTH-IDA-R-87-19 **Zebo Peng:** Construction of Asynchronous Concurrent Systems from their Behavioral Specifications. Also in *Proc. of the 10th World Computer Congress*, Dublin, Ireland, Sept. 1-5, 1986.

LiTH-IDA-R-87-18 **Henrik Nordin:** Reuse and Maintenance Techniques in Knowledge-Based Systems.

LiTH-IDA-R-87-17 **Tony Larsson:** Specification and Verification of VLSI Systems Actional Behaviour This is a close version of a paper presented at *the 8th international conference on Computer Hardware Description Languages, CHDL,* 87.

LiTH-IDA-R-87-16 **Shamsul I. Chowdhury:** State of the Art in Statistical Expert Systems. Also in *Proc. of the Conference on Expert Systems and their Applications*, Avignon, May 1987.

LiTH-IDA-R-87-15 **Nils Dahlbäck:** Kommunikation med datorer i naturligt språk - vad är det och vem behöver det?

LiTH-IDA-R-87-14 **Peter Fritzson:** Window System Architectures - an Overview. Also presented at *A Tutorial on Software Development Environments,* Linköping University, Nov 24-25 1986.

LiTH-IDA-R-87-13 **Mats Wirén:** A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing. Also in *Proc. of the Third Conference of the European Chapter of the Association for Computational Linguistics,* Copenhagen, April 1—3, 1987.

LiTH-IDA-R-87-11 **Sture Hägglund, Christer Hansson, Tomas Sokolnicki:** Knowledge-Based Training of Case Management Routines and Emergency Procedures. Also in *Proc. of the 3rd Int. Conf. on Expert Systems*, London, 1987.

The Department of Computer and Information Science
Linköping University

organizes undergraduate and graduate studies in *Computer Science*, *Telecommunication and Computer Systems*, and *Administrative Data Processing*. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- *ACTLAB - Laboratory for Complexity of Algorithms*, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.
- *AIELAB - Artificial Intelligence Environments Laboratory,* which conducts research on representation and manipulation of knowledge, problem solving techniques and expert systems with mechanical engineering applications.
- *ASLAB - Application Systems Laboratory,* which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.
- *CADLAB - Laboratory for Computer-Aided Design of Electronics*, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.
- *LIBLAB - Laboratory for Library and Information Science,* which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.
- *LOGPRO - Laboratory for Logic Programming*, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.
- *NLPLAB - Natural Language Processing Laboratory*, which conducts research related to the development and use of natural language interfaces to computer software.
- *PELAB - Programming Environments Laboratory*, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.
- *RKLLAB - Laboratory for Representation of Knowledge in Logic*, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

Research Reports 1988

LiTH-IDA-R-88-17 **Bengt Lennartsson:** Multi-Level Software Architectures - Industrial Experiences and Research Directions. This report is a revised version of a paper in *Advance Working Papers of The Second International Workshop on Computer-Aided Software Engineering.* Cambridge, Massachusetts. July 12-15, 1988.

LiTH-IDA-R-88-16 **Lin Padgham:** A Model and Representation for Type Information and Its Use in Reasoning with Defaults. Also in *Proc. of AAAI'88, American Association for Artificial Intelligence*, 1988.

LiTH-IDA-R-88-15 **Lin Padgham, Ralph Rönnquist:** LINCKS: An Imperative Object Oriented System. Also in *Proc. of the 20th Annual Hawaii Int. Conf. on System Sciences*. Hawaii. 1987.

LiTH-IDA-R-88-13 ssystem för autonoma

LiTH-IDA-R-88-12 engvald: Software Robotics Applications.

LiTH-IDA-R-88-11 or PATR.

LiTH-IDA-R-88-10 t Parser for PATR. Revised version of paper appearing in *Proc. of First Scandinavian Conf. on Artificial Intelligence*, Tromsö, Norway, March 9 - 11, 1988.