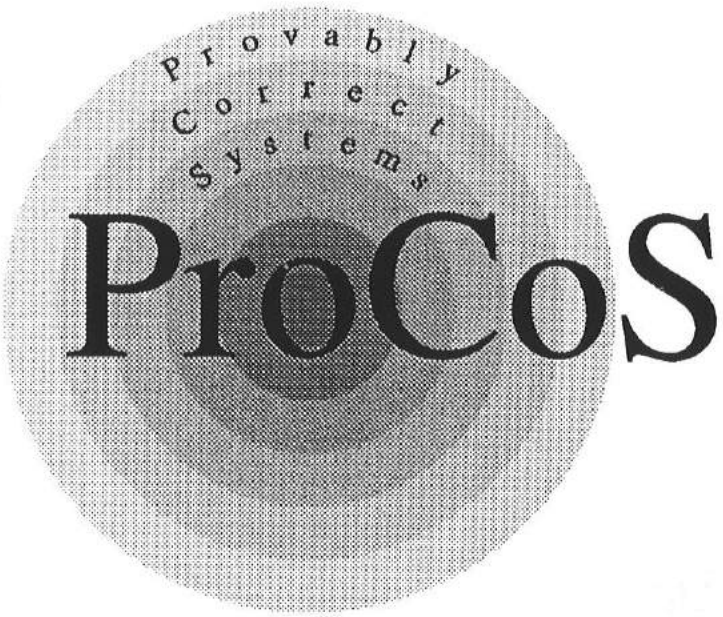


ESPRIT 3104
ProCoS

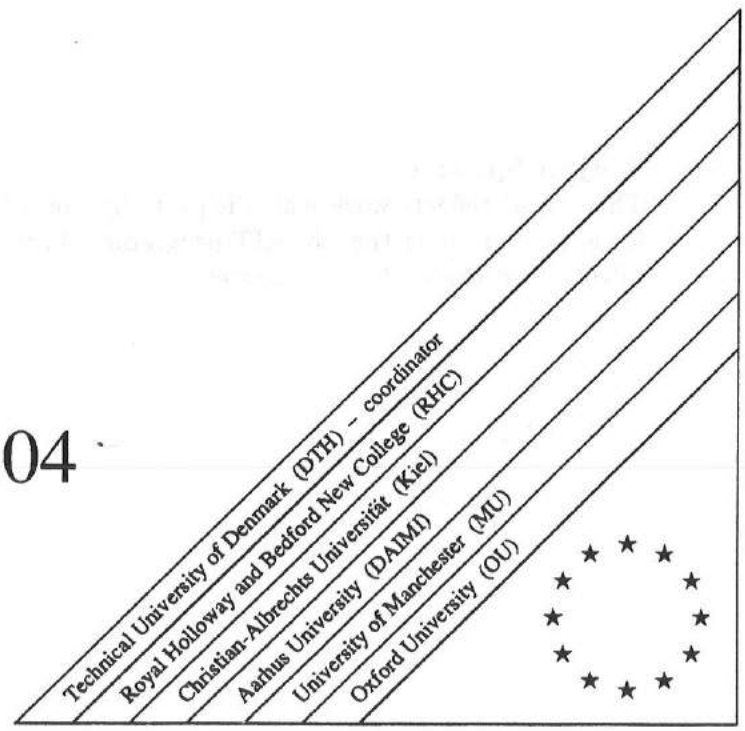
A ProCoS Project Description
ESPRIT BRA 3104

Author: Dines Bjørner

Date: October 1989

Doc. Id.: ID/DTH DB 3 — Version 3

esprit BRA project 3104



Doc. type:	General Information
Status:	Final
Confidentiality:	Free Distribution
Activity-alloc:	Management
Distribution:	**TO BE SPECIFIED**
Copyright © 1989	ID/DTH, Kiel, OU, RHC, DAIMI, MU

Document History
One version of this report was submitted for a CSSR conference, Strebske Pleso, Nov. 6-10, 1989: Proceedings to be publ. by North Holland. That version has DB as single author, but recognizes all present co-authors as collaborators. Present version, slightly augmented, aimed for EATCS Bulletin publication.

Document Purpose
General information for wide circulation.

Project Sponsor
This report reflects work which is partially funded by the Commission of the European Communities (CEC) under the ESPRIT programme in the field of Basic Research Action proj. no. 3104: "ProCoS: Provably Correct Systems"

A ProCoS Project Description
ESPRIT BRA 3104

Dines Bjørner

Abstract

The aims and objectives, baseline and rationale, as well as the detailed structure of a 3 country, 6 university, 30 month research project supported in part by the European Economic Community is described.

The very short aim of the project is to contribute to the science and engineering of constructing mathematically provably correct safety critical computing systems.

We believe that one of several novel features of this project is that it is being planned around a notion of theoretically motivated project component diagram, see figure 1.

Contents

1 Introduction

The acronym title of the research project described here is: **ProCoS** which stands for: Provably Correct Systems.

Project partners, site acronyms, their denotation, and site leaders are:

DTH: Department of Computer Science, Technical University of Denmark, Lyngby, Denmark. **Dines Bjørner.**

Kiel: Institute of Informatics and Applied Mathematics, Christian Albrechts University at Kiel, Federal Republic of Germany. **Hans Langmaack.**

OLD: Institute of Informatics, Oldenburg University, Federal Republic of Germany — subcontractor to Kiel. **Ernst Rüdiger Olderog.**

MU: Department of Computer Science, The University of Manchester, England. **Douglas Edwards.**

OU: Programming Research Group, Computing Laboratory, Oxford University, England. **C. A. R. Hoare.**

RHC: Department of Computer Science, Royal Holloway (and Bedford New) College, London University, England. **Ursula Martin.**

DAIMI: Department of Computer Science, Aarhus University, Denmark. **Flemming Nielson.**

We believe that the current paper is of general interest for the following reasons: (i) it clearly states the synopsis, objectives and hopes for advance, rationale, and baseline of the project; (ii) it enunciates a project planning, and hence subsequently a project management control, scheme in the form of a project component diagram and its mapping onto a project graph, and (iii) it applies state-of-the-art research results.

2 Synopsis

Our objectives are to advance the state of the art of systematic design of complex heterogeneous systems, including both software and hardware; in particular, to reduce the risk of error in the specification, design and implementation of embedded safety-critical systems.

To approach this goal, we plan to develop a concrete system consisting of the following five major components:

- A specification language.
- A programming language.
- A definition of a hardware machine.
- A compiler from the programming language to instructions of that machine.

- A kernel supporting the execution of compiled programs on that machine.

The syntax and semantics of these components will be formalised, and their formal interrelationships will be established.

The project will base its work on the CSP/occam/transputer tradition. During a first 15 month phase, a simple occam subset and a simple transputer-like RISC machine will be selected and formalised as far as possible. During the second 15 month phase, our methods will be generalised and improved, and the system design reiterated.

Since implementation of the machine in hardware is not part of this project, liaison will be made to relevant hardware projects, in particular the IED[1] safemos project.

The work will furthermore be supported by (i) case studies of requirements for and development of safety-critical systems, (ii) experiments in computerised verification support, and (iii) liaison with a number of related projects.

The result of the project is expected to take the form of a book on principles of contructing provably correct systems using the developed system as a special case.

The book could be the scientific basis of the software side of a larger-scale commercial or precompetitive project for the design and production of a thoroughly validated system, suitable for application in a safety-critical environment.

3 Objectives and Advance

The major engineering achievements of this century have been firmly based on an understanding of the relevant branches of science. They have involved cooperation between hundreds or thousands of engineers of various specialisms, and on strict control of correctness of design and accuracy om implementation. For both these purposes, it is essential to make an initial split of the overall task into many subtasks, to define with utmost precision the interfaces between the subtasks, to make an early check of their consistency and completeness of the interfaces. It is necessary also to install technical and managerial procedures to inhibit errors in implementation, and to check that they have been successful in doing so.

This basic research action stems from the belief that similar engineering and management principles should be observed in large-scale software projects, and will result in better control of costs and delays in delivery. These principles are of especial importance in projects which have implications for public safety. But the principles must have a sound basis in the underlying science. The objective of the action is to clarify that basis, to bring together the various relevant scientific disciplines, to identify and concentrate attention on missing links and to gain experience and understanding of the applicability of the principles on component parts of a demonstrator project of moderate size.

The demonstrator project has been selected for its possible relevance to safety-critical systems, with an element of reactive control. We have identified the following important interfaces, which must be specified with great rigour and accuracy:

[1] Information Engineering Directorate, UK

1. The interface to reality: the physical properties and behaviour of the environment within which the software will be embedded, including control objectives and safety criteria.

2. Interface to a compiler: the programming language in which each application-specific program will be expressed.

3. Interface to the hardware: the machine language produced by a compiler which will be executed by the physically embedded hardware.

Each of these interfaces are so important that they should if possible be given two specifications, using independent specification methods, accompanied by a proof of their mutual consistency.

The main work of the action is to develop and formalise the technical procedures to aid in implementing the transition between these major interfaces.

4. Between 1 and 2, we need top-down systematic development methods for extracting designs from specifications, and programs from designs. The use of these will be illustrated by a case study.

5. Between 2 and 3, we need to specify the action of a compiler as a set of correctness-preserving transformations. The design of the compiler will involve specification of interfaces between its various parts, followed by their implementation.

6. Between 3 and the actual electronic(s) of the hardware lies the activity of VLSI design. The work under this heading is not directly funded under the **ProCoS** action; it will be covered by strong technical liaison with other related projects.

The main technical innovation of the project lies in its attemp to co-ordinate the separate specialist scientific theories and engineering skills of the individual participants towards an integrated common objective. We will be able to identify and concentrate on any gaps in the available underlying science. The international nature of the collaboration with six research centres spread over three countries will pose an additional challenge; if this challenge can be met in a scientific fashion, this will contribute to a major advance in understanding of the relevance of scientific and engineering principles to the management and control of large-scale software projects.

In the longer term, the discoveries of this project may serve as the scientific basis of national and international standards for safety critical software.

4 Rationale

The successful and reliable design of an engineering product requires a judicious choice of levels of abstraction, which permit different aspects of the design to be carried out independently at different times by different people; and which permit the correctness of each part of the design to be checked by strongly localised reasoning.

In the **ProCoS** project, we will take advantage of the standard hierarchies of levels of abstraction; starting from machine code, higher level language, operating system, library and continuing right up to methods for reliable development of application programs for embedded systems.

Within each of these levels of abstraction, there is considerable experience of formal design on which we can draw. But the place where the bugs may now congregate is in the interfaces between the levels of abstraction, and between the major components within each level of abstraction.

One of the major goals of the **ProCoS** project is to close these gaps. Obviously in any practical project, this requires great vigilance and hope that from this experience may emerge a general theory of interfacing, using perhaps a generalisation of the methods of data abstraction, which can be studied profitably within a categorical setting.

5 Introduction to Workplan

In accordance with the main components and additional activities referred to in the synopsis we identify eight major project activities:

1. **Case Study:** One or more case studies are to be made of embedded systems found in typical safety critical applications: Railway Signalling, Hydro-electric power station Water Control, etc. Rigorous interface specifications must be developed. These shall reflect the properties and behaviour of the environment, control objectives, and safety criteria. The specifications shall be meaningful to experts of the application area and to safety analysts. Provably correct programs implementing the computerised monitoring and control of such systems conclude this area of the action.

   This part of the action should ideally produce requirements to the delineation of the specification, programming and machine languages, and the transformation rules.

2. **Specification Language:** A specification language capable of expressing salient features of the control and monitoring of safety critical systems is selected, a mathematical model for this language is given, and its transformation rules (proof system) related to the programming language.

3. **Programming Language:** A small number of gradually more and more ambitious programming languages are selected, and consistently given abstract and operational semantics.

   The languages will range from a minimum language roughly corresponding to the Dijkstra/Gries nondeterministic language to a maximum language being more complex than the present occam-2.

   The minimum language is the lower limit to the ambitions of the project, and the maximum language an upper limit. Each subtask (eg. compiler implementation)

should support the minimum language at least, but where necessary may take advantage of the features of the maximum language.

4. **Machine Language:** A transputer-like machine language is delineated, abstract and operational semantics are developed, the two semantics' are proved consistent, and the semantics of the programming and the machine languages are related — the latter as part of the compiler verification effort.

   The machine language is to be selected in co-operation with the `safemos` project conducted by `inmos` plc., Oxford Univ., SRI International Cambridge, and Cambrdige Univ.

   It is expected that the machine is transputer-like, where at the minimum level of ambition, each transputer will support only one `occam`-like process. Programs resulting in several processes will then be handled by a hardware configuration involving at least that number of transputers.

5. **Compiler Development:** The project specifies a compiler, implements this compiler and proves it correct. The compiler compiles programs of the programming language and generates code for the machine language.

   Two facets of verification are identified: compiling verification (compiler specification verification): verifying that the specified code is correct, and compiler verification: verifying that the compiler itself (executable compiler program) correctly generates that code.

6. **Kernel:** A kernel, ie. an absolute minimum operating system, has to be identified, specified and correctly developed. This kernel shall provide minimum support for the execution of PL programs on the ML machine such as communication support.

6 The Workplan

6.1 Logical Structure

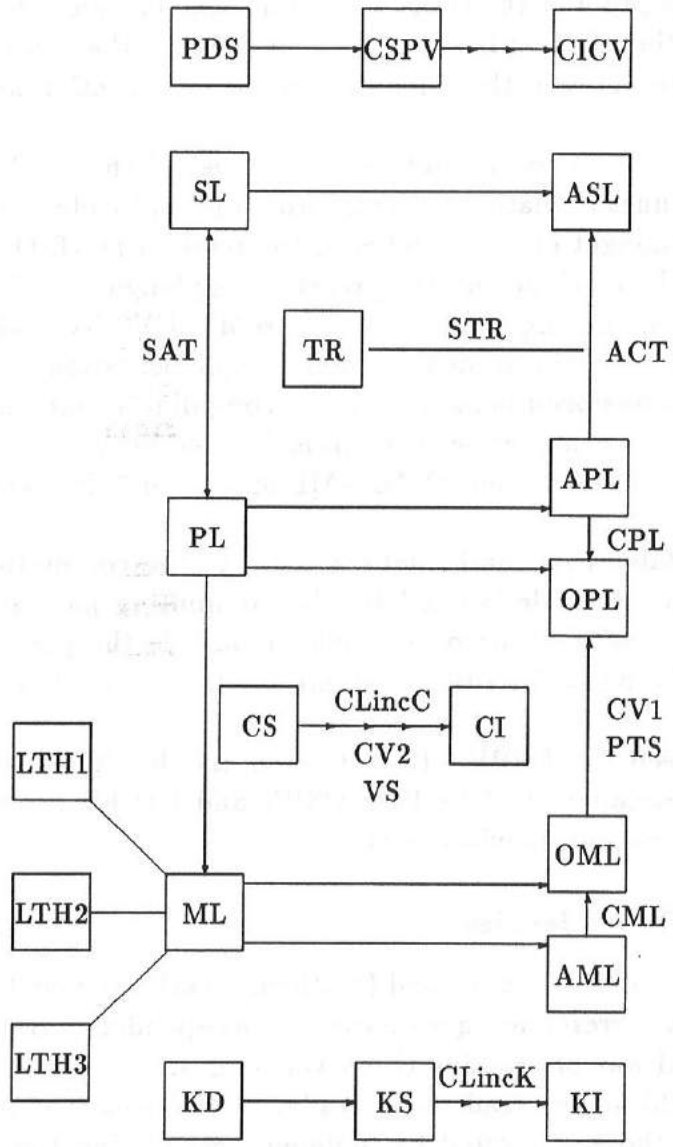
In the following, the major actions are broken down in so-called work-items which form the basic components of the workplan. Figure 1 (page 7) illustrates the basic components of the workplan. Figure 2 (page 9) illustrates a usage dependency graph of the components.

Short Explication of Component Diagram

The Component Diagram (figure 1, page 7) is built around two commuting diagrams with some additional parts. The diagram basically represents our present view of the mathematical relationships among the different levels of abstraction. An important result of the project is to confirm or more likely to change this view!

The upper commuting diagram of figure 1 connects the specification and programming language 'algebras'. The lower commuting diagram connects the programming and

Figure 1: ProCoS Project Component Diagram



Legend:

PDS: Problem Domain Spec.

CSPV: Case Spec.&Prop.Verif.

CICV: Case Impl.&Corr.Verif.

SL: Specification Lang.

ASL: Abstr.SL Sem.

PL: Progr.Lang.

APL: Abstr.PL Sem.

OPL: Operat.PL Sem.

CPL: Consist.: OPL wrt. APL

ACT: Corr.Thm.: SAT to APL,ASL

SAT: Satisf.Rel.: PL to SL

TR: Transf.Rules: SL to PL

STR: Soundness of TR

CS: Compiler Spec.

CI: Compiler Impl.

CLincC: CLinc Comp. Verif. Study

CV1: Compiling Verif.

PTS: Prog.Transf.Support.

CV2: Compiler Verif.

VS: Verif.Supp.

ML: Machine Lang.

OML: Operat.ML Sem.

AML: Abstr.ML Sem.

CML: Consist:OML wrt.AML

LTH1: Liai.to **safemos**

LTH2: Liai.to **ELLA**

LTH3: Liai.to **SSVE**

KD: Kernel Design

KS: Kernel Spec.

KI: Kernel Impl. & Verif.

CLincK: CLinc OS Verif.Study

machine language 'algebras'. Arrows usually designate various mathematical relationships.

SL, PL, ML designate informal definitions of respectively the specification, the programming and the machine languages. A- and O- prefixes to these acronyms designate abstract, respectively operational semantics' for those languages. That completes the corners of the main diagram. Now the edges and arcs.

SAT denotes the satisfaction relation tying programs to specifications. TR is also a label of the PL-SL relationship designating rules for transforming specifications into programs. STR denotes the soundness proof of the transformation rules TR and labels the edge going from the TR box to the APL-ASL arc since soundness of the rules is to be given relative to the relationship between the semantics of the specification and programming languages.

Similarly, in the lower diagram, CS is to be thought of as a label of the PL-ML arc since the compiler specification defines a relation between programs and code. And CV1 (compiling verification) can be thought of as the label of the relation OML-OPL between the operational semantics of the machine and the programming languages. Put in different terms: CV1 verifies the compiling algorithm from PL to ML. CV2 (compiler verification) labels the arc from compiler specification to compiler implementation.

PTS denotes support for program transformations used in the compiling verification CV1. General verification support for the comipler verification is denoted by VS.

CPL, CML labels respectively the OPL-APL and OML-AML arcs — and designate respective consistency proofs.

Additional parts not directly relatable in a mathematical sense to the commuting diagram structure, but supporting the work designated by the commuting part are: LTH1–3 designate liaison to "transformation to hardware" efforts outside the present project. The PDS-CSPV-CICV and the KD-KS-KI triples stand for the case study and the kernel development.

The multiple arrowed lines (between the CSPV-CICV, CS-CI, and KS-KI boxes) stand for stages of proper software development. The PDS-CSPV and KD-KS arrows stand for relations between specifications and 'requirements'.

Short Explication of Work-item Dependencies

While the component diagram of figure 1 describes logical (mathematical) relations between the documents to be produced by corresponding work items, the dependency graph of figure 2 describes a feasible, physical way of ordering these work-items.

The usage dependency graph should not be read too literally: work items that are shown in parallel with others (except those connected by bulleted, vertical lines) need not be pursued concurrently. Also: no absolute timing (ie. no time bar chart) will (yet) be given.

The triples: (ASL-APL-SAT), (ASL-APL-ACT), (APL-OPL-COL), and (OML-AML-CML) somehow overdefine activities related to these work-items. Within each triple the connected work-items are, more or less performed simultaneously.

Finally the dependency graph is only very tentative — its establishment, its further refinement and its use as the basis for project monitoring and control is a matter of project management concern.

Further explication is given under **Legend** in the right-hand side of the graph.

6.2 Deliverables

The project will deliver one major, and one minor deliverable. The major deliverable is a major, technical report. The minor deliverable is a set of demonstrator software.

Major Deliverable

The project will deliver one major technical report. This report will appear in two versions:

Interim version: after 14 months (at milestone 1).

Final version: after 30 months (at milestone 2).

The status of both versions will be **P**: public, available at a nominal charge.

The interim and the final reports are the only proper deliverables from this project. The expectedly very many technical and research notes and reports generated during the project by the work-items are not in themselves deliverables, but shall serve to justify the two deliverables. These notes, however, will have status **C**: available to participants and for action monitoring.

Contents of Main Deliverables

The interim and final versions of the main deliverable will take the form of a monograph-like book on principles of development of provably correct systems. Thus, the book is intended to present the essence of our work in a comprehensive form.

The book will tentatively have the following abstract syntax for chapters:

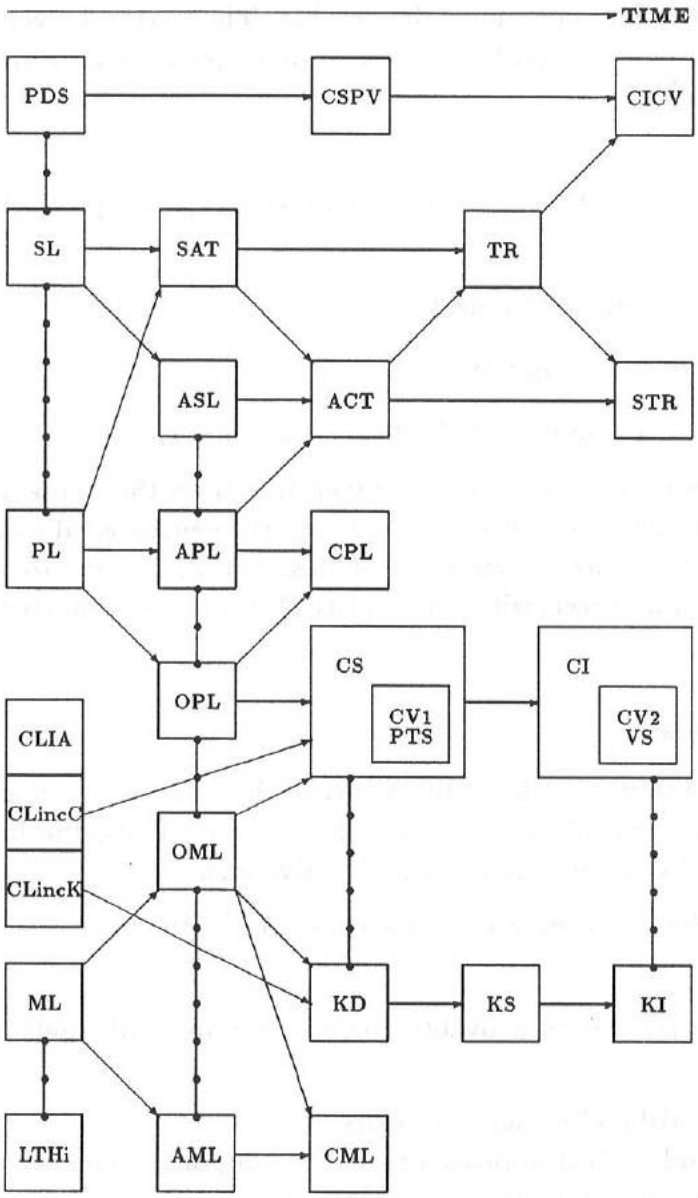
1: Introduction:
Introduction to issues related to provably correct systems and safety-critical systems.

2: Development of Provably Correct Systems:
Overview of the methodological approach (project component diagram and project dependency graph, figures 1-2.), other principles, techniques and tools.

3: Case Studies:
The essence of the PDS-CSPV-CIPV study, research and development.

4: Specification Language Issues:
The essence of the SL-ASL-TR-STR-ACT-SAT study and research.

Figure 2: Work Item Dependency Graph



Legend:

- Time progresses left-to-right.
- Directed arcs designate time dependency.
- Vertical, bulleted lines designate mutual interdepency.
- The **ProCoS** project basically reiterates increasingly larger parts, left-to-right, of the graph in the 3 cycles:
  - May '89 - Oct. '89
  - Oct. '89 - July '90
  - July '90 - Nov. '91
- No more exact time bar chart will be given for the work items shown to the left.
- Some work items (CLinc- and LTH-related) continue beyond their initial [dependency] phases.
- LTHi stands for LTH1-2-3.

5: Programming Language Issues:
The essence of the PL-APL-OPL-CPL study and research.

6: Machine Language Issues:
The essence of the ML-AML-OML-CML-LTH1-LTH2-LTH3 study and research.

7: Compiler Development Issues:
The essence of the CS-CI-CV1-CV2-CLincC study, research and development.

8: Kernel Development Issues:
The essence of the KD-KS-KI-CLincK study, research and development.

9: Verification Support Issues:
The essence of the VS1-VS2 study, research and work.

10: Mathematical Foundations:
A summary of the mathematics underlying the development of provably correct systems, ie. of the ASL-SAT-STR-ACT, APL-OPL-CPL, OML-AML-CML studies and research.

11: Programming Methodology:
A summary of the programming methodological diversities encountered in the CSPV-CIPV, SL-TR, PL, ML, CS-CI-CV1-CV2, KD-KS-KI studies, research and development, as well as in establishing the project components and the process of following the component diagram as a recipe for full development.

Appendix A: Annotated Report Bibliography:
A complete, extensively annotated list of the technical notes and reports, and the research notes and reports generated during the project.

Appendix B: Other References

Appendix C: State-of-the-Art:
An assessment of the State of the Art of the field as obtained through the reading and liaison activities.

Appendix D: Terminology

For reasons of easier flow of reading, certain chapters, eg. 3-4-5-6, may appear rather as chapters $3 + j + 4 * i$ for j from 0 to 3, and i from 0 to 2 — where $3 + j$ is above chapter, and i designate its iteration.

Minor Deliverable

The application of the various development methodologies — for the case study examples, the compiler and the kernel — involves actual development of software, and involves the use of software tools — LALR(K) parser generators, attribute grammar evaluators, OBJ3, other rewrite rule propocessors, etc.

Insofar as the combination of these constitute demonstrable software, the **ProCoS** project will deliver such demonstrables.

We foresee the following such demonstrables:

- 1-2-3 case study safety critical software demonstrators
- A compiler
- A Kernel (Run-time-system)
- A Simulator — for running the kernel, the compiler and the compiled safety critical case study software

7 Reading List

One way of formulating the project baseline is to list a very short set of publications and reports to be read by all project participants, whether contributed or hired. The below list thus exemplifies a *credo* on which the project starts.

1. Anders P. Ravn, Hans Rischel, Hans Henrik Løvengreen: *A Design Method for Embedded Software Systems;* BIT, vol. 28, pp. 427–438, 1988.
2. Ernst-Rüdiger Olderog: *Nets, Terms and Formulas;* Habilitationsschrift, Kiel Univ., 1988. [Selected passages.]
3. Edsger W. Dijkstra: *Guarded Commands, Nondeterminacy and Formal Derivation of Programs;* CACM, vol. 18, no. 8, pp. 453–457, August 1975.
4. C.A.R.Hoare et al.: *Laws of Programming;* CACM, vol. 8, no. 8, pp. 672–686, August 1987. [Corrigenda no. 9, p. 770.]
5. inmos ltd.: `occam-2` *Programming Manual;* Prentice-Hall International 1988.
6. G.D.Plotkin: *An Operational Semantics for CSP;* in: Formal Description of Programming Concepts II, Proceedings of TC-2 Work. Conf., Garmisch Partenkirschen, June 1982 (ed. D.Bjørner); pp. 199–225, North-Holland 1982.
7. Jonathan Bowen: *Formal Specification and Documentation of Microprocessor Instruction Sets;* in Schumny, H., and Mølgaard, J. (eds.): Microprocessing and Microprogramming, Vol. 21, pp. 223–230, North-Holland 1987.
8. Flemming Nielson & Hanne Riis Nielson: *Two-level Semantics and Code Generation;* Journal of Theoretical Computer Science, vol. 56, pp. 59–133, 1988. [Appendix may be skipped.]
9. Dines Bjørner: *Formal Development of Interpreters and Compilers;* in: Intl. Comp. Symp.; Proc. (ed. E.Morlet & D.Ribbens), Liege, Belgium; North-Holland, pp. 1–22, 1977.

10. William R. Bevier: *Kit: A Study in Operating System Verification;* Computational Logic Inc., 1988. (To be published in IEEE Transactions on Software Engineering.)

11. Nachum Derschowitz: *Computing with Rewrite Rules;* Information and Control, vol. 65, pp. 122–157, 1985.

12. P. A. Lindsay: *A Survey of Mechanical Support for Formal Reasoning;* Software Engineering, vol. 3, 1988.

8 Draft Project Reports

Although only 3 months underway, the project has already generated the following internal research and technical notes and reports:

References

[DAIMI KHP 1] Kim H. Pedersen: *Specification and verification of an* occam *compiler*, technical report, draft, 1989-07-14.

[ID/DTH AB 1] Andrzej Blikle: *Designing a Language of Concurrent Processes*, technical report, pre-draft, 1989-07-06.

[ID/DTH AR 1] Annie Rasmussen: *ProCoS Bibliography*, general information, draft.

[ID/DTH AR 2] Annie Rasmussen: *References* , general information, draft.

[ID/DTH BFH 1] Bettina Feldskov Hansen: *ProCoS Staff*, general information, draft, 1989-08-09.

[ID/DTH KMJ 1] Kirsten Mark Jensen: *Note on a small CSP like Language*, technical report, 1989-07-14.

[ID/DTH HHL 2] Hans Henrik Løvengreen: Note on the **ProCos** Programming Language, draft 1.1, 1989-07-14.

[ID/DTH HHL 3] Hans Henrik Løvengreen: Definition of the **ProCoS** Programming Language, Level 0, July 29, 1989, 6 pages.

[ID/DTH JNO 1] Jens Norddahl: *Gasbrænder - Eksempel, Specifikation og verifikation af tidsegneskaber*, notes, 1989-07-06. (Gasburner example: Specification and Verification of Real-time Properties)

[ID/DTH APR 1] A.P.Ravn and Jens Nordahl: *Safety Critical Embedded Software Systems*, July 1989, 14 note pages.

[ID/DTH APR 2] A.P.Ravn: *Further Notes relating to* [ID/DTH APR 1], 20 July 1989, 9 pages.

[ID/DTH ZCC 1] Zhou Chao-Chen: *A note on High Order Communication*, notes, draft, 1989-07-10.

[KIEL BB 1] Bettina Buth et al.: **ProCoS** *Report: Compiling the PrOccam0 Language*, technical report, preliminiary version, 1989-07-14.

[OLD ERO 1] Ernst Rüdiger Olderog: Specification Language Issues, CWI Amsterdam, July 1989, 34 overhead foil presentation pages.

[OX JB 1] Jonathan Bowen: *Z Specification of the μ transputer Instruction Set*, technical report, draft version 0.1, 1989-07.

[OX PP 1] Paritosh Pandya and Jonathan Bowen: *An Operational Semantics for the* **ProCoS** *Level 0 Assembly Language*, technical report, draft version 1.0, 1. August 1989, 12 pages.

[OX HJ 1] He Jifeng and C.A.R.Hoare: *Operational Semantics for* occam, 5. Jul 89, 32 pages

9 ProCoS Project Report Mailing List

The **ProCoS** project is eager to create a mailing list for those EACTS members who are interested in following the project. Please mail your exact mailing address to: Ms. Annie Rasmussen, ProCoS project administrator, Dept. of Computer Science, Bldg. 344, Technical University of Denmark, DK-2800 Lyngby, Denmark; e-mail: procos@iddth.dk.

10 Acknowledgements

Major parts of the present note is extracted from the **ProCoS** technical annex. I wish again to thank the co-authors of that document for an inspiring project initialisation phase. Their names were mentioned on the title page of this note. In particular I thank Tony Hoare for having first inspired the conception of the project.

Sections 3-4 are extracted from material mainly authored by Tony Hoare. The first identification of the diagram of figure 1 was made by Ernst Rüdiger Olderog[2]. Hans Henrik Løvengreen helped in producing the Technical Annex from which this note was culled.

[2]Now professor of theoretical computer science at Oldenburg Univ., FRG.