



LUND UNIVERSITY

The CACE Project -- Steering Committee Meeting 3, 1986-04-15

Mattsson, Sven Erik; Åström, Karl Johan

1986

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Mattsson, S. E., & Åström, K. J. (1986). *The CACE Project -- Steering Committee Meeting 3, 1986-04-15*. (Technical Reports TFRT-7322). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:

2

General rights

Unless other specific re-use rights are stated the following general rights apply:

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

CODEN: LUTFD2/(TFRT-7322)/1-168/(1986)

The CACE Project - Steering Committee Meeting 3

Sven Erik Mattsson
Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology
June 1986

Department of Automatic Control Lund Institute of Technology P.O. Box 118 S-221 00 Lund Sweden		<i>Document name</i> Report	
		<i>Date of issue</i> 1986-06-02	
		<i>Document Number</i> CODEN:LUTFD2/(TFRT-7322)/1-168/(1986)	
<i>Author(s)</i> Sven Erik Mattsson Karl Johan Åström		<i>Supervisor</i>	
		<i>Sponsoring organisation</i> The Swedish Board of Technical Development	
<i>Title and subtitle</i> The CACE project - Steering committee meeting 3, 1986-04-15			
<i>Abstract</i> <p>This report contains documentation handed out to the participants of the 3rd steering committee meeting of the STU Computer Aided Control Engineering Programme (CACE) on April 15, 1986. The minutes of the meeting are also included.</p>			
<i>Key words</i> Computer Aided Control Engineering			
<i>Classification system and/or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i>			<i>ISBN</i>
<i>Language</i> English and Swedish	<i>Number of pages</i> 168	<i>Recipient's notes</i>	
<i>Security classification</i>			

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

PREFACE

This report contains documentation handed out to the participants of the 3rd steering committee meeting of the STU Computer Aided Control Engineering Programme (CACE) on April 15, 1986. The minutes of the meeting are also included.

As seen from the documentation, current projects were reviewed and new ones were proposed and discussed.

The Science and Research Council, U.K. has a programme called The SERC Initiative in Computer and Design Techniques in Control Engineering (CDTCE). Four of the members of the steering committee (Phil Hicks, Jim Andersson, Neil Munro and Mike Denham) participated also in the meeting to get informed about the CACE projects and to present their projects and discuss collaboration.

Table of Contents

AGENDA	7
PROJECT OVERVIEW	
View graphs, Karl Johan Åström	8
Seminar and Visits October 1985 - April 1986 Sven Erik Mattsson and Karl Johan Åström	15
REVIEW OF CURRENT CACE PROJECTS	
New Forms of Man-Machine Interaction	
View graphs, Sven Erik Mattsson	22
A simulator for Dynamical Systems Using Graphics and Equations for Modelling. Hilding Elmqvist and Sven Erik Mattsson. Abstract submitted to the IEEE Control Systems Society 3rd Symposium on CACSD	33
Experiment with Knowledge Based Man-Machine Interfaces	
View graphs, Jan Eric Larsson and Per Persson	36
Knowledge Representation by Scripts in an Expert Interface. Jan Eric Larsson and Per Persson. Paper to be presented at ACC-86, June 18-20, 1986, Seattle, Washington, USA.	45
High Level Problem Solving Languages	
View graphs, Karl Johan Åström	49
System Representations. Karl Johan Åström and Wolfgang Kreutzer. Abstract submitted to the IEEE Control Systems Society 3rd Symposium on CACSD	60
Combination of Formula Manipulation and Numerics	
View graphs, Ulf Holmberg	62
Integrating Symbolic and Numerical Tools for Linear System Analysis. Ulf Holmberg, Mats Lilja and Bengt Mårtensson. Abstract submitted to the SIAM Conference on Linear Algebra, August 12 - 14, 1986, Boston, USA	73
Expert Control	
View graphs, Karl Johan Åström	74
Towards Intelligent Control. Karl Johan Åström. Abstract.	75
View graphs, Karl-Erik Årzén (Presented by Karl Johan Åström)	76

Use of Expert Systems in Closed Loop Feedback Control. Karl-Erik Årzén. Paper to be presented at ACC-86, June 18-20, 1986, Seattle, Washington, USA	87
 NEW CACE PROJECTS	
View graphs, Sven Erik Mattsson	93
Representation and Visualization of Systems and Their Behaviour. Sven Erik Mattsson	109
Expert System Interface Experiments. Jan Eric Larsson and Per Persson	120
Implementation Languages. Karl Johan Åström	122
Numerical Solution of Differential-Algebraic Systems. Bo Kågström	124
 PRESENTATION OF SERC	
Introduction, View graph, Phil Hicks	132
The SERC Initiative in Computer and Design Techniques in Control Engineering (CDTCE).	
View graphs, Jim Andersson	134
Notes	142
U.M.I.S.T Project, View graphs, Neil Munro	146
Current CDTCE Grants, since October 1984	155
Advanced Computing Concepts and Techniques in Control Engineering Mike Denham	158
 MINUTES OF THE MEETING	 164

THE CACE PROJECT - Steering Committee Meeting 3

April 15, 1986

9.00 INTRODUCTION

9.30 REVIEW OF CURRENT PROJECTS

New forms of man-machine interaction

Experiment with knowledge based man-machine interfaces

High level problem solving languages

Combination of formula manipulation and numerics

Expert Control

11.00 DECISION ON FUTURE

12.30 LUNCH

13.30 DEMONSTRATIONS

14.00 PRESENTATION OF SERC

15.30 DISCUSSIONS ON COLLABORATION

PROJECT OVERVIEW

- 1. Introduction**
- 2. Project status**
- 3. Contacts**
- 4. Impressions**
- 5. Decisions**

Numerics

Impl. lang

System Repr

Expert control

Symbolic calc.

High level lang.

Expert system interface

MMI

Background work Simmon..



PERSONNEL

STU Financed

Sven Erik Mattsson

Dag Brück

Per Persson

Jan Eric Larsson

Karl Erik Årzén

Ulf Holmberg

Mats Lilja

LTH Financed

Karl Johan Åström

Per Hagander

Leif Andersson

Tomas Schönthal

Bengt Mårtensson

Guest researchers and consultants

Klas Sigbo

Wolfgang Kreutzer

Hilding Elmqvist

Activate implementation project

Postpone another workstation

Background

FRN, IBM, Department

Unify Lisp Environment

3. CONTACTS

National

Large and small

Ideon

International

5. DECISIONS

Representation and Animation

Expert system interfaces

Implementation languages

Numerics

Dissemination

Public vs private



**IEEE Control Systems Society
3rd Symposium on
Computer-Aided Control System
Design (CACSD)
September 24–26, 1986
Quality Inn, Pentagon City
Arlington, Virginia**

The IEEE CSS Symposium on CACSD provides a technical forum for the open exchange of ideas and information relating to computer-aided control system design. The emphasis of the CACSD symposium is on high quality technical contributions of current results, presented in an informal interactive conference setting. Contributions will be featured that display the synergism of control theory and control algorithms with computer science and computer engineering. Wide participation is encouraged from control system designers in all fields, such as aeronautical, mechanical, and chemical. Exhibits will be presented on software and hardware CACSD capabilities from both commercial and noncommercial offerors.

Call for Papers

Authors of papers on recent developments relevant to CACSD are invited to submit five copies of a summary (approximately 700 words) of their work by March 31, 1986 to:

Kathy L. Lineberry
Business and Technological Systems, Inc.
10210 Greenbelt Road, Suite 440
Seabrook, Maryland 20706
(301) 794-8800

The summaries must be headed with the paper title and the names, affiliations, and complete mailing addresses and telephone numbers of all authors. The first-named author will be used for all correspondence unless otherwise stated. Final manuscripts will be due at registration, September 24, 1986, for publication in the proceedings following the conference.

Call for Exhibits

Exhibitors of software and/or hardware for CACSD are invited to submit three copies of a brief description (approximately 500 words) and any appropriate supporting materials of their exhibit by March 31, 1986 to:

Dr. Malcolm D. Shuster
Business and Technological Systems, Inc.
10210 Greenbelt Road, Suite 440
Seabrook, Maryland 20706
(301) 794-8800

The description should be specific about what is to be displayed and should include an estimate of space requirements. A fee of \$600 for commercial exhibits and \$100 for non-commercial exhibits (i.e., for products available under \$100) will be charged to help defray expenses. Selection of exhibits will be solely on the basis of technical merit. For further information contact Dr. Shuster.

**SEMINARS AND VISITS
OCTOBER 1985 - APRIL 1986**

Sven Erik Mattsson and Karl Johan Åström

**Department of Automatic Control
Lund Institute of Technology
Lund, Sweden**

This is a list of seminars and external contacts the Department of Automatic Control, Lund Institute of Technology has had during the period October 1985 - April 1986, which are of interest for the CACE project. The list includes visits to the department and visits of the staff to companies and other universities, as well as participation in conferences, symposia, workshops, courses etc.

Our visitors are normally given a presentation of our department and our research, as well as live demonstrations of our packages for CACE (Simnon, Idpac etc.), so this is not explicitly mentioned in the list below.

October 2

Professor Rune Gustafsson, Department of Computer Science, Uppsala University visited the department. His research interests include AI and expert systems.

October 2 - 4

Dr. Dean Frederick, RPI, Troy, New York visited the department. He gave a seminar "CACE-III An expert system for the design of control system". He also introduced a bull session on man-machine communication.

October 10

Les Cochron, Sperry visited the department. He is responsible for Sperry's AI Commitment.

October 16 - 21

Dr. James H. Taylor, General Electric, Schenectady, visited our department and gave two seminars "Expert system applications to control system design and implementation" and "Nonlinear controller design based on quasilinear system models".

October 29 - 31

Dag Brück attended the Unix Exhibition in Älvsjö, including the full-day seminar "Unix Internals" by Kaare Christian.

November 4

Karl-Erik Årzén attended a meeting of the Association of Mechanical, Electrical and Electronical Industries in Sweden (Svenska Mekanförbundet) supporting committee on Knowledge based system. The meeting was held at EPITEC, Linköping.

November 11

For his master thesis "An Expert System Interface for Idpac" Jan Eric Larsson received at SCA, Sundsvall the award of Bo Rydin's foundation for research for best master thesis of relevance for the pulp and paper industry.

November 11 - 12

Carl-Wilhelm Welin, Ericsson Telecom, Computer Science Laboratory, Stockholm visited the department. He acts as a discussion partner in the expert system

projects. He also gave a seminar at the Department of Computer Science and Computer Engineering on the expert system activities at LM Ericsson.

November 13

Jan Eric Larsson and Karl-Erik Årzén presented the AI related projects in the CACE project in a seminar on current AI projects at Lund University. The seminar was arranged by the Department of Computer Science.

November 15 - 23

Karl Johan Åström visited the US. He installed Simnon at the Courant Institute of Mathematics at New York University. He also discussed CACE problems with Jack Schwartz.

Karl Johan Åström visited the Mechanical Engineering Department at University of Texas in Austin. The integration of dynamics simulation with conventional mechanical drafting was also discussed.

Karl Johan Åström participated in the ASME annual meeting to receive the Oldenburger Medal. In connection with this he also discussed CACE problems and use of knowledge based engineering. He also established relations with a CACE group at Carnegie Mellon.

November 18 - January 17

Dr. Wolfgang Kreutzer, Department of Computer Science, University of Canterbury, New Zealand visited the department as a guest researcher and participated in the CACE project.

November 26 - 27

Karl-Erik Årzén visited the Department of Computer Science at Uppsala University. He discussed the Expert Control project and studied PICON; a real-time expert system for process control and the expert system shell KEE.

November 28

Mats Jerpander has made a demonstration package for the ASEA Master System as his master thesis project. He gave a seminar and presented the ASEA Master System and his demonstration package.

December 9

Dr. John Cassidy, General Electric, Schenectady, USA gave a seminar "Perspectives on Computer Aided Implementation of Advanced Control Systems". Dr. Cassidy is responsible for research and development in automatic control at General Electric.

December 10

Thomas Rugeland, Tektronix, presented the Tektronix AI-computer 4404 with Smalltalk 80 at the Department of Computer Science and Computer Engineering.

December 11

At a meeting of the AI interest group at Lund University, Karl Johan Åström and Karl-Erik Årzén gave a seminar "Intelligent controllers" (Intelligenta regulatorer)

December 12

In the evening we presented the department and the CACE project for the computer technology students (D-linjen).

December 13

Bjarne Däcker, Ericsson Telecom, Computer Science Laboratory, Stockholm gave a seminar titled "Om realtidsspråk på LM Ericsson" (On Real-Time Programming Languages at LM Ericsson).

December 17

Carl Olin demonstrated Exomatic which is a small system for Direct Digital Control.

December 19 - January 9

Karl-Erik Årzén gave a one week course in Lisp for the members of the department.

January 8

We had an informal meeting with Benima AB to discuss applications of CACE in a small consulting company.

January 13 - 17

Karl Johan Åström participated Chemical Process Control III. He presented an

invited paper on Auto-tuning, Adaptation and Smart Control. Of interest for the CACE project was also a full session devoted to Expert Control.

January 21 - April 30

Dag Brück attended an undergraduate course on Man-Machine Interaction.

January 21

Karl Johan Åström visited Reasoning Systems, Palo Alto to look at their advanced software development environment.

January 22

Karl Johan Åström visited Charlie Herget at Lawrence Livermore National Laboratory to review progress of the CACE project. He was also briefed on the progress of EAGLES. Agreement on informal cooperation was established.

January 23 - 24

Karl Johan Åström visited Naval Weapon Center in China Lake CA to discuss advanced application of CACE.

January 28

Karl-Erik Årzén attended a meeting of The association of Mechanical, Electrical and Electronical Industries in Sweden (swed. Svenska Mekanförbundet) supporting committee on Knowledge based system. The meeting was held at Ericsson Radio in Mölndal.

January 31 - April 18

On Friday mornings between 8 and 10 videotapes from the course "Structure and Interpretation of Computer Programs" with Professors H. Abelson and G.J. Sussman, MIT as lecturers, has been shown at the Department of Computer Engineering. A number of persons from the Department of Automatic Control has attended.

February 5

IBM presented the new IBM PC RT at Lund University.

February 6

Karl-Erik Årzén gave a seminar and presented PICON which is a real-time expert

system for process control from LMI Process Systems, Cambridge, Massachusetts.

Karl Johan Åström gave a seminar titled "Experiences from a trip to US".

February 10

Sven Erik Mattsson presented the CACE project "New Forms of Man-Machine Interactions" in the undergraduate course Man-Machine Interaction (Kurs DA422 Människa-datorinteraktion, 5p) at the Department of Computer Science and Computer Engineering. He also demonstrated the IRIS 2400.

February 13

Karl-Erik Årzén gave a seminar "Expert system applications in automatic control" at the Department of Automatic Control, Linköping.

February 19 - 20

Bengt Skarman, SAAB, Linköping visited the department.

March 6

Staffan Lund, ACIS, IBM, Stockholm and Bengt Herne, IBM, Malmö visited us. We presented the CACE project and the possibilities for IBM to lend us an IBM PC RT for half a year was discussed.

March 11

Johan Westermarck, Scandia Metric AB, Mölndal presented the MASSCOMP 5000 Family.

March 19

Tomas Schönthal gave a seminar titled "Simnon on IBM-PC".

March 20

Karl Johan Åström gave a seminar titled "System representations".

March 21

Dr. Bo Kågström and Anders Barrlund, Institute of Information Processing, University of Umeå, Sweden visited us. Dr. Kågström gave a seminar titled "Can linear systems given by uncertain data be controlled?"

March 24

Sten Bergman, ASEA Relays visited the department and discussed CACE issues.

April 2 - 4

Bjarne Däcker, Ericsson Telecom, Computer Science Laboratory, Stockholm visited the Lund Institute of Technology to discuss education in real-time system. He gave also three seminars: "Experiments with programming languages and techniques for telecommunications applications", "Design of an expert system and man-machine interface for operation and maintenance of AXE telephone exchanges" and "Comparison between Lisp and Pascal for use in developing programming support systems".

April 4 - 8

Professor Jan Willems, Mathematical Institute, University of Groningen, The Netherlands visited the department. He gave three seminars titled "Modelling Dynamical Systems: Fitting a Linear System to an Observed Time Series".

NEW FORMS OF MAN-MACHINE INTERACTION

The man-machine interface is a very important part of CACE system.

The new workstations with high performance, real-time graphics open new possibilities for man-machine interaction.

The purpose of this project is to set up a prototype system so that ideas can be tested and experiences of using graphics for man-machine interaction can be gained.

PILOT PROJECT

Simulator for dynamical system

Graphical description of the model decomposition

Hierarchical block diagrams

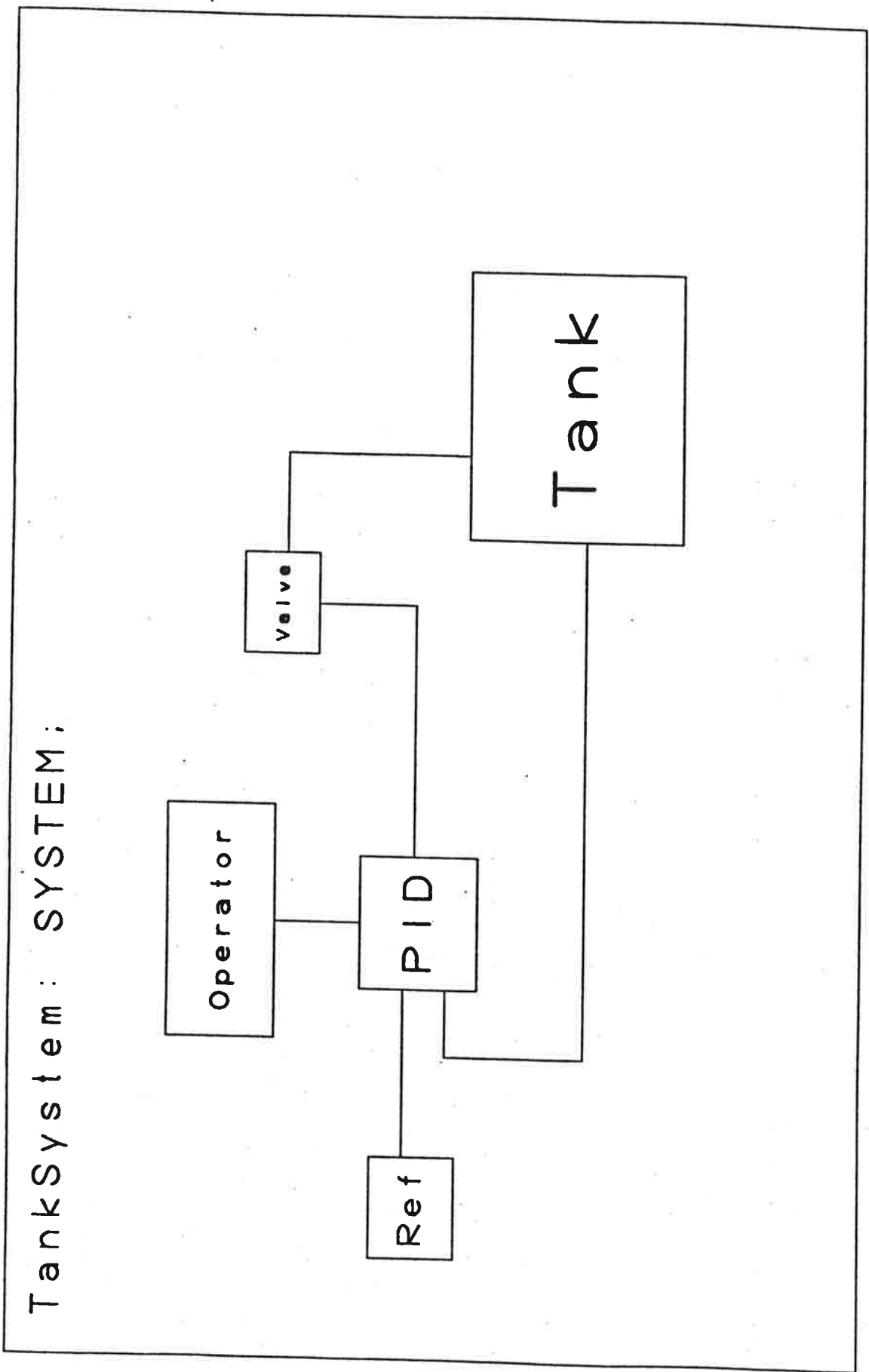
Information zooming

Overview windows

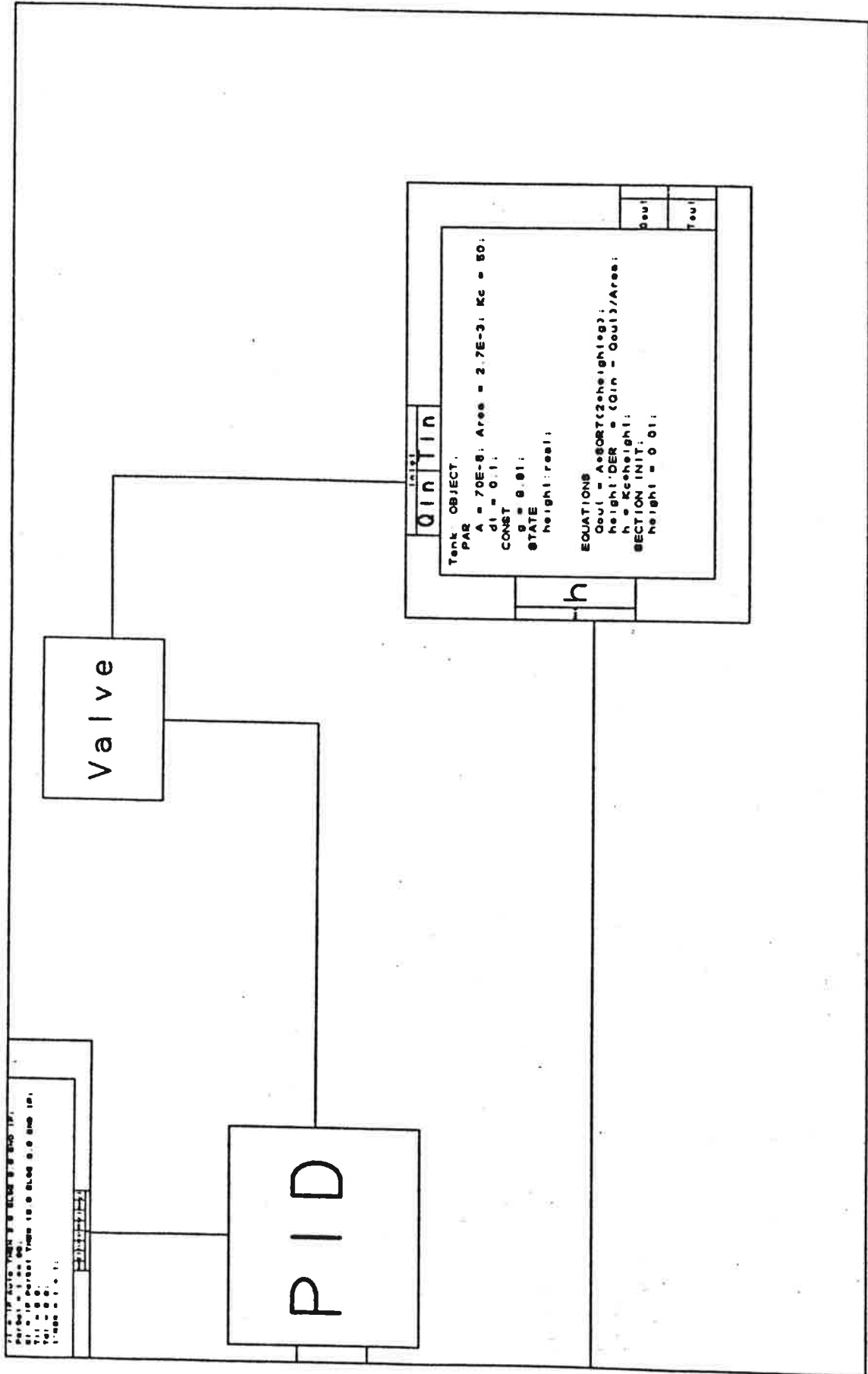
Equation based submodels

Differential-algebraic systems

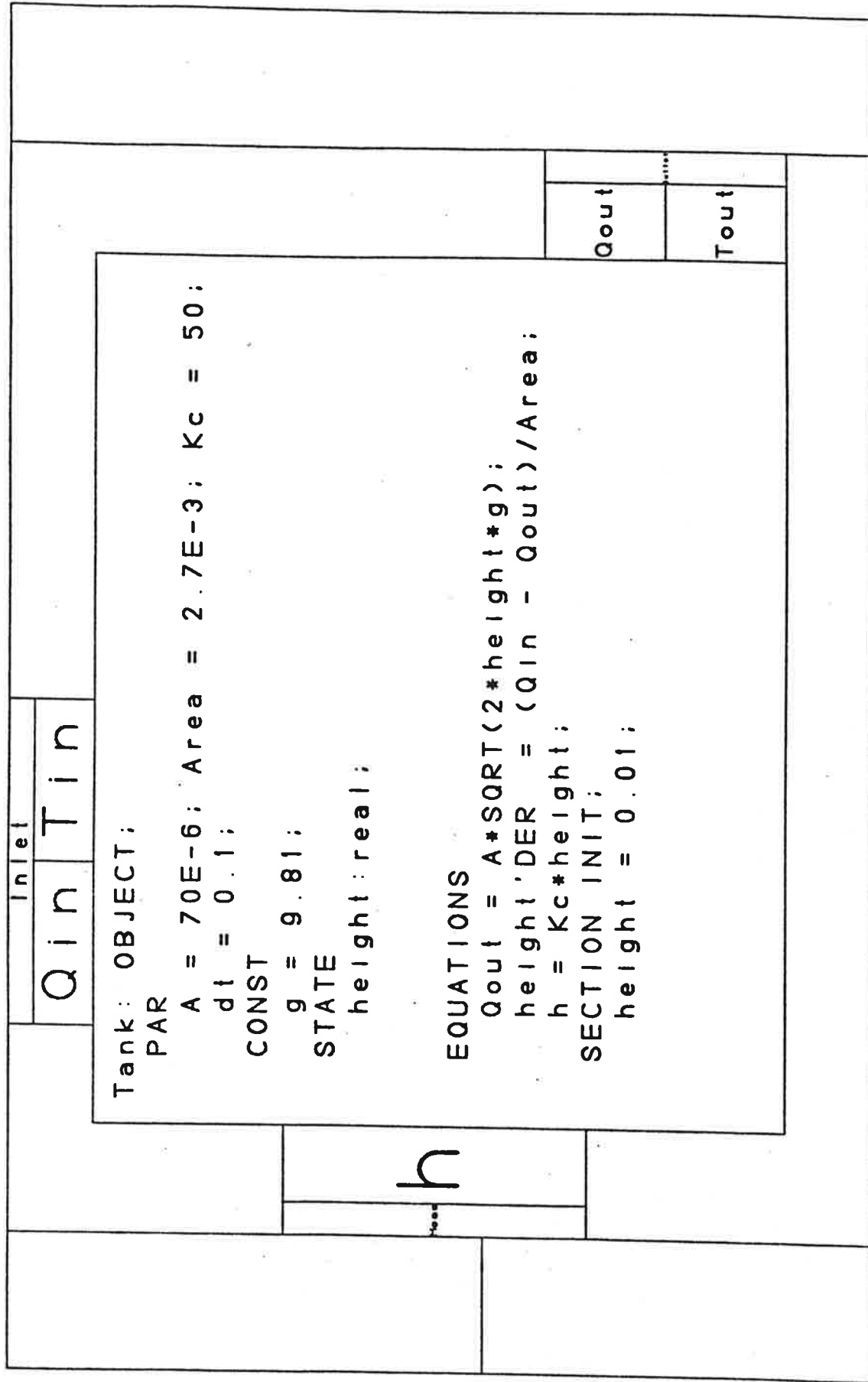
HIERARCHICAL BLOCK DIAGRAM - 1



HIERARCHICAL BLOCK DIAGRAM - 2



HIERARCHICAL BLOCK DIAGRAM - 3



Inlet: (Q_{in}: IN real, T_{in}: real)

Outlet: (Q_{out}: OUT real, T_{out}: real)

WHY EQUATIONS?

Simplifies modelling

 Closer to original formulations

Necessary for 'general' model libraries

 Which variables are 'inputs'?

Allows more sophisticated connection mechanisms

Safer

 Easier to check that the model is entered correctly

 Reduces the risk of introducing transformation errors

Better documentation

 Closer to original formulations

THE PROTOTYPE SIMULATOR

You can create and edit hierarchical block diagrams

Model

Interface

Connect

Copy

Remove

You can inspect the model

Pan

Zoom

Layout

View

You can store and retrieve models

Get

Save

You can simulate

Display

Compile

Simulate

THE IMPLEMENTATION WORK

The simulator is written in Pascal.

The source code is 27 000 lines long.

The code is highly modular.

Related types, variables and procedures
are grouped together.

Machine dependent parts are isolated.

Preprocessor Packman

Generates a standard Pascal program
or code for separate compilation

Supports nested inclusions

THE MODELLING AND LANGUAGE PART

LICS has routines for

- Syntax analysis

- Compilation of interface variables which are connected to each other into a list

The prototype simulator also

- Performs type checking

- Generates equations describing the connections

- Eliminates simple equations; $A = B$

- Makes simple analysis of the equations

- Generates code for a virtual stack machine

- Has an interpreter in Pascal

- Sets initial values

- Has the ODE solver DASSL (converted to Pascal)

GRAPHICS

Software for creating and editing
hierarchical block diagrams from LICS.

LICS uses

Segments

Software for transformation and clipping

IRIS Graphics Library

Implemented the low level graphics routines

Introduced hardware clipping

Generated fonts

Graphical fonts

Raster fonts

The graphics part is currently under redesign

Portability

Explore various hardware facilities

EXPERIENCES

Positive reactions of visitors to the use of graphics for describing structure.

Handling of text is time critical.

The IRIS and the Pascal system have worked well.

Many difficulties were due to missing documentation.

However, Pascal is not convenient for fast prototyping. The turn around time is too long (10 minutes).

Use of Common Lisp may be a solution.

**A SIMULATOR FOR DYNAMICAL SYSTEMS
USING GRAPHICS AND EQUATIONS FOR MODELLING**

**Hilding Elmqvist
SattControl AB,
P.O. Box 9034, S-200 39 Malmö, Sweden
+46 40 226465**

**Sven Erik Mattsson
Department of Automatic Control
Lund Institute of Technology
P.O. Box 118, S-211 00 Lund, Sweden
+46 46 108779**

**Abstract submitted to the IEEE Control Systems Society Third Symposium on
Computer-Aided Control System Design.**

ABSTRACT

The man-machine interface is a very important part of a CACSD-system. The new workstations with high performance, real-time graphics now appearing on the market offer new possibilities. The paper describes a prototype system which explores some of these. The system is a simulator for dynamical systems, which can be described by sets of ordinary differential equations and algebraic equations.

The structural properties of a model are very important particularly when working with large systems. These structures are, however, difficult to represent in an easily apprehendable way when a purely textual description is used. The interconnection structure of submodels is for example much easier described graphically. The idea is to use hierarchical block diagrams to describe the decomposition of the model and the interconnection structure. The system allows continuous zooming to show internal detail. The user can move the cursor with the mouse. He can pan and zoom by pressing mouse buttons when moving the mouse. If he starts to zoom in on a block, the block will open up and show internal details. It will show its interfaces and the interior which is a new diagram or a mathematical description of the submodel. The user can also create new windows for viewing of the model. One of these windows is the interaction window for panning and zooming. To help the user to keep track of where he is, rectangles in the other windows outline what part of a window that is shown in the interaction window. If the user wants to move fast he can point on an object in another window and ask for automatic panning and zooming to this point in the interaction window. The result of a simulation can also be displayed in windows.

New windows are created and the layout is changed by selecting commands from a pop-up menu with the mouse. The mouse is then used to position and stretch the windows in a Macintosh-like fashion. New blocks, interfaces and connections are created in similar ways. The mathematical description of the submodels at the lowest level is entered and modified with a window-oriented editor. It is also possible to copy a block and all its subblocks. The model can be saved as a text file which can be read to recreate the model.

The simulator allows the submodels to be described in the form of equations

instead of assignment statements. This facilitates the use of the simulator since it is closer to the original formulations of the models as a basic set of mass- and energy-balances and other equations. The equation form also has other advantages compared to the assignment form. The documentation is better since the reader will recognize the equations. It is easier to check that the model is entered correctly and it reduces the risk of introducing errors during manual transformation to assignment statements. Furthermore, the equation form is the only reasonable if one want to build model libraries, because different environments of the submodel impose different causality relations, implying that the transformation to assignment form of a submodel depends on its environment. A more sophisticated connection mechanism can be introduced when equations are allowed. The simulator supports two standard types of interface variables: across variables which are equal in the nodes (examples are voltage, pressure and temperature) and through variables which have a direction and are summed to zero in the cut (examples are current, flow, thrust and torque). The simulator interprets the connections drawn by the user and generates appropriate equations automatically. The simulator uses Linda Petzold's differential/algebraic system solver DASSL which has a reputation of being one of the best and most robust solvers for differential/algebraic systems.

The simulator is written in Pascal. The software is highly modular and a preprocessor is used to produce a standard Pascal program from different module files. The source code is 27 000 lines long. The simulator currently runs on an IRIS 2400 from Silicon Graphics, Inc. The IRIS 2400 is a high performance engineering workstation designed for interactive color graphics and computing applications. The machine dependent parts have been carefully isolated to improve portability.

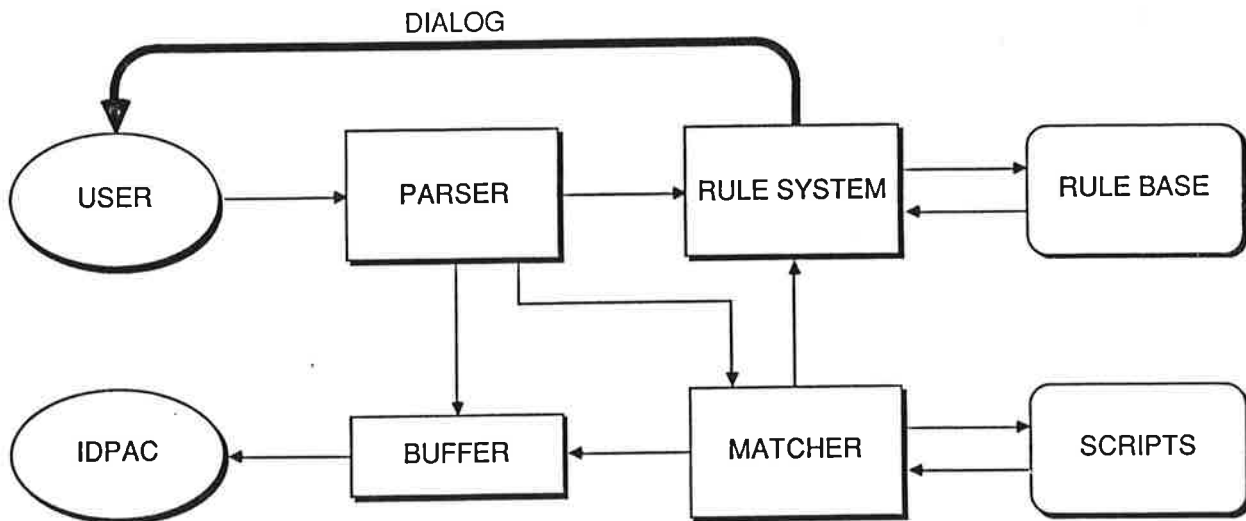
Expert System Interface Experiments

Jan Eric Larsson
Per Persson

Goals of the project

- How does one combine an expert system with a CAD package?
- How does one develop a help system with expert system techniques?
- Build a knowledge base for system identification.
- Requirements on the command language.

The first system

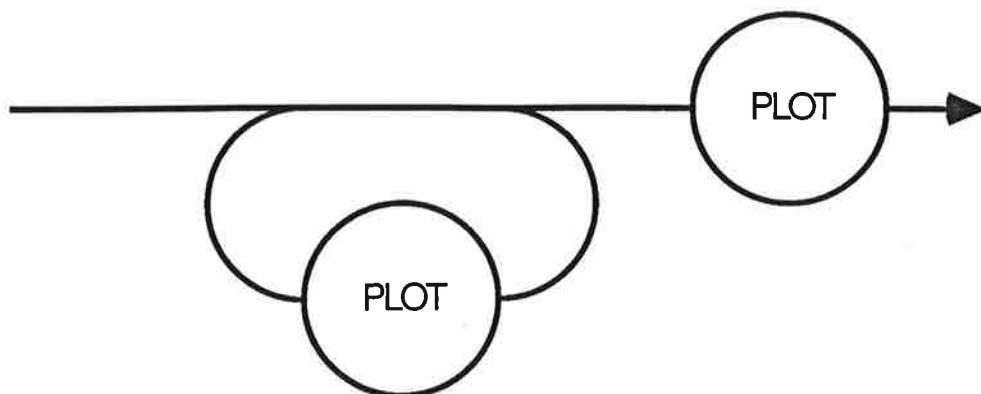
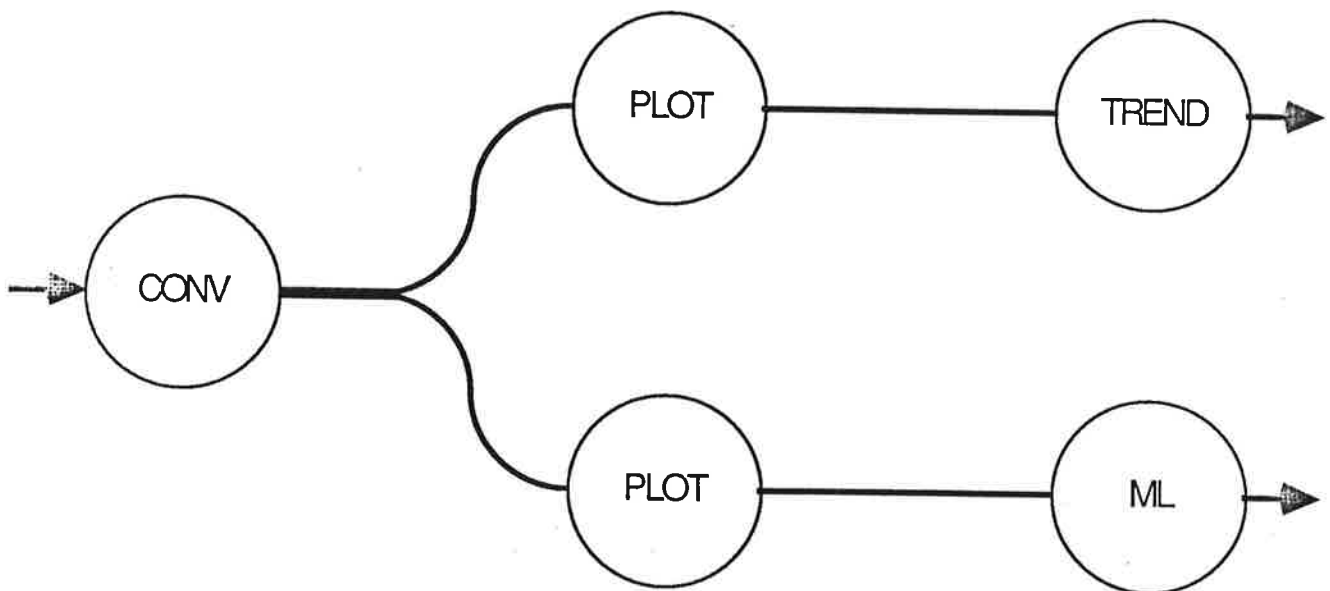


- Parser
- Matcher
- Expert system
- Presented at ACC '86

- Lisp
- VT100 graphics

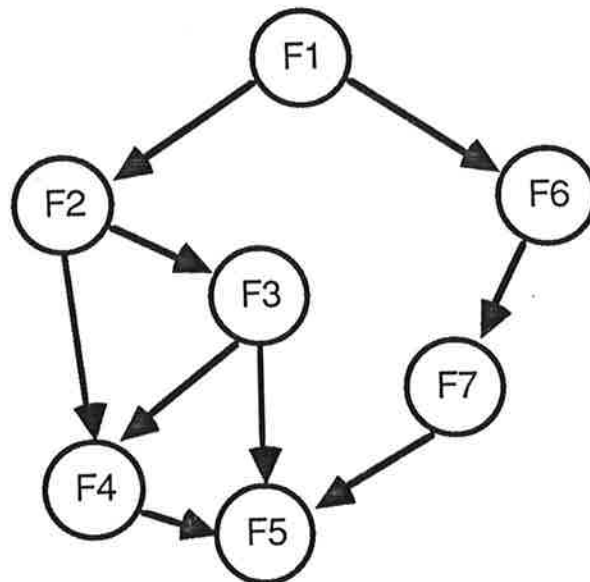
Script Matching

- The script language is very general.
- Standard techniques is not enough.
- No good methods exist.



The File Handler

- Inheritance tree
- Different output
- Parents
- Children
- Graphics
- “Flowchart”



Building blocks

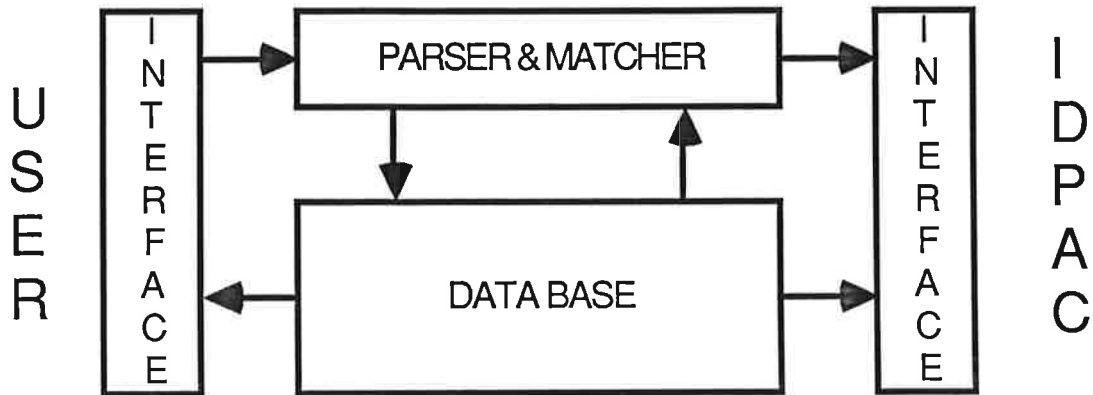
Flavors

- Object oriented programming in Lisp
- Multiple inheritance
- Originates from MIT

YAPS

- Forward chaining framework
- Uses Flavors
- Developed at Maryland University

Design of the new system



- Scripts — objects
- Files — objects
- Graphics interfaces — modules

Graphics

- Have used VT100 graphics.
- Only three lines available on T4025.
- Windows and pictures impossible.
- A high resolution graphics screen is needed.

Knowledge base

- A knowledgebase for test purpose has been developed.
- We want to develop a more realistic knowledge base.

Conclusions

- A small system has been developed.
- Have studied certain problems of the new system.
- Programming with object oriented methods in Lisp.
- Design and implement the new system.
- Build a realistic knowledge base.

Knowledge Representation by Scripts in an Expert Interface

Jan Eric Larsson M.Sc. & Per Persson M.Sc.

Department of Automatic Control
Lund Institute of Technology
Box 118, S-221 00 LUND, Sweden

Abstract

System identification demands skill and experience, and the validity of the results strongly depends on the user's knowledge. For this reason a help facility that uses domain specific knowledge about system identification is needed. This paper deals with the representation of knowledge in an expert interface for system identification, using the interactive identification package "Idpac". The concept of *scripts* is introduced. A small system has been implemented and a description of it is given.

1. Introduction

Knowledge based computer programs, expert systems, are rapidly being developed and will probably be quite common in the near future. When these techniques grow more reliable and become better known, it will be obvious for a CAD program to use them. But in order to incorporate expert knowledge in a program package, several problems must first be solved.

In this paper, we will describe an expert system that runs as an interface to Idpac, a program package for system identification. System identification is described in [1,2]. The Idpac package is built with the interaction module Intrac, a framework that provides an interactive environment for numerical Fortran routines. Idpac and Intrac have been developed at the Department of Automatic Control, Lund Institute of Technology, [3,4,5]. Intrac features a flexible, brief and efficient command style dialog. There is also the possibility for a user to put command sequences together with control statements in a file and have it executed, i.e. a macro facility. This facility is very valuable since it lets the user develop his own methods and helps him document and refine his knowledge. Without macros Idpac would probably not have been used as much as it actually has.

In order to design the expert interface the following goals were set up.

- The expert system should work as an intelligent help system. This means that we want to retain the powerful command dialog rather than using a Q/A style of communication.
- The system should only come into action on the user's request. In this way an experienced user will not be cramped by the system. As long as one doesn't issue a help command to the interface, he won't notice that the expert system is there. We named this concept the *command spy*.
- A large part of the knowledge that an expert interface will be concerned with is sequences. On a macro scale, the user's actions can be viewed as sequences of events, and on a micro scale they can be viewed as sequences of

commands. Sequences may be represented using rules, but this is often very cumbersome. Therefore we have introduced the concept of *scripts*, which is a data structure for describing sequences.

- The knowledge in the system should be split into different groups. Representing the procedural part of the problem, i.e. running Idpac, with scripts and confining the production rule system to handling knowledge specific to system identification gives a natural partitioning of knowledge into different kinds. In this way it might be possible to avoid having the expert knowledge be totally dependent on a specific target system.
- The script facility should capture the main ideas of macros. The popularity of Idpac has been strongly dependent on the writing of increasingly more complicated macros, but the macro facility has a few inherent disadvantages, for example that there is no good way to interact with an executing macro. The script concept is designed for interactive communication. To a great extent it captures the function of the macros and is probably a good complement to the macro facility. This seems to be a better solution than trying to squeeze more and more complicated functions into the existing macros.

According to these design goals, a first step towards a system layout has been taken. The different parts of the expert interface will be the command parser with its command grammar, the command history matcher with its script database, the production rule system with its rule base and a file handler. A first version containing all these elements except the file handler has been implemented.

2. Scripts

In our application there are three types of knowledge.

- Knowledge of command sequences, on what to do next. This depends on which state the user is in, i.e. what has been done so far.
- Identification specific knowledge, i.e. how to interpret results, find remedies to problems, etc.
- Knowledge about the data files created by Idpac.

In short the problem is to describe the semantics of an artificial language, Idpac, and to use expert rules about system identification. To do this a data structure that describes complicated sequences of events, for example Idpac commands, is needed. The expert rules are easily handled in a conventional production system. Inspired by techniques used in natural language understanding, see [6], we decided to try and specify the semantics as patterns. The actual command history is matched against such patterns in order to determine what the user is doing. From this the

system may give advice on what to do next and give expert advice on identification, for example the interpretation of the results. One way of doing this is to introduce scripts. A script is a data structure which consists of events (for example commands) that must occur in a given order. In Lisp a script can be implemented as a list of commands and control symbols.

It is of course possible to keep track of what the user has done, of "what state he is in", with a pure rulebased system, but this requires lots of flags to be set and reset by rules. This makes the rules messy and makes it difficult to introduce new rules in a consistent way. Designers of expert systems have observed this, and have included methods for controlling rules without having to put flags in the rules, see [7]. A better solution is to separate the mechanism for keeping track of what the user has done and the expert rules about identification.

It is possible to regard scripts as a language in which we try to express our knowledge of in what order it is sensible to give different Idpac commands. In order to make a script language a useful tool there must be ways to structure its content. Guided by our experience in using interactive programs we here give some elements which we feel should be part of a script language.

Basically a script consists of identifiers, in our case Idpac commands. Other elements are then used to structure these identifiers. Some pieces of scripts may be so useful and occur so frequently that we want to name them and use them in other scripts. We call such pieces script-procedures. There are command sequences which may have to be repeated several times before an identification is finished. For this we have introduced the REPEAT statement. Sometimes the user has to give a number of commands but the order in which he gives them is not important. This can be expressed with the ALL statement. The OR statement is used when the user may choose one of several possibilities.

The KSCALL statement, (KSCALL stands for Knowledge Source Call), contains facts which should be added to the database of a production system when the previous command has been matched. This is used to fire rules in the production systems that are associated with each script.

If scripts are regarded as a formal language these elements of the script language can be expressed in EBNF syntax. The look of the syntax mirrors the fact that we have implemented a prototype system in Lisp. We have used Lisp atoms and lists to express all language elements.

```
SCRIPT      ::= (SCRIPTCLAUSE [SCRIPTCLAUSE...])
SCRIPTCLAUSE ::= COMMAND | SCRIPTPROCEDURE | KSCALL |
                REPETITION | OR | ALL | EMPTY
COMMAND     ::= <identifier>
KSCALL      ::= (kscall <list> [<list>...])
SCRIPTPROCEDURE ::= (scriptprocedure <identifier>)
REPETITION  ::= (repeat SCRIPT)
OR          ::= (or SCRIPT [SCRIPT...])
ALL         ::= (all SCRIPT [SCRIPT...])
EMPTY      ::=
```

One or more scripts are matched against the commands typed by the user. From the matches the system is able to deduce possible goals for the user. The user does not have to answer any questions, but describes what he is trying to do by typing commands. If the user asks what to do next the system searches down the scripts from the last matched command in the command history until it finds a command. Actually it may find several commands if an ALL or OR statement is found.

There are, of course, several other control constructs that may be implemented. One would be a repetition with a countdown, i.e. specifying the maximum number of itera-

tions allowed. Another would be the SOME construct, matching one or more commands. This construct is easily expressed as (REPEAT (OR ...)) though.

At present the parameters of the commands do not influence the matching.

An Example

An example of what a script may look like, and some command histories that it will match is given below.

```
(CONV
 PLOT
 TREND
 (REPEAT (
  ML
  (REPEAT (
   (OR (RESID) (SPTRF BODE))))))
 STOP)
```

This script describes the following sequence of commands. First an ASCII file is read and converted to the internal Idpac file format with the command CONV, then a PLOT command is used to look at the data. Trends are removed with the TREND command. The actual parameter estimation is carried out with the ML command, and after that a validation is attempted by looking at the residuals with the RESID command or the transfer function of the identified system with SPTRF and BODE. The estimate-validate part of the session may be repeated several times, and we can choose to validate by looking at the residuals or at the transfer function. Finally the user terminates the session by typing STOP.

This is not necessarily the best way to carry out the identification, but is chosen to give an example of what can be expressed with scripts. Many command histories matches the given script, for example

```
> CONV      > CONV      > CONV
> PLOT      > PLOT      > PLOT
> TREND     > TREND     > TREND
> ML        > ML        > ML
> RESID     > SPTRF     > RESID
> ML        > BODE        > SPTRF
> SPTRF     > ML          > BODE
> BODE      > RESID      > ML
> STOP      > STOP      > SPTRF
                > BODE
                > STOP
```

Scripts are easily developed. Another version of the script above could look like this.

```
((OR
 (CONV)
 ()))
 PLOT
 (REPEAT (
  (OR
   (PLOT)
   (PLMAG)
   (TREND))))
 (REPEAT (
  ML
  (ALL
   (RESID)
   (SPTRF BODE))))
 STOP)
```

In this version the identification may start with a data file in either ASCII or internal format. The process of "cleaning" data is performed through a mixture of commands for looking at the data, removing outliers and eliminating trends.

In the parameter estimation this script requires that one looks both at the residuals and the transfer function. Here are some command histories which match the script.

```

> CONV          > PLOT          > PLOT
> PLOT          > PLMAG         > PLOT
> PLMAG        > PLOT          > TREND
> TREND        > TREND        > ML
> PLOT         > ML            > RESID
> ML           > RESID        > SPTRF
> SPTRF        > SPTRF        > BODE
> BODE         > BODE         > ML
> RESID        > STOP         > SPTRF
> STOP         > STOP         > BODE
               > RESID
               > STOP

```

3. The command spy

One of the main ideas of this paper is the concept of a *command spy*. The implications are twofold. First, we keep the command dialog of Idpac. Secondly, the expert system should not be noticed unless the user requests it. In this way, it is possible to retain the advantages of the non-expert communication instead of going into a more rigid Q/A style of dialog. A user who doesn't want any help will not feel any hindrance from the system.

The program module Intrac with the addition of the special Idpac features defines a language. In order to implement the command spy, a parser for the "Idpac language" must be developed. Since the Idpac language is specified in EBNF, this may be done using several different standard tools. One is the language Prolog, [8], another the UNIX™ based YACC, [9]. Parsing is also easily implemented in Lisp, and this approach was chosen in the project. The parser reads the command, checks it for syntactical correctness, and fills in defaults. There will also be a query mode, in which the user gets interactive help with syntax and parameters, [10].

The central part of the interface is the command history matcher. It uses a script database to guess what the user is doing. If the actual command history matches one or more scripts, the matcher is able to offer different kinds of help, for example suggest the next command, name a possible goal, and give information obtained from triggering the production rule system. There are two ways for the interface to get to know what the user is doing. First, the user may tell the system with a special command, secondly, the interface may guess by matching the actual command history against its different scripts.

If the user gives a command that doesn't match any script, the command history matcher issues a warning, but then lets the user continue. In this state no help is available but presumably the user knows what he is doing when he leaves the scripts. The system uses two guessing strategies in order to "get back into the game". First, it throws away the last, non-matched command, retaining the rest of the command history. This covers the case when the user does a few strange things in between but then returns to the main scheme. Secondly, the system tries to start a brand new command history, thus taking care of the case when the user gives up and starts with something else.

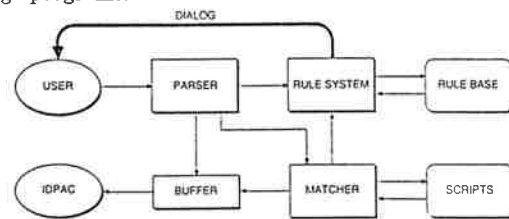
These guessing strategies are not crucial. If the system is not able to understand what the user is doing it won't work, but the user may still go on using Idpac as if nothing has happened. If he wants help, there is always the possibility of setting the help system right by the command that specifies a certain script. This command can also be used in order to reduce the number of matching scripts that may be present early in a session.

Unmatched command histories should be saved for later investigation. In this way it is possible to get ideas for new scripts, especially by checking what experts using the system do.

At certain points in the scripts there are function calls that put facts in a database and activate a production rule system. This is used in order to treat problems more specifically concerned with system identification. A typical example is giving advice about what kind of problems to look for after a maximum likelihood estimation. These depend on both the previous command sequence, captured by the scripts, as well as on earlier identification knowledge, such as whether the data set is short or long, if outliers have been removed, if similar data have been analysed before, and so on. This information is captured by the rule system. The production rule system uses an ordinary, forward chaining strategy to take care of the knowledge not contained in the scripts, see [11]. Its output is available through a special help command.

4. Implementation of a Small System

Some of the previous ideas have been implemented on a subset of Idpac. The system is written entirely in Franz Lisp and runs under VMS on a VAX 11/780. The goal was to build a small test system that would give us a feeling for the problems. We found that it was very convenient to work with Lisp, and that it was quite simple to develop large programs.



Layout of the system.

The command parser is implemented as a pattern matcher. It uses a command grammar to check the legality of the commands and also sticks on defaults in the same way that Idpac does. The command grammar is expressed in a special language, a more Lisp-like version of EBNF with a few additions. The query facility hasn't yet been implemented. It will prompt the user for missing arguments that cannot be defaulted, and in doing so start up an advice-giving dialog.

The parser could also have been implemented in other ways, for example using tools like YACC. But we didn't feel any great need for a more powerful tool. Lisp is perfectly able to handle our needs, and using Lisp we also avoided any interfacing problems. (A strong point, as anyone acquainted with such problems will recognize.)

The command history matcher was also implemented as a pattern matcher, using a script grammar expressed in a special "lispified" language equivalent to EBNF, in order to match the actual command history onto the script database.

Another alternative would be to use a production system to implement the script matcher. Yet another would be to develop a script-to-rules compiler, a program that would take script definitions as input and produce a script matching rule base for output. Such possibilities will be looked into later in the project.

The production rule system used is taken from an example in [11], with a few minor additions. As long as the

rule base is small, this solution is feasible. Since there are several existing frameworks that may easily be built into a Lisp system, an obvious development would be to use one, for example YAPS, [12].

Special attention has to be given to the problem of keeping track of data files. Idpac works with a great number of data files, and almost every command creates a new one. As for now, the user has to come up with a new name for every file created and he also has to keep track of the maze of files by himself. The next step in the project will be the development of a file handler. This device will use a file database, structured as an inheritance tree. It will take care of generating names, hiding uninteresting files, explain what operations have been performed on data in a file, and so on. This also means that the scripts will have to be modified, so that the command history matcher will be sensitive to the filenames given in the commands. Keeping track of the data files is a serious problem in Idpac and the development of a file handler is a most urgent project.

The development of script and rule databases of a realistic size is another part of the project. At present we have just a few scripts and rules for experiments.

A small set of rules for system identification has been developed in a backward chaining environment, see [13,14]. Approximately 1000 rules are needed to cope with most current Idpac applications. The use of scripts will probably reduce the size of the knowledge bases considerably.

To be able to interact with Idpac an interface must be developed for the expert system. One solution is to supply a set of subroutines that can start other processes in VMS, and then run Idpac in such a subprocess. This hasn't yet been done, but we feel that it is possible to do without too many changes in Idpac.

A small set of functions for creating subprocesses, creating mailboxes for communication with the subprocesses and giving other facilities, running under the VAX/VMS operating system, has been developed, see [15].

During the development of the system, we have tried to use an "object oriented" approach. A future step will be to use tools for this, for example a Flavors package, [16]. In this way every script may be viewed as an object with its own goals, associated data files, production system, etc.

5. Conclusions

One conclusion of the project at this stage is that it seems possible to use an expert system and retain the command style dialog. Another is that not all knowledge need be implemented by production rules. Scripts are probably a better way to represent sequences. A third conclusion is that the use of scripts supported by rules in a forward chaining strategy will probably reduce the overall size of the knowledge bases considerably.

Acknowledgements

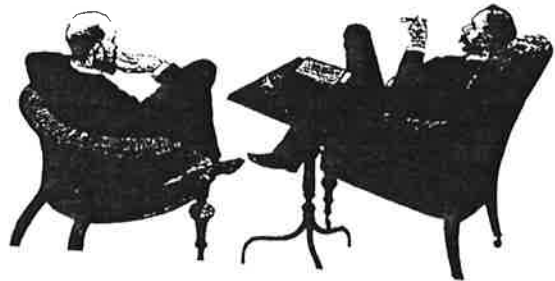
We would like to thank our supervisor Karl Johan Åström and our colleagues Sven Erik Mattsson and Karl-Erik Årzén for supporting us in our work.

The project is part of the Computer Aided Control Engineering Project at the Department of Automatic Control, Lund Institute of Technology, supported by STU, the National Swedish Board for Technical Development, under contract no. 85-3042.

References

1. Eykhoff, *System Identification. Parameter and State Estimation*. John Wiley & Sons, London, 1974.
2. Åström, "Modeling and Simulation Techniques", Agard Lecture Series No. 128, 1983.

3. Wieslander, *Interaction in Computer Aided Analysis and Design of Control Systems*, Ph.D. thesis, LUTFD2/(TFRT-1019), Department of Automatic Control, Lund Institute of Technology, 1979.
4. Åström, "Computer Aided Modeling, Analysis and Design of Control Systems — A Perspective", *Control Systems Magazine*, Vol. 3, No. 2, May 1983, pp. 4-16, 1983.
5. Åström, "Computer Aided Tools for Control System Design", in Jamshidi, Herget, (Eds.), *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam, 1985.
6. Schank, Riesbeck, *Inside Computer Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.
7. Bobrow, Stefik, *The LOOPS Manual*, Xerox PARC, 1983.
8. Clocksin and Mellish, *Programming in Prolog*, Springer-Verlag, Berlin, Heidelberg, New York, 1981.
9. Johnson, Yacc — *Yet Another Compiler Compiler*, Computing Center Technical Report No. 25, Bell Laboratories, Murray Hill, New Jersey, 1975.
10. Rimvall, Bomholt, "A Flexible Man-Machine Interface for CACSD Applications", Preprints of the 3rd IFAC/IFIP International Symposium, pp. 98-103, The Technical University of Denmark, Lyngby, Copenhagen, 1985.
11. Winston, Horn, *Lisp*, Addison-Wesley, Reading, Massachusetts, 1981.
12. Allen, YAPS — *Yet Another Production System*, TR-1146, Department of Computer Science, University of Maryland, 1983.
13. Larsson, *An Expert System Interface for Idpac*, Masters thesis, LUTFD2/(TFRT-5310), Department of Automatic Control, Lund Institute of Technology, 1984.
14. Larsson, Åström, "An Expert System Interface for Idpac", Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design, Santa Barbara, California, 1985.
15. Årzén, *A Real Time Environment for Expert Control*, Internal report, LUTFD2/(TFRT-7314), Department of Automatic Control, Lund Institute of Technology, 1986.
16. Allen et al, *The Maryland Artificial Intelligence Group Franz Lisp Environment*, TR-1226, Department of Computer Science, University of Maryland, 1984.



The authors in discussion.

SYSTEM DESCRIPTIONS

1. Introduction
2. Examples
3. Requirements
4. A prototype
5. Conclusions

MOTIVATION

CACE

Auto-tuning

Expert Interfaces

Expert Control



Qualitative as well as quantitative

Multiple representations

Different properties

Range of validity

SIMNON

Simulation of mixed continuous and discrete time systems.

Continuous system

$$dx/dt = f(x, u, t)$$

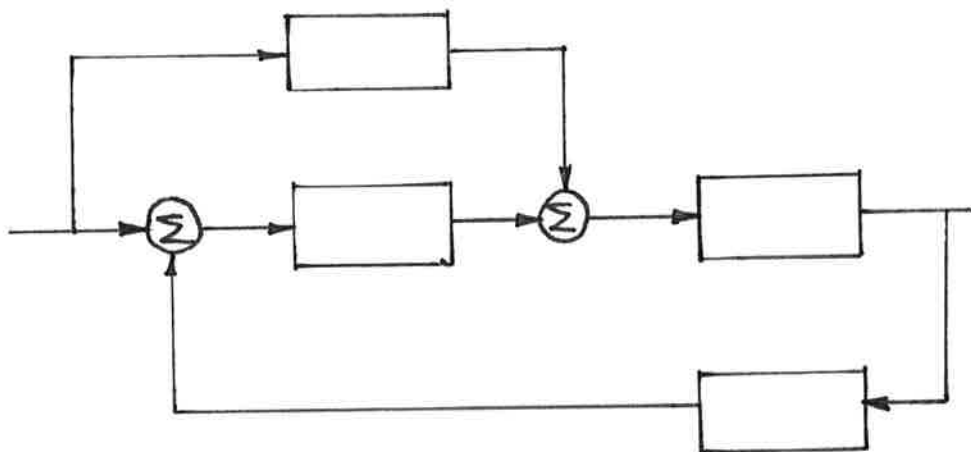
$$y = g(x, u, t)$$

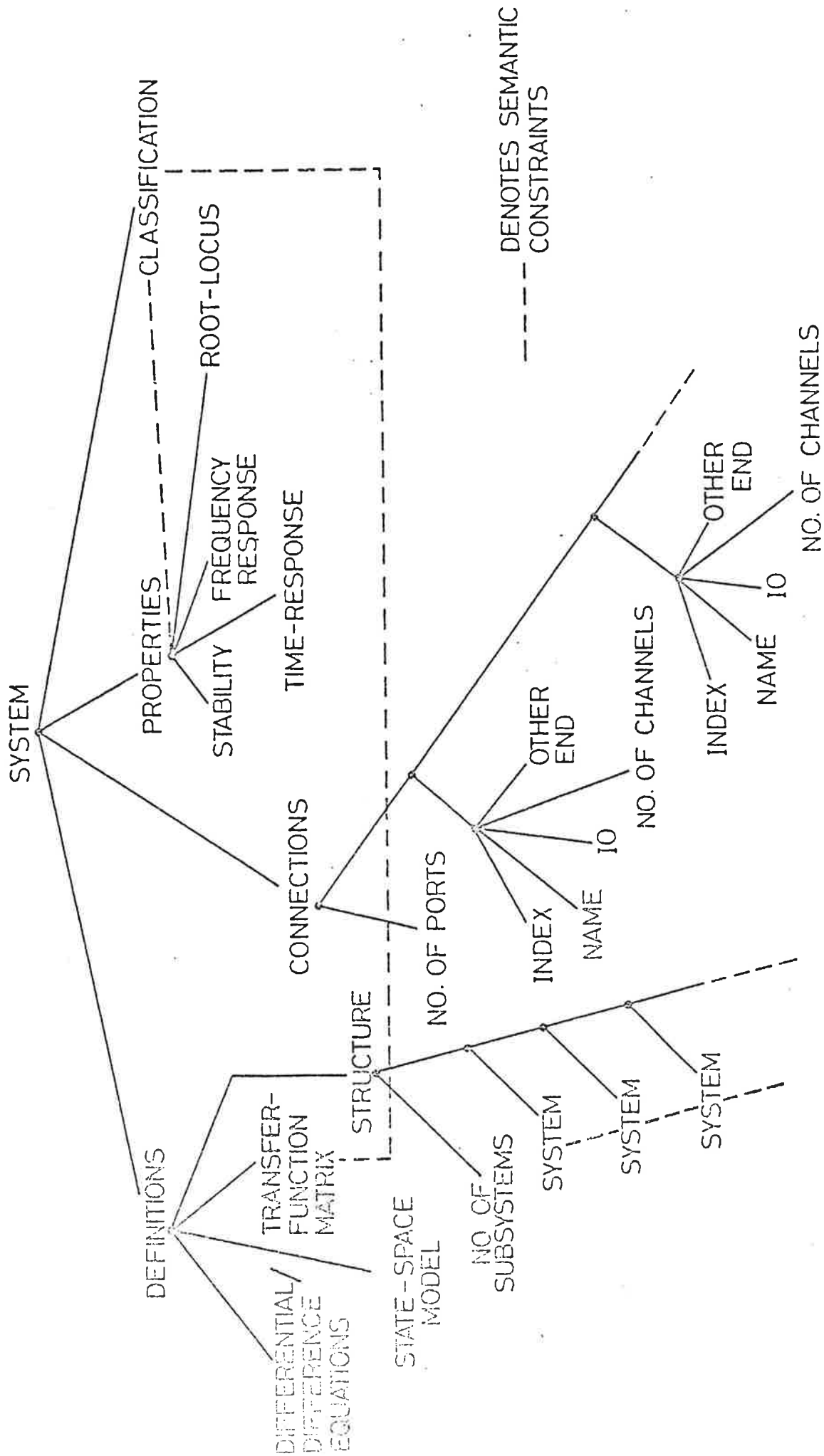
Discrete system

$$x(t_{k+1}) = f(x(t_k), u(t_k), t_k)$$

$$y(t_k) = g(x(t_k), u(t_k), t_k)$$

Connecting system





SEMANTIC DATA STRUCTURE

(NOT COMPLETELY CONSISTENT WITH TEXT)

FIGURE 1

SYSTEM PROPERTIES

Name

Inputs

Outputs

Subsystems

Connections

Behaviour

REPRESENTATIONS

Graphical

Textual

Notice many different versions

Notice different aggregation levels

QUANTITATIVE BEHAVIOUR

Notice several aggregation levels may be useful.

STATIC: $y = F(u)$

DYNAMIC:

Nonlinear State Model

State

Input map

Output map

Linear

Range of validity

Linear state A,B,C,D

Also $E dx/dt = Ax + Bu$

Matrix fraction

Impulse response

Frequency response

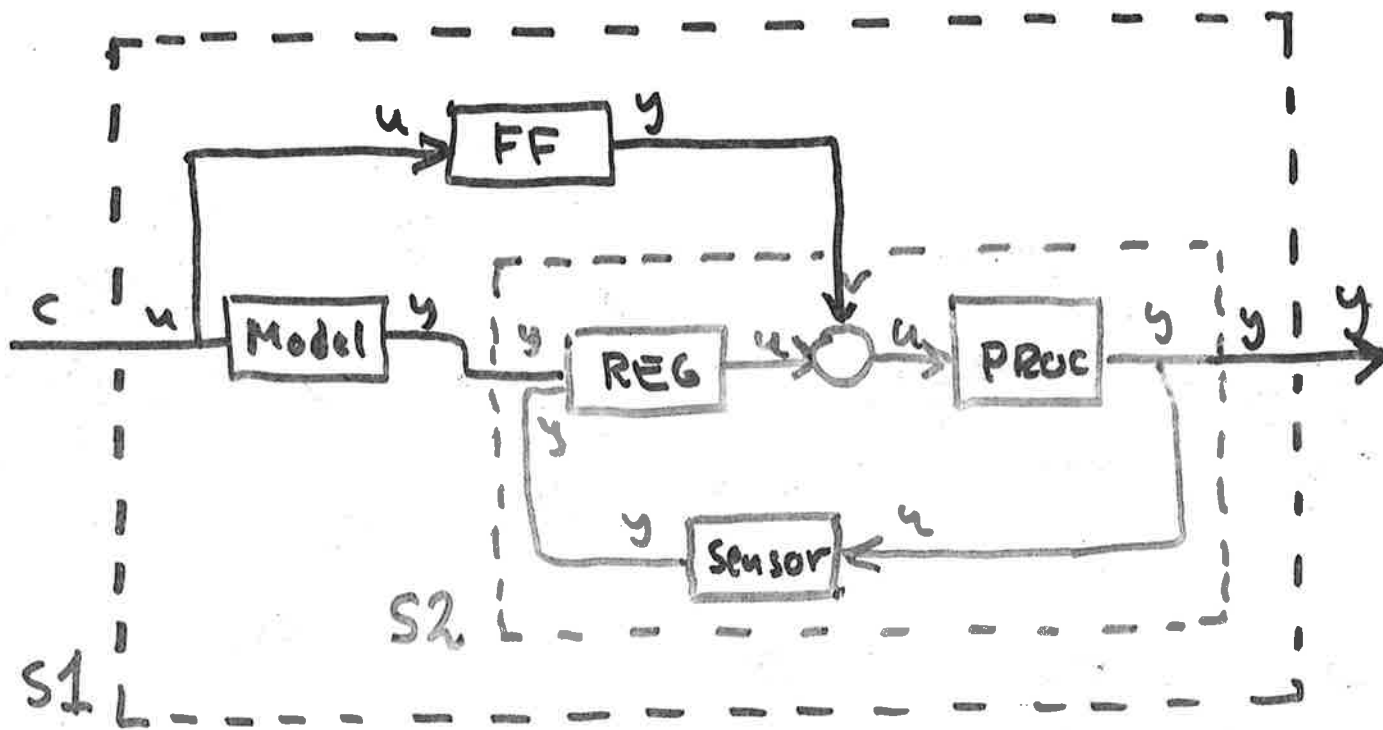
(Lots of properties!)

A PROTOTYPE

Hierarchical system descriptions

Linearization

Experimentation with Lisp



; **example 2:** System Decomposition & ...

(ShowSystem s1)

```
=====
; SYSTEM: S1
=====
```

```
; Inputs : (c)
; Outputs : (y)
; States : nil
```

```
; *** Subsystems ***
```

```
; ++++++
```

```
; (s2 model ff)
```

```
; *** Connections ***
```

```
; ++++++
```

```
; (c S1)
```

```
; (S1 y)
```

```
; *** Behaviour ***
```

```
; ++++++
```

```
; +++ no state equations defined +++
```

```
; +++ no output equations defined +++
```

```
; nil
```

(TableOfConnections proc)

```
; ((u proc)(proc y))
```

(ShowConnectionTable (TableOfConnections proc))

```
; (u proc)
```

```
; (proc y)
```

```
; nil
```

(StateEqns (Behaviour proc))

```
; ((((- (a * (x1 ** (1 / 2)))) + (b * u))((a * (x1 ** (1 / 2))) - (a * (x2 ** (1 / 2)))))
```

(OutputEqns (Behaviour proc))

```
; ((x2))
```

(ShowSubsystemHierarchy s1 7)

+++++

; Subsystems of: S1

+++++

;

;s2

;

; reg

;

; proc

;

; sensor

;

;model

;

;ff

;

;nil

(ShowBehaviourList (Behaviour proc))

;State-Eqns

;((- (a * (x1 ** (1 / 2)))) + (b * u))

;((a * (x1 ** (1 / 2))) - (a * (x2 ** (1 / 2))))

;Output-Eqns

;(x2)

;nil

(ShowBehaviourTable (Linearize proc))

;System - Matrix

;(a * ((1 / 2) * (x1 ** ((1 / 2) - 1)))) 0

;(a * ((1 / 2) * (x1 ** ((1 / 2) - 1)))) (- (a * ((1 / 2) * (x2 ** ((1 / 2) - 1))))

;Input Distribution - Matrix

;b

;0

;Measurement - Matrix

;0 1

;nil

(MeasurementMatrix (Linearize proc))

;((0 1))

(SystemMatrix (Linearize ff))

;((b))

(InputDistMatrix (Linearize sensor))

;(((t ** -1)))

(AllStates 'model)

;((x1 x2) nil nil nil)

(GetAllStates '(reg proc sensor))

;((i d) nil ((x1 x2) nil ((x) nil nil)))

(AllSubsystems s1)

;(s2 model ff reg proc sensor)

(AllSubsystems s2)

;(reg proc sensor)

(GetAllSubsystems '(s1 s2))

;(s2 model ff reg proc sensor)

(GetSubsystemHierarchy '(s1))

;(S1 (s2 model ff) s2 (reg proc sensor) reg nil proc nil sensor nil model nil ff nil)

SYSTEM REPRESENTATIONS

Karl Johan Åström
Department of Automatic Control
Lund Institute of Technology
PO Box 118
S-22100 Lund SWEDEN
phone +46-46-108781
telex S-33533 LUNIVER

Wolfgang Kreutzer
Department of Computer Science
University of Canterbury
Christchurch 1 NEW ZEELAND

Abstract

The notion of system is an essential element of control theory. The representation of systems is also a key issue in computer aided control engineering CACE. Systems can be represented in many different ways. There are graphical representations like block diagrams and signal flow diagrams. There are also mathematical representations like state space models and input-output relations which come in many different forms, matrix fractions, impulse responses, frequency responses. When working with control systems it is frequently useful to use several different representations of a system.

Only fairly primitive ways of representing systems are used in current CACSD systems. Typical examples are the Matlab derivatives where systems are described by matrices. In CTRL-C systems are described as a matrix quadruplet. In Matrix-X they are described as a composite square system matrix and an integer representing the system order. A slightly more sophisticated representation is used in the simulation language Simnon. This representation recognizes that a system has the properties, inputs, outputs and states. Simnon also allows a system to be described as an interconnection of subsystems. However only flat interconnections are allowed. See Åström (1984).

In this paper we will present much more flexible ways of describing a system which are based on object-oriented programming. The discussion is restricted to systems which are composed of subsystems with unidirectional interaction. Such systems can be represented by conventional block diagrams. There are also more general ways to define systems by using equations or relations. This is done in the simulation language Dymola. See Elmqvist (1978) and Elmqvist and Mattsson (This conference).

It is shown that a general structural description of hierarchically connected systems can be constructed from simple ingredients by making a system an object with the properties Name, Inputs, Outputs, Subsystems and Connections. Such a description is very flexible. It is particularly convenient for describing large systems with a regular structure.

The primitive operations on a system include constructors, like MakeSystem, AddSubSystems, NewSystem destructors like ClearSystem and query operators like Inputs?, and ShowSystemHierarchy. There are however also other useful

operations like StructureVerification which checks that all the ports of the subsystems are connected and operators like Loops and SystemTopology that investigate the topological properties of the system. Other operators are TraceStructure.

To provide a good man-machine interface it is also necessary to have tools for graphical display and graphical editing of different system properties.

It is also necessary to add behavioural descriptions to obtain a useful tool. This is done by creating new objects which inherit the structural properties discussed above and in addition describe how the system behaves. The behaviour can be characterized in many different ways. A state description is one of the simpler alternatives. This can be covered by introducing the new properties States, StateTransitionMap and OutputMap. In addition it is useful to add the property Parameters to indicate the parameters of the system which are accessible externally. To cover sampled and discrete time system it is necessary to introduce the properties of Time and SamplingTime as is done in the Simnon language. It is also very useful to introduce the property ValidityRange to indicate the region of validity of the model. This can be described as a subset of the product of the input spaces and the state spaces. With such a feature it is possible to write a simulation program which will raise an exception if the state of the system goes outside the region of validity during a simulation.

Several new operators are needed to deal with behavioural properties. At the lowest level we have the basic constructors, destructors and query functions. In addition we need operators for analysing and simulating a system. Examples of these are StaticModel, OperatingPoint, Linearize, Stable?, Reachable?, Observable?, TransferFunction, Poles, Zeros, SensitivityDerivative, Simulate, etc.

The behavioural descriptions should also allow a given system to be described by different models of different complexity. Apart from the detailed quantitative descriptions it is also useful to be able to deal with less accurate and even qualitative descriptions of behaviour. This includes assessment of gain, time constants, system type, stability margins, degree of nonlinearity etc. Associated with these properties we also need operators to obtain these properties from the more detailed representations.

To see the value of the qualitative characterizations consider e.g. the simulation of a large system with a loop where the dominating time constant is of the order of seconds. It may then be justifiable to use only static descriptions for those subsystems in the loop which have time constants shorter than a millisecond.

The paper also describes a small prototype implementation which admits experimentation with several of the ideas introduced. This prototype which is written in Lisp admits hierarchical system representation and formal as well as numerical manipulation of the systems. Experiences from experiments with the prototype will also be described. The experiments indicate that the approach is quite useful and that CACE systems with the properties discussed can indeed be implemented.

COMBINATION OF FORMULA MANIPULATION AND NUMERICS

Motives for symbolic formula manipulation

Structure information is very important

Analytical expression may give good insights

May eliminate numerical problem

Automatic generation of code

Education

SYMBOLIC FORMULA MANIPULATION IN MACSYMA

Linearization

$$\dot{x} = f(x, u) \rightarrow \dot{x} = Ax + Bu$$

Stability analysis

Routh's criterion

Jury's criterion

Maximum Likelihood method

$$E(\theta - \theta_o)^2$$

Sampling

Transfer function \rightarrow Pulse transfer function

$$G(s) \xrightarrow{h} H(z)$$

State space continuous time \rightarrow discrete time

$$(A, B, C, \tau) \xrightarrow{h} (\Phi, \Gamma_0, \Gamma_1, C)$$

GEOMETRY APPROACH (Time domain)

Observability

Reachability

AB-invariant subspaces

Kernel $\{X|AX = 0\}$

Inverse Image $\{X|AX = B\}$ (A singular)

Gram-Schmidt (orthogonal base)

Gauss elimination $PA=LU$

Normal forms (similar matrices)

Characteristic polynomial $p_A(\lambda)$

Minimal polynomial $\pi_A(\lambda)$

General Matrix functions $f(A)$

FACTORIZATION APPROACH (Frequency domain)

Polynomials

$$AX+BY=C \quad \text{“DAB equation”}$$

Polynomial matrices

Smith form

Smith-McMillan form

Hermite form

Matrix Fraction Decompositions, MFD

Right MFD

Left MFD

Irreducible MFD

Column reduction (-proper)

Multivariable realizations → back to State space

Controller form

Controllability form

Observer form

Observability form

PROGRAM EXAMPLE

```

SAMP(G,[y]):=Block([sum,i,x,s,h,z],

if G=help then return((
print("----- SAMPLING OF A TRANSFER FUNCTION ----- "),
print("Author: Ulf Holmberg                                     "),
print("-----"),
print("The sampling function uses the Residue formula         "),
print("Sum Res (z-1)/(z-exp(sh))*G(s)/s                         "),
print(" Samp(G)                                                  "),
print("   ->H(z)                                                  "),
print(" Samp(G,lambda)                                           "),
print("   ->H(lambda)                                           "),
print(" -----"))),

/* Define independent variable (default s) */,
if length(y)=1 then s:part(y,1),

/* --- Calculate poles --- */,
x:g/s,
pol:transpose(solve(x^-1,s)),

/* --- Put the poles in a vector ---- */,
for i:1 thru length(pol) do pol:setelm(pol,i,1,2),i,1,pol),

/* Calculate the sum of all residues of : */,
x:((z-1)/(z-exp(s*h)))*x,

sum:0,
for i:1 thru length(pol) do sum:residue(x,s,pol[i,1])+sum,
sum:factor(sum)$

```

POLYNOMIAL MATRICES

The following example will serve as an illustration of the above described macsyms functions. The cpu-time was about 4 minutes.

(c18) /* This is a DEMO, involving polynomial matrix manipulation such as

Matrix fraction decomposition (MFD),
Hermite form,
Greatest Common (right) Divisor,
Coprime MFD (Irreducible forms),
Column reduction,

and realizations to corresponding multivariable controller forms.

----- */

loadfile("use:[ulfh.macsyma]h.")\$

(c19) /* Consider the following transfer function matrix */

h;

(d19)

$$\begin{bmatrix} \frac{1}{(s-1)^2} & \frac{1}{(s-1)(s+3)} \\ \frac{6}{(s-1)(s+3)^2} & \frac{s-2}{(s+3)^2} \end{bmatrix}$$

(c20) /* First, let us make a trivial matrix fraction decomposition */

d:denomright(h);

(d35)

$$\begin{bmatrix} (s-1)^2 & (s+3)^2 & 0 \\ 0 & (s-1)(s+3)^2 \end{bmatrix}$$

(c36) n:factor(h.d);

(d36)

$$\begin{bmatrix} (s+3)^2 & s+3 \\ -6(s-1) & (s-2)(s-1) \end{bmatrix}$$

68
(c37) /* Make a (non-minimal) controller realization */

ans1: controller(d,n);

$$(d43) \quad [a = \begin{bmatrix} -4 & 2 & 12 & -9 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & -3 & 9 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, b = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

$$c = \begin{bmatrix} 0 & 1 & 6 & 9 & 0 & 1 & 3 \\ 0 & 0 & -6 & 6 & 1 & -3 & 2 \end{bmatrix}]$$

(c44) /* We now want to make a minimal realization.
Thus we remove the greatest common right divisor of D and N
to get an irreducible matrix fraction decomposition. */

hermite(addrow(d,n));

$$(d44) \begin{bmatrix} 1 & \frac{19}{48} - \frac{7s}{48} \\ 0 & s^2 + 2s - 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

(c45) r: submatrix(3,4,%);

$$(d45) \begin{bmatrix} 1 & \frac{19}{48} - \frac{7s}{48} \\ 0 & s^2 + 2s - 3 \end{bmatrix}$$

(c46) dd: factor(d. (r^^-1));

$$(d46) \begin{bmatrix} (s-1)^2 (s+3)^2 \frac{(s-1)(s+3)(7s-19)}{48} \\ 0 & s+3 \end{bmatrix}$$

(c47) nm: factor(n. (r^^-1));

$$(d47) \begin{bmatrix} (s+3)^2 \frac{7s+9}{48} \\ -6(s-1) \frac{1}{8} \end{bmatrix}$$

(c48) /* Let us now try the realization algorithm */

controller(dd,nn);

(d48) D-matrix not column-reduced

(c49) /* Aha, we first have to make the denominator matrix column reduced. */

ddd: columnreduce(dd);

(d52)

$$\left[\begin{array}{c} \frac{480 s^2}{49} + \frac{960 s}{49} - \frac{1440}{49} \\ \frac{48 s^2}{7} - \frac{2592 s}{49} - \frac{4752}{49} \end{array} \quad \left[\begin{array}{c} \frac{7 s^3}{48} - \frac{5 s^2}{48} - \frac{59 s}{48} + \frac{19}{16} \\ s + 3 \end{array} \right] \right]$$

(c53) /* The corresponding unimodular matrix that made all the elemental column operations was */

u: factor((dd^-1).ddd);

(d53)

$$\left[\begin{array}{c} 1 \\ \frac{48 (7 s + 33)}{49} \end{array} \quad \left[\begin{array}{c} 0 \\ 1 \end{array} \right] \right]$$

(c54) /* We have to do the same operation on the numerator matrix */

nnn: factor(nn.u);

(d54)

$$\left[\begin{array}{c} \frac{144}{49} \\ \frac{48 (7 s - 2)}{49} \end{array} \quad \left[\begin{array}{c} \frac{7 s + 9}{48} \\ 1 \\ 3 \\ 2 \end{array} \right] \right]$$

(c55) /* Construct the minimal controller form */

ans2: controller(ddd,nnn);

$$(d55) \quad a = \begin{bmatrix} \frac{54}{7} & \frac{99}{7} & 0 & \frac{7}{48} & \frac{7}{16} \\ 1 & 0 & 0 & 0 & 0 \\ \frac{921600}{2401} & \frac{2764800}{2401} & 5 & 67 & 1839 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 & \frac{7}{48} \\ 0 & 0 \\ 48 & 480 \\ 7 & 49 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

$$c = \begin{bmatrix} 0 & \frac{144}{49} & 0 & \frac{7}{48} & \frac{3}{16} \\ 48 & 96 & 0 & 0 & 1 \\ \frac{7}{7} & \frac{49}{49} & 0 & 0 & 8 \end{bmatrix}$$

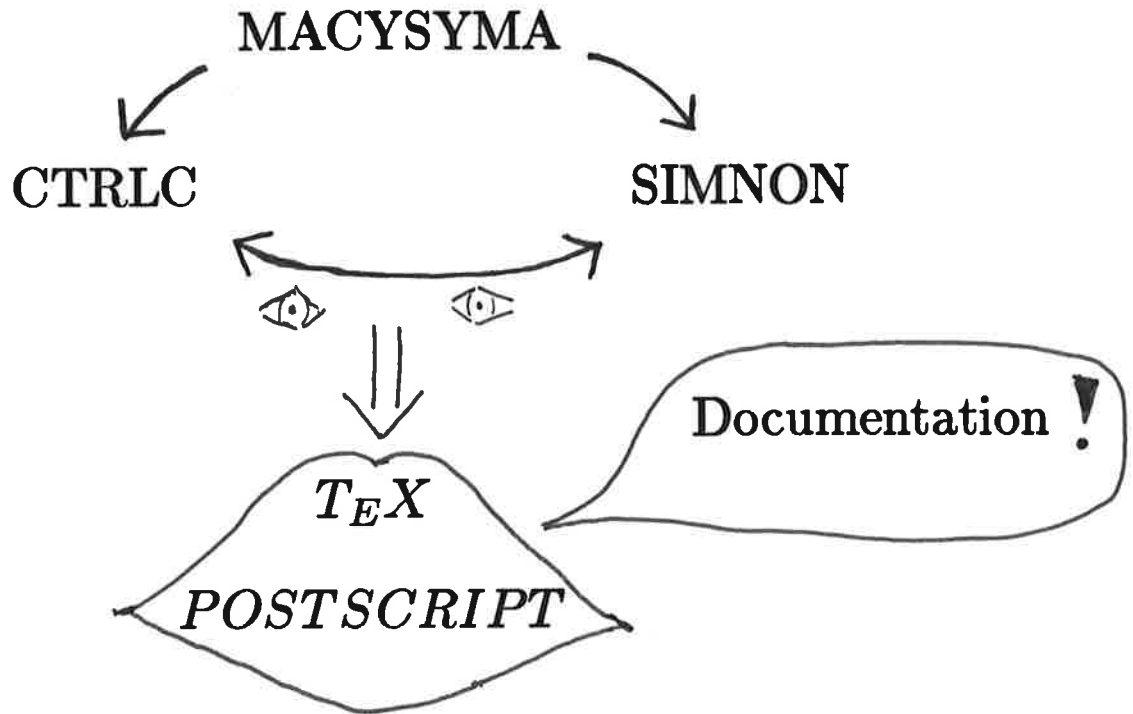
(c56) /* Let us check the result */

factor(c.((s*ident(5)-a)^-1).b),ans2;

$$(d56) \quad \begin{bmatrix} \frac{1}{(s-1)^2} & \frac{1}{(s-1)(s+3)} \\ \frac{6}{(s-1)^2(s+3)^2} & \frac{s-2}{(s+3)^2} \end{bmatrix}$$

(c58) /* End of DEMO */

INTEGRATING SYMBOLIC AND NUMERICAL TOOLS FOR LINEAR SYSTEM ANALYSIS



Submitted to the SIAM Conference on Linear Algebra,
12-14 Aug. 1986.

Integrating Symbolic and Numerical Tools for Linear System Analysis

There are presently several good programs for dealing with numerical and symbolic mathematics. This paper demonstrates some ideas on how to combine existing programs from different categories. As an illustration, these ideas are applied to combining the differential equation program Simnon, the matrix-manipulation program CTRL-C, the symbolic manipulation program MACSYMA, the type-setting program T_EX, and the page description language PostScript.

We demonstrate macros, functions and programs for solving more composite problems arising in linear system analysis, using symbolic and numerical computation. Interface to automatic documentation, T_EX, and PostScript are presented.

Ulf Holmberg

Mats Lilja

Bengt Mårtensson

Department of Automatic Control

Lund Institute of Technology

Box 118

S-221 00 Lund, SWEDEN

*Submitted to the SIAM Conference on Linear Algebra, Boston,
Aug 12-14, 1986.*

EXPERT CONTROL

Character

Response

Results

CPC III

Southampton

ACC

Future

TOWARDS INTELLIGENT CONTROL

Karl Johan Åström
Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Abstract

In spite of significant advances in control theory and computer hardware most of the control loops in operation today are based on the same ideas as the pneumatic regulators of 50 years ago. Similarly many of the distributed control systems in operation are simply implementation of old ideas in new hardware. With modest extrapolation of current trends in computer development we will have the supercomputer power of the seventies packaged in a desktop in the late eighties or early nineties. These drastic improvement of computer hardware opens up new possibilities for control system architectures with drastically different capabilities.

There are some indications that nonconventional control algorithms are practically useful. A number of devices for automatic tuning of regulators have recently been announced by several manufacturers. Adaptive controllers have also been introduced by some manufacturers.

Process knowledge is a key factor in process control. Current systems use only modest amounts of process knowledge. They do not extract and refine process knowledge as they operate. This has far reaching consequences. It is for example very difficult to benefit from the experience of an old system when installing a new system.

In this paper we will speculate on new ideas about system design which result in process control systems with drastically new capabilities. The starting point of our discussion are the devices for automatic tuning of simple control loops which appeared in the early eighties. See Åström and Häggglund (1, 2). These schemes offer interesting possibilities for automatic tuning and information gathering in simple loops. Combined with tables for gain scheduling they also offer possibilities for acquisition and storing of simple knowledge about the process dynamics. Adaptive control is a second element, see Åström (3). It offers the potential to acquire more detailed information about the process through on-line parameter estimation. Adaptive control does however require a priori information about the process to be used in a safe and convenient way. See Isermann and Lachmann (4) and Wittenmark and Åström (5). Such information can however be derived from auto-tuning experiments.

Thinking in along these lines we are immediately led to systems which combine a several different algorithms. We can imagine a system which has a collection of algorithms for monitoring, control, parameter estimation and control design. This leads to a combination of advanced control techniques with the methods of knowledge-based systems in artificial intelligence, see Winston (6), Barr and Feigenbaum (7) and Hayes-Roth et al. (8). We can visualize a system where a collection of algorithms for control, monitoring, parameter estimation, control design, auto-tuning, adaptation and learning are orchestrated by a knowledge based expert system. Such a system was described in Åström and Anton (9), Åström et al. (10) and Åström (11, 12) under the name of expert control. Experiments with prototype systems of this character are described in Årzén (13, 14).

In this paper we start by describing different quantitative and qualitative ways to describe process knowledge. For simple control loops the knowledge can be a crude qualitative characterization of the control problem e.g. if the main difficulty is due to linear dynamics, nonlinearities or disturbances. The linear dynamics is often described by a few parameters like steady state gain, time constants and time delay. The nonlinearities can e.g. be related to actuator saturations. The disturbances can be characterized by some measure of amplitude and frequency. The coupling between loops can in simple cases be captured by simple measures like the relative gain array. More detailed description of dynam-

ics, coupling and disturbances are needed for more sophisticated control loops. The knowledge used in different control strategies is also discussed together with methods for acquiring the knowledge. Different methods for automatic tuning of simple regulators and adaptive control techniques are discussed. The ideas of expert control are then introduced. Simple experiments with it are described.

The paper is concluded by some speculations on the possibilities of the technique. By combining AI methodology with ideas from automatic control it is possible to design systems with several interesting features. It is possible to deal with qualitative as well as quantitative knowledge. The process knowledge can be structured. It is possible to explicitly use the knowledge of operators as well as sensors. The system will also acquire knowledge during its operation. Since the knowledge is structured and accessible it can be extracted from the system both during operation and at replacement time. When the system is running we can ask questions like:

- o How well is the process running?
- o Are the disturbances normal?
- o What control law is currently being used?
- o Why is derivative action not needed on this loop?
- o Tell me the loops where dead time compensation is needed?
- o List all the loops where the tuning constants have been changed significantly during the past two months.
- o Tune the following loops.
- o Monitor the stability margins for this loop.

Knowledge based expert systems have also many other uses in process control. A system where an expert system is put on top of a conventional distributed control system to give advice about alarm and set point control, is given in Moore et al. (15). It has also been proposed to use knowledge based systems to aid in control system design (16) and to reconfigure control systems during operation (17).

References

1. Åström, K.J. and T. Häggglund. Automatic tuning of simple regulators. *Preprints IFAC workshop on Adaptive systems in control and signal Processing*, San Francisco, California, 1983.
2. Åström, K.J. and T. Häggglund. Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica* 20 (1984) 645-651.
3. Åström, K.J. Theory and applications of adaptive control—A survey. *Automatica* 19 (1983) 471-486.
4. Isermann, R., and K.-H. Lachmann. Parameter-adaptive control with configuration aids and supervision functions. *Automatica*, 21 (1985) 625-638.
5. Wittenmark, B. and K.J. Åström. Practical issues in the implementation of adaptive control. *Automatica*, 20 (1984) 595-605.
6. Winston, P.H. *Artificial Intelligence*. Addison-Wesley, Reading, Mass., 1977.
7. Barr, A. and E.A. Feigenbaum (Eds.). *The Handbook of Artificial Intelligence, Volume 2*. William Kaufmann, Inc., Los Altos, California, 1982.
8. Hayes-Roth, F., D. Waterman, and D. Lenat. *Building Expert Systems*. Addison-Wesley, Reading, Mass., 1983.
9. Åström, K.J., and J.J. Anton. Expert control, *Preprints 9th IFAC World Congress*, Budapest, Hungary, 1984.
10. Åström, K.J., J.J. Anton, and K.E. Årzén. Expert control. *Automatica*, 20, (1986) May.
11. Åström, K.J. Auto-tuning, adaptation and expert control. *Proc. American Control Conference*, Boston, Mass., 1985.
12. Åström, K.J. Auto-tuning, adaptation and smart control. *Proc. Chemical Process Control III*, Asilomar, CACHE, Austin, California, 1986.
13. Årzén, K.E. Experiments with expert systems for process control. To appear in *Proc. 1st Int. Conf. on Applications of Artificial Intelligence to Engineering Practice*, Southampton, UK, 1986.
14. Årzén, K.E. Use of expert systems in closed loop feedback control. *American Control Conference*, Seattle, Washington, 1986.
15. Moore, R.L., L.B. Hawkinson, C.G. Knickerbocker, and L.M. Churchman. Expert systems applications in industry. *Proc. ISA Int. Conf.*, Houston, Texas, 1984.
16. James, J.R., D.K. Fredrick, and J.H. Taylor. The use of expert-systems programming techniques for the design of lead-lag compensators. *Proc. IEE Int. Conf. Control 85*, Cambridge, UK, 1985.
17. Trankle, T.L., and L.Z. Markosian. An expert system for control system design. *Proc. IEE Int. Conf. Control 85*, Cambridge, UK, 1985.

Expert Systems for Process Control

Karl-Erik Årzén

**Department of Automatic Control
Lund Institute of Technology
Lund, Sweden**

POTENTIAL APPLICATIONS

Off-line -- on-line

Whole plant -- single loop

Operator tool -- closed loop

Process and control design

*Monitoring and diagnosis of
process upsets*

Alarm analysis - Three Mile Island

Expert controller

EXPERT CONTROL

- **Feedback control with an rule-based expert system.**
- **Acquires knowledge through on-line experiments and from process operators.**
- **Orchestrates numerical algorithms for control, identification and supervision.**
- **Successively increases and refines the knowledge about the plant.**

MOTIVATION

- **Ordinary controller contains a large "safety jacket" of logics to ensure safe operation.**
- **The combination of different algorithms increases the amount of logic.**
- **A division between logic and numerical algorithms is achieved.**
- **The interactive expert system environment provides a convenient workbench for experiments with new control structures.**

LANGUAGES

Most expert systems are written in LISP or PROLOG.

Numerical algorithms available in languages like Pascal, ADA or FORTRAN.

Implementation in two parts.

REALTIME PROBLEM

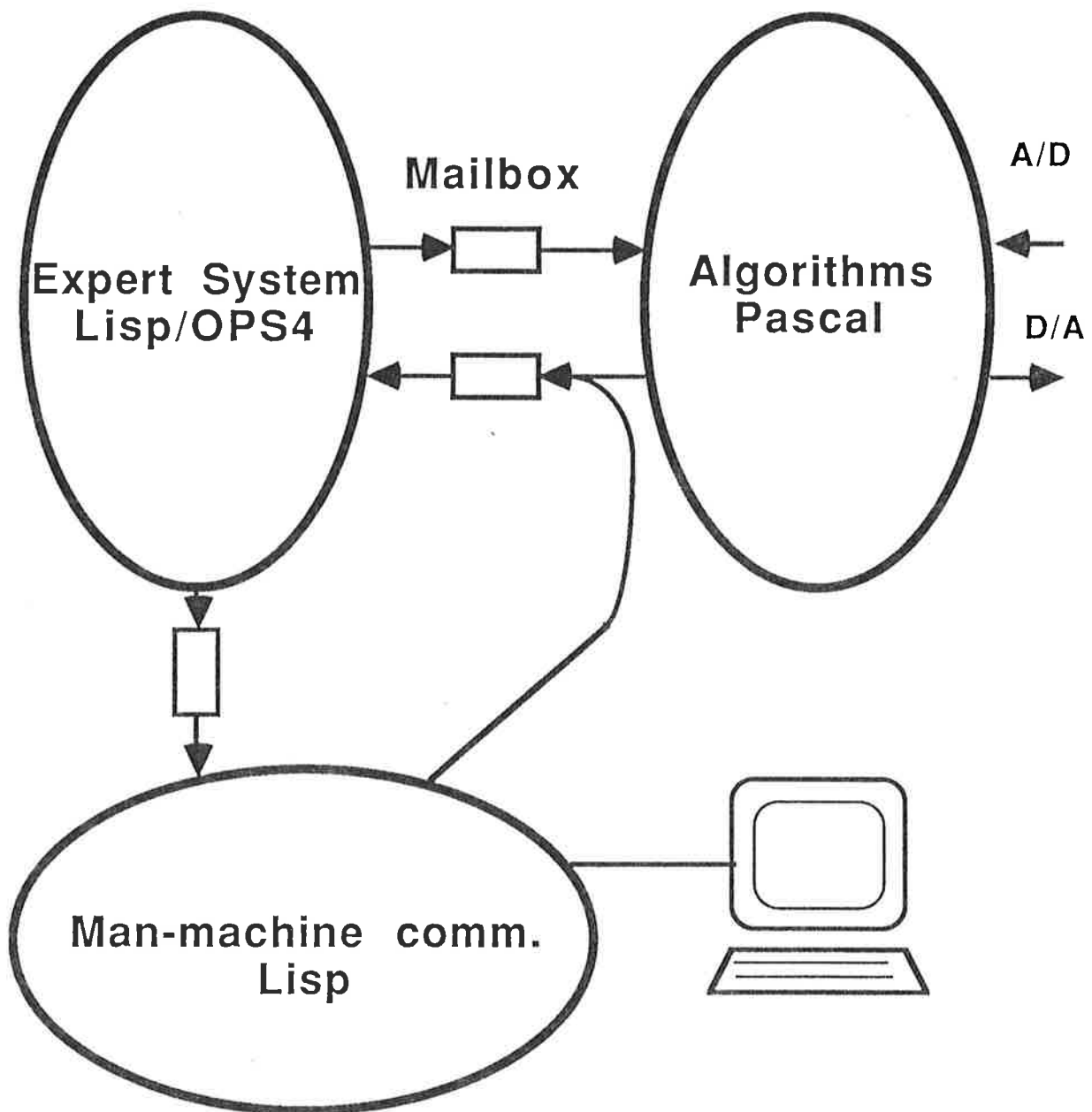
Expert systems -- slow.

Control -- time constraints.

Implementation divided into parallel tasks.

PROTOTYPE SYSTEM

Vax 11/780 VMS & Eunice



CONCLUSIONS

- **Promising prototype.**
- **Knowledgebase of practical control.**
- **Algorithm development.**
- **Expert control architecture under development.**

PLANNING

Operators could be activated in three different ways.

Operator-based activation: The operators suspend and resume each other. No plan generator.

Pre-stored plans: A finite number of plans are stored in the database.

Dynamic plan generation: The plans are dynamically generated. The overall goal is compared with the actual state and the goals and preconditions of the operators.

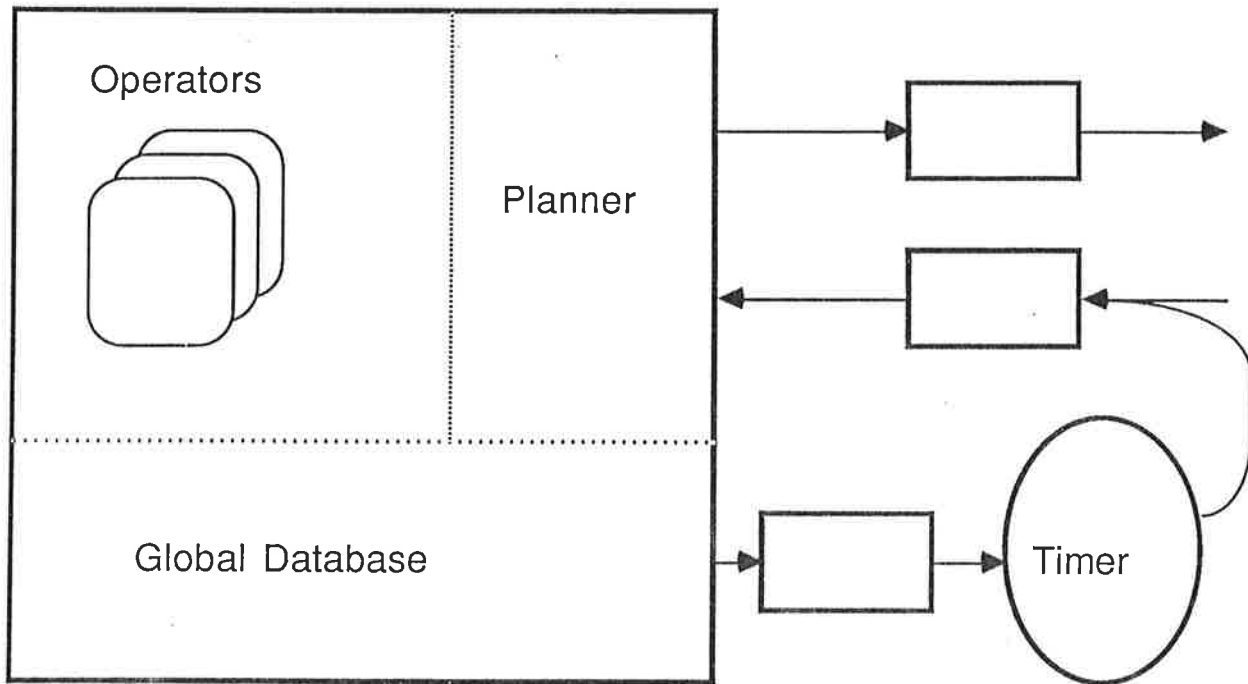
The actual plan is executed by the plan executor.

Replanning is performed when goals are not achieved.

A plan notation has been developed.

IMPLEMENTATION

YAPS



Operators: YAPS or backward chainer.

YAPS OPERATORS

Primitives for:

Waiting a specified time.

Waiting for a specified data element to be entered.

Creating, removing and modifying database objects both locally and globally.

Resuming and suspending other operator.

Manipulating the goal stack.

STATUS

The skeleton of an expert control architecture has been developed.

This will be the basis for experiments with intelligent controllers.

Modifications and further developments will take place during the experiments.

Begin text of second and succeeding pages here

Start second column second and succeeding pages here

USE OF EXPERT SYSTEMS IN CLOSED LOOP FEEDBACK CONTROL

Karl-Erik Årzén

Department of Automatic Control,
Lund Institute of Technology,
Box 118, S-221 00 Lund, Sweden.

Begin text of first page here. Abstract first. Please leave 1/2" space between end of Abstract and first line of text.

Begin second column of text here on first page only.

Abstract: Different uses of expert systems in process control are discussed. The paper concentrates on expert systems for closed single loop control. Motivation for this is given and a prototype experiment is described. The experiment is evaluated and an expert control architecture is proposed. The system can be compared with a real-time operating system. An implementation based on YAPS and object-oriented programming is described.

1. INTRODUCTION

Expert systems have many potential applications in process control. The application domain stretches from the entire plant to the single control loop. Both off-line and real-time problems exist. In this paper the expert system is used in real-time as a part of a single control loop. Many applications incorporate expert operator knowledge into the system. Our application is instead focused on incorporating more control knowledge into the controller. Good control requires high-quality process knowledge. This can be achieved in two ways, either directly from the operator or through experiments with the controlled process. The goal of the expert system is to build up the necessary process knowledge required for good control.

The paper is organized as follows. General expert system applications in process control are discussed in Section 2. Section 3 focuses on the single control loop. A prototype experiment is described. In Section 4 an expert control architecture is proposed and its implementation is discussed in Section 5.

2. EXPERT SYSTEMS IN PROCESS CONTROL

Expert systems is an area of Artificial Intelligence (AI) that has expanded rapidly during the last years, (Hayes-Roth *et al.*, 1983; Waterman 1986). Expert systems are used to solve problems that normally require human expertise and where traditional computer solutions are infeasible. The successful applications have all been in areas where

high-quality knowledge dominates over common sense. This is true for many aspects of process control. The idea of adding heuristics to process control is not new. In Crossman and Cooke (1962), a heuristic decision program that used experience to enhance its performance was proposed for manual control of systems with slow dynamics.

Examples of possible off-line applications in process control are process design and control design. Process design has no clean analytical solution. Expert system assistance has a high potential value since process design strongly affects the achievable control quality. In control design expert systems can guide the selection of appropriate control structures, (Umeda and Niida, 1986), at the global level. Expert systems have also been integrated with computer aided control design packages, (James *et al.*, 1985, Birdwell *et al.*, 1985, Larsson and Åström, 1985). It has also been suggested to use expert systems to assist in the parameter settings of adaptive controllers, (Sanoff and Wellstead, 1985).

With few exceptions, existing expert systems are based on propositional logic or first-order predicate calculus. Sometimes they also allow multi-valued logic, e.g. expressions with the values true, false or unknown. This is however not enough for a real time expert system. Such a system may have to draw conclusions based on incomplete facts. Facts may also change after conclusions have been drawn from them. This means that the system has to backtrack and reconsider these conclusions. Several non-standard logics exist e.g. non-monotonic logic, (McDermott and Doyle, 1980) and temporal logic, (McDermott, 1982, Allen, 1984), that partially solve these problems but they have not yet been applied successfully.

In spite of these problems real-time expert systems exist. These systems tend to circumvent the fundamental problem in different ways. The method used in PICON, (Moore *et al.*, 1984, 1985), is to attach a duration time to all database elements and to test the rules periodically. When the duration of an element ends all concluded elements are withdrawn. It is also possible to assume that database elements are valid only in a certain context. The usual way, however, is to use the engineering, ad hoc method and take

care of these issues explicitly in the rules.

Monitoring and diagnosis are the most common real time applications. Plant-specific knowledge e.g. cause-effect relations, is used to locate a fault which caused a process upset or an alarm and to give advice on corrective action. Work is being done e.g. on nuclear power plants, (Nelson, 1982), and chemical plants, (Palowitch and Kramer, 1985). A monitoring expert system could also give advice on control optimization.

In these applications the expert system is used as a tool for the operator. The expert system loop can also operate in closed loop, i.e. affect the controlled plant directly. On a global level the expert system could e.g. be used for set-point control to optimize the system. Process start-up and production changes, is another possible area. In the aircraft industry expert systems are suggested that reconfigure the flight control system in case of damage, see e.g. Trankle and Markosian (1985). The topic of the rest of this paper is the use of expert systems in closed single loops. The ideas were first outlined in Åström and Anton (1984).

3. CLOSED LOOP EXPERT CONTROL

The present project aims at incorporating a rule based expert system in a feedback control loop. The goal of the expert system is to enhance the performance of the single loop controller and to learn as much as possible about the controlled process. This is achieved by orchestrating the application of different numerical algorithms to the process in an "intelligent" way. The numerical algorithms can be of different types: control algorithms, identification algorithms and monitoring algorithms. The control algorithms may be of varying complexity, from simple PI or relay controllers to more complicated optimal or pole-placement algorithms. The identification algorithms may range from simple algorithms for estimation of static gain to more complex algorithms such as the Least-Squares algorithm, (Åström and Wittenmark 1973). Supervisory algorithms should detect e.g. static errors, alarm level crossings and ringing in the controller output. The expert system should decide in which order the algorithms are applied and calculate their parameter settings. The application of one algorithm increases the knowledge about the physical plant and affects the application of further algorithms.

Existing single loop controllers consists of a combination of a control algorithm and a "safety jacket" of logical conditions. Safety jackets are often logical networks which dominate program code. They can be difficult to modify and often make the code less readable. The goal is to implement as much as possible of this as rules in the expert system. This gives a clean separation between the numerical algorithms and the branching logic that simplifies development and maintenance. It is often desirable to combine different algorithms. Some examples are different identification algorithms in self-tuning controllers, one controller for steady state operation and one robust controller for startup etc. The combination of algorithms imposes additional re-

quirements on the correctness of the logics and enforces the need for a structured implementation.

High quality control requires good process knowledge. Adaptive controllers extract some of this knowledge from the process but they still need much *a priori* information such as system order, number of time delays etc., to perform well, (Isermann, 1982, Åström, 1983, Clark, 1984). One idea behind the expert system approach is to include as much process knowledge as possible in the controller. This knowledge can be collected in two ways: by asking the process operator or by performing experiments on the process. An experienced process operator has knowledge about the physical process. This knowledge is often qualitative. Examples may be estimates of dominant time constants and static gain, the nature of non-linearities etc. Simple identification experiments exist, (Åström and Hägglund, 1984) that can be used to extract knowledge from the process. The information obtained in these ways are diverse and sometimes uncertain or contradictory. The expert system approach gives a possibility to exploit and refine this knowledge.

In most of the examples given in the previous section, the expert system was used to incorporate the knowledge of the process operator. In this paper the emphasis is more on the expert knowledge of the control engineer. A controller of this kind has two possible uses. As an actual industrial controller or, perhaps more interestingly, as a testbench for rapid prototyping of new control structures.

From an AI point of view the on-line control application includes both planning and monitoring. The system should plan how the numerical algorithms should be applied to the process and monitor both the execution of the plan and the actual control. The expert system and the algorithms must be implemented as parallel processes having different priorities. The reason for this is that the different processes operate in different timescales. The response time of the algorithms must match the physical process while the rule interpretation in an expert system is a comparatively slow process.

An environment for experiments with expert control has been developed on a VAX 11/780 running VMS. This is described in more detail in Årzén (1986a). It consists of three parts: the expert system, the numerical algorithms and the man-machine interface. These parts are implemented as subprocesses that communicate by sending messages through mailboxes. The process structure is described in Fig 1. The numerical algorithms are implemented in Pascal and could be viewed as a library of algorithms. This process is connected to A/D and D/A converters. The algorithms works as filters that extract symbolic, qualitative information from the numerical signal flow. The expert system is not involved unless something significant has been detected by the algorithms. The expert system and the man-machine interface are implemented in Lisp. A simulation program, Simnon (Elmqvist, 1975), has been interfaced to the system as an alternative to controlling a physical process. Simnon can simulate

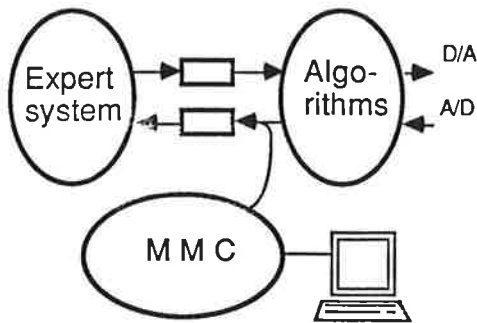


Fig. 1: Process structure. The ellipses represent processes and the rectangles represent mailboxes.

nonlinear differential or difference equations. The program has been modified so that it operates in real time.

This environment has been used in a prototype system where the expert system was implemented with the conventional expert system shell OPS4 (Forgy, 1979). The prototype system is described in more detail in Årzén (1986b). OPS4 is a pure forward chaining system. The database consists of arbitrarily nested list expressions. The condition parts of the rules are patterns that are matched against the contents of the database. There is no possibility of grouping rules according to context. This system was interfaced to the real-time environment by a rule that read new messages from the mailbox and entered them into the database. This rule was executed when no other rules were matched. In this way the rule execution was restarted.

The prototype system was used for experiments with a "smart" PID controller with auto-tuning and gain scheduling. The tuning was based on the Ziegler-Nichols auto-tuner (Åström, 1983). Relay oscillations are used to determine the PID parameters. When the loop is closed with the relay the controlled signals starts to oscillate. This oscillation corresponds to the point where the Nyquist curve of the process crosses the negative real axis. This point gives the ultimate period and the ultimate gain which are then used to compute the PID parameters. The algorithms needed to implement the system were a PID algorithm, a relay algorithm, an oscillation analyzer and a noise estimator. The controller worked in three different modes: manual, tuning and PID. When in PID mode a table was used to schedule the PID parameters for different operating conditions. The operator could change modes and enter new control parameters. It was possible to change the contents of the database and edit the rules on-line. The rules in the system could be divided in the following groups: noise-estimation rules, relay oscillation rules, parameter computation rules, PID supervision rules and command decoding rules. The system contained about 70 rules.

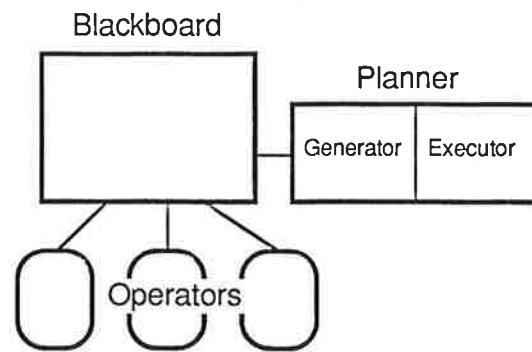


Fig. 2: An expert control architecture.

4. AN EXPERT CONTROL ARCHITECTURE

The experiments performed with the prototype system have been promising. The expert system approach gave a clean implementation that clearly showed the benefits of separating the logic from the algorithms. As an example, the addition of gain-scheduling required only the addition of about 10 new rules to the system. This approach is thus very promising as a testbench for rapid prototyping.

The experiments also showed that a conventional expert system shell is poorly suited for real-time operation. Expert control contains a large element of planning that is not well supported by conventional expert system shells. An example is the tuning phase of the controller that contains a large sequential element. First a noise estimator algorithm is used to collect noise information which is then used to set the relay parameters. A detector is applied to determine that a steady state oscillation is obtained. The PID parameters are determined from the oscillation wave form and PID control is initiated. It is natural to group the rules according to the different stages in this plan. Production systems are generally weak at sequencing problems. The sequencing has to be explicitly expressed in the rules. This often gives the effect that the actual domain knowledge is obscured by the control knowledge i.e. when to apply the rules.

Another disadvantage with the prototype system was the uniform inference strategy. The problem is basically of the data-driven, forward chaining type with data in the form of significant events being sent from the algorithms to the expert system. The monitoring phase can, however, be stated as a diagnosis problem where backward chaining is more appropriate. The same is valid in the phase where the system tries to extract process knowledge from the operator. The lack of structure of the database was a third drawback. A database that allows objects with attributes would be preferable to the nested list structures of OPS4.

A better expert control architecture might be built around a blackboard architecture (Erman *et al.*, 1981) and a planning module, see Fig.2. The blackboard corresponds to a global database that is available to the different operators. The blackboard should allow information to be represented as objects with associated attributes. The

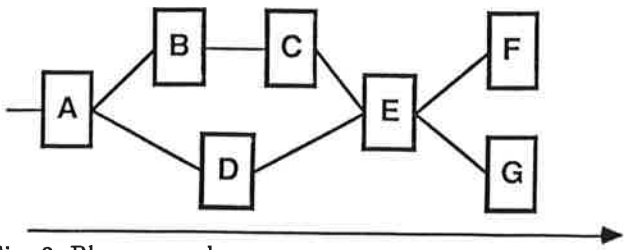


Fig. 3: Plan example.

operators contain the domain knowledge for a certain task. It should be possible to have different problem-solving strategies for different tasks. The operators could be procedural or rule based with different inference strategies depending on their task. This means that the operators could be tailored to their task. They should also be allowed to have their own local databases. The computation of PID parameters from oscillation measurements is an example of an operator task. An operator is often associated with one or more algorithms. An example might be the operator for the relay experiment that contains the domain knowledge for the relay algorithm and the oscillation analyzer algorithm.

The order in which the operators should be used is determined by the planning module. To reach a certain goal state e.g. safe steady-state control, requires in general that a sequence or plan of different operators is used. This plan often contains parallel parts. An example of this is the last part of the plan that contains one operator that takes care of the steady-state control algorithm and one or more operators that handle the monitoring algorithms. An example of a plan is shown in Fig. 3. A plan could be generated in three different ways;

- **Operator-based activation:** The operators start and stop other operators themselves. No separate plan generator is needed in this case.
- **Stored plans:** A finite number of plans for different initial and final goals are stored in the database.
- **Dynamic plan generation:** The plans are dynamically generated by a plan generator. Each operator has an associated set of preconditions that must be fulfilled for the operator to be applicable and a set of goals that will be met by the operator. A plan is generated by comparing the final goal and the initial state with the goals and the preconditions of the operators. This approach can be combined with operator-based activation.

The actual plan is executed by the plan executor. After each stage in the plan the outcome is compared with the expected outcome. Replanning is performed in case of inconsistencies. The dynamic and stochastic nature of control makes this close interaction necessary. The uncertainty in the outcomes of an operator application violates the assumptions of existing domain-independent planning systems (Cohen and Feigenbaum, 1982, Wilkins, 1983). The

goal for the separation between the operators and the planning module is to separate the domain knowledge about different tasks from the control information.

AGE The expert control architecture described can in many respects be compared with an ordinary real-time operating system, (e.g. Brinch Hansen, 1973). The operators are the equivalents of concurrent processes and the plan executor is the equivalent of a scheduler. This is especially true in the parallel phases of a plan. In an operating system the processes can wait for a certain time or for a certain event. Similar features can be provided in this architecture by assigning a state with the values running, ready or waiting to each operator. When an operator calls the function "waittime" in a rule it will be marked waiting by the plan executor. The operator is activated when the time is over. An operator can also wait for a specified element to be inserted in the database, e.g. a certain message from the algorithm part.

The operators could be implemented in two ways. The first is to implement them as concurrent Lisp processes that share a global database. This would probably require a Lisp machine. The second way is to implement the system in a single Lisp process. This has the drawback that the operators can not be interrupted.

5. IMPLEMENTATION

The implementation of a system along the lines of Section 4 will now be described. The system is based on the environment described in Section 3 with a new expert system part. This part is built around the forward chaining production system shell YAPS, (Allen, 1983) and the object-oriented Flavors system (Cannon, 1982).

Object-oriented programming provides a convenient way to implement highly modular systems, see e.g. Stefik and Bobrow (1986). Objects consist of a local state and a behavior. Objects are asked to perform operations by sending appropriate messages to them. The objects have associated methods that handles the messages. Generic algorithms can be implemented using object-oriented programming. A protocol i.e. a set of messages is defined, which specifies the external behavior of the object. This protocol does not define the internal implementation. Objects or instances are created by instantiating their descriptions. There are several different object-oriented add-ons to Lisp such as for example Flavors.

YAPS is a forward chaining shell implemented in Flavors. The database contains arbitrarily nested lists of atoms, integers and flavor objects. The condition part of the rules contains patterns that are matched against the contents of the database. Pattern matching variables are allowed. The condition part may also contain predicates acting on the database that must evaluate to true for the rule to be applicable. The action part of the rules contains ordinary Lisp functions. The inference strategy is event-driven forward chaining. The key feature of YAPS for our purposes is the possibility to use flavors objects in the

database and the fact that YAPS is itself a flavor object. This means that YAPS systems can be used as parts of the database of other YAPS systems. These "internal" systems can be controlled from the rules. In this way expert systems can be used "inside" other expert systems. The internal expert systems could also be of other types e.g. backward chaining systems, as long as they are implemented as flavor objects. Operators are implemented in this way.

The flavor objects that YAPS allows in the database can only be matched as object units. It is not possible to access the attributes of an object in the pattern matching. To allow for objects that can be accessed from the rules according to attributes, YAPS has been extended with static objects. A description of each object type has to be given. This includes the attributes with possible default values and an inheritance order. YAPS has also been extended with a limited explanation facility. A description of how the fact was derived is connected to each fact in the database and to each attribute of the facts that are objects. This gives the possibility to ask HOW questions.

The global database and the planning module are implemented in a YAPS system. The planning module is implemented as rules. The operators are implemented as objects in the database. Only YAPS forward chaining operators are presently allowed. The most important attributes of the operator objects are the following;

- **Status:** Take the values active or inactive. The value is active if the operator is used in the current phase of the plan.
- **State:** Takes the values running, ready or waiting. It is used by the plan executor when the operators is active.
- **Instance:** The flavor instance for this operator.
- **Goal:** The goals which the operator should achieve.
- **Preconditions:** The preconditions that are required for the operator to be applied.

The goal and the preconditions are collections of patterns in disjunctive normal form. The possible uncertainty of an operator shows up in the goal expression. Other key elements of the planning module are the actual plan and the goal stack. The actual plan contains the current plan and the goal stack is a stack of goal entries. A goal entry is a conjunction of patterns to be fulfilled. A goal entry may also consist of a collection of conjunction patterns with associated priorities. The plan generator takes the top goal element and attempts to generate a plan for it. If no plan is found then a new attempt is made with a goal of lower priority. This mechanism could be used to implement backup control. When a plan has been generated its execution starts. There is no guarantee that the plan is unique nor that it is the shortest possible plan. After each stage in the plan execution the goals achieved are compared to the planned goals. When a goal entry has been satisfied the next element of the goal stack is selected. A rule in the plan executor could look as follows.

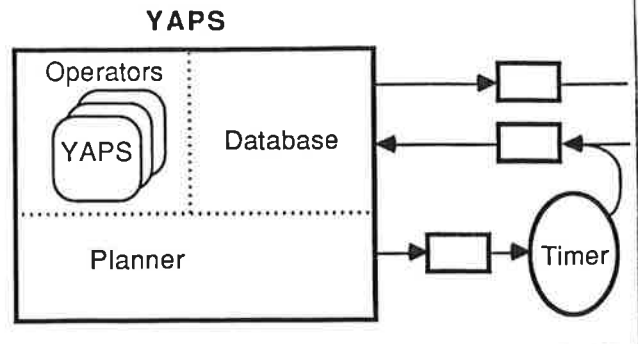


Fig. 4: Implementation structure.

```
(rule schedule1
  (object operator
    status active
    state ready
    instance -x)
  (not (object operator
    state running))
  -->
  (modify 1 state running)
  (<- -x 'run'))
```

In natural language this rule says: if there is an operator which is active and ready and no operator is running then this operator is changed to running and a run message is sent to the actual flavor instance.

The plans are represented as nested list structures of operator names. They can be expressed in the following EBNF syntax where p and s denote parallel and sequential, respectively.

```
PLAN ::= (OPERATOR ARGUMENT [ARGUMENT..])
OPERATOR ::= <p> | <s>
ARGUMENT ::= <operator-name> | PLAN
```

The example in Fig. 3 looks like (s A (p (s B C) D) E (p F G)) in this notation. The rest of the elements in the global database are used for domain information. The knowledge about the actual control loop is well suited to be represented as objects with different attributes.

The operators built on YAPS have predefined functions for activation and deactivation of other operators. They have also functions for waiting a certain time or for a certain element to be inserted in the global database. Each waiting function has two different versions. One that requests a wakeup and suspends the rule execution and one that only requests a wakeup. Furthermore, there are functions for adding and deleting facts locally as well as globally and for pushing and popping elements on the goal stack.

The waittime requests are handled by a separate timer process. When a waittime is requested a message is sent to an associated mailbox. The requests are queued by the process and a message is returned to the expert system when the requested time is over. The structure of the implementation is shown in Fig. 4.

The next step in the implementation will be to add other operators than YAPS. Work on this is currently under progress and will be described in further papers.

6. CONCLUSIONS

The expert system approach simplifies the implementation of controllers based on process knowledge. The knowledge required can be acquired from the process operator or from the process. Extracting knowledge from the process requires experiments. This means that different algorithms are applied to the plant. An expert system is well suited for implementation of the logic needed in this process.

A prototype environment has been built up. Experiment with conventional expert system shells have shown that they are not well suited for expert control. An architecture that is better suited is described. This architecture is under implementation using YAPS and object-oriented programming.

ACKNOWLEDGEMENT

I would like to thank my supervisor Prof. Karl Johan Åström for many useful discussions. The project is part of the Computer-Aided Control Engineering Project at the Department of Automatic Control, Lund Institute of Technology. It is supported by STU, the National Swedish Board for Technical Development under contract 85-3084.

REFERENCES

- Allen, E.M. (1983): YAPS: Yet another production system. TR-1146, Department of Comp. Sc., Univ. of Maryland.
- Allen, J.F. (1984): Towards a General Theory of Action and Time. *AI Journal* 23 pp. 123-154.
- Årzén, K-E. (1986a): A Real-Time Environment for Expert Control. Report TFRT-7314, Dep. of Automatic Control, Lund Institute of Technology, Sweden.
- Årzén, K-E. (1986b): Expert Systems for Process Control. In Proc. 1st Int. Conf. Appl. of AI in Engng. Pract., Southampton, U.K.
- Åström, K.J. (1983): Theory and applications of adaptive control. *Automatica* 19, pp. 471-486.
- Åström, K.J. and J.J. Anton (1984): Expert Control, Proc. 9th IFAC World Congress, Budapest, Hungary.
- Åström, K.J. and T. Hägglund (1984): Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica* 20, pp. 645-651.
- Åström, K.J. and B. Wittenmark (1973): On self-tuning regulators. *Automatica* 9, pp. 185-199.
- Birdwell, J.D., J.R.B. Cockett, R. Heller, R.W. Rochelle, A.J. Laub, M. Athans and L. Hatfield (1985): Expert systems techniques in a computer based control system analysis and design environment. Proc. 3rd IFAC/IFIP Int. Symp. CADCE '85, Lyngby, Copenhagen, Denmark.
- Brinch Hansen, P. (1973): *Operating System Principles*. Prentice Hall, Englewood Cliffs, N.J.
- Cannon, H.I. (1982): Flavors: A non-hierarchical approach to object-oriented programming, unpublished paper, Artificial Intelligence Laboratory, MIT, Cambridge, MA.
- Clark, D.W. (1984): Implementation of adaptive controllers. In Harris and Biddings. (Eds), *Self-tuning and Adaptive Control*. Peter Peregrinus, U.K.
- Cohen, P.R. and E. Feigenbaum (1982): STRIPS and AB-STRIPS. Chapter 15B of *The Handbook of Artificial Intelligence* Vol. 3, William Kaufman Publishing, Los Altos, Ca.
- Crossman, E.R.F.W. and J.E. Cooke (1962): Manual Control of Slow-Response Systems, in E. Edwards and F.P. Lees, *The human operator in process control*, London: Taylor and Francis, Ltd., 1974.
- Elmqvist, H. (1975): SIMNON, An Interactive Simulation Program For Nonlinear Systems, Report TFRT-3091, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.
- Erman, L.D., P.E. London and S.F. Fickas (1981): The design and an example use of HEARSAY-III. Proc. IJCAI 7 pp. 409-415.
- Forgy, C.L. (1979): OPS4 User's Manual. Tech. Rep. CMU-CS-79-132, Department of Comp. Sc., Carnegie-Mellon Univ., USA.
- Hayes-Roth, F., D. Waterman and D. Lenat (1983): *Building expert systems*. Addison-Wesley, Reading, MA.
- Isermann, R. (1982): Parameter adaptive control algorithms - A tutorial. *Automatica* 18, pp. 513-528.
- James, J.R., J.H. Taylor and D.K. Frederick (1985): An expert system architecture for coping with complexity in computer-aided control engineering. Proc. 3rd IFAC/IFIP Int. Symp. CADCE, Lyngby, Copenhagen, Denmark.
- Larsson J.E. and K.J. Åström (1985): An Expert System Interface for IDPAC, Proc. of 2nd IEEE Control Systems Society Symposium on CACSD, Santa Barbara, CA.
- McDermott, D. (1982): A temporal logic for reasoning about processes and plans. *Cognitive Science* 6 pp.101-157.
- McDermott, D. and J. Doyle (1980): Non-monotonic logic I *AI Journal* 13.
- Moore, R.L., L.B. Hawkinson, C.G. Knickerbocker and L.M. Churchman (1984): A Real-Time Expert System for Process Control, Proc. First Conf. on AI Applications, pp. 529-576, IEEE Computer Society, Denver, Colorado.
- Moore, R.L., L.B. Hawkinson, M.E. Levin and C.G. Knickerbocker (1985): Expert Control. In Proc. ACC. pp. 885-887, Boston, MA.
- Nelson, W.R. (1982): REACTOR: An expert system for diagnosis and treatment of nuclear reactor accidents, in Proc. of the National Conference on Artif. Intell., pp. 296-301, Pittsburgh, PA.
- Palowitch Jr., B.L. and M.A. Kramer (1985): The application of a knowledge-based expert system to chemical plant fault diagnosis. In Proc. ACC pp. 646-651, Boston, MA.
- Sanoff, S. P. and P. E. Wellstead (1985): Expert Identification and Control. Proc. IFAC Identification and System Parameter Estimation pp. 1273 - 1278, York, UK.
- Stefik, M. and D.G. Bobrow (1986): Object-oriented Programming: Themes and Variations. *AI Magazine* Vol. 6, No. 4.
- Trankle, T.L. and L.Z. Markosian (1985): An expert system for control system design. Proc. IEE Int. Conf. Control 85, Cambridge, UK.
- Waterman, D.A. (1986): *A Guide to Expert Systems*, Addison-Wesley.
- Wilkins, D. (1983): Domain-Independent Planning: Representation and Plan Generation. Tech. Note 266R, SRI International, Menlo Park, CA.

NEW PROJECTS 86/87

Representation and visualization of systems and their behaviour	659 kSEK
Expert system interfaces	175 kSEK
Implementation languages	161 kSEK
Numerical solution of D/A systems	155 kSEK
	<hr/>
Total	1 150 kSEK

Approved projects 86/87

High level problem solving languages	106 kSEK
Formula manipulation and numerics	53 kSEK
Expert control	295 kSEK
	<hr/>
Total	454 kSEK

D I S P O S I T I O N S P L A N (Belopp i kkr)

	Budgetår 84/85	Budgetår 85/86	Budgetår 86/87	Budgetår 87/88	Budgetår 88/89	Summa
Ursprungligt beviljade medel Därutöver plan.tilldelning	1 800	2 000	1 600	1 550	500	5 400 2 050
Ny disposition av beviljade medel	843	2 000	1 600	957		5 400
Ny disposition av plan. tilldelning				650	1 400	2 050
Summa medel	843	2 000	1 600	1 607	1 400	7 450
Kostnader för beslutade projekt	843	1 046	454			2 343
Återbetalning av utrustnings- anslag		546				546
Start av nya projekt våren -86		408				408
Inköp av ny arbetsstation			400	200		600
Nya projekt			746	1 407	1 400	3 553
Summa kostnader	843	2 000	1 600	1 607	1 400	7 450

REPRESENTATION AND VISUALIZATION OF SYSTEMS AND THEIR BEHAVIOUR

The notion of system is very important.

Structural properties are important.

Graphics is useful to describe structure
and behaviour of systems.

STRUCTURE

Decomposition

Hierarchical block diagrams

Information zooming

Overview windows

Other complementary structuring concepts

Model types

Model classes

Model complexity

Perspectives

Different process designs

Class hierarchies, inheritance

Organization of libraries

Categories

How should these relations be visualized?

Can get inspiration from existing systems for object-oriented programming.

THE MATHEMATICAL DESCRIPTION

Equations - Assignment statements

Higher order derivatives - Transfer functions

Connection mechanisms

Difference equations

Typography

Tex

Function diagrams

Error messages

Colors and highlighting

Error comments below expressions

Handling of parameters

Ranges for parameters

Creation procedures

Default values

Automatically updating of children's parameters

Storing and retrieval

Bookkeeping and documentation

VISUALIZATION OF BEHAVIOUR

Visualization of simulation results

Must be flexible

Trend curves etc.

support editing of axes and curves

Simulate instrumentation - Gauges

Animation of the process simulated

Bookkeeping and documentation

History files

Storage of the model

Text files

Compiled versions

Storage of parameters and values

Storage of simulation results

All variables

User defined

Those calculated by the numeric ODE solver
and the model

PROJECT DESCRIPTION

Investigate how the man-machine interface should represent and visualize systems and their properties. Particularly focus on how graphics can be used.

Flexibility is a keyword.

- User needs

- Hardware

- Command styles

- Separate input/output code from calculations

A flexible and interactive environment is needed

- Common Lisp

- Eagles

Communication with users of CACE programs

OUTCOME

- Ideas and experiences

- Useful program modules

- Circulation of knowledge

EXPERIMENT WITH EXPERT SYSTEM INTERFACES

The results obtained seem promising.

We propose a continuation in order to

Implement a system with
file handler, intelligent defaults,
automatic documentation, etc.

Compile identification knowledge
to form a knowledge database.

IMPLEMENTATION LANGUAGES

Initially planned pilot project.

Given low priority when the expert control project was introduced into the CACE programme.

We feel, however, that it would be useful to have a project devoted to implementation languages.

TASKS

I/O handling

Commando decoding

Graphics

Numerics

Symbolic formula manipulation

Database management

Expert system

ASPECTS

Explorative vs Production programming

Portability

Readability

Security

Efficiency

LANGUAGES

Fortran

C

Pascal, Ada

Lisp, Prolog

Smalltalk

EXPERIENCES

Pascal

New forms of man-machine interaction

Lisp

Experiments with expert system interfaces

Expert Control

The system structure package

Fortran, Ada

Other non-CACE projects

PROJECT DESCRIPTION

Gain experience of Prolog, Smalltalk and Ada.

Implement the system structure package in Prolog, Smalltalk, Ada.

Try to interface it with the graphics developed in the MMI project.

Try to interface it with EAGLES, if we get access to EAGLES.

Guest researchers

Klas Sigbo, DNA, LTH

W. Kreutzer, New Zealand

McKenzie, New Zealand

NUMERICAL SOLUTION OF D/A SYSTEMS

Docent Bo Kågström and Anders Barrlund
Institute of Information Processing
University of Umeå, Sweden

Differential/algebraic systems

$$g(t, dx/dt, x, v, p, c) = 0$$

where

t is the time

x and v are vectors of unknown variables

p is a vector of known parameters

c is a vector of known constants

D/A systems arise naturally when modelling
physical and technical systems.

INDEX (NILPOTENCY) OF D/A SYSTEMS

Consider

$$E y'(t) = F y(t) + f(t)$$

where A and B are constant square matrices.

The index of this system is equal to the size of the largest Jordan block corresponding to the infinity-eigenvalue of the pencil $zE - F$.

NUMERICAL METHODS

DASSL (Linda Petzold)

$$G(t, y, y') = 0$$

$$y(t_0) = y_0$$

$$y'(t_0) = y'_0$$

The user should provide a routine for computing the residual vector $\Delta = G(t, y, y')$ where all arguments of G are known.

The initial vector y_0 ought to be consistent.

The user may give the initial vector y'_0 .

Difficulties

Inconsistent initial conditions

Index > 1

Discontinuities

PROJECT DESCRIPTION

FIRST

Make existing software more robust

Provide the user with information

Index (local and global)

Have software for finding the
structure of the infinity-eigenvalue

Inconsistent initial values

Possible discontinuities

Too ill-conditioned system

Reduce the index by symbolic differentiation

Drift problem

LATER

Focus the attention on efficiency and
and how to explore problem structure

Representation and Visualization of Systems and Their Behaviour

Sven Erik Mattsson

1 INTRODUCTION

The notion of system is an essential element of control theory. The representation and visualization of systems and their behaviour are a key issue in computer aided control engineering, CACE. The structural properties of a system are very important particularly when working with large systems. These structures are, however, difficult to represent in an easily apprehendable way when a purely textual description is used. The interconnection structure of subsystems is for example much easier to describe graphically. The new workstations with high performance, real-time graphics now appearing on the market offer new possibilities for man-machine interaction. Some of these possibilities were explored in the STU project 84-5069, "New forms of man-machine interaction". In that project a prototype simulator was set up, where hierarchical block diagrams are used to describe the decomposition of the model and the interconnection structure. The prototype simulator uses continuous zooming to show internal detail. There is, however, also a need for other supplementary structuring and visualization concepts. The idea of this project is to investigate how systems could be represented and visualized to obtain a good man-machine interface. Representation and visualization of systems and their behaviour are considered in Section 2. Section 3 gives a project description.

2 REPRESENTATION AND VISUALIZATION

In this section we will indicate interesting issues of representation and visualization of systems and their behaviour to get a good man-machine interface. First, representation and visualization of structural properties are considered. Second, the mathematical description is discussed. Third, display of results from analysis and simulation is treated.

2.1 Structure

Modularization is a very important concept. A CACE system supporting modularization of system descriptions gives many advantages. It simplifies the modelling, since the user can focus on a smaller part of the system at a time. It also makes the model more flexible and easier to adapt and manage. Technical systems are often built in a modular way composed of standard components. Their behaviour may be well-known. Even good, generally accepted models may already exist. We can build and use libraries of models. Modularization also makes the model easier to adapt for different simulations. Two conceptually different needs of adaptability can be identified: adaptability with respect to different plant designs, and adaptability with respect to model complexity. During the design of a system, the model has to be updated as the design proceeds. Questions of the type "What happens if we modify the design in this way?" arise frequently in simulation projects. It is impossible to make a model which can simulate all aspects of a given plant. Models of different complexity must be used for simulation of different events. Modularization facilitates testing. It is difficult to verify that a simulation program implements the intended mathematical model. With a modular approach the user can split the problem into smaller parts. He can start with simple models and use them as references when testing the more complex ones.

Hierarchical block diagrams

A block diagram is an excellent way of describing a decomposition, provided it does not contain too much detail. The use of hierarchical block diagrams and the possibility to create different overview windows solve this problem. Sitting at the terminal, the user can try different decompositions of his system. By using the mouse he can create blocks and interfaces and position them and draw

connections until he is satisfied.

An interesting issue is what to do with connections when the user moves a block. It is difficult to make a nice layout by moving the interfaces and connections automatically when a block is moved by the user. However, it is rather easy to let the connections follow if we accept that the connections go criss cross over the blocks. This means that the connections are preserved, and that the user can move the interfaces and edit the connections afterwards to get a nice layout. It is important that an user can change the position of an interface easily, since this will be a common operation when models from a library are used. It may be convenient to be able to move a block over another block when modifying the layout.

If the decomposition and the layout are made properly, most connections are short, but there may be long ones, and when drawing them it would be convenient to be able to zoom and pan.

Use of symbols instead of simple rectangles to denote different parts would probably make the block diagrams more illustrative and easier to grasp. However, this extension implies a more complex CACE system. The user must be able to define (draw) new symbols. How should the interfaces look like? Should the symbols have a rectangular window on their bellies to show their internal information?

Model types

The modular approach allows the user to build and use libraries of models. The prototype simulator handles this by saving and reading files with model descriptions. The models can then be copied and positioned as desired. However, the prototype simulator has no concept for model types. When a model is copied it becomes a new model independent of its father. The lack of model types makes maintenance of the global model more difficult, because it then have to be updated in many places. A CACE system should undoubtedly support the use of model types to facilitate the maintenance of model libraries and models themselves. Many of the aspects of model representation can conveniently be expressed in the framework of object-oriented programming. Some preliminary studies have been made by Åström and Kreutzer (1986). This discipline may give us ideas and

inspiration. A good overview of ideas and concepts of object-oriented programming is given by Stefik and Bobrow (1986). In that terminology model types will be considered to be classes.

Model complexity

There is also a need for other structuring concepts than the hierarchical modeling concept. When developing a new model it is convenient to start with a simple model and to test it, and then extend it stepwise by adding new features, while retaining the old versions for comparison. Furthermore, it is impossible to make a model which can simulate all aspects of a given plant. Models of different complexity must be used for simulation of different events. If all these models have the same interfaces, it is convenient for the user to view them as different versions of the same model. He can then easily adapt the model by selecting an appropriate version for example by using the mouse and an automatically generated menu. One can also visualize a system which can select the particular version automatically. In object-oriented programming the perspective concept can be used to handle the need to have different versions of model. Perspectives are a form of composite object interpreted as views on the same conceptual entity.

Different process designs

We also need structuring concepts so that the model is flexible with respect to different process designs. For example, if we simulate a wind power plant it should be simple to first simulate the system when it has a synchronous generator and then when it has an asynchronous generator. This can be achieved if we define a class generator with properly defined interfaces, and if the simulator offers a facility to exchange the content of the generator easily. There are similarities with the desire to have flexibility with respect to model complexity, so the use of perspectives would be one possibility. However, conceptually the situations are quite different. Maybe such concepts as class hierarchies, inheritance and specialization are more appropriate concepts. This means that we should consider the model classes for synchronous and asynchronous generators to be subclasses of the class of generator models. We can then go on and specialize a synchronous generator further and let it have subclasses to model various designs.

A basic issue is how the CACE system should illustrate the relations between

model classes. Some systems for object-oriented programming draw inheritance lattices. If the user modifies a model, he must be aware of the effects and the CACE system must be able to give support when necessary.

Organization of libraries

An approach without model types has some advantages. The user does not have to consider such abstract concepts as model types when he starts developing a new model. His main interest is then to get the model running. Afterwards he may find that the model may be of general use and that he wants to include it in a library. We think that a simulator should support this way of developing models. The CACE system can at the first stage of model development take care of the definition of a model type and consider the user's intentions to be both a definition of a model type and a model instance. It is probably simple to split the user's model definition into one model type and one model instance.

To avoid a large name space and name conflicts, encapsulation concepts for model types should be introduced, so we can pack and order them in a library in a neat way. The user should not be forced to consider these problems in early stages. The simulator can use the name of the model instance as the name of the model type and put it in an unnamed package. If name conflicts arise, the simulator can solve them automatically in some way. The user should of course be able to rename the model type later if he so prefers. The simulator should have mechanisms for structuring model libraries. It should be possible to reference a library and automatically have a menu of its models.

It should also be possible to supply a model with user information so that the user can go into query mode and ask for information about assumptions made, meaning of different variables and parameters etc. This is particularly important for models in a public model library.

2.2 The Mathematical Description

The prototype simulator accepts model for dynamical systems, which can be described by sets of ordinary differential equations and algebraic equations. The simulator allows the mathematical description of submodels at the lowest level to be in the form of equations instead of assignment statements. This facilitates the

use of the simulator since it corresponds to the original formulations of the models as a basic set of mass- and energy-balances and other equations. The equation form also has other advantages compared to the assignment form. The documentation is better since the reader will recognize the equations. It is easier to check that the model is input correctly and it reduces the risk of introducing errors during manual transformation to assignment statements. Furthermore, the equation form is the only reasonable if one wants to build model libraries, because different environments of a model impose different causality relations so the transformation to assignment form of a model is dependent on its environment. A more sophisticated connection mechanism can be introduced when equations are allowed. The prototype simulator supports two types of interface variables: across variables which are equal in the cuts (examples are voltage, pressure and temperature) and through variables which have a direction and are summed to zero in the cut (examples are current, flow, thrust and torque).

The support of mathematical descriptions on equation form make a simulator more complex. Symbolic formula manipulation or implicit numeric ODE solvers is needed. So it is in many applications relevant to consider descriptions where the interaction between subsystems is unidirectional. For example a block diagram editor for Simnon would be a very useful tool.

The common ways of describing mathematical expressions for a computer are typographically poor. A long term aim could be that greek and script letters as well as common mathematical symbols should be allowed and supported so that the CACE system could display the equations just as they stands in a good textbook. Use of the TeX formalism (Knuth, 1984) for mathematical expressions may be one way to go.

Equations and expressions can also be represented graphically with function diagrams. For special types of systems there are alternative mathematical descriptions like transfer functions.

Error messages

The syntax of equations and expressions is simple to check, as well as type compatibility. The graphics gives excellent possibilities to issue nice error messages. Colors can be used to point out and to highlight erroneous parts of an

expression. The error message can be put below the expression as a special comment, so we can read it in the editor when we try to correct the error. The user can if he likes delete the error message when he is in the editor or he can let the simulator remove it at the next check. To make the errors more easy to find, we can highlight blocks that are erroneous or have erroneous children.

Handling of parameters

The handling of parameter values is an important issue to consider in order to make the simulator useful for realistic applications.

When creating a new model instance, many new parameters are defined and must be given values. The model types may for example have default values for their parameters, or they may be supplied with a more sophisticated mechanism which forces the user to supply parameter values to an creation procedure written by the model designer. The creation procedure should manage the creation of the models on lower levels properly.

It should be possible to set parameters both from the terminal using mouse and menus (tables) and from an user defined macro. It should be possible to store and retrieve parameter values from files.

The handling of parameters raises important semantic issues. For example which parameters should the user be allowed to modify? If a model not being on the lowest level is allowed to have parameters and creation procedure, it may completely define the parameter values of the submodels. Should then an user be allowed to modify the parameter values of the submodels freely, or should the creation procedure be able to prohibit this? Such a mechanism must imply that the parameter values of the submodels are modified automatically when the parameter values of their parent are modified. It may facilitate the use of complex library models.

2.3 Visualization of Behaviour

The possibilities to present results of analysis and simulation must be flexible. The user should be able to present results in windows in the form of trend curves etc. He should be able not only to choose the scale of the axes but also

for example how it should be marked. To be able to read the results more closely, he should be able to move a hair cross along the curve having the numbers displayed. Another possibility to visualize results is to simulate an instrument and to present the results in windows as if you were viewing them on real devices. A natural continuation of this is to present the results as some kind of animation of the process simulated. The IRIS for example is capable of real-time animation of a robot in shaded 3-D graphics. The user can move around in this 3-D world and study how the robot performs its work cycles. The mouse can be used as an input device to control the movement of the user in this animated world. It must be pointed out that the design of an animation must be done carefully so it does not become more spectacular than useful.

Bookkeeping and documentation

Some kind of automatic documentation is desirable. Even simple history files listing all actions and commands may be useful. History files can be combined with some sort of macro facility. It should also be able to save descriptions of the model on files which can be read later to recreate the model. These files can be text files, which can be read, printed or in other ways processed outside the CACE system. It may be of interest to save a compiled version since it may be time consuming to generate a new one.

When using a simulator a basic issue is what signals that should be stored. One possibility is to let the user make his choice. However, he may then find that he forgot to store an interesting variable. The other extreme is to store every variable, which may lead to very large files and storage problems. There are intermediate approaches to the problem. For example it is possible to store just those variables solved by the ODE solver. The simulator can then calculate other variables without assistance of the ODE solver.

3 PROJECT DESCRIPTION

The idea of the project proposed is to investigate how the man-machine interface should represent and visualize systems and their properties. We will particularly focus the intention on how graphics can be used. Interesting issues are outlined in Section 2.

When designing man-machine interaction, flexibility is a keyword for several reasons. Both the ordinary users and the designers of a CACE system need flexibility. However, they may view it from different points. Let us start from the users' standpoint. We may then view the desire to solve complicated control problems as a mother of requirements on a CACE system, since it implies the need of good algorithms, good numerics, good man-machine interface etc. This implies the need of various kinds of flexibility. First, the user must be able to describe the problem in a for him natural language. Second, different users require different kinds of help and support to solve their problem. Third, the hardware for user interface offers several alternatives like keyboard, color screens, mouse, joystick, light pen and others using voice input and sound are to come. A command can be entered in different styles like question/answer dialogue or command dialogue. The commands can be typed from a keyboard or pointed out with a mouse on various kinds of forms or menus.

We in the CACE project need a very flexible and interactive environment for several reasons. First, the CACE project is a research project and we should be able to investigate the different possibilities offered by the hardware and software technology in a fast and convenient way. Second, man-machine interaction is a very important aspect of a CACE system. When evaluating various designs and ideas it is necessary to let different users give their reactions. Thus we must be able to do fast prototyping. The paper by Schneiderman (1983) gives an overview of relevant issues. Third, we have to support various computers and I/O devices.

To retain flexibility with respect to different ways of input/output, it is important to separate input/output code from the code performing the operations. Furthermore, for documentation and to make macros there must be some form of textual description of the user's input sequence, whether he uses the keyboard or the mouse to input his commands. It might be useful to view the command sequence as a sequence of actions. In this context, the use of compiler-compilers must be considered.

Our current proposal to solve our need of a very flexible and interactive environment is to use Common Lisp as the high level implementation language. The experiences from the pilot project "New Forms of Man-Machine Interactions" are that Pascal is too inconvenient for prototyping. It takes too long time to

implement an idea and the turn around time for compilation are too long, while the experiences from the pilot project "Experiments with Expert System Interfaces", "Expert Control" and "Experiments with system structures" indicate that Lisp offers a better environment. There are commercial Common Lisp systems available for both the VAX and the IRIS 2400. We have also requested funding of a Lisp machine from FRN. We will in this project mainly use the IRIS 2400, but when feasible we will use other computers. The costs for a Common Lisp and an upgrade of the CPU memory of our IRIS from 2 Mbyte to 6 Mbyte are included in the application.

Later on another possibility could be to use EAGLES which is an Environment for Advanced Graphical (LLNL) Engineering Systems on workstations as a basis. EAGLES is currently under development at Lawrence Livermore National Laboratory. The β -release is scheduled to this summer and we may get an early release. EAGLES is written in Objective C. It will have user interface and code integration capabilities. It will have facilities for device and system independent design of user interfaces including window management and handling of forms and menus. It will also have tools for integrating new and existing codes at menu level control interface and at subroutine level data interface. The approach is object-oriented. EAGLES will also contain more specific control software. It will support the M language for solving control system problems.

We will in this phase also communicate more closely with users of CACE programs to get their reactions to our ideas. We are now in a much better state to do so since we have the prototype simulator and can illustrate our ideas more clearly.

The outcome of this projects will be ideas and experiences of how to design the user interface with respect to description and visualization of systems and their properties. The project will also give useful program modules that can be incorporated in CACE systems. The communication with users of CACE programs will also circulate knowledge.

REFERENCES

- Åström, K.J. and W. Kreutzer (1986): System Representations. Abstract submitted to the IEEE Control Systems Society 3rd Symposium on Computer-Aided Control System Design (CACSD), September 24-26, 1986, Pentagon City, Arlington, Virginia.
- Årzén K.E. (1986): Use of Expert Systems in Closed Loop Feedback Control. Proc. ACC-86, June 18-20, 1986, Seattle, Washington.
- Elmqvist H. and S.E. Mattsson (1986): A Simulator for Dynamical Systems Using Graphics and Equations for Modelling. Abstract submitted to the IEEE Control Systems Society 3rd Symposium on Computer-Aided Control System Design (CACSD), September 24-26, 1986, Pentagon City, Arlington, Virginia.
- Knuth, D.E. (1984): The TeXbook, Addison-Wesley.
- Larsson J.E. and P. Persson (1986): Knowledge Representation by Scripts in an Expert Interface. Proc. ACC-86, June 18-20, 1986, Seattle, Washington.
- Schneiderman, B. (1983): Direct Manipulation: A Step Beyond Programming Languages. IEEE Computer, August 1983, pp. 57-69.
- Stefik, M. and D.G. Bobrow (1986): Object-Oriented Programming: Themes and Variations. AI Magazine, Vol. 6, No. 4, Winter 1986, pp. 40-62.

Expert System Interface Experiments

Jan Eric Larsson & Per Persson

A continuation of the STU project 85-3042 "Experiments with Expert System Interfaces" is proposed.

The first part of that project was to implement a small prototype expert system interface. The purpose was to get a working prototype quickly. This would help explain what the project was all about, and give a feel for how useful an expert interface could be. Also, the standard technique for building an expert system is to start out with a small example system.

This first prototype contained a command parser, a script matcher and a production system, i.e. a minimal implementation of the "command spy" concept. It uses *scripts*, i.e. command history descriptions, that are matched against the actual command history typed in by the user, to understand what the user is doing and to guide him, tell him what to do next, etc. A more detailed description is given in Larsson and Persson [1986].

The conclusions of the work with the prototype are that it seems possible to develop an expert interface according to our ideas. But the design of such a system is difficult and time-consuming because it is a completely new task. Therefore it takes a lot of time to arrive at a good solution. Considerable effort was also devoted to make the code properly modularized with well defined interfaces.

Matching scripts is a very complex problem. If we want the users to be able to design their own script database, the language which the scripts should recognize must be very general. The ordinary parsing techniques used in compiler-compilers is not enough. Code that handles the complicated type of parsing needed has been developed.

Idpac's command language is relatively easy to parse. The choice of Idpac as the target program was probably very good. A development of the project would be to try and compare Idpac to other languages, for example Matlab, Ctrl-C, Eagles, M and Macsyma.

The need for a file handler and intelligent defaulting quickly became apparent. Also, the script matcher developed was rather inefficient. Very naturally, the next step is to design a new version with a better script matching strategy and including a file handler, intelligent defaults, automatic documentation, etc.

In order to build a well modularized system we will use the object oriented "Flavors" package. We also plan to use "YAPS", an expert system shell developed within Flavors.

The man-machine communication is a crucial part of an expert interface. If possible we would like to implement the actual user-interface in different ways for different types of graphics display. Until now we have

been working with VT100 type terminals, but many interesting possibilities emerge if we get access to high resolution graphics on for example a Lisp machine or a work station. The use of modularizing will allow us to take care of several different interfaces in a uniform way.

Another advantage of modularizing the system is that it will most likely make it easier to use it in the construction of a complete CACE package. This inclusion also means that the project can have a certain impact on the choice of high level languages for communication and the overall design of a CACE program.

A most obvious conclusion is also that as an environment for experimental programming Lisp is excellent.

The design of the new version is well on the way, and several of the critical parts of it have been coded, but the work will not be finished before July 1st, 1986. The results seem promising, and to make the system useful we consider it worthwhile to continue the project.

We therefore propose a continuation of the project until July 1st, 1987. The following tasks will be dealt with.

- An implementation of the system with file handler, intelligent defaults, automatic documentation, etc.
- Acquisition of identification knowledge to form a knowledge database.

References:

- Larsson, Persson, "Knowledge Representation by Scripts in an Expert Interface", Proceedings of the American Control Conference, Seattle, Washington, June 18-20, 1986. To appear.

IMPLEMENTATION LANGUAGES

A pilot project on implementation languages was initially planned. This project was, however, given low priority when the expert control project was introduced into the CACE programme.

We have obtained some insights via the other pilot projects that have been carried out. The project "New forms of man-machine interaction" has given a considerable insight into using Pascal as an implementation language. The projects "Experiments with expert system interfaces", "Expert control" and "High level problem solving languages" have all given experiences with working in a Lisp environment. Additional Lisp experience has been gained from the study of system structures. Other projects carried out by the team members include large Fortran programs. We have also Ada experience from other projects. Our current best guess is that future CACE systems will be implemented by combining large chunks of code and packages written in many different languages. It will also be essential to write the systems so that they can easily be combined with other packages.

We feel, however, that it would be useful to have a project which will be specifically devoted to implementation languages. We have neither experience from Prolog nor from a purely object-oriented language like SmallTalk. In this project we will propose to implement the system structure package from the project "High level problem solving languages" in Prolog and SmallTalk. We will also try to interface it with the graphics developed in the MMI project. It should be noted that the experiences with using SmallTalk and Prolog may give valuable input on how to make man-machine interfaces.

EAGLES is an Environment for Advanced Graphical (LLNL) Engineering Systems on workstations currently being under development at Lawrence Livermore National Laboratory. The β -release is scheduled to this summer. We may get access to an early version. EAGLES is written in Objective C and will have user interface and code integration capabilities. It will have facilities for device and

system independent design of user interfaces including window management and handling of forms and menus. It will also have tools for integrating new and existing codes at menu level control interface and at subroutine level data interface. The approach is object-oriented. EAGLES will also contain more specific control software. It will support the M language for solving control system problems. If we get access to EAGLES, we will also interface that with some version of the system structure package. This would be a good exercise in interfacing of different packages.

Numerical Solution of Differential-Algebraic Systems

In this project we emphasize the numerical solution of the dynamic system derived from the mathematical modeling phase. In general the dynamic system will be modeled by a differential-algebraic (D/A) system

$$(1) \quad g(t, x', x, v, p, c) = 0$$

where t is the time, x and v vectors of unknown variables, p a vector of known (simulation) parameters and c a vector of known constants. The unknown variables are split into the two vectors x and v . The vector v contains the unknown variables which do not appear differentiated in the equations whereas the components of x appear differentiated in the equations.

1. Classification of D/A systems

From the assumptions above the system (1) can be rewritten as

$$(2a) \quad 0 = g_1(t, x', x, v, p, c)$$

$$(2b) \quad 0 = g_2(t, x, v, p, c)$$

where the vector valued functions g_1 and g_2 represent the differential(D) and algebraic(A) parts, respectively, of the system.

If it is possible to rewrite the implicit system (2a) as a system of ordinary differential equations (ODEs) then the D/A

Bo Kågström

1986-03-30

system is posed in semi-explicit form:

$$(3a) \quad x' = f_1(t, x, v, p, c)$$

$$(3b) \quad 0 = g_2(t, x, v, p, c)$$

Sometimes all unknown variables appear differentiated in the equations and the D/A-system (1) is then posed in fully-implicit form (i.e. the subsystem (2.b) does not exist). If the Jacobian matrix dg/dx' is nonsingular then the fully implicit D/A-system is a set of implicit ODEs which theoretically can be reformulated as a system of ODEs. In some cases this can be done by symbolic manipulation, but nonlinearities can make it impossible to give an explicit expression for $f(t, x, v, p, c)$. From a numerical point of view it can still be preferable to handle the system as an implicit ODE system.

However if the Jacobian dg/dx' is singular, the system is a set of D/A equations which may have no solution, one solution, or infinitely many solutions corresponding to a given set of initial values. Similarly the semi-explicit system (3a-b) may have no, one or many solutions for arbitrary initial values.

The behavior of D/A systems is directly related to a property of the system called (nilpotency) index. Uniqueness and existence questions of a solution have only been resolved fully for linear D/A systems with constant coefficients:

$$(4) \quad By'(t) = Ay(t) + f(t)$$

where A and B are constant square matrices. The index m of a system of type (4) is equal to the size of the largest Jordan

block corresponding to the infinity-eigenvalue of the pencil $A-zB$. For all sufficiently smooth $f(t)$ ($m-1$ times differentiable) there exists a unique solution of (4) for each consistent initial vector $y(t_0)=c$. A straight forward generalization to linear variable coefficient systems is not possible. However a local index at time say t_1 is equal to the index of $B(t_1)y'=A(t_1)y+f(t)$. A global index (if its exists) is also defined (see ref 5 and 6). The definition of a local index can be extended to fully implicit and semi explicit systems by linearization. For example for a fully implicit system we have that

- dg/dx' is nonsingular iff $\text{index}=0$
- dg/dx' is singular iff $\text{index}\geq 1$.

2. Numerical methods and software for D/A systems

During the last few years there has been an increasing interest in the numerical solution of D/A-systems (see e.g. ref 2,5,6,8,9 and 10). Linda Petzold has developed a Fortran code DASSL for the numerical solution of fully implicit systems of D/A equations (1) written in the following form:

$$(5a) \quad G(t,y,y') = 0$$

$$(5b) \quad y(t_0) = y_0$$

$$(5c) \quad y'(t_0) = y_0'$$

The user provides a routine for computing the residual vector $G(t,y,y')$ where all arguments of G are known. The initial value $y'(t_0)$ is required in order to get a good starting approximation to the Newton iteration. In many applications we

Bo Kågström

1986-03-30

do not know all information of $y'(t_0)$ and for that reason DASSL has an option for a numerical computation of $y'(t_0)$. However there is no guarantee that the computed approximative initial values are consistent. One major problem in the numerical solution of D/A-systems is to find consistent initial conditions.

As most D/A codes DASSL is based on a k -th order backward differentiation formula (BDF) approximation of the derivative in (5a), and then it solves the resulting nonlinear equation for an approximate solution at the current time using a Newton method.

3. On the robustness of D/A codes

We have numerical experience with DASSL applied to D/A systems of different index (0,1,2 and 3, (see ref 1)). As expected DASSL computes accurate solutions for general index 0 and index 1 problems and for some index 2 problems (linear systems with constant coefficients or mildly varying coefficients). However there is no guarantee that DASSL has computed an approximate solution that is accurate to a prescribed tolerance even if it terminated. When we applied DASSL to a constant coefficient index 3 problem it failed to compute a solution. The theoretical explanation is that DASSL is designed to start integrating with the backward Euler method (BDF-1) and already after one step of integration of an index 3 system the numerical solution contains an $O(1)$ error in the algebraic variables, even if the exact initial values are given. DASSL does not give any indication of the cause of its failure.

3.1 Analyzing the index

We propose to analyze the local index of (5a) at points of time where DASSL fail to compute a satisfactory approximative solution. This corresponds to compute the Jordan structure of the infinity eigenvalue of the pencil $A-zB$, where A and B are partial derivatives dG/dy and dG/dy' , respectively, evaluated in the point of time under consideration. Software in Fortran will soon be available for finding the structure of the infinity-eigenvalue (see ref 3,4 and 7). If the pencil $A-zB$ corresponding to the linearized problem (frozen at time say t_1) is regular this problem has a unique solution for consistent initial values. Otherwise ($A-zB$ is singular) the linearized problem may or may not have a solution, and can even have infinitely many solutions dependent on the Kronecker structure of the underlying pencil (see ref 4). An interesting issue for further consideration is if we can draw any conclusions about the solvability of the nonlinear problem by analyzing the linearized problem.

Another possibility is to make use of the reduction algorithm (see ref 5 and 6) to find out the local index of a linearized system. The reduction algorithm would be most suitable in connection with symbolic manipulation (differentiation). This technique is also sometimes used to reduce the index of a D/A system by differentiation of the algebraic relations. However this frequently leads to a numerical solution in which the algebraic relations are no longer satisfied exactly throughout the course of integration. Thus the computed solution "drifts off" the algebraic constraint.

Bo Kågström

1986-03-30

3.2 Ill-conditioned linear systems and scaling

In order to compute an approximate solution y_n at time t_n DASSL at each time step solves a nonlinear system of equations by a modified version of Newton's method. The iteration matrix

$$(6) \quad JG = dG/dy' + \text{alfa} * dG/dy$$

where alfa is $O(h)$ is computed and factored, and used as many time steps as possible. If dG/dy' is singular this system will be close to singular for small alfa . We have made some preliminary investigations trying to improve the conditioning of the iteration matrix by scaling (see ref 1). In private communications Linda Petzold has proposed that the scaling can be done by someone who understands the problem, or automatically by row equilibration (see also ref 8). We propose to further investigate the matrix conditioning problem and some possible remedies by scaling.

4. Summary

To summarize, in the first part of this project we will concentrate our work on improvements of the robustness of existing D/A-codes, especially DASSL. The aim is to give the user as much information as possible about the underlying D/A system (e.g. solvability of the system, index (local and global) of the system, possible discontinuities in the solution) and the behavior and accuracy of the computed solution. If DASSL does not succeed in computing a solution the user should get as much information as possible about why DASSL broke down (e.g. a high index problem, inconsistent initial

Bo Kågström

1986-03-30

values, tried to solve a too ill-conditioned linear system of equations).

Here symbolic differentiation is of interest when computing exact Jacobians and trying to reduce the index of the D/A system. For example do the linearized systems with exact and approximate Jacobians , respectively, have the same local index.

The insight obtained from a more robust code can then be useful in theoretical analysis and in the development of improved numerical methods for solving D/A systems. In order to handle higher index problems satisfactory this development will probably lead to more specialized methods, especially tailored for a certain class of applications. Having a more robust D/A solver we will in future applications also focus the question of efficiency .

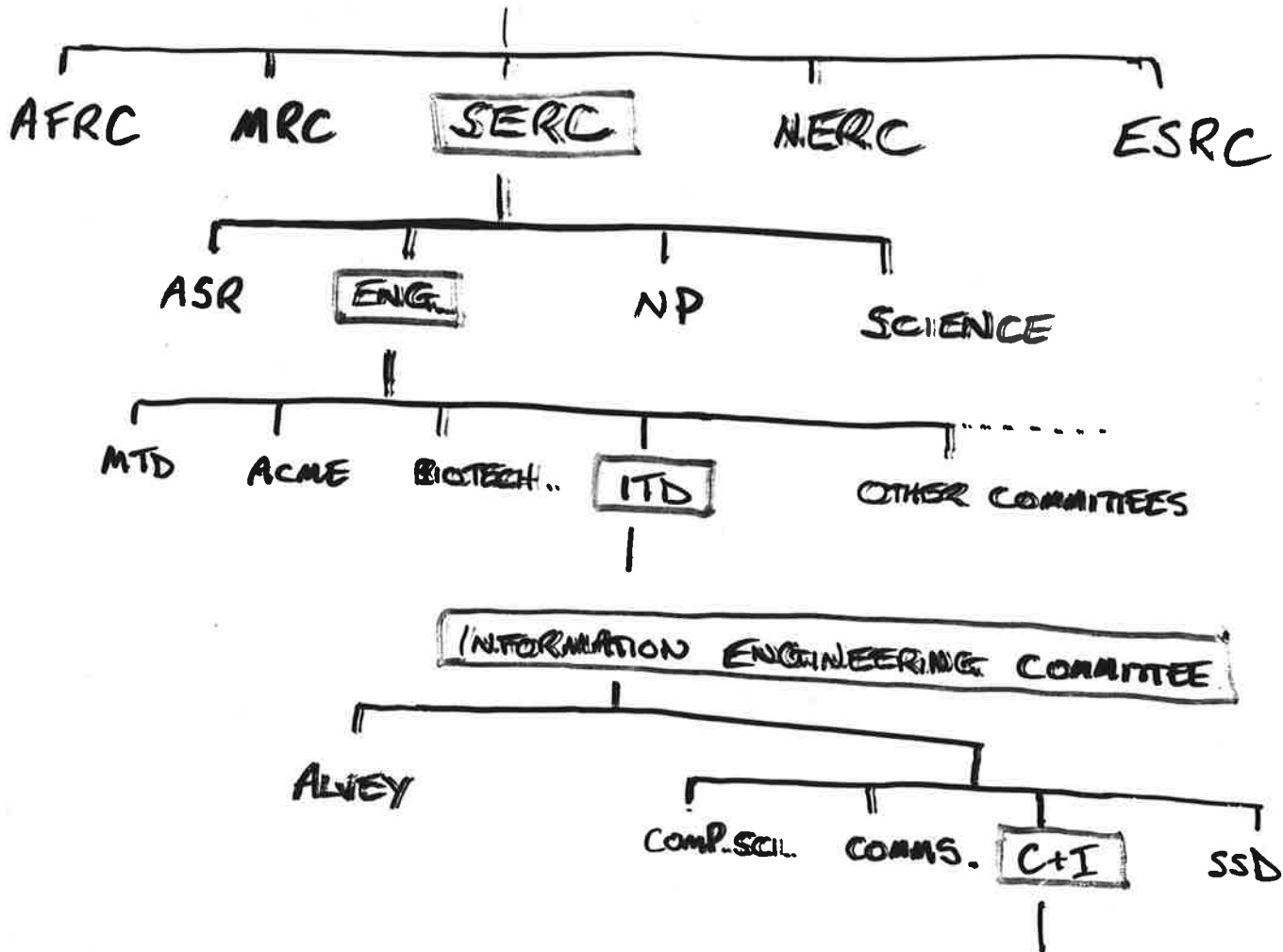
Bo Kågström

1986-03-30

References:

1. Barrlund A., Numerisk behandling av differential-algebraiska system, Slutrapport projektarbete på Matematikerlinjen, UMNAD-16.85, Inst. för Informationsbehandling, Umeå Universitet, 1985.
2. Brenan K.E., Stability and Convergence Approximations for Higher Index Differential-Algebraic Systems with Applications in Trajectory Theory, Ph.D. dissertation, UCLA, 1983.
3. Demmel J., Kågström B., Computing Stable Eigen-Decompositions of Matrix Pencils, submitted to Linear algebra and its applications, Dec 1985.
4. Demmel J., Kågström B., Stably Computing the Kronecker Structure and Reducing Subspaces of Singular Pencils A-zB for Uncertain Data, In Cullum, Wilhoughby(eds) Proceedings of the Conference on Large Eigenvalue Problems, IBM Europe Institute, Oberlech, July 1985, North Holland, 1986.
5. Gear C.W., Petzold L., Differential/algebraic Systems and Matrix Pencils, in Kågström, Ruhe(eds) Proc. Conference, of Matrix Pencils, Piteå, Sweden, 1982, Lecture Notes of Mathematics Vol. 973, 1983, pp 75-89.
6. Gear C.W., Petzold L., ODE Methods for the Solution of Differential/Algebraic Systems, SIAM J. Num. Anal., Vol. 21, No. 4, 1984, pp 716-728.
7. Kågström B., RGSVD - An Algorithm for Computing the Kronecker Structure and Reducing Subspaces of Singular A-zB Pencils, SIAM J. Sci. Stat. Comput., Vol. 7, No. 1, 1986, pp 184-211.
8. Lötstedt P., Petzold L., Numerical Solution of Nonlinear Differential Equations with Algebraic Constraints, Sandia report SAND83-8877, November 1983.
9. Petzold L., A Description of DASSL: A Differential/Algebraic System Solver, Proc. IMACS World Congress, Montreal, Canada, August 1982.
10. Söderlind G., DASP3 - A Program for the Numerical Integration of Partitioned Stiff ODE's and Differential-algebraic Systems, TRITA-NA-8008, Dept. of Numerical Anal. and Computing Science, KTH, Stockholm, 1980.

D.E.S.

SPECIAL PROJECTS :-

i) INSTRUMENTATION AND MEASUREMENT
(NOW ENDED)

ii) COMPUTING AND DESIGN
TECHNIQUES FOR CONTROL
ENGINEERING.
(CDTCE)

↓
5 YEAR PROGRAMME
~ £3.4 M.

C.D.T.C.E

3 MAIN THEMES TO INITIATIVE:

INFRASTRUCTURE

- NEIL MUNRO

DESIGN TOOLS

ADVANCED COMPUTING CONCEPTS.

- MIKE DEINAM

IN UK THERE IS A PARTICULAR
 PROBLEM OF CROSSING THESE RELATED
 ACTIVITIES → INVOLVING INDUSTRY
 → ACADEMIA

PROBLEMS WITH SOFTWARE

- * COMPLEX
- * NOT USER FRIENDLY
- * LARGE AND COSTLY
- * NOT PORTABLE
- * NO SUPPORT
- * NO MAINTENANCE
- * NO DOCUMENTATION
- * NO CONSULTATION

CONCERNS

- * 1 CONTROL ENGINEERING RESEARCH
OUTSTRIPPING APPLICATIONS AND IN LIMITED
AREAS

- * 2 INDEPENDENT DEVELOPMENTS LEADING TO
DUPLICATION

- * 3 INDUSTRIAL RELEVANCE?

OBJECTIVES

of

* 1 PROVISION OR STIMULATE
THE DEVELOPMENT OF
DESIGN SOFTWARE

* 2 DEVELOP A COMPUTER INFRASTRUCTURE TO
ENABLE TECHNOLOGY TRANSFER

(a) BETWEEN UNIVERSITIES

(b) TO INDUSTRY

(c) FROM INDUSTRY

} "FREE ISSUE"
OF INFRASTRUCTURE

ACTIONS

0.

~~DECISION~~

!

DEVELOPMENT INFRASTRUCTUREENVIRONMENT FORCONTRACT DEVELOPMENTS
TO MINIMISE IPR
PROBLEMSCONTROLSYSTEMTHEORYAPPLICATIONSSYNTHESIS

- * (a) STUDY BY UMIST LEADING TO A SPECIFICATION
- * (b) TRIALS BY ACADEMIC COMMUNITY AND INDUSTRY
USING ACSL/CTRL-C
- * (c) DTI SURVEY OF UK NEEDS

PROPOSED ACADEMIC ASSESSMENT EXERCISES using CTRL-C & ACSL
with industrial "uncles"

<u>Team leader</u>	<u>Establishment</u>	<u>Application</u>	<u>Implementation</u>
Prof. M. Grimble	Strathclyde Univ.	Dynamic ship positioning. (Vosper Thorneycroft)	VAX11/750
Dr. M. Denham	Kingston Poly.	Flexible satellite, attitude and vibration control. (General Technology Systems)	VAX11/750
Prof. D. Atherton	Sussex Univ.	Performance evaluation of unmanned aircraft. (GEC Avionics) Missile launcher elevation servo design. (BAe Stevenage)	VAX11/750
Drs. R. P. Jones & M. T. G. Hughes	Warwick Univ.	Automotive transmission. (Lucas Research Centre)	SUN 3
Prof. D. McLean	Southampton Univ.	Active control for battle- field helicopter. (Westland Helicopters PLC)	VAX11/750
Prof. D. J. Murray- Smith	Glasgow Univ.	Parameter identification from helicopter flight data. (RAE Bedford)	VAX11/750
Drs. R. G. Cheetham & D. A. Wilson	Leeds Univ.	Fossil fuelled p/s plant control. (CEGB Harrogate)	VAX 8600
Drs. D. W. Clarke & I. Postlethwaite	Oxford Univ.	Nuclear fuelled p/s plant control. (CEGB Barnwood)	VAX11/780
Dr. J. Norton	Birmingham Univ.	Chemical plant control. (ICI NW Group)	VAX11/750
Dr. G. W. Irwin	Queen's Univ. Belfast	Missile guidance. (Short Bros.)	VAX 8600
Dr. D. P. Stoten	Bristol Univ.	Tyre tread extrusion line control system. (Avon Rubber PLC)	IBM PC/AT?

file CTRL-C/
5/4/86

2

RESEARCH AIMED AT IMPROVED
INFRASTRUCTURE AND DESIGN OF TOOLS

- * 1 DATABASE STRUCTURES - MACIEJOWSKI CAMBRIDGE
- * 2 MAN-MACHINE INTERFACE = MUNRO UMIST
GRAY SALFORD
BARKER SWANEA
- * 3 EXPERT SYSTEMS - MUNRO UMIST
- * 4 LANGUAGES - MARIANI LANCASTER
DENHAM KINGSTON
- * 5 DESIGN TOOLS

3

FEEDBACK FROM ACADEMIC/INDUSTRY TRIALS
IN JULY AND SEPTEMBER 1986

PUBLICATION OF ECSTASY SPECIFICATION
- JULY 1986

4

COLLABORATION WITH STU ?
IDENTIFY COMPLEMENTARY
WORK.

ECSTASY DEVELOPMENT PROGRAMME.

JAN 86 F M A M J J A S O N D 87 F M A M J J A S O N D 88 J A N

UNITS STUDY

6

ACADEMIC/INDUSTRY
UNITS ASSOCIATION

3½ CASD
WORKSHOP

2½ REPORTS

1½ SEMINAR

4½

SELECTION SURVEY

6

ECSTASY SPEC.
EVALUATION
WITH CUSTOMERS

4

FINAL SOFTWARE
DEVELOPMENT



NOTES FOR PRESENTATION TO THE SWEDISH NATIONAL BOARD FOR
TECHNICAL DEVELOPMENT (STU)

UNIVERSITY OF LUND 15TH APRIL 1986

THE SERC INITIATIVE IN COMPUTER AND DESIGN TECHNIQUES IN CONTROL
ENGINEERING (CDTCE)

Introduction

The background to the CDTCE initiative is as follows.

It is recognised that academic work in CDTCE has an increasing economic importance. If the techniques developed in a number of institutions are to be used by industry and built on by other researchers, they need to be usable. That is a statement of the obvious but to date our experience of many of the packages produced by Universities are that they are:-

- 1 Complex, not user friendly and often not finished off too well. The term "academic software" is not a complimentary one.
- 2 They are often of large size and have to be purchased as such and the user may only want to use a part.
- 3 Portability and implementation on alternative machines is often a difficulty.

- :
- 4 Support, maintenance and on documentation are often non existent.
 - 5 Packages are often developed with minimum consultation i e no industry/user feedback.

Other related concerns are that:-

- 1 Control Engineering research is outstripping applications.
- 2 There are many independent developments leading to duplication.
- 3 One must question the industrial relevance of much of the research being carried out.

Objectives

The SERC are concerned about all of these problems and are for example, aiming to identify gaps in our knowledge and to stimulate research in particular areas. It is also believed that a standardized computer environment within which one could embed

the design tools would enable better technology transfer.

between academic institutions

- 2 Out to industry
- 3 To stimulate feedback from industry (very important this one).

The idea is that the infrastructure would be made freely available to academic researchers and to industry. The particular techniques or tools embedded in the structure would be purchased as required by the particular researcher or industrial designer. The need for the software to be of high quality, to be well maintained and the availability of help and assistance together with good documentation is recognised. It is intended that the Rutherford-Appleton laboratory (RAL) would be involved in the generation of the software and in its support.

Actions

Turning now to what is actually being done. Originally DELIGHT was chosen as a vehicle for the infrastructure but although in the public domain no sound ownership could be ascertained and it was concluded, with regret, that it would not be a secure basis on which to build our system particularly if it was to be made available to UK industry.

It has decided therefore to make a critical examination of existing packages to ascertain their strengths and weaknesses and as a result produce a specification for the required package. This package has incidentally been christened ECSTASY which ~~stands for an~~ Environment for Control System Theory Application and Synthesis. Neil Monroe will talk further about this aspect later. The assessment will cover such features as tools implemented within the packages, the user interface and file structures, the command language, the graphics facilities and so on. This work is in hand and is intended to be completed by about mid July this year.

In addition, we are instigating studies in the UK by a number of University departments together with industrial collaborators or "uncles" of a federated package (ACSL/CTRL-C). This is being done to involve the UK academic community as well as a number of leading industrial companies in the SERC initiative following a very positive response from both to recent publicity. It is intended that there will be public feedback or comments from the use of this package in seminars to be held in July and September of this year.

It is worth indicating the current range of research grants in support of the CDTCE initiative which covers developments aimed at improving an infrastructure in the longer term and the development of design tools - they cover:-

- 1 Database structures
- 2 Man-machine interface
- 3 Expert systems
- 4 Languages
- 5 Design tools

The CACSD workshop is being held at Salford University on 2nd, 3rd and 4th July 1986 and it would be useful if there was participation by representatives from the STU. Finally of course, we are very interested in collaboration with the STU.

I should like know to hand over to Neil Munro who will describe the current status of the UMIST project work and preliminary ideas for the specification of ECSTASY.

U.M.I.S.T.

PROJECT

1. To critically examine various control system design software facilities

2. To produce a preliminary specification for ECSTASY
ie Environment for
Control
System
Theory
Applications and
Synthesis.

Packages being examined

1. CONCENTRIC **UMIST's Design Suite**
2. CLADP **Cambridge's Design Suite**
3. CSS **UMIST's Control System Software**
4. SIMIS **LUND's SIMNON, INTRAC, MODPAC,
IDPAC, SYN PAC**
5. DELIGHT **Berkeley's Infrastructure**
6. MATRIX-X **Integrated Systems Inc.'s
Design Package**
7. CONTROL-C **Systems Technology Inc.'s
Design Package**
8. ACSL **Mitchell and Cauthier's
Simulation Package**

Feature being examined

1. User Interface.
2. Infrastructure.
3. Tools.

USER INTERFACE.

1. Dialogue facility (ie the command language).
2. Graphics facilities (including windowing).
3. Data input.
4. Data output.
5. Help features and documentation.
6. Dialogue/history features.
7. Internal Editor.
8. MACRO features and structure.

INFRASTRUCTURE.

1. Database management.
2. File structure.
3. Command decoder.
4. Programmer tools.
5. Interface to 'foreign' code

TOOLS.

1. Model construction.
2. Model identification.
3. Analysis techniques (SISO).
4. Analysis techniques (MIMO).
5. Design methods (SISO).
6. Design methods (MIMO).
7. Synthesis methods.
8. Simulation facilities.
9. Numerical aspects.
10. Graphical aspects.
11. Failure response.

PRELIMINARY INFRASTRUCTURE SPECIFICATION.

1. USER FRIENDLY DIALOGUE FACILITIES.
2. FAST GRAPHICS FACILITIES.
3. FLEXIBLE MODELLING TOOLS.
4. ROBUST DESIGN TOOLS.
5. ROBUST SYNTHESIS TOOLS.
6. RELIABLE ANALYSIS TOOLS.
7. POWERFULL SIMULATION FACILITIES.
8. RELIABLE NUMERICAL ALGORITHMS.
9. EXPERT SYSTEM FACILITIES.

OVERALL FACILITIES:
FOR C.D.T.C.E.

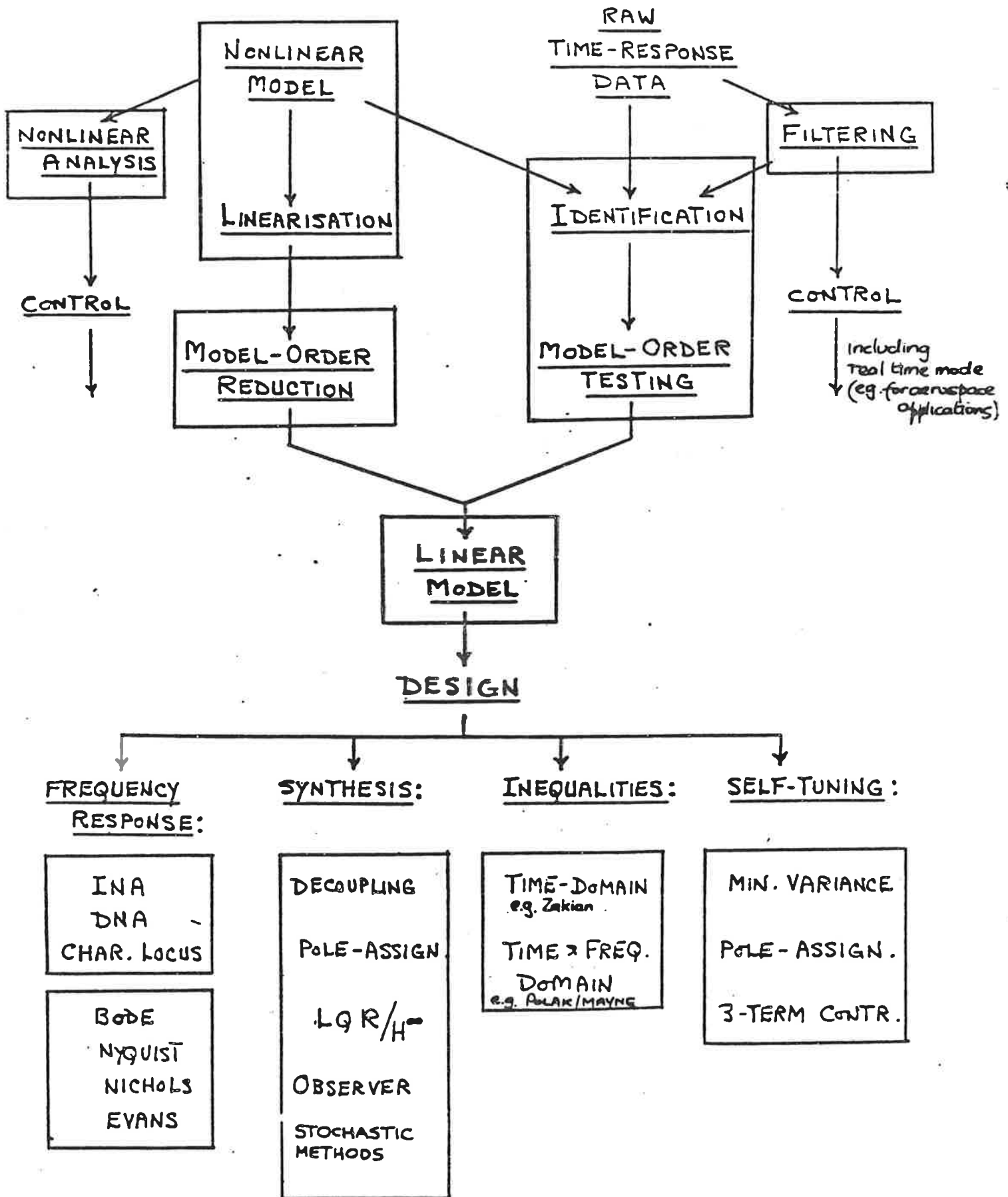


FIGURE 1

N. MUNRO

31-5-85.

ECSTASY

ENVIRONMENT for CONTROL SYSTEM THEORY, ANALYSIS and SYNTHESIS

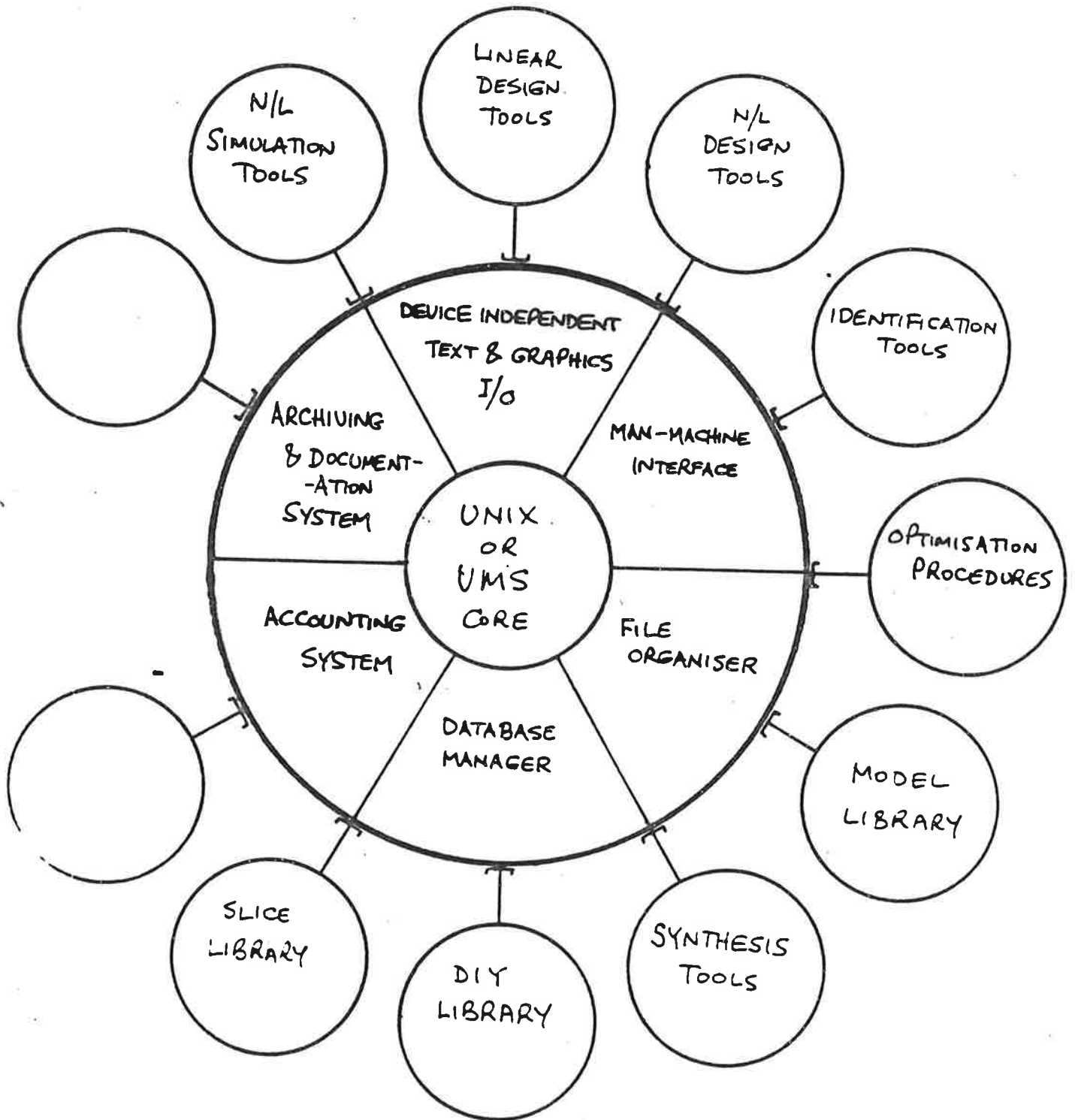


FIGURE 2

Aspects of Software Packages to be addressed:

	<u>SDG</u>	<u>GB</u>	<u>NM</u>	<u>BS/DL</u>	<u>JME</u>	<u>JR</u>
<u>1. User Interface:</u>						
a) Dialogue facility (ie the command language).	x					
b) Graphics facilities (including windowing)	x					
c) Data input				x		
d) Data output				x		
e) Help features and documentation					x	
f) Dialogue/history features				x		
g) Internal Editor					x	
h) MACRO features and structure					x	
<u>2. Infrastructure:</u>						
a) Database management					x	
b) File structure					x	
c) Command decoder	x					
d) Programmer tools				x		
e) Interface to 'foreign' code					x	
<u>3. Tools:</u>						
a) Model construction				x		
b) Model identification					x	
c) Analysis techniques (SISO)						x
d) Analysis techniques (MIMO)					x	
e) Design methods (SISO)						x
f) Design methods (MIMO)					x	
g) Synthesis methods				x	(x)	
h) Simulation facilities						x
i) Numerical aspects						x
j) Graphical aspects				x	(x)	
k) Failure response						x
<u>4. Critical Comparison of Features in:</u>						
a) Matrix-X or Control C	}			and CSS (Control System Software)		
b) DELIGHT						
c) Lund Software						
d) Reports and Demonstrations						
<u>5. Preliminary Specification of ECSTACY:</u>						
a) Visits to Swansea/Salford/Cambridge						
b) Specification						
c) Report						

Professor N. Munro
25 January, 1986.

Current CDTCE Grants, since October 1984
(excluding short term visitors)

	INVESTIGATOR INSTITUTION	TITLE	TOTAL AWARD	DATE
DATA BASE				
GR/D/1783.0	Maciejowski Cambridge	Data structures for control system design	£51,825	Oct '84
GR/D/6078.2	Maciejowski Cambridge	Data structures for control system design	£67,675	Oct '85
MAN-MACHINE INTERFACE				
GR/D/2845.4	Munro UMIST	An Interactive MMI for control system computer aided engineering	£46,204	Oct '84
GR/D/6960.0	Gray Salford	The development of a graphical input interpreter for the CAD of non-linear control systems	£73,572	Oct '85
→ Add	Banker Swansea.			
EXPERT SYSTEMS				
GR/D/3551.3	Munro UMIST	Appln. of expert systems to a new interactive MMI for control system CAD	£10,685	Feb '85
LANGUAGES				
GR/D/8356.9	Mariani Lancaster	A high level object oriented approach to distributed system design and implementn. for process control.	£77,488	Feb '86
DESIGN TOOLS				
GR/D/1823.3	Glover Cambridge	Model reduction in H [∞] Control system design.	£79,780	Oct '84
GR/D/3087.7	Postlethwaite Oxford	An H [∞] approach to control system design.	£43,520	Feb '85
GR/D/3312.0	O'Reilly Strathclyde	Robust non-linear Control	£27,290	Feb '85

PTO

GR/D/3937.5	Grimble Strathclyde	H [∞] robust controller and industrial applns	£24,923	Feb '85
GR/D/5543.6	Patton York	An assessment of the application of sliding mode control to a/c flight systems.	£13,780	Feb '85
GR/D/5545.0	Patton York	Extended Kalman filtering applied to helicopter modelling and control	£11,470	Feb '85
GR/D/4571.0	Owens Strathclyde	Parameterised modelling methods for linear and non-linear control.	£38,270	May '85
GR/D/4564.2	Gawthrop Sussex	Integrated design and implementation of robust multi-loop self tuning controllers	£83,266	May '85
GR/D/5635.8	Atherton Sussex	Further development of describing function modelling, analysis and design.	£54,700	May '85
GR/D/8206.7	Douce Warwick	Frequency response estimates using short time records.	£32,997	Feb '86
GR/D/7885.5	Limebeer Imperial	Theory and design of H [∞] control systems.	£46,570	Feb '86
GR/D/7580.9	Billings Sheffield	Time and frequency domain estimation for non-linear systems with applns. to modal analysis.	£86,302	Feb '86
GR/D/8482.5	Johnson Strathclyde	Control structures- assessment, validation -software & theory.	£20,000	Feb '86
GR/D/7884.8	Mayne Imperial	Optimisation based design of control systems.	£85,044	Feb '86
XG 10602	Brinson Bristol Poly.	Rotor-state feedback in helicopter flight control systems.	£16,826	Feb '86

PTO

XG 10732	Patton York	Development & eval- -uation of a param. identification facility using a scaled remote piloted helicopter.	£48,945	Feb '86
XG 10022	Bradshaw Salford	The use of transputer technology in flight control system design.	£56,957	Feb '86

file GRANT1
5/4/86

Advanced Computing Concepts and Techniques in Control Engineering*M. J. Denham, Kingston Poly.*Introduction

Computational methods and techniques have always played a major role in control engineering since the first computer-based control was put into operation in the mid-60's. This role has increased over the intervening years as the sophistication of the computing methods and tools available has also increased. [1] In particular, the introduction of the microprocessor and its use as a low cost computing element in a distributed computer control system, has had a profound effect on the way in which the design and implementation of a control system is carried out, and, to some extent, on the theory which underlies the basic design strategies. The development of interactive computing has encouraged a substantial growth in the use of computer aided design methods and robust and efficient numerical algorithms have been produced to support these methods. Major advances have also taken place in the languages used for control system implementation, notably the recent introduction of Ada, a language whose design is based on some very fundamental computer science concepts derived and developed over the past decade.

With the enormous expansion in the use of computing in all fields of commerce, business and industry, the more recent developments in computing have outpaced their incorporation into new control system design and implementation techniques. This is particularly true in those areas which have been the subject of intensive study in the last few years under various strategic governmental programmes, most of which were started in response to the Japanese "Fifth Generation" initiative. These areas are:

- * software engineering methods and tools
- * artificial intelligence techniques
- * new computer architectures, based on VLSI circuits

There is an increasingly urgent requirement to ensure that recent developments in these major computing areas are investigated, studied and assessed in relation to their potential benefit to control system design and implementation.

In the following, each of the areas, and related matters, eg languages, will be reviewed briefly and the potential relevance of recent developments to control engineering will be indicated.

Software engineering methods and tools

The increasing complexity of systems and the extent to which they are dependent on software control, has led to the requirement for effective methods and tools to aid the software development process. The aim is to produce reliable, maintainable and expandable software which will perform accurately accordingly to the specification of the required system behaviour. This is of paramount importance in the case of real-time control software, where the occurrence of errors can have expensive, if not catastrophic effects on system performance. To quote A J Macina, the manager of flight operations

for IBM's On-board Space Shuttle Program, when asked in a recent article [2] what computer science research and development could do to aid programmes like his in the future: "The most important area is software engineering. We need more structured ways of designing software and coding, and more automated ways for validating the correctness of software early in the life cycle."

One of the most important developments in recent years in this area has been in software specification techniques [3]. Of prime importance to the system designer is that the software controlling the system will perform in the way in which the designer intended. In order to ensure this, a precise description of the intended behaviour is required. Various techniques have been proposed for this and many system designers employ some form of specification procedure, either informal, including the use of semi-formal system description languages and graphical models of system behaviour, or formal, whereby a mathematical theory is used to provide a precise model of the intended behaviour, eg the theory of sets. The use of a mathematical model is important since it also provides the opportunity to construct a formal proof of correctness of the software produced to meet the specification. To do this it is necessary to produce the software in a language whose semantics are also precisely defined and can be interpreted in relation to the theory within which the specification is stated, ie a function exists between the objects, and the operations on objects, used in the specification model and the objects and operations available in the implementation language. It is interesting to note that in 1983 the US Department of Defense, in setting criteria for classifying software integrity, awarded the highest classification to systems that could be shown to have been formally specified and developed and proved correct.

Developments in this area have led to a number of formal specification languages, eg Z, CSP, OBJ, CLEAR, GYPSY, and to complete development methods based on formal languages, eg VDM. The reader is referred to [3] for descriptions of some of these languages and of their use, and to the references contain therein. Although at an advanced state of development for specifying purely sequential programs, there is still much active research on the concepts and techniques for parallel, or concurrent, program specification. The latter is, of course, of considerable relevance to control systems software specification, which invariably takes the form of a large number of programs operating in parallel, with both synchronous and asynchronous interaction between the programs and their environment and with time-critical constraints on the required system behaviour.

Control system specifications are normally expressed in terms of the continuous-time dynamic behaviour of the system, eg response time, settling times, frequency response, disturbance attenuation, and the application of design techniques translates this into a specification for a controller, usually in terms of a transfer function, which must be realised in software or special purpose hardware. Software specification techniques play a role at this stage, not only in ensuring that the software specification matches the required transfer function but that the input and output functions of the controller software are precisely stated. Most control systems also have a discrete event component, ie control is effected by individual actions taken by the controller at discrete moments in time, often determined by external events. Software specification techniques play a direct and important role in this case, since there is a much closer relationship between the specification of the discrete-event system behaviour and the software required to realise it. In fact, the mathematical concepts and techniques underlying concurrent software specification and the modelling of discrete-event dynamic systems are essentially identical, eg language and automata theory, net theory, domain

theory, temporal logic, etc, and there is a strong symbiosis between these two areas to be investigated and exploited [4].

Closely related to control software specification techniques is the language for implementation, for, as mentioned previously, to prove software correct there must be a well-defined function mapping between the semantics of the specification language and the semantics of the implementation language. The more closely the latter embodies the abstract constructs of the former in a concrete, realisable form, the simpler this mapping is and the easier the correctness proof. In this respect, Ada has proved to be valuable as an implementation language since it has a rich set of constructs which closely model the abstract data structures used in many software specification languages, eg VDM, OBJ. It has also been proposed that the specification language and the implementation language should be identical, ie the specification should be compiled directly into machine code, thus avoiding any possible mismatch between the required and actual behaviour and the need to prove correctness. This assumes, of course, the existence of a compiler for the specification language which provides provably correct code. However such a verification has only to be carried out once for each compiler produced, rather than for every piece of control software produced.

Other important related developments are in new languages and programming paradigms. Notable amongst these is the concept of object oriented programming, which is finding increasing favour in interactive computing in general, eg the Smalltalk system, and in application areas such as office automation and CAD [5]. The semantics of object oriented languages such as Smalltalk are, as yet, not well defined and their extension to use in programming concurrent systems, as in control systems, is at an early stage of investigation. However, the object orientated language BETA [6] has the latter as one of its objectives, and the apparent and widely promoted advantages of this approach to programming demand an investigation into its relevance to control systems. This is reinforced also firstly by the fact that object orientated languages are of considerable interest to the AI community, eg languages such as LOOPS, thus providing a straightforward mechanism for introducing AI techniques into control software, and secondly by the attractiveness of such languages in interactive computing and hence in CAD, thus providing perhaps a common language for modelling a system and specifying and implementing a its control software.

Other languages and programming approaches to be investigated include those based on the applicative style of programming, also known as functional programming languages. Interesting in this context, again because of its relevance to AI, is the logic programming language Prolog.

In summary, recent advances in software specification techniques, in particular for concurrent systems, and new programming approaches and languages are likely to have a fundamental relevance to control system development, in terms of improvements in integrity, reliability, correctness of behaviour, programming efficiency and productivity, etc. Many of these new developments are now appearing in the form of actual software specification and development tools, often in an integrated programming support environment. There is a strong need to investigate the relevance and potential benefits of these software engineering methods and tools to the control engineering community.

Artificial Intelligence Techniques

Software engineering has been an important issue in control engineering for many years but a more recent addition to the set of possible techniques and tools available to the control system designer is the use of artificial intelligence concepts. AI itself has existed as a discipline for many years, but it is only recently that other developments, notably in computer architecture, ie in processing speed and memory capacity, have made AI methods feasible computationally. We are now at the very early stages of seeing how such methods can be used in control engineering, both in design and implementation.

The use of AI methods in design of control systems has focussed so far on the use of expert systems to aid the user of a computer aided design system in selecting design strategies and in making design decisions within a particular strategy. [7],[8] Early efforts in the area have concentrated on traditional design strategies using frequency response methods to produce classical, transfer function based controllers and on system identification. The emphasis is on providing an intelligent, supportive man-machine interface for the non-expert designer. A major problem exists in creating the knowledge base required, both in terms of eliciting and formalising the inherently heuristic concepts and techniques used by the expert control system designer, and also in representing this knowledge in a computationally accessible and efficient manner. Over the last few years there has been considerable advances made in the area of knowledge representation and in eliciting knowledge in a form suitable for a particular representation, eg as formal logic statements, semantic nets, production or rule-based systems, frames, etc. [9] Many of the concepts involved in selecting and using an appropriate representation have yet to be investigated and are very closely related to other issues, such as the way in which the control system designer reasons about his design, in particular the form which the specification of required system behaviour takes, as described in the previous sections. It would appear that for traditional continuous-time dynamic systems the required formal specification methods based on mathematical objects such as transfer functions and state space models are sufficient to build a knowledge base for design. However, for more advanced adaptive and distributed control and for discrete-event control, it will be necessary to use formal specification techniques based on dynamic logic, set theory, etc as a foundation for representing and reasoning about system behaviour in an expert CAD system.

The same concepts extend to the use of AI techniques in the implementation of control systems. [10],[11],[12] In this case knowledge about the behaviour of the system or plant to be controlled must be elicited and formalised using some form of knowledge representation method. To this needs to be added a knowledge of how the system responds to applied control or extraneous disturbances. Since this involves essentially an infinite dimensional problem space, it may be necessary to employ some form of learning system for building additions to the knowledge base dependant on operational experience, either manually or automatically. There are enormous problems to be overcome in this area in respect of system security, reliability, robustness, etc before the control engineer can confidently design a system based on these techniques. However such systems, with the necessary safeguards, have been built and proved successful in operation. The substantial advances now being made in the AI research community, in respect of knowledge representation, search techniques, learning and inductive inference, planning and problem solving concepts and methods, need careful monitoring and investigation in terms of their relevance to control engineering. This is not only with regard to

traditional dynamic control but also in areas such as adaptive control, distributed and related areas of plant condition monitoring, failure mode analysis, alarm interpretation, etc.

New computer architectures

The enormous increase in the power of computer systems in recent years has made feasible in a practical sense a wide range of computational methods which were previously regarded as purely theoretical. Artificial intelligence methods are just one area where this has been the case. Other areas important to future applications of control engineering include system simulation [13], for design, implementation, verification and operator training, and processing of system data, for monitoring, control, alarm analysis, etc. The most important developments in the last few years have been the availability of relatively low-cost, high speed parallel processing computers [14] and the practicality in several applications, eg the automobile industry, of producing custom-designed processors using VLSI, which implement specific computational algorithms [15]. The importance of these developments, together with other more fundamental advances in computer architecture, eg optical methods, dataflow and reduction machines, is reflected in the massive funding that this area is receiving in the US under, for example, the DARPA Strategic Computing Initiative (SCI) programme [16].

For control engineering, new computer architectures are important both in the design and in the implementation stages. In design, there is now the potential for very high speed computation on parallel machines of design algorithms, eg for optimisation, which were previously considered to be impractical owing to their long computation time. Both optimisation and simulation, for even relatively simple systems, has always been in doubt as an activity in CAD of control systems which can be carried out in a practical interactive way. With fast, parallel numerical integration algorithms, simulation in particular could well play a more important role in interactive design methods.

For the implementation of control systems in aerospace applications, special computer architectures, using VLSI or VHSIC, are currently being used for special purpose signal processors. It is likely that as control applications in other sectors such as manufacturing and process control, automobiles, robots, etc become more complex and the control strategies computationally (numerically or symbolically) more demanding, the need for the use of novel computer architectures will arise, in particular fast, parallel-processing machines.

Conclusion

The rapid development of novel computing concepts and techniques in recent years, together with the enormous growth in computing power, has made a rich set of potential new tools available to the control engineer. In addition to the major areas covered above, advances in data communication networks and protocols are influencing the design of distributed computer control systems and the introduction of new concepts in interactive computing, eg the use of icons, windows, etc on high resolution graphical workstations, together with developments in computer graphics, eg GKS, are changing the nature of the man-machine interaction with control system CAD tools. The interface between the disciplines of computing and control engineering is steadily growing, but unless a substantial effort is devoted to the study of the way in which the

powerful new advanced computing techniques can best be applied in control engineering, as is happening now in other fields such as office automation, it is likely that control system designers and implementors will be limited to using outdated techniques based on concepts which are even now no longer current in the computing field. This will severely limit their access to new hardware and software technology and generally degrade the efficiency and productivity of both designers and the systems they design in the future.

References

- [1] Software for Industrial Process Control, collection of papers in Computer, vol. 17, pp6-74, February 1984.
- [2] Spector and Gifford: "The space shuttle primary computer system", Comm. ACM, vol. 27, pp874-900, September 1984.
- [3] Gehani and McGettrick [ed.]: Software Specification Techniques, Addison-Wesley, 1986.
- [4] Ostroff and Wonham: "A temporal logic approach to real-time control", Proc. 24th IEEE Control and Decision Conference, Fort Lauderdale, December 1985.
- [5] Degano and Sandewall [ed.]: Integrated Interactive Computing Systems, North-Holland, 1983.
- [6] Krittensen, et al.: "A survey of the BETA programming language", Norwegian Computing Centre, Oslo, 1981.
- [7] Taylor and Frederick: "An expert system architecture for computer aided control engineering" Proc. IEEE, vol. 72, pp1795-1805, December 1984.
- [8] Larsson and Astrom: "A expert system interface for IDPAC" Proc. 2nd IEEE Symposium on Computer-Aided Control System Design, Santa Barbara, March 1985.
- [9] Barr and Feigenbaum [ed.]: The Handbook of Artificial Intelligence, William Kaufmann, 1981.
- [10] Mamdani, et al: "Developments in fuzzy logic control" Proc. 23rd IEEE Conference on Decision and Control, Las Vegas, December 1984.
- [11] Francis and Leitch: "Intelligent Knowledge-Based Process Control", Proc. IEE Control 85 Conference, Cambridge, July 1985.
- [12] Moore: "Adding real-time expert system capabilities to large distributed control systems", Control Eng., vol. 32, pp118-121, April 1985.
- [13] Singh, et al.: "Parallel processing techniques for large scale simulation problems", Proc. IEE Control 85 Conference, Cambridge, July 1985.
- [14] Mokhoff: "Concurrent computer makes scientific computing affordable" Comput. Des., vol. 24, pp59-60, April 1985.
- [15] Kung, Whitehouse and Kailath [ed.]: VLSI and Modern Signal Processing, Prentice-Hall, 1985.
- [16] Schneck, et al.: "Parallel processor programs in the Federal Government", Computer, vol.18, pp43-56, June 1985.

Arne Otteblad/ksb

PROTOKOLL

från möte med STUs styrgrupp för ramprogram CACE den 15/4 1986
 kl 9 00 - 16 15 vid Institutionen för reglerteknik, Lunds tekniska
 högskola.

Närvarande:

Styrgruppsmedlemmar:

Sven-Gunnar Edlund
 Claes Källström
 Gustaf Söderlind
 Karl Johan Åström
 Arne Otteblad
 Eric Sandewall (per telefon)

Projektengagerade:

Sven-Erik Mattsson
 Dag Brück
 Ulf Holmberg
 Jan-Erik Larsson
 Per Persson
 Klas Sigbo

Engelska gäster:

Jim Andersson, ICI	§§ 1, 7, 8
Mike Denham, Kingston Polytechnic	§§ 1, 7, 8
Phil Hicks, SERC	§§ 1, 7, 8
Neil Munro, UMIST	§§ 1, 7, 8

§ 1 Projektpresentationer

Forskarna vid institutionen för reglerteknik gjorde mycket intressanta presentationer av de pågående projekten och de resultat som hittills kommit fram.

Pärmar med material rörande projekten utdelades till mötesdeltagarna. Följande projekt behandlades:

Nya former av man-maskin kommunikation

Experiment med expertsysteminterface

Högnivåspråk för lösning av reglerproblem

Kombination av formelbehandling och numerik

Expertreglering

Paragrafen innefattade även ett längre intressant inlägg av Eric Sandewall berörande kopplingen mellan reglerteknik och expertsystem.

§ 2 Härmed startade det formella styrgruppsmötet med godkännande av föregående protokoll, val av Arne Otteblad, STU, till sekreterare och Sven-Erik Mattsson, LTH, till justeringsman.

§ 3 En genomgång av det ekonomiska läget gjordes. Det noterades att för innevarande budgetår finns 402 723 kronor tillgängliga efter det att återbetalning av utrustningsanslaget har skett. För nästa budgetår finns utöver de redan beslutade projekten cirka 1146 kkr tillgängliga.

Det konstaterades vidare att köp av arbetsstation bör anstå till nästa budgetår. Bland annat bör man avvakta resultatet av en FRN-ansökan på 1500 kkr för en Lisp-maskin.

§ 4 Ansökningshandlingar för nedan angivna projekt hade sänts ut till styrgruppen. De diskuterades och styrgruppen rekommenderade stöd till samtliga projekt.

a) Representation och visualisering av system och deras beteende.

659 kkr för tiden 1/5-86 - 30/6-87.

Det konstaterades att det är viktigt att en testning sker mot system-användare. Styrgruppen uppmanades därför att sända förslag på intressenter, som kan medverka i testningen, till K J Åström.

Det uttalades också att vi vid nästa sammanträde i styrgruppen bör diskutera i vilken utsträckning tids- och händelsestyrda system hör hemma inom ramprogrammet.

b) Experiment med expertsysteminterface.

175 kkr för tiden 1/7-86 - 30/6-87.

Detta är ett pågående projekt som styrgruppen anser bör fortsätta även nästa budgetår med experiment och framtagning av hjälpfaciliteter. Det konstaterades att ett expertsystem kan kräva 1000 - 2000 regler och att detta inte kan åstadkommas inom den här kostnadsramen. Regelbasen får utvecklas senare.

c) Implementeringsspråk.

161 kkr för tiden 1/5-86 - 30/6-87.

Detta är ett projekt som prof Kreutzer skall arbeta med som gästforskare vid institutionen för reglerteknik. En viktig fråga i detta projekt är hur man hanterar osäkerheten om vilka språk som kommer till användning i framtiden.


d) Numerisk behandling av differential-algebraiska system.

155 kkr för tiden 1/7-86 - 30/6-87.

Detta är ett projekt som skall genomföras av Bo Kågström och Anders Barrlund i Umeå. Det uttalades inom styrgruppen vissa farhågor för att projektet blir av alltför långsiktig grundforskningskaraktär och att man inte får fram alla resultat inom ramprogrammets tidsperiod. Man beslöt dock att rekommendera stöd till projektet för ett år. Därefter har man underlag för bedömning om det är meningsfullt att arbeta vidare med projektet inom ramprogrammet.

- § 5 Styrgruppen diskuterade policy för samarbetet med motsvarande program inom SERC i England. Det konstaterades att deras verksamhet är 4-5 gånger större än vår och att det är värdefullt för oss att ta del av den. Styrgruppens allmänna mening var att man bör arbeta med "öppna kanaler" gentemot SERC. Speciellt uttalade sig styrgruppen positivt till ett utbyte av gästforskare och även till anordnande av gemensamma workshop. Beträffande utbyte av dataprogram ansåg man att detta får diskuteras från fall till fall.
- § 6 Beträffande framtida styrgruppsaktiviteter diskuterades och bestämdes följande:
- a) Karl Johan Åströms forskare kommer att ha ett möte med Eric Sandewalls grupp i början av hösten -86. Styrgruppen kommer att inbjudas.
 - b) En workshop, där projekten presenteras och den fortsatta inriktningen diskuteras, bör anordnas under budgetåret 86/87. Karl Johan Åström och Arne Otteblad fick i uppdrag att finna lämplig tidpunkt i början av 1987. Planeringen inriktas mot att kostnaderna skall klaras inom befintlig budget.
 - c) Nästa styrgruppsmöte bestämdes till torsdagen den 27 november 1986.
- § 7 De engelska gästerna presenterade öppet, utförligt och intressant SERC-programmet. Här skall endast nämnas att programmet "Computer and Design Techniques in Control Engineering" är planerat för en total insats på 3,4 miljoner pund under 5 år. Start skedde sensommaren 1984.
- § 8 Samarbetet mellan forskargrupperna i STUs och SERCs program diskuterades. Samtliga var positiva till ett mycket öppet samarbete, där deltagande i varandras styrgruppsmöten, gemensamma seminarier och forskarutbyten ansågs vara lämpliga ingredienser. Först måste dock lämpliga problemområden för samarbetet identifieras. Denna fråga liksom lämpliga seminarieaktiviteter hänsköts till kommande dags överläggningar mellan Karl Johan Åströms forskargrupp och den engelska gruppen. Se vidare protokoll från dessa överläggningar i bilaga 1.

Vid protokollet


 Arne Otteblad
 STU

Justeras

 Sven-Erik Mattsson
 LTH

Notes on
COLLABORATION STU - SERC

Sven Erik Mattsson

This note summarizes the discussions on April 16, 1986 between Jim Andersson, Phil Hicks, Mike Denham and Neil Munro, members of the Steering Committee of the SERC Initiative in Computer and Design Techniques in Control Engineering (CDTCE), and Karl Johan Åström, Sven Erik Mattsson, Dag Brück, Per Persson and Jan-Eric Larsson, members of the STU Computer Aided Control Engineering Programme (CACE), on collaboration.

Areas of Interest

Four main areas of joint interest were identified:

1. User Interface
 - Graphics
 - Graphics Standards
 - Expert System Interfaces
 - Natural Languages
2. Implementation
 - Infrastructure
 - Implementation languages
 - Workstations
 - High Level Problem Solving Languages
 - New Tools
 - System Representation
 - Database
3. Design Tools
 - Algorithms
 - Methods
 - Numerics
4. Control Systems Realization
 - Automatic Control
 - Expert Control
 - Languages
 - New Computer Architectures

Modes of Cooperation

Different ways to cooperate were discussed:

- Exchange of information
- Participation in each others steering committee meetings
- Workshops
- Exchange of researchers
- Joint projects

It was felt that good cooperation always builds on strong joint interests. It was decided to start a number of activities that could create a good environment.

Workshop on Graphical Front Ends

It was decided to arrange a workshop on graphical front ends in the week July 14 - 18, 1986 at UMIST. The goal of this workshop should be to set the course, identify the state of the art and identify relations for possible joint work. A report should be ready by September 1, 1986. Professor Neil Munro, UMIST is responsible for the organization. It was anticipated to have a direct influence on the projects planned for 86/87 in England and Sweden.

The meeting is intended to be a workshop for active researchers in the SERC CDTCE initiative and the STU CACE programme. However, external researchers and people industry people will also be invited. The size of the workshop is estimated to 20 - 30 persons.

In the opening of the workshop the researchers will present their work and plans in this area. The programme will also include presentations of graphics standards (GKS, PHIGS etc). Important issues are: "What are they?", "How do they compare?" and "Can we guess industrial acceptance?".

Time will be reserved for formal and informal discussions on future directions both on short and long terms. The workshop will end with a concluding session which will include a discussion of the structure of the final report.

Workshop on Expert Systems

There is also an interest to arrange a workshop on expert systems and expert system interfaces. We will discuss the arranging of such a workshop at the workshop in July 1986.

Collaboration

Professor Neil Munro's work on specification for an Environment for Control System Theory Applications an SYnthesis (ECSTASY) is of great interest for us. In this project they evaluate various CACE programs as we do in the CACE project "High level problem solving languages". We can here exchange various experiences on programs, environments and workstations.

Mike Denham and Karl Johan Åström will think about a programme for collaboration on "Control System Realization".

Further collaboration will of course also be considered at the workshop in July 1986.