DYMOLA - A Structured Model Language for Large Continuous Systems

Elmqvist, Hilding

1979

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Elmqvist, H. (1979). *DYMOLA - A Structured Model Language for Large Continuous Systems*. (Technical Reports TFRT-7175). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

CODEN: LUTFD2/(TFRT-7175)/1-008/(1979)

DYMOLA -

A STRUCTURED MODEL LANGUAGE FOR LARGE CONTINUOUS SYSTEMS

HILDING ELMQVIST

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
SEPTEMBER 1979

Dokumentutgivare Lund Institute of Technology Dept of Automatic Control	Dokumentnamn REPORT
Handläggare Hilding Elmqvist	Dokumentbeteckning LUTFD2/(TFRT-7175)/1-008/(1979)
Författare Hilding Elmqvist	Utgivningsdatum Sept 1979
	Ärendebeteckning 06T6

10T4

Dokumenttitel och undertitel

DYMOLA - A Structured Model Language for large Continuous Systems

Referat (sammandrag)

A model language, called DYMOLA, for continuous dynamical systems is presented. Large models are conveniently described hierarchically using a submodel concept. The ordinary differential equations and algebraic equations of the model need not be converted to assignment statements. There is a concept, cut, which corresponds to connection mechanisms of complex type, and there are facilities to conveniently describe the connection structure of a model. A model can be manipulated for different purposes such as simulation or static design calculations. The model equations are sorted and they are converted to assignment statements using formula manipulation.

Referat skrivet av

author

Förslag till ytterligare nyckelord

44T0

Klassifikationssystem och -klass(er)

50T0

Indextermer (ange källa)

Mathematical models, Simulation languages, Computerized simulation, Nonlinear systems, Ordinary differential equations, Compilers. (Thesaurus of Engineering and Scientific Terms, Eng Joint Council, USA.

Omfång 86 pages	Övriga bibliografiska uppgifter 56T2
Språk English	
Sekretessuppgifter 60T0	ISSN 60T4
	ISBN 60T6
Dokumentet kan erhållas från Department of Automatic Control Lund Institute of Technology Box 725, S-220 07 LUND 7, Sweden	Mottagarens uppgifter 62T4
Pris 66T0	

DYMOLA - A STRUCTURED MODEL LANGUAGE FOR LARGE CONTINUOUS SYSTEMS

Hilding Elmqvist

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

ABSTRACT

A model language, called DYMOLA, for continuous dynamical systems is presented. Large models are conveniently described hierarchically using a submodel concept. The ordinary differential equations and algebraic equations of the model need not be converted to assignment statements. There is a concept, cut, which corresponds to connection mechanisms of complex type, and there are facilities to conveniently describe the connection structure of a model. A model can be manipulated for different purposes such as simulation or static design calculations. The model equations are sorted and they are converted to assignment statements using formula manipulation.

INTRODUCTION

A common principle for solving large problems is *decomposition* of the problem into a set of smaller subproblems which are either solved directly or decomposed further. The original problem is then solved by combining the subproblem solutions.

This principle is used when modelling large systems. The system is considered as a set of subsystems. This decomposition is often inherent in the physical system. An industrial plant, for example, is in fact designed according to the decomposition principle. The language to describe the model should reflect this and encourage the use of *submodels.* The languages of CSSL-type have a MACRO-concept which corresponds to submodels.

A model must also contain a description of how submodels interact with each other. The introduction of submodels are often done in a way that the interactions between submodels are rather limited. The interaction is often restricted to a set of *connection mechanisms.* Such connection mechanisms often correspond to some physical devices such as shafts, pipes,electrical wires etc.

In the languages of CSSL-type there are no constructs corresponding to such connection mechanisms. The connections are done by means of variables. Each macro has formal input and output variables. The connection of two submodels is done by having the same variable appear as actual variable in both of the corresponding macro calls. This way of describing the connections between submodels tends to hide the connection structure of the model. One reason is that the details of the connection mechanisms, such as the variables involved, are considered at the same time. A model language ought to have a means of defining abstract connection mechanisms and a means of describing the connection structure in a natural way.

The fundamental way of describing submodels is by equations. Physical laws are formulated as for example mass and energy balances and phenomenological equations. In CSSL such equations must be entered as assignment statements which, after automatic sorting, give an algorithm for assignment of derivatives and auxiliary variables.

This paper describes a model language called DYMOLA (Dynamic-Modelling Language). It is designed to overcome some of the drawbacks with languages of CSSL-type.

Dymola has a hierarchical submodel concept. Abstract connection mechanisms are introduced by means of the concept *cut* which defines the variables associated with each connection. The connection structure of the model is described by a *connection statement.* The model equations are entered in their original form. They need not be converted to assigment statements.

The compiler translates the connection statements to equations. The equations obtained can then be used for different calculations such as simulation or static design calculations. The user specifies which variables are considered known. The equations are automatically sorted and transformed to assignment statements to get an algorithm that assigns the unknown variables. Systems of equations that have to be solved simultaneously are then detected. The assignment statements are obtained by automatic formula manipulation.

The complete definition of the Dymola language can be found in Elmqvist [1].

SUBMODELS

A model can be defined hierarchically according to the following pattern.

```
model <model ident>
  declaration of submodels
  declaration of variables and connection mechanisms
  equations and connection statements
end
```

If several subsystems have the same model, their descriptions do not have to be duplicated. It is possible to define a model type with the same structure as model . Such a model type can then be duplicated with a submodel -statement.

VARIABLES AND EQUATIONS

The following types of variables can be declared in a model: parameter, constant, local, terminal, input and output (see the syntax in the appendix).

The terminal variables decribe the interdependence between a submodel and its environment. The input and output variables are special cases of terminal variables. They are introduced to make it possible to indicate causalities in the model.

Some types of variables can be referenced from the outside of a submodel using a dot-notation.

The equations of the submodels have the form

```
<expression> = <expression>
```

where <expression> is defined as for Algol-60. An equation can thus contain the if-then-else construction. It is also possible to call functions and procedures written in some algorithmic language. Derivatives are denoted by x', x'' or der(x), der2(x) etc.

`C1 from <C2 - C3>`	`C2`	`C3 at C1`
`<C1 - C2> from C3`	`C2`	`C3 at C1`
`C1 from C2`	`none`	`C2 at C1`

5. `<C1 - C2> par <C3 - C4>`	`<C1 - C2>`	`C1 at C3`
		`C2 at C4`

6. `<C1 - C2> loop <C3 - C4>`	`<C1 - C2>`	`C1 at C4`
		`C2 at C3`

7. `<C1 - C2> branch <[C3 C4 ...] - C5>`	`<C1 - C5>`	`C2 at C3`
		`C2 at C4`
		`...`

8. `<C1 - [C2 C3 ...]> join <C4 - C5>`	`<C1 - C5>`	`C2 at C4`
		`C3 at C4`
		`...`

9. `(C1)`	`C1`	`none`
`(<C1 - C2>)`	`<C1 - C2>`	`none`
`(C1 C2 ...)`	`[C1 C2 ...]`	`none`
`(<C1 - C2> <C3 - C4> ...)`	`<[C1 C3 ...] - [C2 C4 ...]>`	`none`

Example

Consider the electrical network in fig 3.

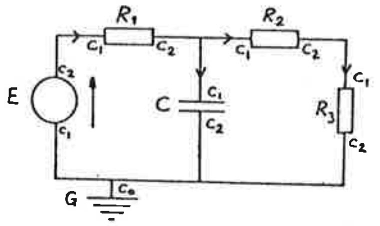

Fig 3. Electrical network

The system can for example be described with the following connection statement.

```
connect G to E to R1 to (C par (R2 to R3) ) to G
```

In order to explain how this statement is evaluated, assume that the submodel G has the cut declaration

```
main cut CO(...)
```

and the other submodels have the path declaration

```
main path P<C1 - C2>
```

The connection statement could then be represented as

```
G:CO to <E:C1 - E:C2> to <R1:C1 - R1:C2> to (<C:C1 - C:C2> par
  (<R2:C1 - R2:C2> to <R3:C1 - R3:C2>) ) to G:CO
```

The steps performed when evaluating this expression are given below.

1. Effect `G:CO at E:C1`
   Result `E:C2 to <R1:C1 - R1:C2> to (<C:C1 - C:C2> par (<R2:C1 - R2:C2> to <R3:C1 - R3:C2>) ) to G:CO`

2. Effect `E:C2 at R1:C1`
   Result `R1:C2 to (<C:C1 - C:C2> par (<R2:C1 - R2:C2> to <R3:C1 - R3:C2>) ) to G:CO`

3. Effect `R2:C2 at R3:C1`
   Result `R1:C2 to (<C:C1 - C:C2> par <R2:C1 - R3:C2>) to G:CO`

4. Effect `C:C1 at R2:C1, C:C2 at R3:C2`
   Result `R1:C2 to <C:C1 - C:C2> to G:CO`

5. Effect `R1:C2 at C:C1`
   Result `C:C2 to G:CO`

6. Effect `C:C2 at G:CO`
   Result `none`

■

MANIPULATION OF EQUATIONS

The manipulation of the equations is described in Elmqvist [1], [2]. Only a brief survey is given here.

The connection statements are translated to equations by the compiler. This means that the model then consists of only equations. They can formally be written as

$$f(t, \dot{x}, x, z, p) = 0$$

where t is time, x is a vector of state variables, z is a vector of auxiliary variables and p is a vector of parameters.

In order to use an explicit integration algorithm, $\dot{x}$ and z should be solved for. The model equations often have special characteristics. All variables are not present in each equation. The Jacobian of f with respect to $\dot{x}$ and z thus contains many zero elements, i.e. it is sparse. This means that the solution could be obtained more efficiently by partitioning the system of equations into a set of smaller systems of equations. In fact, many of the systems of equations will be scalar.

The algorithm for partitioning uses only structural information, i.e. whether a variable is present in an equation or not. The first problem is to determine which variable to solve for in each equation. It is called finding an assignment. The next step is to find a partitioning of the equations into minimal systems of equations that must be solved simultaneously and to sort them for correct computational order. This is called finding the strong components of the associated bipartite graph.

The total effect of these two algorithms is finding two permutation matrices operating on the columns and the rows of the Jacobian and making it block triangular.

When this procedure has been performed it is thus known in which order the equations should be solved and which variables to solve for. The blocks correspond to systems of equations that must be solved simultaneously.

The equations are then converted to assignment statements for the unknown variable. This is done only for equations which are linear in this variable. An important observation for model equations is that there is often such a linear dependence in the simulation case.

One important advantage with automatic manipulation of the equations is that the same basic equations could be used for different calculations such as simulation and static design calculations. The algorithms only have to know which variables are known and which are unknown.

The structural analysis is very useful for the modeller. It gives information about causalities and algebraic loops in the model. This can then be compared with the modeller's perception of these properties in the real system.

IMPLEMENTATION

A compiler for the language and the algorithms for the manipulation of the equations have been implemented using the programming language Simula. The complete program listing can be found in Elmqvist [1]. The input to the program is a model in Dymola and the output is

The description of the IPturb is shown below.

```
model type IPturb

submodel (turbsection) IP1 IP2 IP3 IP4

path steam <IP1:insteam - IP4:outsteam>
cut extract1 [IP1:extract IP2:extract IP3:extract]
cut extract2 [IP4:extract]
path power <IP1:inpower - IP4:outpower>

connect (steam) IP1 to IP2 to IP3 to IP4
connect (power) IP1 to IP2 to IP3 to IP4

end
```

The overall description of the power station in accordance with fig 4 is shown below.

```
model powerstation

submodel (superheater) superh1 superh2 superh3
submodel (attemperator) attemp1 attemp2
submodel (turbsecton) HPturb
submodel drumsyst reheater controlvalve
submodel IPturb LPturb
submodel condensor splitsteam dearator
submodel preh1 preh2
submodel feedwaterpump feedwatervalve
submodel combchamber economizer

connect (heat) combchamber to (economizer ->
        drumsyst::risers  superh1  superh2  superh3  reheater)

connect (steam) drumsyst::drum to superh1 to attemp1 to   ->
        superh2 to attemp2 to superh3 to controlvalve to   ->
        HPturb to reheater to IPturb to LPturb to condensor

connect (extract) (HPturb  IPturb:extract1) to preh2  ->
        IP:extract2 to splitsteam to (dearator  preh1:extract2)->
        LP to preh1:extract1

connect (feedwater) condensor to preh1 to dearator to   ->
        feedwaterpump to preh2 to feedwatervalve to   ->
        (economizer to drumsyst::drum  attemp1  attemp2)

connect (condensate) preh2 to dearator  ->
        preh1 to condensor

connect (power) HPturb to IPturb to LPturb

end
```

The total number of equations for this model is about 400. The equations was sorted for the simulation case. Eleven systems of equations was discovered. The largest contained 17 nontrivial equations.

CONCLUSIONS

The Dymola language contains several new constructs that correspond to the way large dynamical models are developed.

A model can be decomposed into a set of submodels. The interaction of a submodel with its environment is often naturally considered as a set of interactions from different other submodels through distinct connection mechanisms. This corresponds to cut declarations. Interactions might be further decomposed by introducing hierarchical cuts. The basic level of defining an interaction is by associating a set of terminal variables with it.

The connection of submodels is often viewed as a block diagram or a graph. The connection statement makes it possible to conveniently describe such diagrams as text. Since directions are often associated with this kind of diagrams, it is natural to introduce the concept of a path.

Because the basic means of describing models is equations, the models can be written in a form that is independent of what calculations they are used for.

ACKNOWLEDGEMENTS

The author wants to thank Professor Karl Johan Åström for his support and guidance during this work.

This research was supported by the Swedish Institute of Applied Mathematics (ITM).

REFERENCES

[1] Elmqvist, H., A Structured Model Language for Large Continuous Systems, Ph.D. Thesis, TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden (May 1978).

[2] Elmqvist, H., Manipulation of Continuous Models Based on Equations to Assignment Statements, Proc. IMACS Congress 1979 / Simulation of Systems, Sorrento, Italy (September 1979).

[3] Lindahl, S., A Nonlinear Drum Boiler - Turbine Model, Report TFRT-3132, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden (1976).

APPENDIX

Syntax Notation

The following syntax notation is used: | means or, { } groups terms together, [ ] means that a group of terms is optional, { }* means repetition one or more times, [ ]* means repetition none or more times.

To make the syntax smaller, it has purposly been left incomplete in some respects. It does not contain the definitions of basic items such as `<identifier>` and `<number>`. Trivial productions such as `<model ident>::=<identifier>` are omitted.

New line is treated as ;. Continuation of a statement on the next line is indicated by -> at the end of the line.

Syntax For The Dymola Language

```
<model spec>::=[<model type>; ]*<model>

<model>::=model <model ident>; <model body> end
<model type>::=model type <model type ident>; <model body> end
  <model body>::=<submodel part> <declaration part>
          <statement part>

    <submodel part>::=[<model>; | <model type>; |
          <submodel incorp>;]*
      <submodel incorp>::=submodel [(<model type ident> ) ]
          {<model ident> [(<parameter list>)]}*
      <parameter list>::={<number>}*|
          {<parameter>=<number>}*

    <declaration part>::=[<variable declaration>; |
          <cut declaration>; | <node declaration>; |
          <path declaration>; ]*
```

```
<variable declaration>::=
    parameter {<variable> [=<number>]}*|
    constant {<variable>=<number>}*|
    local {<variable>}*|
    terminal {<variable>}*|
    input {<variable>}*|
    output {<variable>}*|
    default {<variable>=<number>}*|
    external {<variable>}*|
    internal {<variable>}*

  <variable>::=<identifier>

 <cut declaration>::=[main] cut {<cut ident> [<cut>]}*
   <cut>::=<cut clause> | <cut spec>
     <cut clause>::=( <variable cut> ) |
       [ <hierarchical cut> ]
     <variable cut>::=[<cut element>]*
       [/ [<cut element>]*]
     <cut element>::=<variable>|-<variable>|.
     <hierarchical cut>::={<cut>|.}*
     <cut spec>::=<model spec>[:<cut ident>]|
       <cut ident>
     <model spec>::=<model ident>[::<model ident>]*

  <node declaration>::=node {<node ident>
      [<node clause>]}*
    <node clause>::=( <variable cut>) |
      [ <hierarchical node> ]
    <hierarchical node>::={<node clause> |
      <node ident> | .}*
```

```
<path declaration>::=[main] path {<path ident>
    {<path clause> | <path spec>} }*
  <path clause>::=< {<cut>|.} - {<cut>|.} >
  <path spec>::=<model spec>[..<path ident>]|
    <path ident>

<statement part>::=[<equation>; | <procedure call>; |
     <connection statement>; ]*

  <equation>::=<expression> = <expression>
    <variable spec>::=[<model spec>.]<variable>

  <procedure call>::={<variable spec>}*=
      <procedure ident> ( {<expression>}* )

  <connection statement>::=connect [(<ident>)]
      {<connection expression>}*
    <connection expression>::=<connection secondary>
      { {at|=|to|-|from|par|//| loop|branch|join}
      <connection secondary> }*
    <connection secondary>::=
      [reversed]<connection primary>
    <connection primary>::=<connection operand> |
      ( {<connection expression>|.}* )
    <connection operand>::=<cut spec> | <path spec> |
      <node ident>
```