On an Interface Between OmSim and IdKit

Sørlie, James A.

1997

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

ISSN 0280–5316
ISRN LUTFD2/TFRT--7562--SE

On an Interface Between OmSim and IdKit

James A. Sørlie

Department of Automatic Control
Lund Institute of Technology
June 1997

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

Document name
TECHNICAL REPORT

Date of issue
June 1997

Document Number
ISRN LUTFD2/TFRT--7562--SE

Author(s)
James A. Sørlie

Supervisor

Sponsoring organisation
NUTEK, Dnr 96-10653

Title and subtitle
On an Interface Between OmSim and IdKit.

Abstract

This report describes a code-generation interface between OmSim and IdKit which has been developed during a nine month post-doctoral visit at the department. The interface consists of command-line tools which transform model equations exported from OmSim into C-code subroutines that can be used by IdKit for parameter optimization. The tools make use of batch calls to Maple and its packages MacroC and TplC for symbolic manipulation and code generation. These tools were used in a case-study documented in a companion report.

Key words
nonlinear modeling,optimization,parameter estimation,system identification

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
31

Recipient's notes

Security classification

Contents

Chapter 1

Introduction

This document is my final progress report and is intended to document the results of my postdoctoral visit at LTH Reglerteknik. While planned as a seven month stay, my visit was extended two months. The original goal of my visit was to further investigate and develop a software interface [1, Ch. 5] between the LTH's model definition tools [2] and KTH's identification tools [3].

In late November 1996, with the then coming visit of Professor Rodney Bell from Australia, a decision was made to investigate parameter optimization of the Åström–Bell drum boiler model [4]. Since then, this has been my primary goal. This has been a collaborative effort with Jonas Eborn, a doctoral candidate at the department, and has yielded one conference paper [5] and a final report [6].

The main components in this report are:

(*i*) A final status report on the OMSIM–IDKIT interface. Because of the change in plans, I adopted the "kiss" strategy in developing the software, i.e. do only that needed to get on with the case study. Minimally, this equated to command-line oriented tools for automatic code generation of model equations according to the existing IDKIT model subroutine specification. Development of an interface based on symbolically derived Kalman filter equations has been left incomplete.

(*ii*) A concluding discussion of the directions that this work could be continued.

(*iii*) Attached as an appendix are listings of the various files discussed in the report.

Acknowledgments

The work has been supported by the Swedish National Board for Industrial and Technical Development under contract Dnr 96-10653, Project P9304688-4.

I would like to thank Docent Sven Erik Mattsson and Professor Karl Johan Åström for the invitation to visit LTH's Department of Automatic Control, and gratefully acknowledges the arrangements made for financial support.

Professionally, I very much enjoyed the interaction with Jonas Eborn. We first met in 1996 at a mini-course in Uppsala and enjoyed talking about OMOLA. I thank him for his patience in acting as my sounding board, and also for his efforts in keeping me attuned to happenings at the department.

Finally, the visit would not have been the same without the attention paid by the charming women of the fifth floor.

Chapter 2

Status of Software Development

A preliminary proposal written in October 1996 listed the following software design issues:

- internal versus external symbolic manipulation/code generation
- continuous and/or discrete-time model support
- level of the interface, i.e. model versus predictor equations
- degree of task automation

Time constraints and the change of focus of my visit have forced a number of design decisions. Rather than diving into the internals of OMSIM's implementation, I elected to proceed developing **modular external tools** and thereby build upon the products of my doctoral research. These tools were developed for **continuous-time models** and the case study involves a continuous-time process. Hence, no effort has been directed toward either discrete-time or hybrid systems.

The shift in emphasis to *using* rather than *developing* tools dictated that I do only that needed for the case study. Minimally this requires generation of the C-language subroutines for IDKIT. Again, time constraints dictated that we stick with the existing IDKIT **subroutine interface for model equations** and forego development of a predictor equation level interface.

As for the degree of task automation, the modular external tools have been packaged, where possible, as **command-line oriented** shell scripts. The only exception is the creation of the OMSIM log file, i.e. the "input" to the automatic code generation. Creation of the log file requires user interaction via the mouse.

2.1 Project Planning and Progress Made

The preliminary proposal's project planning was summarized in a Gant chart and is reproduced here in figure 2.1. Below we reconsider these issues and document the progress made in developing the interface.

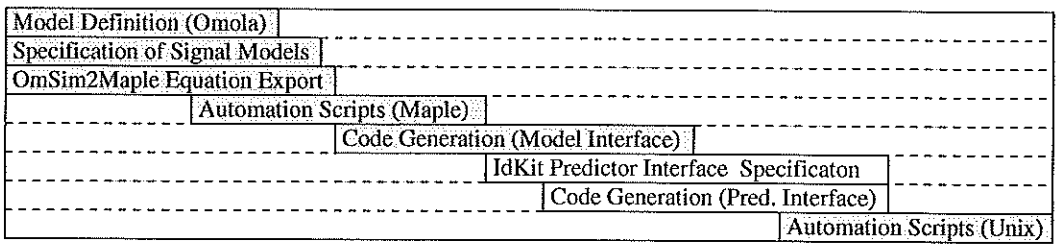

Figure 2.1: Original Gant chart for project planning of an interface between OMSIM and IDKIT. Shading indicates tasks which have been addressed and/or completed.

2.1.1 Model Definition (OMOLA)

Originally this block was included to indicate the need for test cases. In October, I upgraded the test cases in my Ph.D. thesis [1, Ch. 4] to the latest version of OMSIM (3.6). With regard to the drum boiler case study, Jonas Eborn and I have developed a family of OMOLA definitions of the model structure. The complexity of the definitions ranges from first through fifth-order state equations; cf. [4, 7–9]. Class inheritance, the K2 library [10], and OMOLA's freedom to over-write attributes have proven very useful in making the model definitions concise.

2.1.2 Specification of Signal Models

To facilitate symbolic processing of equations exported from OMSIM, I proposed in my Ph.D. thesis a signal model "wrapper" class library. Its purpose was to standardize the definition of signal model mappings, i.e. the mapping of long hierarchic names to short symbolic names. Such a class library proves to be over-kill. The name mappings can be more clearly programmed as a sub-model in the definition of a models simulation interface.

An example of this revised approach is shown in appendix listings A.1 and A.2. When viewed in OMSIM's graphical editor MED, this sub-model approach takes on the flavor of a "Data Store" block in SystemBuild [11]. Unless a formal equation export mechanism for OMSIM were planned, this simple approach is flexible, practical, and suffices.

2.1.3 OMSIM2MAPLE Equation Export

As part of my thesis work, I developed a shell-script translator which extracted the sorted modeling equations from a log file containing OMSIM's debug output, specifically, the "Simulation code" and "Parameter part." In version 3.6 of OMSIM, the "Flat model" debug option was added and the functionality of the `initial` attribute of variables was abandoned in favor of handling state variable initialization as a discrete event. These changes necessitated changes in the export procedure as well as the translator.

The new export procedure, i.e. creation of the log file, is visualized in figure 2.2. The "Flat model" output contains all information previously groped from the "Simulation code" plus "Parameter part" outputs combined. The new discrete state initialization necessitates including the "Event part" in the OMSIM log file.

The revised version of the shell-script translator, more appropriately named flat-model2maple, is listed in the appendix; see listing A.3, pages 11–12. Although the syntax of OMSIM's compiled equations (and hence the "Flat model" output) is very close to MAPLE's syntax, there are some key incongruities. Table 2.1 provides a summary of the translations performed by the shell script.

Syntax	OmSim	Maple
model hierarchy†	`m.attribute.attribute`	`m_attribute_attribute`
time derivative†	`x', X[1]'`	`x_prime, X_prime[1]`
function names	`asin, acos, atan, atan2`	`arcsin, arccos, arctan, arctan`
column vectors	`[x1; x2; ...; xn]`	`[[x1], [x2], ..., [xn]]`
matrices	`[x11, x12; x21, x22]`	`[[x11, x12], [x21, x22]]`
conditional statement	`IF expr1 THEN expr2 ELSE expr3`	`piecewise(cont.,expr1,expr2,expr3)`

Table 2.1: Summary of syntactic incongruities between the OMSIM and MAPLE; syntactic translations marked † indicate arbitrary design choice in the translation.

Conditional Branches The most troublesome translation is conditional statements. OMOLA lacks a tag marking the end of the final branch expression. Since this allows a cleaner nesting of conditional branches, the language is better without one. It is however difficult to program the translation in a shell-script. The following simple example breaks the translator as implemented today:

```
y := m*(IF x < a THEN x-a ELSE IF x < b THEN 0 ELSE x-b) + c;
```

In MAPLE, this would be expressed as follows:

```
y := m*piecewise(0, x<a, x-a, x<b, 0, x-b) + c;
```

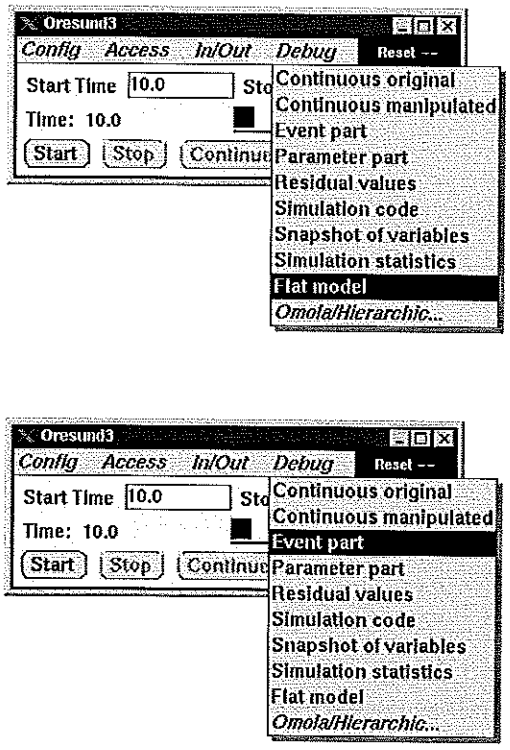

Figure 2.2: The two steps of mouse interaction involved in exporting modeling equations from OMSIM to a log file; after translation using the shell-script in listing A.3, the equations can be loaded into MAPLE.

The problem is simple: the conditional expression appears as a sub-expression in a composite expression. The correct translation of such requires the intelligence of a lexical analyzer. The work-around "solution" has been to avoid programming such composite expressions in OMOLA. This is done simply by introducing interim variables. In the example above, the "solution" looks like this:

```
xx ISA Variable WITH value := IF x < a THEN x-a ELSE IF x < b
  THEN 0 ELSE x-b; END; y := m*xx + c;
```

Note that all these problems existed in the original version of the shell-script translator. They were not as well understood before. Adapting to "Flat model" export greatly simplified the translator. The "Flat model" debug output is a tangible step towards providing a useful high-level equation export mechanism. Nevertheless, the need for a translator to MAPLE persists.

2.1.4 Automation Scripts (MAPLE)

This block refers to the post-processing task that is necessary after one has created the OMSIM log file. The task has been automated as a MAPLE command script. Because the post-processing is an integral part of the code generation, the MAPLE command script is itself embedded as a "here-document" inside a Unix shell script; see lines 17–140 of listing A.4 (pp. 13–15). Note that this command script sources (in line 28) the utility procedures shown in listing A.5 (pp. 16–17).

Event Initialization Switching to OMSIM's "Flat model" output homogenized the post-processing task somewhat. The bulk of the post-processing, specifically lines 59–118 in listing A.4, is spent manipulating the "Event part" i.e. the *initialization event equations*. Lines 59–71 of the script yield expressions of the initial state conditions in terms of initial input conditions, numerical constants and the model parameters. Similarly, lines 78–111 construct expressions for possible initial event equations, i.e. parameter equations. To un-

derstand how event equations arise, consider the following OMOLA code (taken from the drum-boiler case study):

```
OnEvent Init DO
  % Initialize state variables.
  new(Vwt) := 1/2*Vd + (1-ar0)*Vr + Vdc + dVwt0; new(P) := P0;
  schedule(Start,0.0); % fire immediately after OMSIM's init-solver
END; OnEvent Start DO
  % Initialize variables dependent upon the initial conditions
  new(ar0) := ar; new(qct0) := qct; new(qfw0) := qfw; new(qr0) :=
  qr; new(qs0) := qs; new(rs0) := rs; new(rw0) := rw; new(Vwd0)
  := Vwd-Tsd*(hw-hfw)*qfw/rs/hc; END;
```

During the built-in `Init` event, the two state variables are initialized; note that one state is parameterized. During the scheduled `Start` event, eight variables are initialized, capturing the initial values of these signals. Although only `Vwd0` appears to be parameterized, all are parametric functions. These event equations depend solely on the values of the model parameters, the initial input and state conditions.

It should be emphasized that the *export of event equations from* OMSIM *is limited to* ***initialization*** *events*. Including the "Event part" in the OMSIM log file is necessary to export state variable initialization. Exporting the initialization events is an added benefit. The exported equations do remain however completely continuous-time.

2.1.5 Code Generation (Model Interface)

Automatic encoding of OMSIM model equations according to the IDKIT model subroutine specifications has been implemented. This work is an extension of the TPLC package for MAPLE [1, App. C] which provides code generation according to the Simulink SimStruct S-function subroutine interface [12]. TPLC stands for Template-C. The package was strongly inspired by the template programming language behind AutoCode [13] from Integrated Systems. Architecturally, this package is built on top of the MACROC share-ware package for MAPLE [14].

The newly developed code generation template for IDKIT is included in the appendix; see listing A.6, pp. 18–24. Half of the code (lines 31–269) is spent processing the model equations, checking for errors, and constructing lists of C pre-processor (CPP) definitions and include files. The code generation template, i.e. MACROC definitions of IDKIT subroutines, is in lines 277–433. Processing the template using the MACROC code generator occurs in line 436. The usage of the code generation template (listing A.6) is shown in listing A.4; see lines 145–168 on page 15.

Sub-Expression Optimization One may wonder "what sets code generation using MAPLE and MACROC apart from AutoCode [13] and the Real Time Workshop [15] from MathWorks?" The answer is *sub-expression optimization* [14]. Optimal numerical computation of a set of equations is achieved by collecting interim variables, i.e. common sub-expressions, evaluating them once, and re-using the computed values. In MAPLE, the deduction of these common sub-expressions is performed by the `optimize` package [16].

The result of using MAPLE and MACROC is best shown by example. Consider the implicit third-order state equations from the Åström-Bell drum boiler model [4]:

$$\begin{bmatrix} e11 & e12 & 0 \\ e21 & e22 & 0 \\ 0 & e32 & e33 \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial t} V_{wt} \\ \frac{\partial}{\partial t} P \\ \frac{\partial}{\partial t} x_r \end{bmatrix} = \begin{bmatrix} q_{fw} - q_s \\ Q + h_{fw}\, q_{fw} - h_s\, q_s + \Delta_{geb} \\ -h_c\, x_r\, q_{dc} + Q + \Delta_{reb} \end{bmatrix} \tag{2.1}$$

Solving explicitly for the state derivatives, we have:

$$\begin{bmatrix} \frac{\partial}{\partial t} V_{wt} \\ \frac{\partial}{\partial t} P \\ \frac{\partial}{\partial t} x_r \end{bmatrix} = \begin{bmatrix} \frac{(Q + h_{fw}\, q_{fw} - h_s\, q_s + \Delta_{geb})\ e12 + (-q_{fw} + q_s)\ e22}{-e11\ e22 + e12\ e21} \\ -\frac{(Q + h_{fw}\, q_{fw} - h_s\, q_s + \Delta_{geb})\ e11 + (-q_{fw} + q_s)\ e21}{-e11\ e22 + e12\ e21} \\ \frac{-h_c\, x_r\, q_{dc} + Q + \Delta_{reb} - e32\ \left(\frac{\partial}{\partial t} P\right)}{e33} \end{bmatrix} \tag{2.2}$$

Specifically, note the dependency of the third state equation on $\frac{\partial}{\partial t}P$, the time-derivative of pressure. An excerpt from the output of the TPLC code generation template is shown in the appendix listing A.7, pp. 25–26. Lines 22–79 of the listing show the computation of the common sub-expressions. Lines 80–90 perform the actual computation of the state derivatives.[1] As in equation 2.2 above, notice how the expression for $\frac{\partial}{\partial t}P$(= `t116*t111`) reappears in the computations; look at lines 85 and 89 respectively in the listing. In these lines, `t111` corresponds to the denominator expression $-e11\ e22 + e12\ e21$.

The explicit formulation in equation 2.2 is the most compact solution for the system of state equations and corresponds the result generated by the OMSIM compiler. As shown above, this is also the result yielded using MAPLE for code generation. However fortunate, this conclusion is contrary to what I thought in mid-April this year. The reason has to do with complexity introduced through multiple realizations [1, Ch. 4] and the translation of (nested) branch statements to (nested) `piecewise` functional statements.

Conditional Branches *(revisited)* As implemented today, my approach to code generation is to take the sorted OMSIM model equations and translate them into MAPLE assignment statements. When loaded into the MAPLE interpreter, this leads to an automatic substitution of the sorted definitions. The explicit state equations are thereby constructed and the sorted ordering provided by OMSIM is lost.

In complex cases, such as the drum boiler model, this approach leads to extremely large composite expressions. As demonstrated by listing A.7, MAPLE's `optimize` facility can handle the complex job of reconstructing the (lost) sorted information. However, the facility does have its limitations.

As mentioned above, in mid-April, I believed these limitations (i.e. the loss of information) to be more of a problem than they are. The problem manifested itself as follows:

(i) Code generation was only possible on the departments most powerful computer, Bellman, equipped with 128 MByte of RAM memory.

(ii) Processing times were on the order of two minutes.

(iii) Although the resulting numerical code functioned properly, it seemed extremely complex for "optimized" code.

The cause of the problem turns out to be the complexity introduced by conditional statements. Consider the following OMOLA code:

```
value := IF Sw < 1 THEN hypothesis[1] ELSE IF Sw < 2 THEN
    hypothesis[2] ELSE IF Sw < 3 THEN hypothesis[3] ELSE
    hypothesis[4];
```

The output of the shell-script translator (listing A.3) is then:

```
value := piecewise( 0, Sw<1, hypothesis[1], piecewise( 0, Sw<2,
         hypothesis[2], piecewise( 0, Sw<3, hypothesis[3],
         hypothesis[4] ) ) );
```

Note the nesting of the `piecewise` operators. This functional nesting aggravates the job of the `optimize` facility. Common sub-expressions are found by recursively comparing the branch arguments of the functional operators. At each level, the first hypothesis is compared against a functional expression. If each of the hypotheses above represents a complex composite expression, then the comparisons fail (i.e. `optimize` fails to dissect the complexity).

Before revealing the work-around "solution", it should be mentioned that this problem could have been avoided (at least to some extent). The complexity is compounded by the simple approach taken in translating from OMSIM to MAPLE. The formal syntax of MAPLE's `piecewise` operator is a follows:

```
piecewise( c_1, f_1, c_2, f_2, ..., c_n, f_n, f_(n+1) )
```

Hence, a better translation from OMOLA to MAPLE, which avoids the unnecessary nesting, would be:

```
value := piecewise( 0, Sw<1, hypothesis[1], Sw<2, hypothesis[2],
                       Sw<3, hypothesis[3], hypothesis[4] );
```

The simplistic approach taken in the translation was necessitated by the limitations of the lexical logic that can be implemented in a shell-script.

[1] The first four states in the IDKIT model belong to stochastic disturbance models not shown above.

Fixed Realization Compilation The work-around to the complexity problem is to "hard-wire" the value of the realization switches prior to model compilation. Specifically, the parameterized definition of the switching parameter is programmed as

```
Sw TYPE STATIC Real := 1;                    % Hard-wired during compilation.
```

rather than:

```
Sw ISA Parameter WITH default := 1; END; % Modifiable after compilation.
```

With the switching parameter `Sw` hard-wired, conditional branches are eliminated automatically by the OMSIM compiler. The chief disadvantage of this "solution" is that one cannot switch between realizations after compilation. One must manually edit the OMOLA source code *before compilation* and equation export.

By using the OMSIM compiler to eliminate the realization branches, we totally circumvent the problems MAPLE's `optimize` has during code generation. With this concession of convenience, MAPLE's memory requirements during code generation decrease drastically. For the most complex drum boiler model (sixth-order), execution times drop to 10–15 seconds on a 120 MHz Pentium running Linux and equipped with 24 MByte RAM.

Finally, we note that the concession of hard wiring realizations before code generation is necessary for another reason. If one wishes to derive predictor equations from the OMSIM model equations, fixing the active realization is necessary because of the symbolic differentiation involved in the derivation. Automatic derivation of first-order EKF predictor equations is demonstrated in lines 157–160 of listing A.4, page 15; cf. [1, App. C].

2.1.6 IDKIT Predictor Interface Specification

Creating a new code generation template is relatively easy using the MACROC and TPLC packages. However, defining a new application interface requires deeper thought. For this reason, and because of the change of focus of my visit at LTH, I have not developed a specification for the predictor equation interface for IDKIT. In the Gant chart on page 2, this remains one of the un-shaded blocks, i.e. work left incomplete.

2.1.7 Code Generation (Pred. Interface)

Without the interface specification, it is not possible to develop the code generation template. Hence, the Simulink S-function code generation template remains the only support for predictor equations derived from OMSIM model equations.

2.1.8 Automation Scripts (Unix)

To make code generation as automatic as possible, the steps of:

(*i*) OMSIM log file translation,

(*ii*) MAPLE post-processing of the equations, and

(*iii*) MAPLE code generation,

have been embedded within a single Unix shell-script; see listing A.4, pp. 13–15. The command script is simple to use; it takes no arguments. One merely executes it in the directory where the OMSIM log file resides. If successful, the script will create two files in the same directory:

`model.c` — a subroutine module for use with IDKIT,

`Smodel.c` — a Simulink S-function for Matlab.

The results of running the script are logged in two files: `prep.c` and `codegen.log`. These logs can be useful in diagnosing problems, should they arise.

Chapter 3

Conclusion

A fully functional interface between OMSIM and IDKIT has been designed, prototyped and tested. The interface consists of tools for translation of model equations exported from OMSIM into C-language subroutines—the format of the equations required by IDKIT. With these subroutines, computer-aided parameter optimization of OMOLA models becomes possible. The tools have been tested and extensively used in a case study. A companion report [6] documents these experiences.

Time constraints forced the decision to only develop tools for IDKIT's existing subroutine interface, i.e. for the model equations. Plans had included the development of a subroutine interface for predictor equations, i.e. an extended Kalman filter derived from the model equations. The chief purpose of such an interface would be to allow efficient and proper treatment of numerically stiff systems.

3.1 Directions for Continued Development

As I see it, there exist two distinct directions to continue this work. One is to continue developing modular external tools as done here. Were someone to do this, things to prioritize would be:

(*i*) finalizing a subroutine specification for an IDKIT predictor equation interface;

(*ii*) implementing code generation for such a subroutine interface.

The other direction would be to develop tools internally within OMSIM and the realm of the CACE project. Topics for study would be:

(*i*) investigate how the symbolic derivation of the Kalman filter equations, and code generation, could be implemented utilizing the sorted equations provided by OMSIM;

(*ii*) finalizing a specification for *high-level* equation export, i.e. define a nonlinear control data object (CDO).

Appendix A

Listings

A.1 Model Definition and Equation Export

```
1   SignalModelMapping ISA Base::Model WITH
2     Graphic ISA super::Graphic WITH
3       bitmap TYPE String := "SignalModelMapping"; x_pos := 250; y_pos := 300;
4     END;
5   input_signals:
6     U, U0, Wv, Wy ISA Std::VectorVar;
7   state_signals:
8     X ISA Std::VectorVar;
9   output_signals:
10    Y ISA Std::VectorVar;
11  data_store:
12    PM ISA Std::MatrixVar;
13    NaN ISA Parameter;
14  END;
```

Listing A.1: DrumBoiler/SignalModelMapping.om—An Omola class that defines the signal-model name mappping required by the equation export utilities.

```
1   Oresund2 ISA DrumBoiler::OresundSimIC WITH
2
3     Boiler ISA DrumBoiler::Boiler2FM;
4
5   equation_export:
6     SMM ISA DrumBoiler::SignalModelMapping WITH
7     input_signals:
8       U.n  := 5;  U  := [qs1;  qs2;  qf;  qfw;  Tfw ];
9       U0.n := 5;  U0 := [qs10; qs20; qf0; qfw0; Tfw0];
10      Wv.n := 4;  Wv := [CN1; CN2; CN3; CN4];
11      Wy.n := 2;  Wy := [DN1; DN2];
12    output_signals:
13      Y.n  := 2;  Y  := [dp; dl];
14    state_signals:
15      X ISA Std::VectorVar WITH
16        n := 6;
17        value := [LPF1.x; Wp1.W; Wp2.W; Wp3.W; Boiler.Vwt; Boiler.P];
18      END;
19    parameter_matrix_mapping:
20      PM ISA Std::MatrixVar WITH
21        m := 23;
22        n := 5;
23        value := [qs10,             qs20,        qf0,        qfw0,       Tfw0;
24                  Boiler.mdc,  Boiler.Vd,  Boiler.Vr, Boiler.Vdc,        NaN;
25                  Tcf,             qfrng,       NaN,        NaN,        NaN;
26                  Boiler.Sw0, Boiler.Sw1, Boiler.Sw2, Boiler.Sw3,        NaN;
27                  qscf,              NaN,        NaN,        NaN,        NaN;
28                  qfcf,              NaN,        NaN,        NaN,        NaN;
29                  Sigma1,            NaN,        NaN,        NaN,        NaN;
30                  Sigma2,            NaN,        NaN,        NaN,        NaN;
31                  Sigma3,            NaN,        NaN,        NaN,        NaN;
32                  Sigma4,            NaN,        NaN,        NaN,        NaN;
33                  Sigma5,            NaN,        NaN,        NaN,        NaN;
34                  Boiler.md,         NaN,        NaN,        NaN,        NaN;
35                  Boiler.mr,         NaN,        NaN,        NaN,        NaN;
36                  Boiler.Ad,         NaN,        NaN,        NaN,        NaN;
37                  Boiler.kf,         NaN,        NaN,        NaN,        NaN;
38                  Boiler.ks,         NaN,        NaN,        NaN,        NaN;
39                  Boiler.L0,         NaN,        NaN,        NaN,        NaN;
40                  Boiler.xi0,        NaN,        NaN,        NaN,        NaN;
41                  Boiler.b1,         NaN,        NaN,        NaN,        NaN;
42                  Boiler.b2,         NaN,        NaN,        NaN,        NaN;
43                  Boiler.dVwt0,      NaN,        NaN,        NaN,        NaN;
44                  Boiler.P0,         NaN,        NaN,        NaN,        NaN;
45                  Boiler.xr0,        NaN,        NaN,        NaN,        NaN];
46      END;
47    END;
48  END;
```

Listing A.2: DrumBoiler/Oresund2.om—An example demonstrating how the signal model mapping is included as a sub-model attribute in the simulation interface; specifically, see lines 6–47 and listing A.1.

```
1   #!/bin/sh
2   # $Id: flat-model2maple,v 1.7 1997/06/24 11:32:55 sorliej Exp $
3   ######################################################################
4   # First process the "flat-model" output.
5   ######################################################################
6   cat $1 |\
7   sed '1,/^model .*;$/d;
8        /^.*= /!d;
9        /^.*= ?;$/d;
10       /^Non-linear equation: .*$/d;
11       s/;$/:/g;
12
13  # Strip lines from the Event part of the log-file.
14
15       /^.*:=.*[^:]$/d;
16
17  # Convert the end-of-line character (^M) to a Maple command terminator (:).
18
19    s/^M//g;
20
21  # Convert any equalities (=) into Maple bindings (:=).
22
23    s/ = / := /g;
24
25  # Convert differential syntax (prime notation) to something unique.
26  # First process vector-valued (indexed) symbols, then scalars.
27
28    s/\(\[[0-9]*\]\)'\''/__prime\1/g;
29    s/'\''/__prime/g;
30
31  # Convert the period in Omola names to an underscore.
32
33    s/\.\([^0-9]\)/_\1/g;
34
35  # Convert semi-colons in between square brackets to commas.
36  # This converts Omola vectors and matrices into Maple lists.
37  # NOTE:  one _should_ be able to program this as a loop.
38
39    s/\[\(.*\);\(.*\)\]/[\1,\2]/g;
40    s/\[\(.*\);\(.*\)\]/[\1,\2]/g;
41    s/\[\(.*\);\(.*\)\]/[\1,\2]/g;


... (duplicate lines deleted) ...


105   s/\[\(.*\);\(.*\)\]/[\1,\2]/g;
106   s/\[\(.*\);\(.*\)\]/[\1,\2]/g;
107   s/\[\(.*\);\(.*\)\]/[\1,\2]/g;
108
109 # Finally, comment out equations designating the OmSim inputs.  This
110 # makes the script-output suitable as "include file" input for Maple.
111
112   s/^\(.*\)\*continuousinput\*.*\,.*\,[ ]*\(.*\)\,.*$//;
113
```

Listing A.3: flat-model2maple—A shell script which translates OmSim's "Flat model" and "Event part" debug information into a format that can be loaded into Maple.

```
114  # Convert conditional assignment statements to piecewise function calls.
115  # Through repitition of the sed command, conditional statements with up to
116  # ten branches are translated into nested two-branch piecewise operators.
117  # NOTE!! This translation breaks if the conditional block is a component
118  # sub-expression of a larger right-hand side expression.  This is a con-
119  # sequence of the Omola conditional statement not having an end delimiter.
120
121    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
122    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
123    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
124    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
125    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
126    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
127    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
128    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
129    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
130    s/^\(.*:=.*\)if\(.*\)then\(.*\)else\(.*\):$/\1 piecewise\(2,\2,\3,\4\):/g
131
132  # Translate function names
133
134    s/acos/arccos/g;
135    s/asin/arcsin/g;
136    s/atan/arctan/g;
137    s/atan2/arctan/g;
138
139  # Finish by translating relational operators.
140
141    s/~=/<>/g;
142    s/^[  ]*//g;
143    /^$/d;
144    s/==/=/g' |\
145  cat
146
147  ########################################################################
148  # Second, process the "Event part" output.
149  ########################################################################
150  # Begin by deleting everything upto the beginning of the Event part.
151  cat $1 |\
152  sed '1,/^ Event model instantiated from .*$/d;
153
154  # Delete the message OmSim outputs under Linux (if necessary).
155       /^Non-linear equation: .*$/d;
156
157  # Delete lines that are not Event-part assignment statements.
158       /^.*:=.*[^;]$/!d;
159
160  # Convert the period in Omola names to an underscore.
161       s/\.\([^0-9]\)/_\1/g;
162
163  # Strip the first hierarchic level from every name in the Event dump.
164       s/[A-Za-z0-9]*_\([_A-Za-z0-9]*\)/\1/g;
165
166  # A more portable form of the previous step is, using GNUs regexps,
167  #    s/[[:alnum:]]+_\([_[:alnum:]]+\)/\1/g;
168
169  # Finally, post-fix a colon to each Event-dump assignment statement.
170       s/^\(.*\)$/\1:/g' |\
171  cat
```

Listing A.3: flat-model2maple (*continued*).

A.2 Symbolic Processing and Code Generation

```
1   # $Id: CodeGen.sh,v 1.10 1997/06/24 11:43:57 sorliej Exp $
2   #-----------------------------------------------------------------#
3   # Needed for LTH-Regler's version of the maple script
4   DISPLAY="";export DISPLAY;
5   #-----------------------------------------------------------------#
6   # 1) Translate the equations in the log-file into Maple syntax.
7   #-----------------------------------------------------------------#
8   echo "Running the OmSim log-file translator."
9   flat-model2maple omsim.log > omsim.mpl
10  #-----------------------------------------------------------------#
11  # 2) Start Maple and process the translated equations.
12  #-----------------------------------------------------------------#
13  echo "Post-processing the equations in Maple."
14  maple -q > prep.log <<"___PostProcess___"
15
16  # Define a utility function and then load the translated equations
17  # and post-translation processing utilities.
18      REVERSE_LST := lst -> map( rel->rhs(rel)=lhs(rel), lst );
19      read `omsim.mpl`;
20      read `/home/sorliej/maple/post-processing-utilities`:
21
22  # Compensate for case-changes made by the Omola compiler.
23      if not assigned(SMM_U) then SMM_U := SMM_u: fi:
24      if not assigned(SMM_X) then SMM_X := SMM_x: fi:
25      if not assigned(SMM_Y) then SMM_Y := SMM_y: fi:
26
27  # Process the signal model mapping definitions.
28      state_mapping := table2mapping(SMM_X,x);
29      input_mapping := table2mapping(SMM_U,u);
30      disturb_mapping := table2mapping(SMM_Wv,wv)
31                      union table2mapping(SMM_Wy,wy);
32      parameter_mapping := table2mapping(SMM_PM,theta):
33      numeric_mapping := defaults2mapping(SMM_PM,theta):
34      name_mapping :=  state_mapping
35                  union  input_mapping
36                  union  disturb_mapping
37                  union  parameter_mapping:
38
39  # Format the parameter matrix and default values.
40      par[default] := defaults2matrix(SMM_PM);
41
42  # Construct lists of state and output equations, substituing indexed vector
43  # names (i.e. x[], u[], wv[], wy[], theta[]) for the hierarchic Omola names.
44      eqns_dX := subs( name_mapping, table2primelist(SMM_X) ):
45      eqns_Y  := subs( name_mapping, table2list(SMM_Y) ):
46
47  # Construct a list of the initial input and state conditions.
48      REVERSE_LST( table2mapping(SMM_U0,u0) ):
49      subs( name_mapping, eval(")):
50      eqns_U0 := convert(table(map(r->op(1,lhs(r))=rhs(r),")), list);
51      table2newmap(SMM_X,x0):
52      subs( name_mapping, eval(")):
53      eqns_X0 := convert(convert(table(map(r->op(1,lhs(r))=rhs(r),")), array), list):
```

Listing A.4: CodeGen.sh—Top-level shell script for translation of an OmSim log file into a Simulink MEX-file S-function and an IdKit model subroutine module.

```
54
55   # If the initial conditions are functions of other initial input
56   # and/or state conditions, make them functions of parameters.
57       indets(eqns_X0,u[integer]);
58       if nops(") <> 0 then
59           REVERSE_LST( table2mapping(SMM_U0,u0) ):
60           eqns_X0 := subs(u=u0, ", parameter_mapping, eqns_X0 ):
61       fi:
62       indets(eqns_X0,x[integer]);
63       if nops(") <> 0 then
64           eqns_X0:
65           {seq(x0[i]=op(i,"),i=1..nops("))}:
66           eqns_X0 := subs( x=x0, ", eqns_X0 ):
67       fi:
68
69   # Identify any remaining indeterminate names and construct their
70   # "new" names.  The equation assignments for these names are
71   # obtained from the Event-part of the log file.  These are the
72   # event equations computed either at the "Init" or "Start" events.
73       indets(eqns_dX,name) union indets(eqns_Y,name) union indets(eqns_X0,name):
74       " minus indets(",indexed);
75       {seq(op(i,")=insert_new(op(i,")), i=1..nops(") )}:
76       subs( name_mapping, eval(") ):
77
78   # Substitute state vector references in the event equations with
79   # the initial state conditions.
80       table2newmap(SMM_X,x0):
81       eval("):
82       subs( x=x0,", parameter_mapping, """ ):
83
84   # Also substitute the initial input values into the relations,
85   # making them functions of only the model parameters.
86       REVERSE_LST( table2mapping(SMM_U0,u0) ):
87       event_eqns := subs( u=u0, ", parameter_mapping, "" ):
88
89   # Check if the event equations involve other event definitions.
90       map(lhs,event_eqns);
91       indets(map(rhs,event_eqns),   name) minus
92       indets(map(rhs,event_eqns),indexed);
93
94   # Iteratively substitute the event definitions into their own
95   # right-hand-side's.  This is necessary to completely reduce them
96   # to functions of the model parameters.
97       tmp := ":
98       for i from 1 to 10 while nops(tmp) <> 0 do
99           print(i);
100          map(r->lhs(r)=subs(event_eqns,rhs(r)), event_eqns ):
101          event_eqns := ":
102          indets(map(rhs,event_eqns),   name) minus
103          indets(map(rhs,event_eqns),indexed):
104          tmp := ":
105      od:
106      if i > 9 then ERROR(`self-referencing parameter equation probable.`): fi;
107      indets(map(rhs,event_eqns), name);
```

Listing A.4: CodeGen.sh (*continued*).

```
103
109  # Finally substitute the relations into the existing definitions
110  # that contain the unassigned names.
111      event_eqns:
112      eqns_dX := subs(   ", eqns_dX ):
113      eqns_Y  := subs(  "", eqns_Y  ):
114      eqns_X0 := subs( """, eqns_X0 ):
115
116  # Convert the lists of equations into functionals.
117      f[m] := unapply( eqns_dX, x, u, wv, theta ):
118      h[m] := unapply( eqns_Y, x, u, wy, theta ):
119      x0[m] := unapply( eqns_X0, theta ):
120      u0[m] := unapply( eqns_U0, theta ):
121
122  # Collect the model definition in a list-of-lists, plus the default
123  # parameter values.  Then turn this into an operator (internally, a
124  # Maple procedure) and save it to a file using Maple's internal
125  # format.  We do this so we can restart the Maple interpreter
126  # "fresh" before beginning the code generation step.
127      [ f[m](x,u,beta,theta),
128       x0[m](theta),
129        h[m](x,u,nu,theta),
130       u0[m](theta), rhs(par[default])]:
131      model := unapply( ", x,u,beta,nu,theta ):
132
133  # Save the model equations and exit.
134      save model, `model.m`:
135      quit:
136  ___PostProcess___
137
138  #------------------------------------------------------------------#
139  # 3) Restart Maple and run the code-generation batch.
140  #------------------------------------------------------------------#
141  echo "Running the Maple code-generation utilities."
142  maple -q > codegen.log <<"___CodeGen___"
143
144  # Reload the signal model, and load the EKF and TplC packages.
145      read `model.m`;
146      read `/home/sorliej/maple/EKF`:
147      read `/home/sorliej/maple/TplC`:
148
149  # Encode the model as a Simulink S-function.
150      model(x,[seq(`u`[i],i=5..9)], [u[1],u[2],u[3],u[4]],[u[11],u[11]], p):
151      SimStructModel( `Smodel`, [op(1..4,")], [x,u,p], evalm("[5]) );
152
153  # Encode the model for use with IdKit.
154      model(xy,u,wv,wy,p):
155      IdKitModel( `model`, [op(1..4,")], [xy,u,wv,wy,p], evalm("[5]) );
156
157  # Derive and encode a CKF predictor as a Simulink S-function.
158  #   model(x,u,w,v,p):
159  #   filter[1,1] := ckf( [1,1], [op(1..3,")], [x,u,w,v,p,X] ):
160  #   SimStructCKF(`Sfilter`, filter[1,1], [x,u,p], evalm(""[5]) );
161
162  # Exit Maple
163      quit:
164  ___CodeGen___
```

Listing A.4: CodeGen.sh (*continued*).

```
1   concat_lists := (L1,L2) -> [op(L1), op(L2)]:
2
3   table2list := tbl -> convert(convert(tbl,array),list):
4
5   list2matrix := lst -> convert(convert(lst,vector),matrix):
6
7   table2mapping := proc(Tbl,Name)
8     local lst, j;
9     if not type(Tbl,table)  then ERROR(`bad arguments`); fi;
10    lst := [ indices(Tbl) ]:
11    RETURN( {seq(Tbl[op(op(j,lst))]=Name[op(op(j,lst))], j=1..nops(lst))} );
12  end:
13
14  # Return a set of equalities from a name-d parameter matrix to default numeric values.
15  defaults2mapping := proc(Tbl,Name)
16    local lst, j;
17    if not type(Tbl,table)  then ERROR(`bad arguments`); fi;
18    lst := [ indices(Tbl) ]:
19    RETURN( {seq(Name[op(op(j,lst))]=eval(cat( Tbl[ op(op(j,lst)) ],
20                  `_default` )), j=1..nops(lst))} );
21  end:
22
23  defaults2matrix := proc(Tbl)
24    local lst, j, m, n, M, N, MaxRowIndex, MaxColIndex;
25    if not type(Tbl,table)  then ERROR(`bad arguments`); fi;
26    lst := [ indices(Tbl) ]:
27    MaxRowIndex := < max(0,seq(op(1,op(i,set)),i=1..nops(set)))|set|i>:
28    MaxColIndex := < max(0,seq(op(2,op(i,set)),i=1..nops(set)))|set|i>:
29    m := MaxRowIndex(convert(lst,set));
30    n := MaxColIndex(convert(lst,set));
31    M := linalg[matrix](m,n,(i,j)->`NaN`):
32    N := linalg[matrix](m,n,(i,j)->`-`):
33    for j from 1 to nops(lst) do
34      Tbl[op(op(j,lst))]:
35      if type(",indexed) or searchtext(`SMM_NaN`,") = 0 then
36        M[op(op(j,lst))] := eval(cat( Tbl[ op(op(j,lst)) ], `_default` )):
37        N[op(op(j,lst))] := substring( eval(Tbl[ op(op(j,lst)) ]), 1..10^4):
38      fi:
39    od:
40    RETURN( evalm(N) = evalm(M) ):
41  end:
42
43  reverse_searchtext := proc(str,c)
44    local i, n:
45    n := length(str):
46    for i from 1 to n
47      while searchtext(c,str,-i..-1) = 0
48        do od:
49    RETURN(n-i+1):
50  end:
```

Listing A.5: post-processing-utilities—A collection of Maple procedures used in processing the OmSim modelling equations once loaded into Maple; see listing A.4.

```
51
52  insert_new := proc(str)
53    local s:
54    if type(str,string)  then
55      reverse_searchtext(str,'_'):
56      cat( substring(str,0.."), 'new_',
57           substring(str,"+1..length(str)) ):
58    elif type(str,indexed) then
59      s := op(0,str):
60      reverse_searchtext(s,'_'):
61      cat( substring(s,0.."), 'new_',
62           substring(s,"+1..length(s)) )[op(str)]:
63    else ERROR('bad arguments'); fi;
64    RETURN("):
65  end:
66
67  postfix_prime := proc(str)
68    local s:
69    if   type(str,string)  then RETURN(cat(      str, '__prime' )):
70    elif type(str,indexed) then RETURN(cat(op(0,str), '__prime' )[op(str)]):
71    else ERROR('bad arguments'); fi;
72  end:
73
74  set2newmap := proc(Set,Name)
75    local lst, j;
76    if not type(Set,set)  then ERROR('bad arguments'); fi;
77    lst := [ indices(Tbl) ]:
78    RETURN({
79      seq(Name[op(op(j,lst))]=insert_new( Tbl[op(op(j,lst))]), j=1..nops(lst)
80    });
81  end:
82
83  table2newmap := proc(Tbl,Name)
84    local lst, j;
85    if not type(Tbl,table)  then ERROR('bad arguments'); fi;
86    lst := [ indices(Tbl) ]:
87    RETURN({
88      seq(Name[op(op(j,lst))]=insert_new( Tbl[op(op(j,lst))]), j=1..nops(lst)
89    });
90  end:
91
92  table2primelist := proc(Tbl)
93    local lst, j;
94    if not type(Tbl,table)  then ERROR('bad arguments'); fi;
95    lst := [ indices(Tbl) ]:
96    RETURN(convert(convert(table({
97      seq(op(op(j,lst))=postfix_prime( Tbl[op(op(j,lst))]), j=1..nops(lst))
98    }),array),list));
99  end:
```

Listing A.5: post-processing-utilities (*continued*).

```
#-------------------------------------------------------------------------#
# $Source: /home/sorliej/maple/src/TplC/RCS/IdKitModel.src,v $
# $Revision: 1.18 $
# $Date: 1997/03/06 08:24:31 $
# $Author: sorliej $
# $Locker:  $
#-------------------------------------------------------------------------#
`type/IdKitModelargs` := {
#  [string, list({list,vector})],
  [string, list({list,vector}), list(string)],
  [string, list({list,vector}), list(string), {list,vector,array,matrix}]
}:
#-------------------------------------------------------------------------#
IdKitModel := proc( FileName, SignalModel, SignalNames, ParameterMatrix )
#-------------------------------------------------------------------------#
 local  dX, Y, X0, N, NList, Pvals, Eqns, EqNames, name_mapping, Fcns,
        numeric_mapping, par_index_mapping, Defines, Includes,
        newlineC, IdKitHead, IdKitBody, IdKitFoot,
        IdKit_initcond, IdKit_init_u, IdKit_yequ, IdKit_xyequ,
        IdKit_setm, IdKit_setp, ij, Coldims, Passign,
# Declare names (strings) used as local in order to avoid evaluation of
# possibly existing variables from the calling (global) namespace.
        i, par, xy, u, wv, wy, y, p, Pm, Pn, feedthru, C, S,
# Declare variables different from the SimStructModel template.
        m, n, U0, t, t0, x0, u0, dx_dt, y_of_t:

  options `© 1996, 1997 J.A.Sørlie, KTH S3-Reglerteknik, Stockholm Sweden`:

  newlineC := [commentC, ``]:

  if not type([args],IdKitModelargs) then
    ERROR(cat(`invalid arguments.  See ?`, procname)):
  fi:

  userinfo(1,'TplCcomment',
    `Parsing the signal model equations, converting fractions to floats.`):

  dX := evalf( convert(SignalModel[1],list) ):  N[xy] := nops(dX):
  userinfo(3,'TplCresult', `state derivatives:`, print(dX)):

  if nops(SignalModel)<2 or type(SignalModel[2],[]) then
    userinfo(2,'TplCcomment',`Assuming initial conditions are zero.`):
    X0 := evalf([seq(0,i=1..N[xy])]):
  else
    X0 := evalf( convert(SignalModel[2],list) ):
    if nops(X0)<>N[xy] then ERROR(`wrong number of initial conditions`) fi:
  fi:
  userinfo(3,'TplCresult', `state initialization:`, print(X0)):

  if nops(SignalModel)<3 or type(SignalModel[3],[]) then
    userinfo(2,'TplCcomment',`Assuming outputs to be state vector.`):
    Y := evalf([seq(`xy`[i],i=1..N[xy])]):       N[y] := N[xy]:
  else
    Y := evalf( convert(SignalModel[3],list) ):  N[y] := nops(Y):
  fi:
  userinfo(3,'TplCresult', `measured outputs:`, print(Y)):

  if nops(SignalModel)<4 or type(SignalModel[4],[]) then
    userinfo(2,'TplCcomment',`Unspecified initial inputs left unassigned.`):
  else
    U0 := evalf( convert(SignalModel[4],list) ):
    userinfo(3,'TplCresult', `input initialization:`, print(U0)):
  fi:
```

Listing A.6: IdKitModel.src—Template that automatically encodes the IdKit model subroutines.

```
64
65      if nops(SignalModel)>=5 then
66        userinfo(2,'TplCcomment',`Ignoring extra equations in the signal model.`):
67      fi:
68
69      if nargs >= 3 and not type(SignalNames,[]) then
70        if nops(SignalNames) >= 5 then
71          NList :=[op(1..5,SignalNames)]:
72          userinfo(1,'TplCcomment',`Using parameter names:`,print(NList)):
73        else
74          ERROR(`expecting a list of signal names e.g. [xy, u, wv, wy, p].`):
75        fi:
76      else
77        NList := [`xy`, `u`, `wv`, `wy`, `p`]:
78        userinfo(1,'TplCcomment',`Using default parameter names:`,print(NList)):
79      fi:
80
81      if nargs >= 4 and not type(ParameterMatrix,[]) then
82        if type(ParameterMatrix,{list,vector}) then
83             Pvals := evalf(convert(convert(ParameterMatrix,vector),matrix)):
84        else Pvals := evalf(convert(ParameterMatrix,matrix)): fi:
85        userinfo(2,'TplCcomment',`Using default parameter values:`,print(Pvals)):
86      else
87        userinfo(2,'TplCcomment',`Unspecified parameter values left unassigned.`):
88      fi:
89
90      userinfo(1,'TplCcomment', `Mapping signal names in the equations.` ):
91      Eqns := { op(dX), op(X0), op(Y) }:
92      if assigned(U0) then Eqns := Eqns union { op(U0) } fi:
93      EqNames := indets(Eqns,name):
94      indets(EqNames,NList[1][integer]):
95        {seq( op(i,")=`xy`[op(1,op(i,"))], i=1..nops(")}:
96      indets(EqNames,NList[2][integer]):
97        "" union {seq(op(i,")=`u`[op(1,op(i,"))],i=1..nops(")}:
98      indets(EqNames,NList[3][integer]):
99        "" union {seq(op(i,")=`wv`[op(1,op(i,"))],i=1..nops(")}:
100     indets(EqNames,NList[4][integer]):
101       "" union {seq(op(i,")=`wy`[op(1,op(i,"))],i=1..nops(")}:
102     indets(EqNames,NList[5][integer,integer]):
103     name_mapping := "" union {seq(op(i,")=`p`[op(1..2,op(i,"))],i=1..nops(")}:
104     userinfo(3,'TplCresult',print('name_mapping'=name_mapping)):
105     dX := subs( name_mapping, dX ):
106     X0 := subs( name_mapping, X0 ):
107      Y := subs( name_mapping, Y  ):
108     Eqns := { op(dX), op(X0), op(Y) }:
109     if assigned(U0) then
110       U0 := subs( name_mapping, U0   ):
111       Eqns := Eqns union { op(U0) }:
112     fi:
113     if assigned(Pvals) then  Eqns := Eqns union convert(Pvals,set): fi:
114
115     EqNames := indets(Eqns,name):
116     userinfo(3,'TplCresult',print('EqNames'=EqNames)):
117
118     userinfo(1,'TplCcomment', 'Determining input vector dimension.`):
119     N[u] := `TplC/MaxRowIndex`( indets(EqNames,`u`[integer]) ):
120     userinfo(2,'TplCresult',print('N[u]'=N[u])):
121     if not assigned(U0) then
122       if N[u] = 0 then U0 := [0] else U0 := evalf([seq(0,i=1..N[u])]) fi:
123     elif nops(U0) <> N[u] then ERROR(
124       `Vector dim of initial input vector not = to the max index in equations.`)
125     fi:
126
```

Listing A.6: IdKltModel.src (*continued*).

```
127   userinfo(1,'TplCcomment', 'Determining stochastic input vector dimensions.'):
128   N[wv] := 'TplC/MaxRowIndex'( indets(EqNames,'wv'[integer]) ):
129   N[wy] := 'TplC/MaxRowIndex'( indets(EqNames,'wy'[integer]) ):
130   userinfo(2,'TplCresult',print('N[wv]'=N[wv],'N[wy]'=N[wy])):
131
132   userinfo(1,'TplCcomment', 'Determining parameter matrix dimensions.' ):
133   indets(EqNames,'p'[integer,integer]):
134   N[Pm] := 'TplC/MaxRowIndex'( " ):
135   N[Pn] := 'TplC/MaxColIndex'( "" ):
136   userinfo(3,'TplCcomment',print('Params'="""")):
137   userinfo(4,'TplCcomment',print('Eqns'=Eqns)):
138   userinfo(2,'TplCcomment',print('N[Pm]'=N[Pm],'N[Pn]'=N[Pn])):
139   if assigned(Pvals) then
140     if linalg[rowdim](Pvals) <> N[Pm] or linalg[coldim](Pvals) <> N[Pn] then
141       ERROR('Dimensions of numerical parameter matrix are wrong.'):
142     fi:
143   else
144     userinfo(0,'TplCwarning', print(''),
145       'WARNING:  filling default numeric parameter matrix with NaN:s.',
146       print('') ):
147     Pvals := matrix(N[Pm],N[Pn],(i,j)->'NaN'):
148     EqNames := EqNames union {'NaN'}:
149   fi:
150
151   userinfo(1,'TplCcomment',
152     'Checking functional structure of the signal model.' ):
153   indets({op(dX)},indexed) minus indets(EqNames,
154     {'xy'[integer],'u'[integer],'wv'[integer],'p'[integer,integer]}):
155   if nops(") <> 0 then ERROR(
156     cat('state equations contain unallowed names:\n\t',convert(",string))):
157   fi:
158   indets({op(Y)},indexed) minus indets(EqNames,
159     {'xy'[integer],'u'[integer],'wy'[integer],'p'[integer,integer]}):
160   if nops(") <> 0 then ERROR(
161     cat('output equations contain unallowed names:\n\t',convert(",string))):
162   fi:
163   indets({op(X0)},indexed) minus indets(EqNames,'p'[integer,integer]):
164   if nops(") <> 0 then ERROR(
165     cat('initial conditions contain unallowed names:\n\t',convert(",string))):
166   fi:
167   indets({op(U0)},indexed) minus indets(EqNames,'p'[integer,integer]):
168   if nops(") <> 0 then ERROR(
169     cat('initial inputs contain unallowed names:\n\t',convert(",string))):
170   fi:
171
172   userinfo(1,'TplCcomment',
173     'Checking for any unassigned variables in the equations.' ):
174   EqNames
175     minus indets(EqNames,{'xy'[integer],'u'[integer],
176       'wv'[integer],'wy'[integer],'p'[integer,integer]})
177     minus { 'EPS', 'TRUE', 'FALSE', 'NaN', epsilon, pi, tau, '' }:
178   if nops(") <> 0 then
179     ERROR(cat('equations contain unassigned names:\n\t',convert(",string))):
180   fi:
181
182   Passign := []:
183   for ij  in [indices(Pvals)] do
184     if type(Pvals[op(ij)],numeric) then
185       Passign := [ op(Passign),
186         [equalC, cat('p(',ij[1]-1,',',ij[2]-1,')'), Pvals[op(ij)] ]]:
187     fi:
188   od;
189   if nops(Passign) < 1 then
```

Listing A.6: IdKitModel.src (*continued*).

```
190       Passign :=
191         [commentC,` No parameter values were given at code-gen time. `]:
192     fi:
193
194     indets(EqNames,`p`[integer,integer]):
195     Coldims := convert([seq(0,i=1..N[Pm])],array):
196     for ij  in "" do
197       Coldims[ op(1,ij) ] := max( Coldims[op(1,ij)], op(2,ij)  ):
198     od:
199
200     userinfo(1,'TplCcomment',
201       `Mapping numeric constants to CPP definitions, e.g. C1, C2, etc.`):
202     convert(indets(Eqns,float),list):
203     numeric_mapping := [seq("[i] = cat(`C`,i), i=1..nops("))]:
204   # numeric_mapping := [seq("[i] = cat(`C`,printf(`%02d`,i)), i=1..nops("))]:
205     dX := subs(numeric_mapping,dX):
206     X0 := subs(numeric_mapping,X0):
207     Y  := subs(numeric_mapping,Y):
208     U0 := subs(numeric_mapping,U0):
209
210     userinfo(1,'TplCcomment', `Constructing list of CPP macro definitions.`):
211     Defines :=
212       [commentC, ` Definitions of numeric constants. `],
213       seq( [`defineC`, cat(`C`,i,`                    `), lhs(numeric_mapping[i])],
214         i=1..nops(numeric_mapping) ),
215       newlineC:
216
217     userinfo(1,'TplCcomment', `Constructing list of include files.`):
218     Includes :=
219       [commentC,
220         ` Define interface for a continuous-time single-block model. `],
221       [includeC, `"cs.h"`]:
222
223     userinfo(1,'TplCcomment',
224       `Building a list of functional operators used in the equations.`):
225     convert( indets(Eqns,function), list ):
226     Fcns := { seq(op(0,"[i]), i=1..nops(")) }:
227     userinfo(2,'TplCcomment',
228       `The equations contain the following functional operators:`,print(Fcns)):
229
230     if member(`piecewise`,Fcns) then
231       userinfo(2,'TplCcomment',
232         `Checking that piecewise() calls have either 3 or 4 arguments.`):
233       convert( indets(Eqns,specfunc(anything,`piecewise`)),list);
234       if nops({seq(nops("[i]), i=1..nops(")) } minus {3,4}) <> 0 then ERROR(
235         cat(`support for piecewise() is limited to if-then-else constructs.\n`,
236           `Its arguments should be: smoothness-order, condition, expr1, expr2`)):
237       fi:
238       Defines := Defines, [defineC,
239         `piecewise(dummy,cond,expr1,expr2)`,`((cond)?(expr1):(expr2))`]:
240       Fcns := Fcns minus {`piecewise`}:
241     fi:
242
243     userinfo(1,'TplCcomment', `Adding define/includes for supported constants.`):
244     if member(`EPS`,EqNames) then
245       Defines  := Defines,  [defineC,  `EPS`, `DBL_EPSILON`]:
246     fi:
247     if member(`epsilon`,EqNames) then
248       Defines  := Defines,  [defineC,  `epsilon`, `DBL_EPSILON`]:
249     fi:
250     if member(`pi`,EqNames) then
251       Defines  := Defines,  [defineC,  `pi`, `M_PI`]:
252     fi:
```

Listing A.6: ldKitModel.src (*continued*).

```
253     if member('tau',EqNames) then
254       Defines  := Defines,  [defineC,  'tau', 'time_c']:
255     fi:
256     if member('NaN',EqNames) then
257       Defines  := Defines,  [defineC,  'NaN', '(0.0*(1.0/0.0))']:
258     fi:
259
260     userinfo(1,'TplCcomment',
261       'Map parameter references to a CPP macro, i.e. p[i,j] -> p(i-1,j-1).'):
262     convert(indets(EqNames,'p'[integer,integer]),list);
263     par_index_mapping :=
264       {seq(op(i,'')='p'(op(1,op(i,''))-1,op(2,op(i,''))-1), i=1..nops(''))}:
265
266     dX := convert( subs(par_index_mapping,dX), vector ):
267     X0 := convert( subs(par_index_mapping,X0), vector ):
268     Y  := convert( subs(par_index_mapping,Y),  vector ):
269     U0 := convert( subs(par_index_mapping,U0), vector ):
270
271     #------------------------------------------------------------------------#
272     userinfo(1,'TplCcomment', 'Defining a template in ''macroC'.'):
273     #------------------------------------------------------------------------#
274
275     i := 'i':
276
277     IdKitHead :=
278     [ [commentC, cat(
279         '*********************************************************************',
280       '\n * FILE:  ', FileName, '.c',
281       '\n * DATE:  ', 'TplC/GetDate'(),
282       '\n *',
283       '\n * To compile, issue the command: "mcompile ', FileName, '.c"',
284       '\n *',
285       '\n * This file was automatically generated using Maple and MacroC,',
286       '\n * a shareware package developed by Patrick Capolsini.  The code',
287       '\n * generation template, part of the TplC package, was developed',
288       '\n * by J.A.Sørlie, KTH S3-Reglerteknik, Stockholm SWEDEN.',
289       '\n *********************************************************************')],
290       newlineC,
291       Includes,
292       newlineC,
293       Defines,
294       newlineC
295     ]:
296     'TplC/debugger'(IdKitHead):
297
298     IdKit_initcond :=
299     [ [commentC, cat(
300         '*********************************************************************',
301       '\n * initcond - user supplied state-vector initialization',
302       '\n *',
303       '\n * \tx0[*] = fcn(t0,p(*,*),constants)',
304       '\n *',
305       '\n * where:',
306       '\n * x0[*]\thas indices 0..',eval(N[xy]-1),
307       '\n * t0\t\tis the start time of the simulation',
308       '\n * p(i,j)\thas indices i=0..',eval(N[Pm]-1),' and j=0..',eval(N[Pn]-1),
309       '\n * constants\tare numerical values or CPP definitions',
310       '\n *********************************************************************')],
311       newlineC,
312       [functionm, 'void', 'initcond',
313         [ ['int   ', ['dim']],
314           ['double', ['t0','*x0'] ] ],
315         [ [matrixm,'x0',X0] ] ],
```

Listing A.6: IdKitModel.src (*continued*).

```
316       newlineC ]:
317     `TplC/debugger`(IdKit_initcond):
318
319     IdKit_init_u :=
320     [ [commentC, cat(
321           `*****************************************************************`,
322       `\n * init_u - user supplied input-vector initialization`,
323       `\n *`,
324       `\n * \tu0[*] = fcn(t0,p(*,*),constants)`,
325       `\n *`,
326       `\n * where:`,
327       `\n * u0[*]\thas indices 0..`,eval(N[u]-1),
328       `\n * t0\t\tis the start time of the simulation`,
329       `\n * p(i,j)\thas indices i=0..`,eval(N[Pm]-1),` and j=0..`,eval(N[Pn]-1),
330       `\n * constants\tare numerical values or CPP definitions`,
331       `\n *****************************************************************`)],
332       newlineC,
333       [functionm, `void`, `init_u`,
334         [ [`double`, [`t0`, `*u0`] ] ],
335         [ [matrixm,`u0`,U0] ] ],
336       newlineC ]:
337     `TplC/debugger`(IdKit_init_u):
338
339     IdKit_yequ :=
340     [ [commentC, cat(
341           `*****************************************************************`,
342       `\n * yequ - user supplied measurement equations`,
343       `\n *`,
344       `\n * \ty_of_t[*] = fcn(xy[*],u[*],wy[*],t0,t,p(*,*))`,
345       `\n *`,
346       `\n * where:`,
347       `\n * y_of_t[*]\thas indices 0..`,eval(N[y]-1),
348       `\n * xy[*]\thas indices 0..`,eval(N[xy]-1),
349       `\n * u[*]\t\thas indices 0..`,eval(N[u]-1),
350       `\n * wy[*]\thas indices 0..`,eval(N[wy]-1),
351       `\n * t0\t\tis the start time of the simulation`,
352       `\n * t\t\tis the current time of the simulation`,
353       `\n * p(i,j)\thas indices i=0..`,eval(N[Pm]-1),` and j=0..`,eval(N[Pn]-1),
354       `\n *****************************************************************`)],
355       newlineC,
356       [functionm, `void`, `yequ`,
357         [ [`double`, [`*y_of_t`, `t0`, `t`] ] ],
358         [ [matrixm,`y_of_t`,Y] ] ],
359       newlineC ]:
360     `TplC/debugger`(IdKit_yequ):
361
362     IdKit_xyequ :=
363     [ [commentC, cat(
364           `*****************************************************************`,
365       `\n * xyequ - user supplied state equations`,
366       `\n *`,
367       `\n * \tdx_dt[*] = fcn(xy[*],u[*],wv[*],t0,t,p(*,*))`,
368       `\n *`,
369       `\n * where:`,
370       `\n * dx_dt[*]\thas indices 0..`,eval(N[xy]-1),
371       `\n * xy[*]\thas indices 0..`,eval(N[xy]-1),
372       `\n * u[*]\t\thas indices 0..`,eval(N[u]-1),
373       `\n * wv[*]\thas indices 0..`,eval(N[wv]-1),
374       `\n * t0\t\tis the start time of the simulation`,
375       `\n * t\t\tis the current time of the simulation`,
376       `\n * p(i,j)\thas indices i=0..`,eval(N[Pm]-1),` and j=0..`,eval(N[Pn]-1),
377       `\n *****************************************************************`)],
378       newlineC,
```

Listing A.6: IdKitModel.src (*continued*).

```
379       [functionm, 'void', 'xyequ',
380         [ ['double', ['*t', '*xy', '*dx_dt'] ] ],
381         [ [matrixm,'dx_dt',dX] ] ],
382       newlineC ]:
383     'TplC/debugger'(IdKit_xyequ):
384
385     IdKit_setm :=
386     [ [commentC, cat(
387         '*****************************************************************',
388       '\n * InitializeSizes - set signal model dimensions',
389       '\n *',
390       '\n * A subroutine which could be used to automate initialization',
391       '\n * of the IdKit model interface.  This is just a proposal.',
392       '\n *****************************************************************')],
393       newlineC,
394       [functionm, 'static void', 'InitializeSizes', [ ],
395         [ [equalC, 'V.dim   ', '0'   ],
396           [equalC, 'Z.dim   ', '0'   ],
397           [equalC, 'Y.dim   ', N[y]  ],
398           [equalC, 'X[0].dim', '0'   ],
399           [equalC, 'X[1].dim', '0'   ],
400           [equalC, 'X[2].dim', N[xy] ],
401           [equalC, 'Wv.dim  ', N[wv] ],
402           [equalC, 'Wy.dim  ', N[wy] ],
403           [equalC, 'U.dim   ', N[u]  ],
404           [equalC, 'P.nvect ', N[Pm] ],
405           [matrixm,'P.dim',    op(Coldims)]
406         ]
407       ],
408       newlineC ]:
409     'TplC/debugger'(IdKit_setm):
410
411     IdKit_setp :=
412     [ [commentC, cat(
413         '*****************************************************************',
414       '\n * InitializeParameters - set model's default parameters',
415       '\n *',
416       '\n * A subroutine which could be used to automate initialization',
417       '\n * of the IdKit model interface.  This is just a proposal.',
418       '\n *****************************************************************')],
419       newlineC,
420       [functionm, 'static void', 'InitializeParameters', [], [Passign]],
421       newlineC ]:
422     'TplC/debugger'(IdKit_setp):
423
424     IdKitBody :=
425       IdKit_initcond,
426       IdKit_init_u,
427       IdKit_yequ,
428       IdKit_xyequ,
429       IdKit_setm,
430       IdKit_setp:
431
432     IdKitFoot := [commentC, cat(
433       '**********************  End-Of-File  *************************')],
434
435     userinfo(1,'TplCcomment', 'Running the ''macroC' code generator.'):
436     genC(0,[IdKitHead, IdKitBody, IdKitFoot], filename=cat(FileName,'.c')):
437     userinfo(1,'TplCcomment', 'Code generation completed successfully!'):
438
439     RETURN(NULL):
440   end:
```

Listing A.6: IdKitModel.src (*continued*).

```
1   /*********************************************************************
2    * xyequ - user supplied state equations
3    *
4    *       dx_dt[*] = fcn(xy[*],u[*],wv[*],t0,t,p(*,*))
5    *
6    * where:
7    * dx_dt[*]      has indices 0..6
8    * xy[*]         has indices 0..6
9    * u[*]          has indices 0..4
10   * wv[*]         has indices 0..2
11   * t0            is the start time of the simulation
12   * t             is the current time of the simulation
13   * p(i,j)        has indices i=0..22 and j=0..4
14   *********************************************************************/
15  void xyequ(t,xy,dx_dt)
16  double *t,*xy,*dx_dt;
17    {
18    static double t2,t3,t16,t21,t22,t23,t24,t32,t37,t41,t42,t47,t48,t50,t51,t52,
19  t54,t57,t59,t64,t69,t70,t71,t74,t81,t82,t87,t90,t94,t97,t104,t107,t111,t116,
20  t118,t121,t123,t124,t126,t127,t128,t129,t131,t132,t134,t140,t147,t152,t153,t154
21  ,t166,t168,t176,t180,t188,t189,t196;
22    t2 = 1/p(2,0);
23    t3 = sqrt(t2);
24    t16 = u[2]*(p(5,0)*C2+p(2,1)*atan(xy[0])*C1);
25    t21 = C7+xy[5]*C57;
26    t22 = (xy[5]*C5+C6)*t21;
27    t23 = t22+C24;
28    t24 = 1/t23;
29    t32 = p(4,0)*(u[0]+u[1]);
30    t37 = p(15,0)*(xy[5]+p(21,0)*C57);
31    t41 = (xy[5]*C19+C20)*t21;
32    t42 = t41+C21;
33    t47 = t16+u[3]*(u[4]*C4+xy[5]*t24*C8)*C9+(t32+t37)*t42*C57+p(6,0)*xy[1];
34    t48 = p(1,1);
35    t50 = p(1,3);
36    t51 = p(1,2);
37    t52 = t48+xy[4]*C57+t50+t51;
38    t54 = xy[5]*C22+C23;
39    t57 = xy[5]*C10+C11;
40    t59 = t52*t54+xy[4]*t57;
41    t64 = u[3]*C9+t32*C57+t37*C57;
42    t69 = xy[5]+C14;
43    t70 = (xy[5]*C12+C13)*t69;
44    t71 = t70+C15;
45    t74 = C16+xy[5]*C17;
46    t81 = (C18+xy[5]*C25)*t69;
47    t82 = t81+C26;
48    t87 = C27+xy[5]*C28;
49    t90 = p(12,0);
50    t94 = C29+xy[5]*C30;
51    t97 = t52*t42*t54+t52*t71*t74+t48*C57+t50*C57+t51*C57+xy[4]*t82*t57+xy[4]*t23
52  *t87+(p(11,0)+t90+p(1,0))*t94*C31;
53    t104 = t23*t82+t71*t42*C57;
54    t107 = t22+C32+t70*C57;
55    t111 = 1/(t59*t104+t107*t97*C57);
56    t116 = t64*t104+t47*t107*C57;
```

Listing A.7: Example of the sub-expression optimization, excerpted from the IdKit subroutine module produced by the TplC `IdKitModel` code generation template; cf. lines 363–377 of listing A.6.

```
57      t118 = t23*t23;
58      t121 = C33+p(17,0)*C57;
59      t123 = t107*xy[6];
60      t124 = 1/t71;
61      t126 = t123*t124+C33;
62      t127 = log(t126);
63      t128 = 1/t107;
64      t129 = t127*t128;
65      t131 = 1/xy[6]*t71;
66      t132 = t131*C57;
67      t134 = C33+t129*t132;
68      t140 = sqrt(t118*t121*t134*t51/p(14,0));
69      t147 = t41+C35+t81*C57;
70      t152 = t51*t121;
71      t153 = t152*t23;
72      t154 = t128*t134;
73      t166 = t154*C57;
74      t168 = t121*t23;
75      t176 = t51*(t168*t154+C36);
76      t180 = t70+C15+t123;
77      t188 = t23*t124;
78      t189 = 1/t126;
79      t196 = t107*t107;
80      dx_dt[0] = t3*wv[0]*C56+t2*xy[0]*C57;
81      dx_dt[1] = wv[1];
82      dx_dt[2] = wv[2];
83      dx_dt[3] = wv[2];
84      dx_dt[4] = (t47*t59+t64*t97*C57)*t111;
85      dx_dt[5] = t116*t111;
86      dx_dt[6] = (t16+(t140*C34+p(8,0)*xy[3])*xy[6]*t147*C57+p(7,0)*xy[2]+(t153*
87    t154*t71*t74+t152*t23*t134*t128*(C33+xy[6]*C57)*t147*t54+t153*t166+t51*(C33+
88    t168*t166)*t23*t87+t176*xy[6]*t147*t57+t176+t51*t180*t147*t121*(t23*t54+t71*t57
89    *C57)*(t188*t189+(t188+C33)*t127*t128*t132+C33)/t196+t90*t94*C31)*t116*t111*C57
90    )/t51/t180/t147/t121*t24/(t129*t131+t189*C57)*t107*xy[6];
91      }
```

Listing A.7: (*continued*)

References

[1] J. A. Sørlie. *On Grey-Box Model Definition and Symbolic Derivation of Extended Kalman Filters.* Ph.d. thesis TRITA–REG–9601, S3–Automatic Control, Royal Institute of Technology, Stockholm, Sweden, 1996.

[2] M. Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems.* Ph.D. thesis TFRT–1043–SE, Dept. of Automatic Control, Lund Institute of Technology, Sweden, 1994.

[3] S. F. Graebe. *Theory and Implementation of Gray Box Identification.* Ph.D. thesis TRITA–REG–9006, Dept. of Automatic Control, Royal Institute of Technology, Stockholm, Sweden, 1990.

[4] K. J. Åström and R. D. Bell. Simple drum-boiler models. In *IFAC Symp. Power Systems: Modelling and Control Applications*, Brussels, 1988.

[5] J. Eborn and J. Sørlie. Parameter optimization of a non-linear boiler model. In *Proc. IMACS World Congress*, Berlin, Aug 1997.

[6] J. Sørlie and J. Eborn. A grey-box identification case study: The Åström–Bell drum-boiler model. Technical Report TFRT–7563–SE, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1998.

[7] K. J. Åström and R. D. Bell. A nonlinear model for steam generation processes. In *IFAC World Congress*, Sydney, 1993.

[8] R. D. Bell and K. J. Åström. A fourth order non-linear model for drum-boiler dynamics. In *IFAC World Congress*, San Francisco, 1996.

[9] R. D. Bell and K. J. Åström. Drum-boiler dynamics. *Submitted to Automatica*, 1997.

[10] B. Nilsson and J. Eborn. K2 Model Database - Tutorial and Reference Manual. Technical Report TFRT–7528–SE, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1994. Available at URL: `http://control.lth.se/~cace/`.

[11] ISI. *SystemBuild User's Guide.* Integrated Systems Inc., 1994.

[12] MWI. *Simulink 1.3 Release Notes.* MathWorks Inc., 1994.

[13] ISI. *AutoCode User's Guide.* Integrated Systems, Inc., 1994.

[14] P. Capolsini. *MacroC – génération de code C depuis Maple.* INRIA – Université de Nice, Laboratoire I3S, June 1992. In Maple, type `with(share): readshare(macroC,numerics):`.

[15] MWI. *Real-Time Workshop User's Guide.* MathWorks, Inc., 1994.

[16] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Library Reference Manual.* Springer Verlag, 1991.