

# CHAPTER 16 A CONDITIONAL MODELING EXAMPLE: REPRESENTING A SUPERSTRUCTURE

To give an example of the application of the conditional modeling tool in ASCEND -the WHEN statement-, we developed a simplified model for the superstructure given in Figure 16-1. The code listed below exists in a file in the ASCEND models subdirectory entitled *when\_demo.a4c*. You could run this example by loading this file and using it and its corresponding script *when\_demo.a4s*.

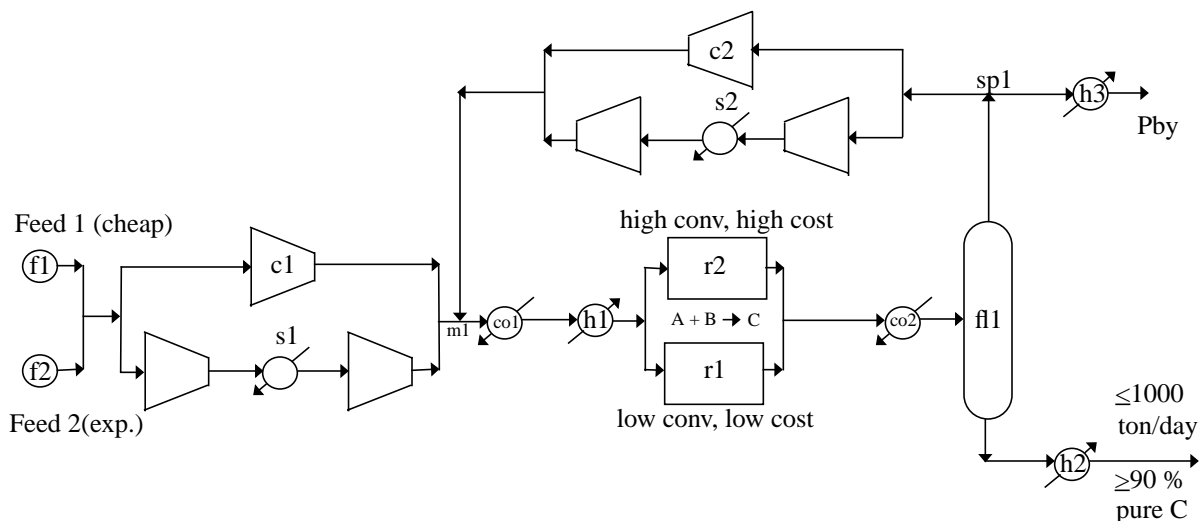


Figure 16-1 Superstructure used in the example of the application of the when statement

## 16.1 THE WHEN STATEMENT

Before showing the example, we want to start by giving a brief explanation about the semantics of the WHEN statement, a tool which allows ASCEND to represent conditional models efficiently.

In the WHEN statement, we take advantage of the fact that ASCEND is based on object oriented concepts where model definitions can contain parts that contain parts to any level. Furthermore, in ASCEND, a simple

relation is treated as an object by itself and can have a name. Based on these ideas, the syntax for the WHEN statement is:

```

WHEN (list_of_variables)
  CASE list_of_values_1:
    USE name_of_equation_1;
    USE name_of_model_1;
  CASE list_of_values_2:
    USE name_of_equation_2;
    USE name_of_model_2;
  CASE list_of_values_nminus1:
    USE name_of_equation_nminus1;
    USE name_of_model_nminus1;
  OTHERWISE:
    USE name_of_equation_n;
    USE name_of_model_n;
END;

```

The following are important observations about the implementation:

- 1 The WHEN statement does not mean conditional compilation. We create and have available the data structures for all of the variables and equations in each of the models. This is actually a requirement for the solution algorithms of conditional models. All the models and equations whose name is given in each of the cases should be declared inside the model which contains the WHEN statement.
- 2 The variables in the list of variables can be of any type among boolean, integer or symbol or any combination of them. That is, we are not limited to the use of boolean variables. Obviously, The list of values in each case must be in agreement with the list of variables in the number of elements and type of each element. In other words, order matters in the list of variables of the WHEN statement, and parentheses are enclosing this list to make clear such a feature.
- 3 Names of arrays of models or equations are also allowed inside the scope of each CASE.

The WHEN statement represents an important contribution to modeling: it allows the user to define the domain of validity of both *models* and *equations* inside the cases of a WHEN statement. This feature enormously increases the scope of modeling in an equation based modeling environment.

Mainly, there are two different ways in which the WHEN statement can be used.:

- First, the WHEN statement can be used to select a configuration of a problem among several alternative configurations.
- Second, in combination with logical relations, the WHEN statement can be used for conditional programming. That is, a problem in which the system of equations to be solved depends on the solution of the problem. A typical example of this situation is the laminar-turbulent flow transition. The selection of the equation to calculate the friction factor depends on the value of the Reynolds number, which is an unknown in the problem.

## 16.2 THE PROBLEM DESCRIPTION

In the example, there are two alternative feedstocks, two possible choices of the reactor and two choices for each of the compression systems. The user has to make 4 decisions (for example, using either the cheap feed or the expensive feed), therefore, there are  $2^4 = 16$  feasible configurations of the problem. All these 16 configurations are encapsulated in one ASCEND model containing 4 WHEN statements which depend on the value of 4 boolean variables.

The value of the four boolean variables will determine the structure of the problem to be solved. In this example, those values are defined by the modeler, but they also could be defined by some logic inference algorithm which would allow the automatic change of the structure of the problem.

The following section gives the code for this model. The first models correspond to the different types of unit operations existing in the superstructure. Those model are very simplified. You may want to skip them and analyze only the model *flowsheet*, in which the use and syntax of the WHEN statement as well as the configuration of the superstructure become evident.

## 16.3 THE CODE

As the code is in our ASCEND examples subdirectory, it has header information that we required of all such files included as one large comment extending over several lines. Comments are in the form (\* comment \*). The last item in this header information is a list of the files one must load before loading this one, i.e., *system.a4l* and *atoms.a4l*.

```

REQUIRE "atoms.a4l";                                     35
(* --> measures,system *)                                36
PROVIDE "when_demo.a4c";                                 37
(*****\                                                 38
    when_demo.a4c                                        39
    by Vicente Rico-Ramirez                             40
    Part of the Ascend Library                           41
                                                    42
This file is part of the Ascend modeling library.        43
                                                    44
The Ascend modeling library is free software; you can redistribute 45
it and/or modify it under the terms of the GNU General Public License as 46
published by the Free Software Foundation; either version 2 of the      47
License, or (at your option) any later version.          48
                                                    49

The Ascend Language Interpreter is distributed in hope that it will be   50
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of   51
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU       52
General Public License for more details.                   53
                                                    54

You should have received a copy of the GNU General Public License along with55
the program; if not, write to the Free Software Foundation, Inc., 675    56
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.       57
                                                    58

Use of this module is demonstrated by the associated script file          59
when_demo.a4s.                                                    60
\*****\                                                 61
                                                    62
(*****\                                                 63
    $Date: 1998/05/14 21:39:44 $                                       64
    $Revision: 1.5 $                                                    65
    $Author: rv2a $                                                     66
    $Source: /afs/cs.cmu.edu/project/ascend/Repository/models/when_demo.a4c,v $67
\*****\                                                 68
                                                    69
(*                                                                 70
    This model is intended to demonstrate the degree of flexibility      71
    that the use of conditional statements -when statement- provides      72
    to the representation of superstructures. We hope that this          73
    application will become clear by looking at the MODEL flowsheet,     74
    in which the existence/nonexistence of some of the unit operations    75
    is represented by when statements. A particular combination of       76
    user defined boolean variables -see method values, configuration2,    77
    configuration3- will a define a particular configuration of the       78
    problem.                                                            79
                                                    80

```

```

This model requires:
    "system.a41"
    "atoms.a41"
*)
(* ***** *)

MODEL mixture;

    components          IS_A set OF symbol_constant;
    Cpi[components]    IS_A heat_capacity;
    y[components]      IS_A fraction;
    P                   IS_A pressure;
    T                   IS_A temperature;
    Cp                  IS_A heat_capacity;

    SUM[y[i] | i IN components] = 1.0;
    Cp = SUM[Cpi[i] * y[i] | i IN components];

METHODS

    METHOD default_self;
    END default_self;

    METHOD specify;
        Cpi[components].fixed := TRUE;
        P.fixed := TRUE;
        T.fixed := TRUE;
        y[components].fixed := TRUE;
        y[CHOICE[components]].fixed := FALSE;
    END specify;

END mixture;

(* ***** *)

MODEL molar_stream;

    state          IS_A mixture;
    Ftot,f[components] IS_A molar_rate;
    components     IS_A set OF symbol_constant;
    P              IS_A pressure;
    T              IS_A temperature;
    Cp             IS_A heat_capacity;

```

```

components, state.components      ARE_THE_SAME;      127
P, state.P                        ARE_THE_SAME;      128
T, state.T                        ARE_THE_SAME;      129
Cp, state.Cp                      ARE_THE_SAME;      130
                                  131
FOR i IN components CREATE        132
    f_def[i]: f[i] = Ftot*state.y[i]; 133
END FOR;                          134
                                  135
METHODS                            136
                                  137
METHOD default_self;              138
END default_self;                 139
                                  140
METHOD specify;                   141
    RUN state.specify;            142
    state.y[components].fixed := FALSE; 143
    f[components].fixed := TRUE;   144
END specify;                      145
                                  146
END molar_stream;                 147
                                  148
(* ***** *)                    149
                                  150
                                  151
MODEL cheap_feed;                 152
    stream          IS_A molar_stream; 153
    cost_factor     IS_A cost_per_mole; 154
    cost            IS_A cost_per_time; 155
                                  156
    stream.f['A'] = 0.060 {kg_mole/s}; 157
    stream.f['B'] = 0.025 {kg_mole/s}; 158
    stream.f['D'] = 0.015 {kg_mole/s}; 159
    stream.f['C'] = 0.00 {kg_mole/s}; 160
    stream.T = 300 {K};            161
    stream.P = 5 {bar};            162
                                  163
    cost = cost_factor * stream.Ftot; 164
METHODS                            165
                                  166
METHOD default_self;              167
END default_self;                 168
                                  169
METHOD specify;                   170
    RUN stream.specify;           171
    stream.f[stream.components].fixed := FALSE; 172

```

```

        cost_factor.fixed := TRUE;           173
        stream.T.fixed := FALSE;           174
        stream.P.fixed := FALSE;           175
    END specify;                             176
END cheap_feed;                             177
                                           178
                                           179
                                           180
(* ***** *)                             181
                                           182
MODEL expensive_feed;                       183
    stream          IS_A molar_stream;      184
    cost_factor     IS_A cost_per_mole;     185
    cost            IS_A cost_per_time;     186
                                           187
    stream.f['A'] = 0.065 {kg_mole/s};      188
    stream.f['B'] = 0.030 {kg_mole/s};      189
    stream.f['D'] = 0.05 {kg_mole/s};      190
    stream.f['C'] = 0.00 {kg_mole/s};      191
    stream.T = 320 {K};                     192
    stream.P = 6 {bar};                     193
                                           194
    cost = 3 * cost_factor * stream.Ftot;   195
                                           196
METHODS                                       197
                                           198
    METHOD default_self;                      199
    END default_self;                       200
                                           201
    METHOD specify;                          202
        RUN stream.specify;                 203
        stream.f[stream.components].fixed := FALSE; 204
        cost_factor.fixed := TRUE;         205
        stream.T.fixed := FALSE;           206
        stream.P.fixed := FALSE;           207
    END specify;                             208
                                           209
END expensive_feed;                         210
                                           211
(* ***** *)                             212
                                           213
                                           214
MODEL heater;                               215
    input,output   IS_A molar_stream;      216
    heat_supplied  IS_A energy_rate;       217
    components     IS_A set OF symbol_constant; 218

```

```

cost          IS_A cost_per_time;          219
cost_factor   IS_A cost_per_energy;       220
                                                    221
components, input.components, output.components ARE_THE_SAME; 222
FOR i IN components CREATE                223
    input.state.Cpi[i], output.state.Cpi[i] ARE_THE_SAME;    224
END FOR;                                    225
                                                    226
FOR i IN components CREATE                227
    input.f[i] = output.f[i];             228
END FOR;                                    229
                                                    230
input.P = output.P;                        231
                                                    232
heat_supplied = input.Cp *(output.T - input.T) * input.Ftot; 233
                                                    234
cost = cost_factor * heat_supplied;       235
                                                    236
METHODS                                     237
                                                    238
METHOD default_self;                       239
END default_self;                          240
                                                    241
METHOD specify;                            242
    RUN input.specify;                     243
    cost_factor.fixed := TRUE;             244
    heat_supplied.fixed := TRUE;          245
END specify;                               246
                                                    247
METHOD seqmod;                             248
    cost_factor.fixed := TRUE;            249
    heat_supplied.fixed := TRUE;          250
END seqmod;                                251
                                                    252
END heater;                                253
                                                    254
(* ***** *)                             255
                                                    256
MODEL cooler;                              257
                                                    258
input, output    IS_A molar_stream;       259
heat_removed     IS_A energy_rate;        260
components       IS_A set OF symbol_constant; 261
cost             IS_A cost_per_time;      262
cost_factor      IS_A cost_per_energy;    263
                                                    264

```



```

components,input.components,output.components      ARE_THE_SAME;      265
FOR i IN components CREATE                          266
  input.state.Cpi[i],output.state.Cpi[i]          ARE_THE_SAME;      267
END FOR;                                           268
                                                    269
FOR i IN components CREATE                          270
  input.f[i] = output.f[i];                       271
END FOR;                                           272
                                                    273
input.P = output.P;                                274
heat_removed = input.Cp *(input.T - output.T) * input.Ftot; 275
cost = cost_factor * heat_removed;                276
                                                    277
METHODS                                           278
                                                    279
METHOD default_self;                               280
END default_self;                                  281
                                                    282
METHOD specify;                                    283
  RUN input.specify;                               284
  cost_factor.fixed := TRUE;                       285
  heat_removed.fixed := TRUE;                     286
END specify;                                       287
                                                    288
METHOD seqmod;                                     289
  cost_factor.fixed := TRUE;                       290
  heat_removed.fixed := TRUE;                     291
END seqmod;                                       292
                                                    293
END cooler;                                       294
                                                    295
(* ***** *)                                    296
                                                    297
MODEL single_compressor; (* Adiabatic Compression *) 298
                                                    299
input,output      IS_A molar_stream;              300
components        IS_A set OF symbol_constant;   301
work_supplied     IS_A energy_rate;              302
pressure_rate     IS_A factor;                   303
R                 IS_A gas_constant;             304
cost              IS_A cost_per_time;            305
cost_factor       IS_A cost_per_energy;          306
                                                    307
components,input.components,output.components      ARE_THE_SAME;      308
FOR i IN components CREATE                          309

```

```

    input.state.Cpi[i],output.state.Cpi[i]          ARE_THE_SAME;          311
END FOR;                                           312
                                                    313
FOR i IN components CREATE                        314
    input.f[i] = output.f[i];                    315
END FOR;                                           316
                                                    317
pressure_rate = output.P / input.P;              318
                                                    319
output.T = input.T * (pressure_rate ^(R/input.Cp) ); 320
                                                    321
work_supplied = input.Ftot * input.Cp * (output.T - input.T); 322
                                                    323
cost = cost_factor * work_supplied;              324
                                                    325
METHODS                                           326
                                                    327
METHOD default_self;                              328
END default_self;                                329
                                                    330
METHOD specify;                                   331
    RUN input.specify;                            332
    cost_factor.fixed := TRUE;                    333
    pressure_rate.fixed := TRUE;                  334
END specify;                                     335
                                                    336
METHOD seqmod;                                    337
    cost_factor.fixed := TRUE;                    338
    pressure_rate.fixed := TRUE;                  339
END seqmod;                                      340
                                                    341
END single_compressor;                            342
                                                    343
(* ***** *)                                   344
                                                    345
MODEL staged_compressor;                          346
                                                    347
input,output          IS_A molar_stream;         348
components            IS_A set OF symbol_constant; 349
work_supplied         IS_A energy_rate;          350
heat_removed          IS_A energy_rate;          351
T_middle              IS_A temperature;          352
n_stages              IS_A factor;               353
pressure_rate         IS_A factor;               354
stage_pressure_rate   IS_A factor;               355
R                    IS_A gas_constant;         356

```

```

cost                IS_A cost_per_time;           357
cost_factor_work    IS_A cost_per_energy;        358
cost_factor_heat    IS_A cost_per_energy;        359
                                                            360
components,input.components,output.components    ARE_THE_SAME; 361
FOR i IN components CREATE                               362
  input.state.Cpi[i],output.state.Cpi[i]          ARE_THE_SAME; 363
END FOR;                                               364
                                                            365
FOR i IN components CREATE                               366
  input.f[i] = output.f[i];                         367
END FOR;                                               368
                                                            369
output.T = input.T;                                    370
                                                            371
pressure_rate = output.P / input.P;                   372
                                                            373
stage_pressure_rate =(pressure_rate)^(1.0/n_stages); 374
                                                            375
T_middle = input.T * (stage_pressure_rate ^ (R/input.Cp)); 376
                                                            377
work_supplied = input.Ftot * n_stages * input.Cp *   378
  (T_middle - input.T);                               379
                                                            380
heat_removed = input.Ftot * (n_stages - 1.0) *      381
  input.Cp * (T_middle - input.T);                   382
                                                            383
cost = cost_factor_work * work_supplied +           384
  cost_factor_heat * heat_removed;                   385
                                                            386
METHODS                                                387
                                                            388
METHOD default_self;                                   389
END default_self;                                     390
                                                            391
METHOD specify;                                        392
  RUN input.specify;                                  393
  n_stages.fixed := TRUE;                             394
  cost_factor_heat.fixed := TRUE;                     395
  cost_factor_work.fixed := TRUE;                     396
  pressure_rate.fixed := TRUE;                        397
END specify;                                          398
                                                            399
METHOD seqmod;                                        400
  n_stages.fixed := TRUE;                             401
  cost_factor_heat.fixed := TRUE;                     402

```

```

        cost_factor_work.fixed := TRUE;           403
        pressure_rate.fixed := TRUE;           404
    END seqmod;                                   405
                                                406
END staged_compressor;                           407
                                                408

(* ***** *)                                  409
                                                410
MODEL mixer;                                     411
                                                412

    components          IS_A set OF symbol_constant; 413
    n_inputs            IS_A integer_constant;      414
    feed[1..n_inputs], out IS_A molar_stream;      415
    To                  IS_A temperature;          416
                                                417

    components,feed[1..n_inputs].components,      418
    out.components     ARE_THE_SAME;              419
    FOR i IN components CREATE                    420
        feed[1..n_inputs].state.Cpi[i],out.state.Cpi[i] ARE_THE_SAME; 421
    END FOR;                                       422
                                                423

    FOR i IN components CREATE                    424
        cmb[i]: out.f[i] = SUM[feed[1..n_inputs].f[i]]; 425
    END FOR;                                       426
                                                427

    SUM[(feed[i].Cp *feed[i].Ftot * (feed[i].T - To))|i IN [1..n_inputs]]= 428
        out.Cp *out.Ftot * (out.T - To);          429
                                                430

    SUM[( feed[i].Ftot * feed[i].T / feed[i].P )|i IN [1..n_inputs]] = 431
        out.Ftot * out.T / out.P;                432
                                                433

METHODS                                           434
                                                435

    METHOD default_self;                           436
    END default_self;                              437
                                                438

    METHOD specify;                                439
        To.fixed := TRUE;                         440
        RUN feed[1..n_inputs].specify;           441
    END specify;                                   442
                                                443

    METHOD seqmod;                                 444
        To.fixed := TRUE;                         445
    END seqmod;                                   446
                                                447

END mixer;                                        448

```

```

(* ***** *)
MODEL splitter;

    components          IS_A set OF symbol_constant;
    n_outputs           IS_A integer_constant;
    feed, out[1..n_outputs] IS_A molar_stream;
    split[1..n_outputs] IS_A fraction;

    components, feed.components,
        out[1..n_outputs].components ARE_THE_SAME;
    feed.state,
        out[1..n_outputs].state ARE_THE_SAME;

    FOR j IN [1..n_outputs] CREATE
        out[j].Ftot = split[j]*feed.Ftot;
    END FOR;

    SUM[split[1..n_outputs]] = 1.0;

METHODS

    METHOD default_self;
    END default_self;

    METHOD specify;
        RUN feed.specify;
        split[1..n_outputs-1].fixed:=TRUE;
    END specify;

    METHOD seqmod;
        split[1..n_outputs-1].fixed:=TRUE;
    END seqmod;

END splitter;

(* ***** *)

MODEL cheap_reactor;

    components          IS_A set OF symbol_constant;
    input, output       IS_A molar_stream;
    low_turnover        IS_A molar_rate;
    stoich_coef[input.components] IS_A factor;
    cost_factor         IS_A cost_per_mole;
```

```

cost                IS_A cost_per_time;                495
                                                            496
components,input.components, output.components        ARE_THE_SAME;    497
FOR i IN components CREATE                               498
    input.state.Cpi[i], output.state.Cpi[i]          ARE_THE_SAME;    499
END FOR;                                                500
                                                            501
FOR i IN components CREATE                               502
    output.f[i] = input.f[i] + stoich_coef[i]*low_turnover; 503
END FOR;                                                504
                                                            505
input.T = output.T;                                     506
(* ideal gas constant volume *)                         507
input.Ftot * input.T / input.P = output.Ftot * output.T/output.P; 508
                                                            509
cost = cost_factor * low_turnover;                     510
                                                            511
METHODS                                                  512
                                                            513
METHOD default_self;                                    514
END default_self;                                       515
                                                            516
METHOD specify;                                         517
    RUN input.specify;                                   518
    low_turnover.fixed:= TRUE;                          519
    stoich_coef[input.components].fixed:= TRUE;         520
    cost_factor.fixed := TRUE;                          521
END specify;                                            522
                                                            523
METHOD seqmod;                                          524
    low_turnover.fixed:= TRUE;                          525
    stoich_coef[input.components].fixed:= TRUE;         526
    cost_factor.fixed := TRUE;                          527
END seqmod;                                             528
                                                            529
END cheap_reactor;                                      530
                                                            531
                                                            532
(* ***** *)                                         533
                                                            534
MODEL expensive_reactor;                                535
                                                            536
components                IS_A set OF symbol_constant; 537
input, output             IS_A molar_stream;           538
high_turnover             IS_A molar_rate;             539
stoich_coef[input.components] IS_A factor;            540

```

```

cost_factor          IS_A cost_per_mole;          541
cost                 IS_A cost_per_time;         542
                                                            543
components,input.components, output.components ARE_THE_SAME; 544
FOR i IN components CREATE                               545
  input.state.Cpi[i], output.state.Cpi[i]           ARE_THE_SAME; 546
END FOR;                                               547
                                                            548
FOR i IN components CREATE                               549
  output.f[i] = input.f[i] + stoich_coef[i]*high_turnover; 550
END FOR;                                               551
                                                            552
input.T = output.T;                                    553
(* ideal gas constant volume *)                        554
input.Ftot * input.T / input.P = output.Ftot * output.T/output.P; 555
                                                            556
cost = cost_factor * high_turnover;                   557
                                                            558
METHODS                                                559
                                                            560
METHOD default_self;                                   561
END default_self;                                     562
                                                            563
METHOD specify;                                       564
  RUN input.specify;                                  565
  high_turnover.fixed:= TRUE;                         566
  stoich_coef[input.components].fixed:= TRUE;        567
  cost_factor.fixed := TRUE;                          568
END specify;                                          569
                                                            570
METHOD seqmod;                                        571
  high_turnover.fixed:= TRUE;                         572
  stoich_coef[input.components].fixed:= TRUE;        573
  cost_factor.fixed := TRUE;                          574
END seqmod;                                          575
                                                            576
END expensive_reactor;                                577
                                                            578
(* ***** *)                                       579
                                                            580
MODEL flash;                                          581
                                                            582
  components          IS_A set OF symbol_constant;  583
  feed,vap,liq       IS_A molar_stream;            584
  alpha[feed.components] IS_A factor;              585
  ave_alpha          IS_A factor;                  586

```

```

vap_to_feed_ratio      IS_A fraction;                               587
                                                                588
components,feed.components,                                       589
vap.components,                                                 590
liq.components          ARE_THE_SAME;                             591
FOR i IN components CREATE                                       592
    feed.state.Cpi[i],                                          593
    vap.state.Cpi[i],                                          594
        liq.state.Cpi[i]   ARE_THE_SAME;                         595
END FOR;                                                         596
                                                                597
vap_to_feed_ratio*feed.Ftot = vap.Ftot;                          598
                                                                599
FOR i IN components CREATE                                       600
    cmb[i]: feed.f[i] = vap.f[i] + liq.f[i];                   601
    eq[i]:  vap.state.y[i]*ave_alpha = alpha[i]*liq.state.y[i]; 602
END FOR;                                                         603
                                                                604
feed.T = vap.T;                                                 605
feed.T = liq.T;                                                 606
feed.P = vap.P;                                                 607
feed.P = liq.P;                                                 608
                                                                609
METHODS                                                         610
                                                                611
METHOD default_self;                                           612
END default_self;                                             613
                                                                614
METHOD specify;                                               615
    RUN feed.specify;                                          616
    alpha[feed.components].fixed:=TRUE;                       617
    vap_to_feed_ratio.fixed:= TRUE;                            618
END specify;                                                  619
                                                                620
METHOD seqmod;                                                621
    alpha[feed.components].fixed:=TRUE;                       622
    vap_to_feed_ratio.fixed:= TRUE;                            623
END seqmod;                                                  624
                                                                625
END flash;                                                    626
                                                                627
(* ***** *)                                               628
                                                                629

```

Next, the model *flowsheet* is presented. This model represents one of the applications of the WHEN statement. Namely, selecting among



alternative configurations of the problem. Note that in each of the WHEN statements we define the conditional existence of complete ASCEND models. A specific combination for each of the conditional variables -boolean\_vars in the example- will define a specific configuration of the problem. Once a configuration has been selected, it will be kept until the user decides to change it. Note that the user does not have to recompile the model to switch among alternative configurations. The reconfiguration of the system can be done automatically by simply changing the values of the conditional variables. An obvious application of this would be the synthesis of process networks. While running the script *when\_demo.a4s*, note the changes in the number of active equations, active variables and fixed variables for the different configurations. For example, the configuration defined by one of the feeds, two single-stage compressors and one of the reactors contains 169 active equations.

```
( * ***** *) 630
MODEL flowsheet; 631
(* units *) 632
f1 IS_A cheap_feed; 633
f2 IS_A expensive_feed; 634
c1 IS_A single_compressor; 635
s1 IS_A staged_compressor; 636
c2 IS_A single_compressor; 637
s2 IS_A staged_compressor; 638
r1 IS_A cheap_reactor; 639
r2 IS_A expensive_reactor; 640
col,co2 IS_A cooler; 641
h1,h2,h3 IS_A heater; 642
f11 IS_A flash; 643
sp1 IS_A splitter; 644
m1 IS_A mixer; 645
(* boolean variables *) 646
select_feed1 IS_A boolean_var; 647
select_single1 IS_A boolean_var; 648
select_cheapr1 IS_A boolean_var; 649
select_single2 IS_A boolean_var; 650
651
652
653
654
655
656
657
658
659
```

```

660
(* define sets *) 661
662
m1.n_inputs :=2; 663
spl.n_outputs := 2; 664
665
(* wire up flowsheet *) 666
667
f1.stream, f2.stream, c1.input, s1.input ARE_THE_SAME; 668
c1.output, s1.output, m1.feed[2] ARE_THE_SAME; 669
m1.out, col.input ARE_THE_SAME; 670
col.output, h1.input ARE_THE_SAME; 671
h1.output, r1.input, r2.input ARE_THE_SAME; 672
r1.output, r2.output, co2.input ARE_THE_SAME; 673
co2.output, fl1.feed ARE_THE_SAME; 674
fl1.liq, h2.input ARE_THE_SAME; 675
fl1.vap, spl.feed ARE_THE_SAME; 676
spl.out[1], h3.input ARE_THE_SAME; 677
spl.out[2], c2.input, s2.input ARE_THE_SAME; 678
c2.output, s2.output, m1.feed[1] ARE_THE_SAME; 679
680
681
(* Conditional statements *) 682
683
WHEN (select_feed1) 684
CASE TRUE: 685
USE f1; 686
CASE FALSE: 687
USE f2; 688
END WHEN; 689
690
WHEN (select_single1) 691
CASE TRUE: 692
USE c1; 693
CASE FALSE: 694
USE s1; 695
END WHEN; 696
697
WHEN (select_cheapr1) 698
CASE TRUE: 699
USE r1; 700
CASE FALSE: 701
USE r2; 702
END WHEN; 703
704
WHEN (select_single2) 705

```

```

    CASE TRUE:
        USE c2;
    CASE FALSE:
        USE s2;
END WHEN;

METHODS

METHOD default_self;
END default_self;

METHOD seqmod;
    RUN c1.seqmod;
    RUN c2.seqmod;
    RUN s1.seqmod;
    RUN s2.seqmod;
    RUN co1.seqmod;
    RUN co2.seqmod;
    RUN h1.seqmod;
    RUN h2.seqmod;
    RUN h3.seqmod;
    RUN r1.seqmod;
    RUN r2.seqmod;
    RUN fl1.seqmod;
    RUN sp1.seqmod;
    RUN m1.seqmod;
END seqmod;

METHOD specify;
    RUN seqmod;
    RUN f1.specify;
    RUN f2.specify;
END specify;

END flowsheet;

(* ***** *)

MODEL test_flowsheet REFINES flowsheet;

    f1.stream.components := ['A','B','C','D'];

METHODS

METHOD default_self;
```

```

END default_self;                                752
                                                753
METHOD values;                                  754
                                                755
    (* Initial Configuration *)                  756
    select_feed1 := TRUE;                        757
    select_single1 := TRUE;                      758
    select_cheapr1 := TRUE;                      759
    select_single2 := TRUE;                      760
                                                761
    (* Fixed Values *)                           762
                                                763
(* Physical Properties of Components *)          764
                                                765
    f1.stream.state.Cpi['A'] := 0.04 {BTU/mole/K}; 766
    f1.stream.state.Cpi['B'] := 0.05 {BTU/mole/K}; 767
    f1.stream.state.Cpi['C'] := 0.06 {BTU/mole/K}; 768
    f1.stream.state.Cpi['D'] := 0.055 {BTU/mole/K}; 769
                                                770
(* Feed 1 *)                                    771
    f1.cost_factor := 0.026 {dollar/kg_mole};    772
                                                773
(* Feed 2 *)                                    774
    f2.cost_factor := 0.033 {dollar/kg_mole};    775
                                                776

(* Cooler 1 *)                                  777
    col.cost_factor := 0.7e-06 {dollar/kJ};      778
    col.heat_removed := 100 {BTU/s};             779
                                                780

(* Cooler 2 *)                                  781
    co2.heat_removed := 150 {BTU/s};            782
    co2.cost_factor := 0.7e-06 {dollar/kJ};      783
                                                784

(* Heater 1 *)                                  785
    h1.heat_supplied := 200 {BTU/s};            786
    h1.cost_factor := 8e-06 {dollar/kJ};         787
                                                788

(* Heater 2 *)                                  789
    h2.heat_supplied := 180 {BTU/s};            790
    h2.cost_factor := 8e-06 {dollar/kJ};         791
                                                792

(* Heater 3 *)                                  793
    h3.heat_supplied := 190 {BTU/s};            794
    h3.cost_factor := 8e-06 {dollar/kJ};         795
                                                796

(* Flash *)                                     797

```

```
f11.alpha['A'] := 12.0;          798
f11.alpha['B'] := 10.0;          799
f11.alpha['C'] := 1.0;          800
f11.alpha['D'] := 6.0;          801
f11.vap_to_feed_ratio :=0.9;    802
                                803
(* Splitter *)                   804
    spl.split[1] :=0.05;         805
                                806
(* Mixer *)                       807
    m1.To := 298 {K};           808
                                809
(* Single Compressor 1 *)         810
    c1.cost_factor := 8.33333e-06 {dollar/kJ}; 811
    c1.pressure_rate := 2.5;     812
                                813
(* Single Compressor 2 *)         814
    c2.cost_factor := 8.33333e-06 {dollar/kJ}; 815
    c2.pressure_rate := 1.5;     816
                                817
(* Staged Compressor 1 *)        818
    s1.cost_factor_work := 8.33333e-06 {dollar/kJ}; 819
    s1.cost_factor_heat := 0.7e-06 {dollar/kJ}; 820
    s1.pressure_rate := 2.5;     821
    s1.n_stages := 2.0;         822
                                823
(* Staged Compressor 2 *)        824
    s2.cost_factor_work := 8.33333e-06 {dollar/kJ}; 825
    s2.cost_factor_heat := 0.7e-06 {dollar/kJ}; 826
    s2.pressure_rate := 1.5;     827
    s2.n_stages := 2.0;         828
                                829
(* Reactor 1 *)                   830
    r1.stoich_coef['A']:= -1;    831
    r1.stoich_coef['B']:= -1;    832
    r1.stoich_coef['C']:= 1;     833
    r1.stoich_coef['D']:= 0;     834
    r1.low_turnover := 0.0069 {kg_mole/s}; 835
                                836
(* Reactor 2 *)                   837
    r2.stoich_coef['A']:= -1;    838
    r2.stoich_coef['B']:= -1;    839
    r2.stoich_coef['C']:= 1;     840
    r2.stoich_coef['D']:= 0;     841
    r2.high_turnover := 0.00828 {kg_mole/s}; 842
                                843
```

```
      (* Initial Guess *)                                844
                                                    845
(* Flash *)                                           846
    fl1.ave_alpha:= 5.0;                               847
                                                    848
END values;                                           849
                                                    850
METHOD configuration2;                                851
(* alternative configuration *)                        852
    select_feed1 := FALSE;                             853
    select_single1 := FALSE;                           854
    select_cheapr1 := FALSE;                           855
    select_single2 := FALSE;                           856
END configuration2;                                   857
                                                    858
METHOD configuration3;                                859
(* alternative configuration *)                        860
    select_feed1 := FALSE;                             861
    select_single1 := TRUE;                             862
    select_cheapr1 := TRUE;                             863
    select_single2 := FALSE;                           864
END configuration3;                                   865
                                                    866
END test_flowsheet;                                  867
                                                    868
(* ***** *)                                       869
                                                    870
```