OCCAM-Oriented Software Tools

Algirdas Pakštas *, Danutė Paketūraitė,†Artūras Tamkevičius
Lithuanian Academy of Sciences
Institute of Mathematics and Informatics ‡

Abstract

For many years software engineers have been providing the engineers of other fields with advanced design tools. But the tools used by software designers themselves look quite primitive in comparison. Only CASE-like systems developed during last years can provide reasonable help to software developers.

Experimental software development systems ALADDIN/LAMP is oriented to creation of distributed computer control systems (DCCS). Proposed approach to development distributed software configurations (DSCs) is based on the model of *virtual distributed software configuration* (VDSC) with *information-transport ports* (ITPs) as interconnecting servers.

Transputer networks and OCCAM-written software are often used for creation of DCCS. An approach to ALADDIN/LAMP tools extention for OCCAM-written DSCs handling is discussed in this paper. Survey of tools is presented (OCCAM-oriented structure editor OSE, OCCAM structure extractor OSX, ALADDIN/OOB translator and deadlock locator and analyser DLA). A comparison with related works is given.

*This work was partially funded by NTNF. Dr.Algirdas Pakštas now with IDT NTH as NTNF post-doctoral research fellow. Current address: Division of Computer Systems and Telematics (IDT), The Norwegian Institute of Technology (NTH), The University of Trondheim, N-7034 Trondheim, Norway. Phone: +47 7 594485. Fax: +47 7 594466. E-mail: Algirdas.Pakstas@idt.unit.no.

†Danutė Paketūraitė now with Computing Center of the Bank of Lithuania, Vilnius

‡Detailed address: Dept. of Software Engineering for Distributed Computer Systems, Institute of Mathematics and Informatics, Akademijos 4, 2600 Vilnius, Lithuania. Phone +7 0122 359702, Fax +7 0122 359909, TELEX 261131 IMC SU, e-mail: {a.pakstas,arturas}@sedcs.mii2.lt

1 Introduction

For many years software engineers have been providing the engineers of other fields with advanced design tools that considerably facilitate their jobs, relieve them of tedious tasks, put them in control of the design process and help them to turn out quality products. But the tools used by software designers themselves look quite primitive in comparison. The typical design environment includes a text editor, a compiler, perhaps a debugger, but hardly anything that could in fairness be characterized as a design system. Only CASE-like systems developed during last years can provide reasonable help to software developers.

In order to increase productivity of program development, different approaches and related CASE-like tools have been developed. One of such approaches is *"building"* of *"distributed software configurations"* (DSCs) from already existing blocks [19].

Transputer networks and OCCAM-written software are often used for distributed computer control systems (DCCS) creation. According to the OCCAM-language semantics the processes can interact via channels by sending messages [3] what arised the deadlocks problem (see [29, 30]).

Concerning the operation of ALADDIN/LAMP system tools the interests in deadlock problems are the following:

- automated DSCs production (building) from the prepared components requires the guarantee of DSC correct operation;
- deadlocks can arise and should be eliminated in the debugging phase of DSC development.

The present paper considers an approach to the OCCAM-oriented extention of ALADDIN/LAMP software tools and deadlocks analysis in the transputer networks when DSCs are written in OCCAM.

2 Model and Related Tools

Proposed approach to development and analysis of OCCAM-programs is based on the model of virtual distributed software configuration (VDSC) with information-transport ports (ITPs) as interconnecting servers. The model was initially proposed for the distributed computer control systems development by means of the ALADDIN/LAMP system tools.

VDSC is a quadruplet

$$VDSC =< Blk_i, Rapr_i, ITP_j, CS >,$$

where Blk_i - program blocks, $Rapr_i$ - program blocks requirements for the hardware or software using, ITP_j - information-transport ports of the i-th program block, CS - communication scheme which defines the topological structure of the links between ITPs, $i = 1...NB$ (NB - the number of program blocks in DCCS) and $j = 0...NP$ (NP - the number of ports in DCCS).

The CS is presented by directed graph CS = (PORTS, LINKS).

$$PORTS = < ITP_i | i = 1...NP >,$$
$$LINKS = < (ITP_i, ITP_j) | ITP_i := ITP_j, i, j = 1...NP >,$$

where NP is total number of ITPs. An expression like "$ITP_i := ITP_j$" designates that ITP_i receives messages from ITP_j, i.e. that these ITPs are linked.

Such VDSC representation is directly mapped to ALADDIN language syntax and semantics. More details about this model are presented in [19]. For a practical VDSC use the fact that all links between ITPs are defined in CS is very important. Therefore, the VDSC description can be used as the data for the statical analysis of potential deadlocks.

An approach to ALADDIN/LAMP tools extention for OCCAM-written DSCs handling is shown in Fig. 1 where the following subsystems are shown:

- OSE (OCCAM-Oriented Structure Editor);
- OSX (OCCAM Structure eXtractor);
- translator of system architect's language ALADDIN/OOB;
- DLA (Deadlock Locator and Analyser).

The Example of Small DSC in OCCAM. The adopted approach will be illustrated with the DSC example (see Fig. 2 from [30]). The DSC has three processes that are executed in parallel: `Producer`, `Buffer` and `Consumer`. The process `Producer` produces the messages and sends them via the channel `prod` to the process `Buffer` up to filling buffer of this process, afterwards sends the message about the completed reception via the channel `endprod`.

The process `Buffer` has limited size buffer - till 10 one byte messages (see Fig. 3). The process `Consumer` receives these messages via the channel `cons` and sends to

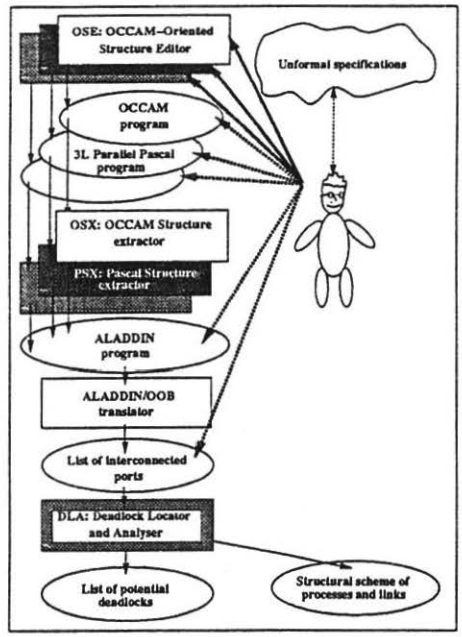

Figure 1: OCCAM-oriented extention of ALADDIN/LAMP tools

the screen via the channel `screen`. The requests for following messages are sent via the channel `more`. When all messages are passed, the process `Buffer` informs about this process `Consumer` via the channel `endcons`.

3 OCCAM-Oriented Structure Editor OSE

3.1 Advantages of using a structure editor

Advantages of using a structure editor are the following[14]:

- All documents produced are guaranted to be syntactically correct.
- As syntactic aspects are handled by the editor, users may concentrate on more interesting issues. A programmer, for example, has better things to do than keyword typing, worrying about proper indentation, etc.

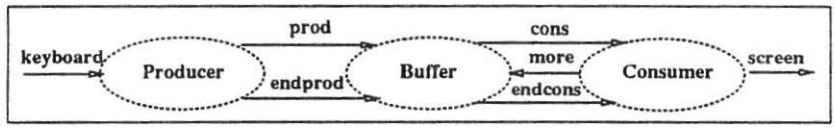

Figure 2: An example of distributed software configuration with a process `Buffer`, which has limited size

- Many language-dependent operations, which are hard or impossible to achieve with text editors, become possible.
- A flexible structural editor is a powerful tool for implementing programming or documentation standards.
- More generally, a structural editor brings all the benefits of a "smart" tool that knows about the structure of the document it manipulates.
- When the languages considered are software languages such as programming or design languages, structural editor simplifies the task of writing other software tools. Most software tools acting on programs, designs, specifications, and the like must at some point perform some parsing to get the input documents into a suitable internal form. If a structural editor is used, this task is no longer necessary; the structural editor has its own internal form, usually some kind of tree structure, that can be used by other tools.

More detailed review of structure-oriented concepts is presented in [24].

3.2 Implementation and user interface

The editor runs on IBM PC/XT/AT under MS-DOS. The editor was implemented in Turbo Pascal on the top of Turbo Professional package text editor, which operates similar to Borland International Turbo Pascal editor [4].

User interface was developed on the basis of Borland's windows and menus libraries. General work style with OSE is very similar to Borland's Turbo compilers (the same names of menus and ALT-keys for the same functions). Editor OSE can operate in two main modes: *text mode*, *structure-oriented mode*.

In the text mode the editor operates as conventional full screen text editor. Commands set is very similar to Turbo Pascal's editor commands set.

```
PROC BoundedBuffer
     (CHAN OF BYTE producer, consumer,
      CHAN OF ANY endprod, endcons, more)
[10]BYTE buffer  :
INT in, out      :
BOOL termination :
BYTE any         :
SEQ
  in := 0
  out := 0
  termination := FALSE
  WHILE NOT (termination)
    SEQ
      ALT
        (in<(out+10))&producer?buffer[in REM 10]
          in := in + 1
        (out < in) & more ? any
          SEQ
            consumer ! buffer [out REM 10]
            out := out + 1
        (out = in) & endprod ? any
          SEQ
            more ? any
            endcons ! any
            termination := TRUE
:
```

Figure 3: An OCCAM-program fragment with limited size `Buffer` process

In the syntax-oriented mode the commands set is very similar, but these commands operate a little bit different, because in this mode the editor works not with characters but with OCCAM-2 syntax constructs. The main editor task is to support correct program structure until editing. In this mode programmer is restricted in freedom. He/she can't use commands that break correct program structure. Program is developed by selecting an appropriate template. The programmer can make choices from a menu or use ALT-keys.

4 OSX and ALADDIN/OOB translator

OCCAM Structure eXtractor OSX has the following functions:

1. *Analysis of the OCCAM-programs, extraction of names of interconnected processes and used channels.* The OCCAM syntax analyser for OSX is under implementation by means of grammar-based technique as described in [20, 25]. It should be noted that only this analysis and extraction functions of OSX are language-dependent. Other two functions are universal.

2. *ITPs interconnection scheme reconstruction.* According to virtual DSC model each channel in the OCCAM-program is associated with two information-transport ports that belong to the different processes. In the used example for the processes `Buffer` and `Consumer` the channel `cons` is mapped into the 2 ports: output ITP `Buffer.cons` and input ITP `Consumer.cons`.

3. *ALADDIN-written VDSC description preparation* (example is shown in Fig. 4). Neither information about distributed hardware configuration nor assignments of program blocks on workstations are presented here.

It's easy to see that LINK-operators contain all information which is necessary for potential wait-for-graph constructing. For the more deep analysis and OCCAM-program simulation all necessary information presented in OCCAM-programs leaves in their ALADDIN-descriptions. For example, the `Buffer.prod` (input ITP) has a limited queue length which is equal to the buffer array size as defined in the OCCAM-program (Fig. 3).

ALADDIN/OOB translator task is to build DSC using already existing components and to prepare the data for DLA (a list of interconnected ports according to LINK-operators, see example in Fig. 5). This list also can be prepared manually according to available non-standard specifications.

5 Deadlock Locator and Analyser DLA

One of the most important problem, which arised during the creation of distributed software configurations (DSCs), is deadlocks between interacting processes[9, 32]. Deadlocks were and still should be the subject of research, because their influence on normal computer systems behaviour is obvious.

5.1 Deadlock detection problem

Deadlock detection has been studied extensively in the context of concurrent operation of monoprocessor time-sharing systems. Deadlock can occur in such systems due

to cyclic patterns of requests for exclusive access to system resources. Algorithms for deadlock prevention, detection and recovery are standard material in operating systems textbooks (e.g. [11]).

Such algorithms have also been extended to distributed systems, where concurrent processes may execute on different processors. As was discovered in [15, 17, 18] during DSC debugging and maintenance in the DCCS-oriented software development system ALADDIN/LAMP (see [16, 19]) the problem of deadlock elimination was also faced.

In this work we'll use the folowing deadlock definition which is formed on the basis of [16]: *"The deadlock arises when all processes (two or more) of the same group sending the messages are infinitely blocked and can not be executed without special system interactions because of waiting messages from the processes of this group".*

So, according to [10] this type of deadlock may be called a *communication* deadlock where a process waits to communicate with *any one* from a set of neighbors in comparison with *resource* deadlock, which assumes that a process becomes unblocked only after it receives *all* the resources for which it is waiting.

At present there are a number of methods for prevention, avoidance and detection-repairing of the deadlocks (see [9, 32, 10]). Very often deadlock detection methods are based on distributed maintenance of the *"wait-for-graph"*, that shows which process is waiting for a resource held by which other process [13, 28, 5, 26, 27]. Deadlock detection is reduced to finding cycles in this graph.

Programming language OCCAM was created for development of DSCs running on transputer networks. According to the OCCAM-language semantics presented in [3] the processes can interact via channels by message sending. And naturally deadlock detection problem was also faced (see [29, 30, 31]).

Although the approach of maintaining a wait-for-graph and searching for cycles in it is estimated as problematic for development of run-time monitors for parallel programming environments [6] this approach is still acceptable for static analysis of parallel programs. The similar method for static deadlock analysis was adopted in the ALADDIN/LAMP system tools [21, 22, 23].

5.2 OCCAM Language and Deadlocks Features

An OCCAM program comprises the collection of processes that can be executed in a serial or parallel fashion [3]. OCCAM defines the basic processes to be assignement process, input process and output process.

Input and output processes operate via OCCAM channels and provide the synchronised interprocess communication between concurrent processes. The channel is used to pass data from one concurrent process to another.

According to the proposed definition, the deadlock arises as a result of infinite waiting for the messages. In OCCAM-programs we have 3 typical cases:

1. **Incorrect interprocess protocol.** Process A completed the data passing to the process B, but didn't inform about it, and as result the process B waits for messages and blocks the other processes that are interconnected with him. This case is typical not only for transputer networks, but also for other types of distributed systems.

2. **Incorrect process-coordinator implementation.** If the "process-coordinator" hasn't the possibility to accept more than one message from coordinated processes, then the other messages will be lost. The processes, that sent "lost" messages, will wait infinitely for confirmation and will block other processes. This case is typical for OCCAM-language, when variable value is assigned from several channels in parallel.

3. **Incorrect external environment ("human inspired deadlocks").** This special kind of deadlocks arises as result of deadlock cycle closing via external non-machinery environment, i.e. when humans are also active processes in distributed system (terminal I/O, manual switching, etc.).

According to this classification in the example presented above (Fig. 2) cases 1 and 3 may occur. Case 2 wasn't presented.

5.3 Implementation

DLA [21] uses recursive algorithm for the search of cycles in potential wait-for-graph. This graph is constructed of the two parts: external (according to interconnection scheme CS described in ALADDIN-program by LINK-operators) and internal (derived from the same original information taking into account semantics of process-to-ITP calls, i.e. "internal links").

DLA outputs potential deadlocks list (see example in Fig. 6) and structural scheme of processes interconnection (pseudographical, see example in Fig. 7). In the Fig. 7 program blocks are shown as rectangles, output and input ITPs are marked by symbols ">" and "<". The lines like "..." mean the "internal port connections" that DLA adds to the interconnections list according to the principle "all inputs

with all outputs". The lines "---" mean external links between ITPs that belong to different blocks.

Human inspired deadlocks can be analysed according to corresponding ITPs parameters `external-in` and `external-out` (see Fig. 4). This possibility is optional and should be defined before running DLA.

6 Related works

In this section we'll present a survey of systems, related to development of OCCAM-programs. The typical environments for OCCAM programs development are Inmos Transputer Development System TDS (oriented to IBM PC-based computers with connected transputers) and Inmos OCCAM Programming System OPS (oriented to IBM PC-based computers without transputers) [8, 7]. However, alternative OCCAM-environments and supporting tools also are created [12, 2, 1].

"Folding editor" concept. In the Inmos TDS and Inmos OPS are included "folding editors". A "folding editor" extends the principle of tree structured directories into a text file. This allows the simultaneous display of large text amounts, by "folding" text sections away behind a descriptive heading. This results in a tree structure very similar to the sub-directory structure of, for example, MS-DOS.

With suitable text structuring it should be possible in most circumstances to ensure that no display exceeds a single screen at any one time. To access text which is folded away you can either ENTER the containing fold, in which case the contents of the fold and its header are the only displayed text, or you can OPEN the fold, in which case the contents are displayed in the context of the surrounding text.

The advantage of this system is that it eliminates the need for seemingly endless paging through long files to find the section of interest, allowing you to move down the tree structure, following the (hopefully) descriptive headers, to locate the text you require. "Folding editors" combine the syntactic elision and holophrasting concepts for program display.

Origami. System Origami [2] is a "folding editor", similar to, and inspired by, the editor included in the Inmos TDS. Origami is not a word processor (although many of its features would be useful in a word processor), but it comes into its own as a program editor for structured languages. The code structuring is obvious from the screen display, although no actual code may be visible (i.e. "full elision").

Existing files, not produced using Origami, may be imported into Origami, and then folded up for future use, and ease of development. These files can be further edited either using Origami or some other editor, and compiled in the normal way.

CODE. In the [1] is presented system CODE as a user-friendly (pull-down menus for user interface) PC-based learning tool for parallel programming. CODE helps users learn OCCAM-2 by writing programs for a single transputer, checking for errors and debugging during execution.

Components of CODE are: **Folding Editor** compatible with the Inmos TDS folding editor for generating Inmos TDS compatible source files; **Syntax Checker** for checking syntax errors in OCCAM-2, reporting if the errors encountered and generating the object file for the **Interpreter**; **Interpreter and Debugger** for program executing in *debug* and *non-debug* modes, deadlock detecting and process monitoring, simulating of computing and communicating times for any process; **Lister** for preparing of program source code listings with or without line numbers, opening all nested folds in program.

Main emphasis of this system is learning of parallel programming in OCCAM. At the moment system CODE is only related work covering deadlock detection in OCCAM-programs which is known by the authors.

7 Conclusions

The problem of software development for parallel systems and distributed computer control systems was discussed in the paper in respect to creation of distributed software configurations. The model of virtual DSC was used as a basis for proposed "building" approach. The Deadlock Locator and Analyser DLA and OCCAM Structure Extractor OSX were offered as helpful tools for reliable OCCAM-based DSC development, because they provide faster DSC checking and support the visualization of program structure. OCCAM language features that are the most important for deadlocks analysis were shown in the paper.

The OCCAM-oriented structure editor OSE was presented as helpful tool for program writing in OCCAM-2, because it provides faster program development and supports program structure. The editor can be used for learning aims too.

DLA and OSX implementations for IBM PC under MS-DOS were developed as components of integrated OCCAM programming environment including structure editor, compiler, debugger, deadlock analyser and process configurator. DLA can

be used not only for OCCAM but for other parallel and object-oriented languages too. All what is necessary - to develop corresponding structure extractor for the new target language (PSX for Pascal, etc., see Fig. 1). DLA and OSX can be useful also as components of reverse engineering technology.

References

[1] CODE - user-friendly PC-based learning tool for parallel programming - the first of its kind in the world! C-DAC - Centre for Development of Advanced Computing, Pune University Campus, Ganesh Khind, Pune 411 007, INDIA, 1989.

[2] *Origami User Guide*, 1990. 9p.

[3] G. Barett. *OCCAM-3 Reference Manual.* INMOS Limited, Mar. 1992. 190p.

[4] Borland International. *Turbo Pascal. User's Guide. Version 5.0*, 1988. XIII+350p.

[5] A. Elmagarmid and A. Datta. Two-phase deadlock detection algorithm. *IEEE Trans. Comput.*, 37:1454–1458, 1988.

[6] D. G. Feitelson. Deadlock detection without wait-for-graphs. *Parallel Computing*, 17:1377–1383, 1991.

[7] INMOS Limited. *TDS 2.0. TDS tools. Beta release documentation*, July 1986.

[8] INMOS Limited. *TDS 2.0. User manual. Beta release documentation*, June 1986.

[9] S. Isloor and T. Marsland. The deadlock problem: An overview. *Computer*, 13(9):58–78, 1980.

[10] E. Knapp. Deadlock detection in distributed databases. *ACM Computing Surveys*, 19(4):303–328, Dec. 1987.

[11] S. Krakowiak. *Principles of Operating Systems.* MIT Press, Cambridge, MA, 1988.

[12] D. Macfarlane, M. Webb-Johnson, and J. Galletly. PC-OCCAM. *Journal of Microcomputer Applications*, (12):191–212, 1989.

[13] D. Menasce and R. Muntz. Locking and deadlock detection in distributed data base. *IEEE Trans. on Software Eng.*, SE-5(3):195–202, 1979.

[14] B. Meyer. Cepage: Toward computer-aided design of software. *The Journal of Systems and Software*, (8):419–429, 1988.

[15] A. Pakštas. Programming support of reserving and handling deadlock and other exceptional situations in distributed systems. In *FTSD-10, Proc. 10th Int. Conf. on Fault-Tolerant Systems and Diagnostics, Varna, Bulgaria*, pages 142–147, Sept. 1987.

[16] A. Pakštas. *Distributed Software Configurations: Analysis and Development.* Mokslas, Vilnius, 1989. 223p., (in Russian).

[17] A. Pakštas. Exceptional situations mechanism for interaction environment. In *FTSD-12: Proc. 12th Inter. Conf. on Fault-Tolerant Systems and Diagnostics, Prague, Czechoslovakia*, page 357, Sept. 1989.

[18] A. Pakštas. Methods and algorithms of distributed deadlock detection for DCCS on-line diagnostic subsystem. In *Technical Diagnostics'90: Proc. 7th IMECO TC 10 Symp. on Technical Diagnostics, Helsinki, Finland*, Sept. 1990.

[19] A. Pakštas. Architecture, organization and building of distributed software configurations for the microcomputer control network. In U. Jaakso and V. Utkin, editors, *Automatic Control. World congress 1990. "In the Service of Mankind". Proceedings of the 11th Triennial World Congress of the International Federation of Automatic Control, August 1990, Tallinn, Estonia*, pages 123–128, Oxford, UK, 1991. Pergamon Press. Vol.4.

[20] A. Pakštas, R. Meidutė, and G. Stradalov. A software system for parser constructing with full-screen debugging facilities. In *System Sciences X, Inter. Conf. on Systems Sciences*, page 145, Wroclaw, Poland, Sept. 1989. (Abstr. of Papers).

[21] A. Pakštas and D. Paketūraitė. DLA: Locator and analyser of deadlocks in distributed software configurations. In *FTSD-13, Proc. 13th International Conference on Fault-Tolerant Systems and Diagnostics, Varna, Bulgaria*, pages 171–176, June 1989.

[22] A. Pakštas, D. Paketūraitė, and S. Pakštienė. Tools for Analysis and Simulation of Distributed Computer Control System in Object-Oriented Software Development Environment ALADDIN/OOB. In *9th IEEE Workshop on Real-time Operating Systems and Software, 13–14 May, 1992, Pittsburg, PA, USA*, May 1992.

[23] A. Pakštas, D. Paketūraitė, and A. Tamkevičius. OCCAM-Oriented Extention of ALADDIN/LAMP Software Tools. In *TAPA-92: Proc. Transputer and Parallel Applications Conf., Nov. 4–5, Melbourne, Australia*, Nov. 1992.

[24] A. Pakštas and A. Tamkevičius. OSE: OCCAM-Oriented Structure Editor (preliminary version). In *Proc. 3rd Nordic Workshop on Programming Environment Research, Tampere, Finland*, Jan. 1992.

[25] A. Pakštas and N. Zolotariov. *Syntax-oriented Components of Distributed Systems: Development and Debugging Tools on the Basis of Formal Descriptions.* Nauka, Moscow, 1991. 280p. (in Russian).

[26] M. Roesler and W. Burkhard. Resolution of deadlocks in object-oriented distributed systems. *IEEE Trans. Comput.*, 38:1212–1224, 1989.

[27] M. Singhal. Deadlock detection in distributed systems. *Computer*, 22(11):37–48, Nov. 1989.

[28] K. Sugihara, T. Kikuno, and N. Yoshida. Deadlock detection and recovery in distributed database systems. *Systems, Computers, Controls. Scripta Electronica Japonica*, 15(1):48–56, 1984.

[29] M. Surridge. A topology independent, minimal memory, deadlock-free, general message passing harness. EMS Report: Software Migration Aids for Transputer Systems (Contract Extension), Dept. of Electronics and Computer Science, University of Southampton, 1990. The SERC/DTI Initiative in the Engineering Applications of Transputers. P15.

[30] D. Talia. Notes on termination of Occam processes. *Sigplan Notices*, 25(9):17–24, 1990.

[31] L. Waring. A general purpose communications shell for a network of transputers. *Microprocessing and Microprogramming*, 29:107–119, 1990.

[32] D. Zöbel. The deadlock problem: A classifying bibliography. *SIGOPS Operating Systems Rev.*, 17(4):6–15, 1983.

```
-- software components
BLOCK Producer;    -- BLOCK description
                   -- PORT descriptions
  PORT keyboard (EXTERNAL_IN; ... );
                                END PORT;
  PORT prod (OUT; FORMAT = BYTE;
       SIZE = 1; QUEUE = 1);  END PORT;
  PORT endprod (OUT; ... );   END PORT;
END BLOCK;
BLOCK Buffer;
  PORT prod (IN; FORMAT = BYTE;
       SIZE = 1; QUEUE = 10); END PORT;
  PORT endprod (IN; ... );    END PORT;
  PORT cons (OUT; ... );      END PORT;
  PORT more (IN; ... );       END PORT;
  PORT endcons (OUT; ... );   END PORT;
END BLOCK;
BLOCK Consumer;
  PORT cons (IN; FORMAT = BYTE;
       SIZE = 1; QUEUE = 1);  END PORT;
  PORT more (OUT; ... );      END PORT;
  PORT endcons (IN; ... );    END PORT;
  PORT screen (EXTERNAL_OUT; ... );
                                END PORT;
END BLOCK;
-- scheme of interconnections
LINK (Producer.prod => Buffer.prod)
LINK (Producer.endprod => Buffer.endprod)
LINK (Buffer.cons => Consumer.cons)
LINK (Buffer.endcons => Consumer.endcons)
LINK (Consumer.more => Buffer.more)
```

Figure 4: A fragment of ALADDIN-program corresponding to DSC with limited size Buffer process

```
  # PORTS = 12,      # EXTERNAL LINKS = 5
  OUTPUT:            INPUT:
  Producer.prod      Buffer.prod
  Producer.endprod   Buffer.endprod
  Buffer.cons        Consumer.cons
  Buffer.endcons     Consumer.endcons
  Consumer.more      Buffer.more
```

Figure 5: The example of DLA input data file format

```
# PORTS = 12, # ALL LINKS = 17

LIST OF POTENTIAL DEADLOCKS:
1. Consumer.more Buffer.more Buffer.cons Consumer.cons Consumer.more
2. Buffer.endcons Consumer.endcons Consumer.more Buffer.more Buffer.endcons
```

Figure 6: The list of potential deadlocks

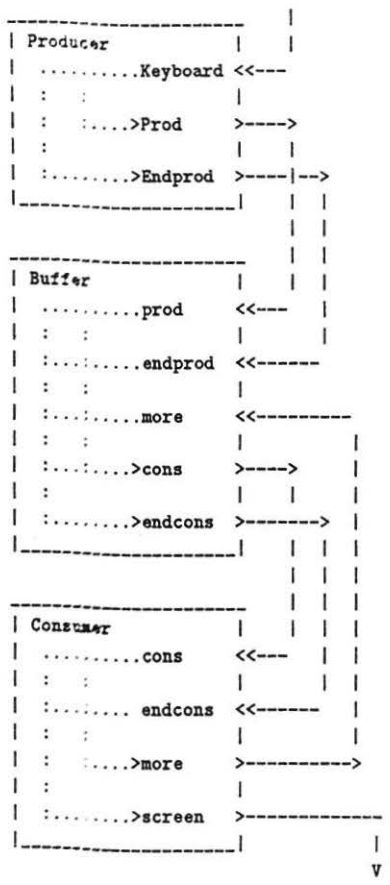

Figure 7: Structural scheme of DCS processes and their interconnections