PRECOMPILED SUBMODELS: A GENERAL SORTING PROCEDURE

Guasch A., Huber R.
Institut de Cibernética (UPC/CSIC)
Barcelona, Spain

ABSTRACT

Simulation languages supporting hierarchical submodel structures, face sorting problems not present in monolithic CSSL-67 models. These problems, result from algebraic relationships between submodels leading to information loops (1) in the model code.

This paper proposes a global solution, based on the segmentation of the submodels through graph analysis, and its promising application to sorting problems.

INTRODUCTION

The separate compilation of declarative submodels is still not widely accepted to be included in a desirable new Continuous System Simulation Language (CSSL) specification (TC3-IMACS Newsletter No. 12).

The main point against its inclusion is the problem of sorting the code relatively to other submodels. Using the classical methods, information loops (those that disappear if the called lower level submodels are handled like macros) appear in the calling submodel

The authors contribution is an attempt to solve the sorting problem by means of a segmentation procedure. The report, first introduces elementary graph concepts; second, the submodel digraph is defined; third, a general segmentation procedure is described; fourth, a classical sorting procedure is introduced to sort the sentences within each segment; and finally, a few related points are analyzed: side effects, submodel object code, and an example.

ELEMENTARY GRAPH CONCEPTS

The CSSL sorting problem can be solved through **directed graph (digraph)** sorting procedures. A digraph is a collection of **vertices** V={v1,v2,...}, a collection of edges E={e1,e2,...} and a mapping Ψ that maps every edge onto some ordered pair of vertices (vi,vj). Vertices are simple objects and an edge is a connection between vi and vj with an arrow directed from vi to vj.

A **path** from vertex vi to vj in a graph is a list of vertices in which succesive vertices are connected by edges in the graph. A path is directed -**directed path**- if the edges have the same orientation, otherwise, it is a **semipath**. A graph is **connected** if there is a path from every vertex to every other vertex in the graph. A graph which is not connected is made up of **connected components**. A connected graph is **strongly connected** if there is at least one directed path from every vertex to other vertex, otherwise, it is **weakly connected**. A **simple directed path** is a directed path in which no vertex is repeated and a **directed cycle** is a directed path which is simple except that the first and last vertex are the same.

A vertex vj is **reachable** from a vertex vi if there is a directed path from vi to vj.

Graphs with all edges present are called **complete** graphs; graphs with relatively few edges are called **sparse**; graphs with relatively few of the possible edges missing are called **dense**.

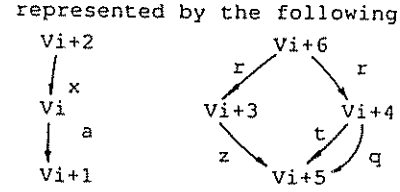
In order to be segmented and sorted, graphs have to be directed and acyclic. These graphs are called **directed acyclic graphs (dags)**.

In our CSSL problem the sentences are the vertices and the edges are the relationships between sentences due to its input and output variables -**explicit edges**-. The implicit relationships between segments constitute the **implicit edges or hidden edges** which will be defined later.

The following code (*), where k1, k2 and k3 are constants, y is a state variable (**), and a, x, y, z, r, r, q and z are real variables,

```
                    .....................
Sentence i  (Vi  )  a    := Fun(x,y)   ;
            (Vi+1)  y'   := a*k3       ;
            (Vi+2)  x    := ...        ;
            (Vi+3)  z    := r*k1       ;
            (Vi+4)  t,q  := Sub(r,k2)  ;
            (Vi+5)  v    := z+t*q      ;
             Vi+6)  r    := Time       ;
                    .....................
```

will be represented by the following digraph:



*) The syntax used for coding examples is unformal.
**) Integration output variables.

The state variables of the system do not appear in this graph because their values are known in advance.

The graphs associated with CSSL models (CSSL-graphs) are:

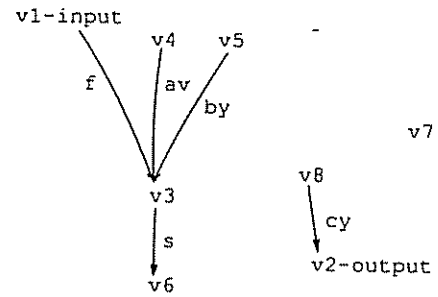
- Sparse: Graph with relatively few edges (< V log V).
- Directed: The edges are one-way.
- Acyclic: Algebraic implicit loops leading to directed cycles may appear in incorrect models. They have to be eliminated beforehand.
- In general, they are not connected.

SUBMODEL DIGRAPH

Let be a submodel coded in a CSSL language,

```
                        /*   v1          v2 */
SUBMODEL example(in:REAL f; out:REAL cy;)IS
   CONSTANT
      REAL a,b,c;
   END CONSTANT;
   STATE v,y; REAL av,by;
   DYNAMIC
/*v3*/  s  := f+av+by;
/*v4*/  av := -a*v;
/*v5*/  by := -b*y;
/*v6*/  v' := s;
/*v7*/  y' := v;
/*v8*/  cy := c*y;
   END DYNAMIC;
    END example;
```

Its associated digraph Gm=(Vm,Em) will be:



where:

- Vm = (Im,Om,Tm) is the set of vertices,

  Vm = {v1,v2,v3,v4,v5,v6,v7,v8}

  being:

  Im: set of input vertices. Each formal parameter in the input list becomes an input vertex.

  Im = {v1}

  Om: set of output vertices. Each formal parameter in the output list becomes an output vertex.

  Om = {v2}

  Tm: set of executable vertices.

  Im = {v3,v4,v5,v6,v7,v8}

- Em is the set of directed edges.

  Em = {(v1,v3),(v4,v3),(v5,v3),....}

Given a vertex vi belonging to Vm, R(vi) is the set of vertices reachable from the vertex vi,

i. e.

R(v1) = {v1,v3,v6}

the reaching set Q(vi) is the set of vertices from which vertex vi can be reached

i. e.

Q(v6) = {v6,v1,v4,v5,v3}

p(vi) is the set of predecessors of vi

i. e.

p(v6) = {v3}

and s(vi) is the set of successors from vi

i. e.

s(v3) = {v6}

SEGMENTATION

Some Definitions

- The output weight of a vertex vi of the submodel digraph is the number of output vertices reachable from vertex vi.

  Ow(vi) = |R(vi) ∩ Om|

- The input weight of a vertex vi of the submodel digraph is the number of input vertices which can reach vi.

  Iw(vi) = |Q(vi) ∩ Im|

- A pair of vertices vi and vj are said to be fused, if both are replaced by a single new vertex such that every vertex that is incident on vi or vj or both is now incident on the new vertex (2).

Once the executable vertices have been clustered (fused) using the rules explained later in order to get a reduced digraph (Gr), a segment S=(V,E) will be a subdigraph of the reduced digraph.

According to the nature of its vertices, three types of segments are defined:

- State segment (Ss=(Vs,Es)): Its executable vertices only depend on state variables or constants.

- Derivative segment (Sd=(Vd,Ed)): Its executable vertices only contribute to derivative computations.

- Algebraic segments (Sa=(Va,Ea)): Its executable vertices belong to input/output vertices directed paths.

Fusion Rules

The following fusion steps are proposed to get the the reduced digraph from the submodel digraph.

- Step 1:

  Fuse the weakly connected executable vertices that have the same output weight.

- Step 2:

  Afer completing the previous step, fuse the weakly connected executable vertices that have the same input weight.

- Step 3:

  Fuse the executable vertices having zero input weight.

  Fuse the executable vertices having zero output weight.

  The executable vertices having input and output weights equal to zero are fused with any executable vertex.

Segments

Each executable vertex ti of the reduced digraph has an associated segment, its vertices being,

$$Ti = ti$$

$$Ii = p(ti) \cap Im$$

$$Oi = s(ti) \cap Om$$

- If ts is an executable vertex of the reduced digraph with input weight equal to zero, it has an associated state segment (Ss=(Vs,Es)), its vertices being,

$$Ts = ts$$
$$Is = p(ts) \cap Im \quad \text{(empty)}$$
$$Os = s(ts) \cap Om$$

- If td is an executable vertex of the reduced digraph with output weight equal to zero, it has an associated derivative segment (Sd=(Vd,Ed)), its vertices being,

$$Id = td$$
$$Id = p(td) \cap Im$$
$$Od = s(td) \cap Om \quad \text{(empty)}$$

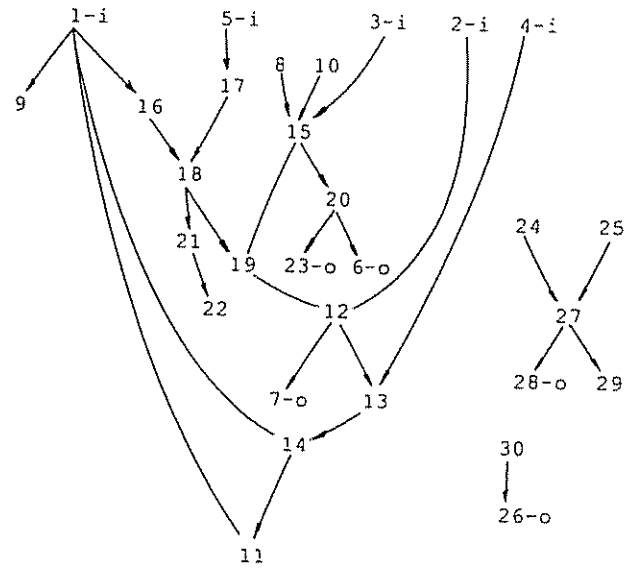
- All the other executable vertices tai have associated algebraic segments because they belong, at least, to a directed path from an input vertex to an output vertex

Each vertex tai has an associated algebraic segment (Sai=(Vai,Eai)), its vertices being,

$$Tai = tai$$
$$Iai = p(tai) \cap Im$$
$$Oai = s(tai) \cap Om$$

Example

Let be the submodel digraph shown in the figure



where

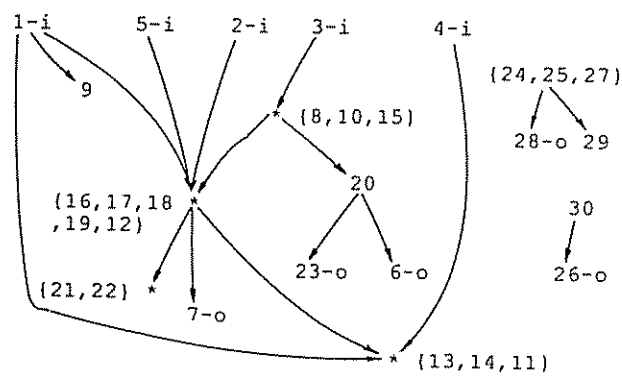
Im={1,2,3,4,5}

Om={6,7,23,28,26}

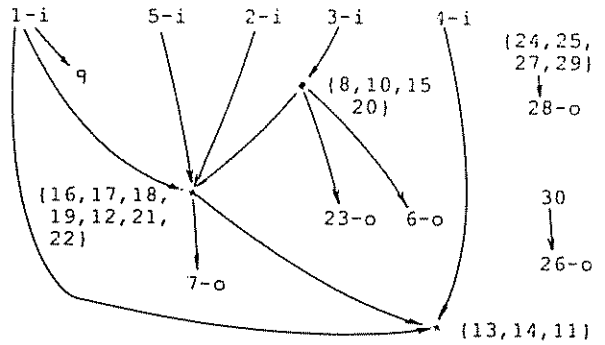
Im={8,9,10,11,12,13,14,15,16,17,
18,19,20,21,22,24,25,27,29,30}

Fusion: The fusion steps are applied to cluster executable vertices with equivalent sorting properties:

- Step 1: Fuse weakly connected executable vertices that have the same output weight.

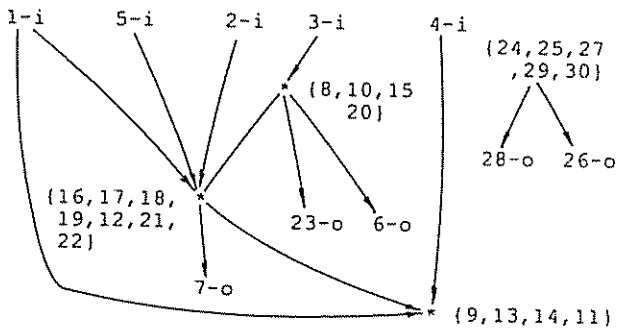


- Step 2: Fuse weakly connected executable vertices that have the same input weight.



- Step 3: Fuse the executable vertices having zero input weight and fuse the executable vertices with zero output weight, even if they are not weakly connected.

The reduced digraph is:



Resulting Segments: The segments will be:

- Vertices of the state segment Vs=(Is,Os,Ts)

  Ts = {24,25,27,29,30}
  Is = p(Ts) Im = Empty
  Os = s(Ts) Om = {28,26}

- Vertices of the derivative segment Vd=(Id,Od,Td)

  Td = {9,13,14,11}
  Id = p(Td) Im = {1,4}
  Od = s(Td) Om = Empty

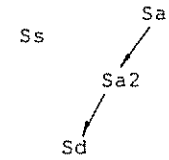
- Vertices of the algebraic segment Val=(Ial,Oal,Tal)

  Tal = {8,10,15,20}
  Ial = p(Tal) Im = {3}
  Oal = s(Tal) Om = {23,6}

- Vertices of the algebraic segment Va2=(Ia2,Oa2,Ta2)

  Ta2 = {16,17,18,19,12,21,22}
  Ia2 = p(Ta2) Im = {1,5,2}
  Oa2 = s(Ta2) Om = {7}

A segment digraph: Is defined to show the sorting relationship between segments.



The call to the previous submodel from a higher level submodel will be split into four calls, one for each segment.

The edges of the segment digraph are called hidden edges because they can not be inferred from the higher level submodel through the input-output variables in the segment calls.

Furthermore, to properly sort both submodels (high level submodel and low level submodel) as a whole, the knowledge of the segment digraph is required. This knowledge about submodels interfacing can be stored in a Submodel Information Library (SIL).

SUBMODEL BLOCKS: A CRITIQUE FROM THE SEGMENTATION POINT OF VIEW

Although combined system simulation languages can be used to describe systems composed by continuous and discrete subsystems, at present we will restrict ourselves to piecewise continuous system simulation languages.

A first criticism of the blocks used in CSSL67 languages from the segmentation point of view is shown in the next points. The authors feeling is that a deeper analysis is necessary before attempting the definition of the blocks in a new CSSL specification if the sorting of precompiled submodels is wanted.

Initial Region

"The Initial region -subregion- is procedural code that performs initialization calculations,input,..." (3).

If we accept the above CSSL67 definition the initial region has to be managed as a single vertex -initial vertex-, nevertheless, information loops may appear if the INPUT vertices to the initial vertex are not constants.

If the initial region is to be preserved, some modifications or restrictions have to be introduced in its definition in order to avoid information loops.

At first glance, two approaches are possible:

1. To preserve the initial region definition but restrict the INPUT vertices to it to constants.

2. To define the initial region as non procedural code. The sentences of the

initial region will be placed in different segments according to the stated segmentation steps.

The proposed segmentation method allows the initialitation of models in a single pass.

Terminal Region

Computations done within the terminal region are always part of the experiment.

Besides, the existance of a terminal region in a submodel increases the difficulty of segmentation.

The terminal region is likely to disappear in a model or submodel structure definition.

Procedural Pseudoblock

The procedural pseudoblock has to be viewed as a sigle vertex in the submodel digraph. This vertex will be connected to the other vertices through edges defined by the formal parameters of the input/output procedural list.

MACRO Pseudoblock

The MACRO facility is not needed anymore since the proposed segmentation method avoids the necessity of spreading the submodel sentences throughout the calling submodel in order to sort it (4) (5).

NOSORT Pseudoblock

The NOSORT pseudoblock is a potential source of errors for non expert users.

This type of procedural block is not compatible with the proposed segmentation strategy.

Sample Block

The authors conceive the SAMPLE block as a submodel to be called within the dynamic region of a higher level submodel (6) rather than a block that shares the dynamic region with the derivative segments (7) (8).

In that case, calling a SAMPLE submodel will be viewed as a single vertex.

SUBMODEL SORTING

Detection of directed cycles (algebraic implicit loops in the model) and sorting is done before the segmentation of a submodel digraph.

The following steps are taken to sort a submodel:

1. Replace the calls to lower-level submodels by calls to its segments.
2. Create the submodel digraph of the submodel being sorted.
3. Apply the modified depth-first search algorithm (9) to detect directed cycles and sort the digraph.

At the end of the segmentation which will follow, the sentences within each segment have to maintain the precedence in the sorted submodel digraph.

SIDE EFFECTS

If a submodel is to be called more than once within a model the generated code has to be reentrant. In this case, the consideration below has to be taken into account.

The code in the segments of a submodel holds regular sentences and segment calls to lower level submodels.

Therefore, in general, the hidden edges in the segment digraph associated to the higher level submodel can be divided into two groups:

- Those coming from the regular sentences.
- Those coming from hidden edges in the segment digraph of the lower level submodels.

Dynamic memory is needed only for the variables associated to the edges coming from the regular sentences in order to get the submodel reentrancy.

SUBMODEL OBJECT CODE

FORTRAN 77 has been selected as the preprocessor target language looking for the compatibility with the FORTRAN 77 CSSL67 (3) simulation language developed at the Institut de Cibernetica.

Two ways of coding a submodel are possible:

- To use a subprogram with different entry points to the segments.
- To use a subprogram for each segment with a local COMMON in order to share the variables with global scope within the submodel.

The first solution seams to be neater and easier to implement.

EXAMPLE

The example proposed by Prof. Rimvall (10) is very suitable to support the outlined segmentation procedure.

It consists of two coupled masses moving on a frictionless wall (fig. 1). The left mass is connected with a spring and a damping to a wall and the second mass is externally forced.

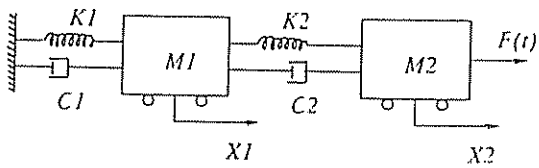


fig. 1: Coupled masses system.

the equations for the system are:

$m1\ x1''+k1\ x1+c1\ x1'-k2(x2-x1)-c2(x2'-x1')=0$

$m2\ x2''+k2(x2-x1)+c2(x2'-x1')=\sin(wt)$

and the code in a CSSL language may be:

```
v1'     := (fl1+fr1)/M1;
x1'     := v1;
fl1     := -K1*x1-C1*v1;
fr1     := -fl2

v2'     := (fl2+fr2)/M2;
x2'     := v2;
fl2     := -K2*x1-C2(v2-v1);
fr2     := sine(OMEGA,PHASE);
```

The above mass system submodel can be extended to increase its generality (fig. 2).

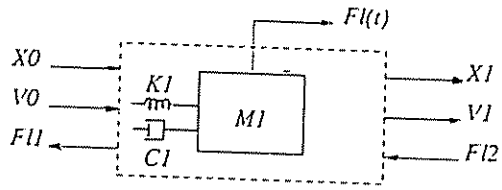


fig. 2: Generalized mass subsystem

The code associated to the generalized mass subsystem is

```
/*------------------------------ mass_system
*/
                          /*   v0,v1,v2, v3*/
SUBMODEL mass_system(in:REAL x0,v0,fl2,f1;
                     /*    v4,v5        ,v6 */
                 out:SIATE x1,v1; REAL fl1;)IS
   REAL fr1;
   CONSTANT
      REAL M1,K1,C1;
   END CONSTANT;
   DYNAMIC
/*v7 */    v1' := (fr1+fl1+f1)/M1;
/*v8 */    fr1 := -fl2;
/*v9 */    fl1 := -K1*(x1-x0)-C1*(v1-v0);
/*v10*/    x1' := v1;
   END DYNAMIC;
END mass_system;
```

and the coupled masses system model will become:

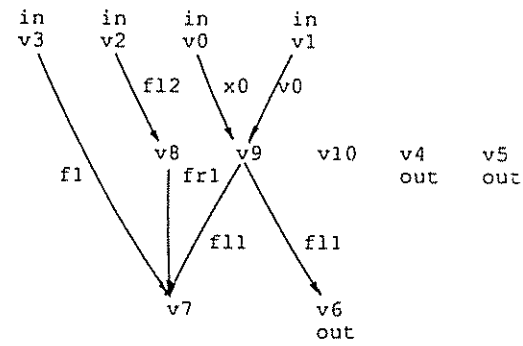
```
/*---------------------- coupled_mass_system
*/
SUBMODEL coupled_mass_system () IS

   CONSTANT
      REAL X0 := 0.0, V0 := 0.0, F1 := 0.0,
           FL3 := 0.0,OMEGA := 1.0,
           PHASE := 0.0;
   END CONSTANT;
   SUBMODEL
      mass_system.left WITH CONSTANT
         REAL M1 := 10.0 , C1 := 10.0,
              K1 := 25.0   ;
      END WITH;
      mass_system.right WITH CONSTANT
         REAL M1 :=  8.0 , C1 := 6.0,
              K1 := 35.0   ;
      END WITH;
   END SUBMODEL;

   DYNAMIC
      f2          :=sine(OMEGA,PHASE);
      x1,v1,fl1 :=
            mass_system.left(X0,V0,f12,F1);
      x2,v2,fl2 :=
            mass_system.right(x1,v1,FL3,f2);
   END DYNAMIC;
END coupled_mass_system;
```

The call to the generalized mass subsystem submodel has a qualifier field that is used to initialize the constants. The constants of a submodel can be initialized within the submodel or from higher level submodels. The highest level initialitation of a given constant overrides lower level initialitations of the same constant.

The digraph associated to the generalized mass subsystem submodel is



and the segments are:

1. State segment Ss=(Is,Os,Ts)

   Is = Empty
   Os = (v4,v5)
   Ts = Empty

2. Derivative segment Sd=(Id,Od,Td)

   Id = (v2,v3)
   Od = Empty
   Td = (v8,v7,v10)

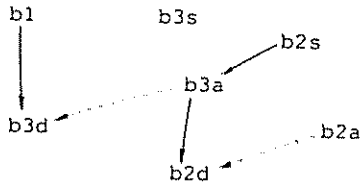
3. Algebraic segment Sa=(Ia,Oa,Ta)

```
Ia = (v0,v1)
Oa = (v6)
Ta = (v9)
```

Writing the calls to the preceding segments using the CSSL syntax, the shape of the dynamic region becomes:

```
     DYNAMIC
b1   f2 := sine(OMEGA,PHASE);
b2s  x1,v1 := mass_system_state.left();
b2d  mass_system_derivative.left(f12,F1);
b2a  f11:=mass system algebraic.left(X0,V0);
b3s  x2,v2 := mass_system_state.right();
b3d  mass_system_derivative.right(FL3,f2);
b3a  f12:=mass_system_algebraic.right(x1,v1);
     END DYNAMIC;
```

The associated digraph is:



Edges represented as continuous arrows are inferred from the input/output variables of each sentence (explicit edges) and dotted edges arise from the segment digraph (hidden edges).

Since the coupled mass system submodel does not have input/output variables, all the sentences within the submodel will be placed in the state segment.

CONCLUSIONS

The outlined segmentation procedure has been succesfully applied to sort a complex electrical network model. The model has 17 submodels and the hierarchical structure has four levels.

Nowadays, a CSSL language with segmentation capability is under development.

REFERENCES

(1) Baker N.J. and Smart P.J., Proceedings of the European Simulation Congress ESC 83 - "The SYSMOD simulation language", September, 1983, pp. 281-286.

(2) Deo N., Graph Theory With Applications to Engineering and Computer Science, Prentice-Hall, 1974.

(3) SCI, "The SCi Continuous-System simulation languaje", Simulation, Vol. 9 (6), December, 1967, pp. 281-303.

(4) Cellier F. E. and Bongulielmi L. P., "The COSY simulation language", Simulation of systems : Proc. of the 9th IMACS Conf. on Simulation of Systems, ed. North Holland pub. co., Sorrento, Italy, 1979, pp. 271-281.

(5) Hay J. L., "A new CSSL standard -an implementation view", Proc. of the UKSC Conference on Computer Simulation, Harrogate , May, 1981, pp. 35-43.

(6) Huber R. M. and Guasch A., "Towards a specification of the structure for continuous system simulation languages", 11th IMACS WORLD CONGRESS, Oslo, August, 1985,

(7) Mitchell E. E. L. and Gauthier J. S., "ACSL User Guide/Reference Manual", Mitchell and Gauthier Assoc. Inc., 1981.

(8) Syn W. M.,and Dost M. H., "On the dynamic simulation language (DSL/VS) and its use in the IBM corporation.", 11th IMACS WORLD CONGRESS, Oslo, August, 1985,

(9) Tarjan R., "Depth-First Search and Linear Graph Algorithms", Siam J., Vol. 1, (2), June, 1972, pp. 146-160

(10) Rimvall M., "Computer aided modeling and simulation using a direct-executing simulation language.", 10th Imacs World Congress on Syste. Simul. and Scient. Comp., Oslo, August, 1985.