Reimplementation of MMS–Software in DYMOLA II

Martin Schöckle

Advisor:
Dr. François E. Cellier

A research report submitted to the
Department of Electrical and Computer Engineering
University of Arizona
and the
Institut für Energiewirtschaft und Rationelle Energieanwendung
Universität Stuttgart

in partial fulfillment of the requirements
for the Degree of
Diplomingenieur
at the
Universität Stuttgart

1 9 9 2

Acknowledgments

The work that resulted in this thesis was done during my stay in the USA. Therefore, I wish to thank everybody who made this stay possible for me, namely Professor A. Voß who organized and arranged the exchange program I participated in, and my parents.
My wife Annette, who was able to join me most of the time, deserves my special gratitude.
I wish to thank my advisor Dr. F. E. Cellier, who supplied most of the ideas to be included in this thesis.
Of course, I have to thank all my friends to join me in Tucson and a lot of other places in the US., preventing me from getting homesick, especially Johannes Kreißig and my classmate Markus Weiner.

Contents

List of Figures

List of Tables

Abstract

The Modular Modeling System MMS for dynamic power plant simulation is used as a basis to develop a new simulation system. This new system makes use of a new modeling language (DYMOLA) that offers structurized models and a more readable code for model descriptions than the originally used simulation language ACSL. DYMOLA proves to be an adequate tool to model large continuous systems. In addition to a pure recoding of the given ACSL module library of MMS, bond graph models are developed for some basic MMS modules. Although the bond graph modeling technique has been applied to a large variety of problems, the attempt to model the dynamic behavior of compressible fluid motion and steam/water equilibrium failed. The bond graph is successfully applied to incompressible fluid motion problems including dynamic wave phenomena occurring in power plant components, e.g. pipes.

1 Introduction

Modeling and simulation have become indispensable design tools since they permit the engineer to predict the behavior of a system before it is actually built. In fact, modeling and simulation are the only techniques available that make it possible to analyze arbitrarily nonlinear systems accurately even under varying experimental conditions.

Computer simulation has become more and more common in the past 30 years due to the increased availability of more and more powerful digital computers. Several modeling techniques have been developed for different purposes, e.g the Finite Element and Finite Difference Methods or multibody dynamic system simulation techniques in mechanical engineering or simulation, and design systems for integrated circuit design in electrical engineering.

In this text, a special topic in continuous systems simulation is addressed, namely power plant dynamic performance simulation. In this field, simulation becomes extremely important due to the importance and the impact of recent decisions for the present and for the future and because of the huge investments that are necessary in this engineering field. For power plant simulation, a simulation system was developed by EPRI in the early 1980's, called the Modular Modeling System or simply MMS. MMS consists of a set of pre–programmed modules representing various power plant components like pumps, pipes, heat exchangers or turbines. The modules are combined in a module library that is the kernel of the MMS. An actual power plant model is built by selecting the desired modules, supplying specific parameterization data to turn the generic modules in actual representations of power plant components and finally combining the modules in a simulation program defining the module interconnections.

Once a model of a power plant is built, it needs to be executed in order to determine the desired transients of plant or single component performance. This is done using a so called simulation language. MMS was and still is based on the simulation language ACSL. ACSL is an excellent software product to deal with the numerical solution of systems of ordinary differential and algebraic equations and provides some useful options like several integration algorithms and the eigenvalue determination of a given system. The language is not so well suited, however, to deal with large, hierarchically structured models itself. For this purpose, several so called modeling languages like DYMOLA were introduced. DYMOLA has a very sophisticated system to build hierarchically structured models, and it provides for a very easy interconnectability of the single model parts to a larger model. DYMOLA is unable to execute a model, therefore it provides for interfaces for simulation languages, e.g. ACSL.

This enables DYMOLA to turn a model into a specific simulation program. DYMOLA can be regarded as a very powerful macro handler for simulation languages. Large scale models like powerplant models become much easier to handle if they are coded in DYMOLA rather than in ACSL.

Yet another part of this thesis expands on modeling techniques by themselves rather than on the modeling tools mentioned so far. The MMS model equations are derived from the basic laws of conservation, like conservation of mass, energy and momentum. In this thesis, the bond graph modeling technique is applied to derive models for power plant simulation. The bond graph modeling technique is based on power conservation rather than on energy conservation. It supports the modeler in building more structured and error–free models, because it is based on a restricted set of well defined and very simple model elements and because the bond graph represents the topological structure of a model as well as its computational structure. Also, the bond graph is a mainly graphical technique and therefore the model structure becomes much clearer from a bond graph than from a set of conservation equations.

2 MMS and the implementation as ACSL macros

The first version of MMS was developed in the early 1980's by EPRI (Electric Power Research Institute, Palo Alto, California). The first release of MMS was available in 1984. In this chapter, a short introduction to MMS is given. The implementation of MMS in the current form is as ACSL macros. In order to get an understanding how MMS works, an introduction to ACSL and to the usage of ACSL macros is given in this chapter, too.

2.1 The Modular Modeling System

Designers and operators of electric power plants face difficult problems at all stages of plant development and operation. From specification of components, to checkout of control systems, to predictions of dynamic performance, to diagnostics during troubleshooting, it is necessary to have a clear picture of what might happen if disturbances occur. The traditional tools — steady state analysis, piecemeal simulation of components and subsystems, and cumbersome manual calculations — were simply not adequate to simulate or predict dynamic system performance.

The need for an efficient and economical code to perform long term analysis was recognized by the Electric Power Research Institute (EPRI), the primary research and development agency of the U.S. electric power generation industry. Under the guidance of EPRI, MMS was developed to provide utility engineers and operating personnel with a code to be used for:

- design and checkout of control systems
- improved diagnosis of plant performance
- procedure evaluation
- specification, selection, and integration of plant components
- analysis to expedite plant commissioning and retrofit
- best estimate plant safety analysis
- plant simulator qualification

MMS consists of a library of pre-programmed generic component models, called *modules*. A model of a actual plant may be assembled from these modules. Each significant physical

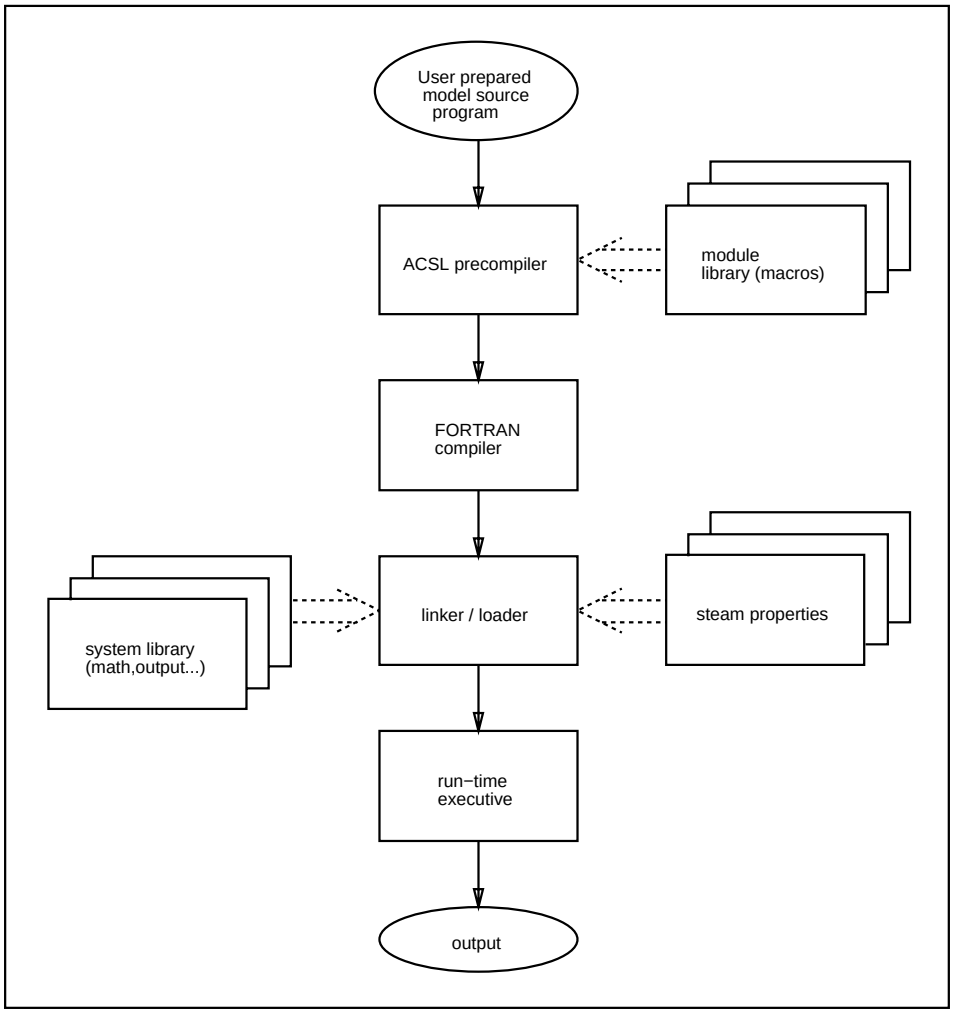

Figure 1: MMS simulation code structure

component of the plant under consideration is represented by one or more modules. The user defines their interconnections and provides sufficient data to describe the physical component. In this way, the generic modules are turned into models of components in the particular plant to be modeled.

After the model has been defined, it is processed by the ACSL precompiler that generates a FORTRAN program of the model, which is automatically compiled and linked with all necessary system libraries. Then the interactive runtime executive of ACSL is entered. It allows the user to run the model and to create the desired output like plots or printed output. The procedure of model compilation and execution is shown in Fig.1.

2.2 Implementation of MMS as ACSL macros

MMS in its currently available version is still implemented as a set of ACSL macros as it was when it was first developed. In this chapter, it is shown why this implementation technique is not very well suited for a large and complex model, e.g. for a power plant. In the following chapter, the basic features of DYMOLA are introduced in contrast to the description given here.

The ACSL macro language allows the user of ACSL to expand the language capability by defining new *operators* to be used in a simulation run. Therefore, the ACSL macros are to be used like subroutines in a procedural high level programming language, e.g. FORTRAN or PASCAL. ACSL provides for library features for the macros, that look a lot like libraries for the above mentioned languages. This makes it possible to define a subroutine– or function–like macro once and then call it from many places. There is a big difference, however, between an ACSL macro and a subroutine or a function, that makes it possible to use macros to structure large continuous models; macros are not executed like procedures, but the macro handler of ACSL just places the equations coded in the macro into the rest of the simulation program upon each macro call [1].

ACSL provides for concatenated variable names in the macros, which enables the user to call the same macro several times without intervening memory space. In other words, unique variable names are generated from the macro handler itself if a macro is defined and called properly. Here an example might be helpful to demonstrate this point, which distinguishes the ACSL macro language from FORTRAN. In Fig.2, an example of an ACSL macro is listed, as well as a part of an ACSL program calling this macro.

Note that in the macro PUMPMD all variables that appear in mathematical expressions like `ZDH_ID=144.*(P_WL-P_WE)/R_WE` are concatenated with the arguments appearing in the macro definition `MACRO PUMPMD (ID,WE,WL)` using the concatenation operator (_). Upon invocation of the macro with the statement `PUMPMD("CND","COND","LCND")` , all concatenation symbols (e.g. `ID`) are replaced by the argument appearing in the invocation statement (`CND`). In the generated program, the mathematical expression then reads `ZDHCND=144.*(PLCND-PCOND)/RCOND` .

This example shows two other features of the ACSL macros, namely how the macros are connected to each other and how parameters are defined for the macros. In the ACSL program, the model PUMPMD is connected to the model VALVE or more technically spoken, the outlet of a pump is supposed to be connected to the inlet of a valve. In order to do that, the user has to make sure that the argument that denotes the concatenation for the

outlet variables of the macro PUMPMD (`LCND`) is the same as the argument that denotes the concatenation for the inlet variables of the macro VALVEI (`LCND`). As an additional constraint, the variable names used in both macros for the in– and outlet properties as `P`, `H`, `W` or `R` have to be identical. By using this trick, the outlet property variables for macro PUMPMD will have the same names as the inlet property variables for the macro VALVEI and therefore will be in fact identical. The assignment of parameter values works similarly. For the macro PUMPMD, a value of 2 has to be assigned to the parameter `KNP_ID`. Since the argument `CND` is used for concatenation of this variable, the full name of the variable is `KNPCND`. This is exactly the name used in the `CONSTANT`–statement related to the invocation of macro PUMPMD, as is seen from the listing.

This clumsy procedure of connecting macros or modules is due to the fact that ACSL macros were not intended to be used that way. As will be shown in the next chapter, software tools are available that provide for a much more sophisticated way of dividing large systems into subsystems and connecting them with each other.

In appendix A, a complete MMS example is printed. Note that the macros are defined before they are used by the simulation program.

```
MACRO PUMPMD(ID,WE,WL)
     MACRO RELABEL L100,L200,L300
     PROCEDURAL(ZQW_ID=ZDH_ID)
     ZQW_ID=KHC_ID(ZDH_ID)
     IF(ZZFRFL.AND.KCK_ID.AND.(ZQW_ID.LE.0.)) ZQW_ID=0.
     END
     ZDH_ID=144.*(P_WL-P_WE)/R_WE
     W_WE=8.02*KNP_ID*R_WE*ZQW_ID
     ZEP_ID=KEP_ID-KPR_ID*(ZQW_ID-KQR_ID)**2
     H_WL=H_WE+.0013*ZDH_ID/ZEP_ID
     W_WL=W_WE
     R_WL=RLOFPH(P_WL,H_WL)
     T_WL=TLOFPH(P_WL,H_WL)
" VALIDITY CHECKS "
     PROCEDURAL(OWW_ID=W_WE)
     CONSTANT OWW_ID=.FALSE.
     IF(OMASK.OR.(.NOT.ZZFRFL)) GO TO L300
     IF(OWW_ID) GO TO L200
     IF(W_WE.LT.0.) OWW_ID=.TRUE.
     IF(OWW_ID) PRINT L100
  L100..FORMAT(//,...
     '***EXECUTION TERMINATED; REVERSE FLOW IN PUMP _ID***',//)
  L200..CONTINUE
     TERMT(OWW_ID)
  L300..CONTINUE
     END
MACRO END
                         .
                         .
                         .

" MACRO INVOCATION FOR CONSTANT SPEED MOTOR DRIVEN PUMP "
" "
     PUMPMD("CND","COND","LCND")
" PARAMETERS FOR THE MODEL ARE: "
     CONSTANT KEPCND=0.85,KNPCND=2.,KPRCND=3.4E-8,KQRCND=5000.
" BOUNDARY CONDITIONS "
     CONSTANT PCOND=1.42,HCOND=81.7,RCOND=61.8
" "
" MACRO INVOVATION FOR VALVE "
" "
     VALVEI("VLV","LCND","LVLV")
" PARAMETERS FOR THE MODEL ARE: "
     CONSTANT KCPVLV=4.403E4,KCVVLV=1.044E5,KVAVLV=1
" "
                         .
                         .
                         .
```

Figure 2: ACSL program section

3 The modeling language DYMOLA

In the last chapter it was shown how ACSL can be used to structure a large model, e.g. a power plant model. However, ACSL was not intended to be used to build large and heavily hierarchically structured models. In this chapter a new tool (DYMOLA) will be presented, which is a stand–alone program that can be used as a front end to several different simulation languages.

DYMOLA is not a simulation language in its own right, since it does not provide for a simulation engine of its own. Instead, DYMOLA is a modeling language, since it supports the user in coding more–readable and better–modularized hierarchically structured model descriptions. From these model descriptions simulation code for several simulation languages like SIMNON, ACSL or DESIRE can be generated.

First, some basic principles of DYMOLA models are shown in contrast to comparable ACSL models. After that, it is shown that basically the whole MMS macro library can be transformed into a set of DYMOLA submodels, resulting in more understandable and more error–free simulation programs.

3.1 Basic properties of DYMOLA model descriptions

If a large system of any kind is considered, one seldom can have a look at the whole system, because it is too complex to be examined at one time. Instead, people usually split large systems into smaller subsystems to be considered individually. After one has learned enough about the parts of the system, one can hope to assemble the gained knowledge to get an understanding of how the whole system works. Engineers use this concept as well as biologists or social scientists. If a large system is to be considered, the only way to get around with it is to split it in smaller and smaller subsystems, which can be understood one at a time, and afterwards assemble the submodels to the whole system under consideration.

Consequently, in simulation of technical systems, the simulation software should enable the modeler to build the computer model the way he built the physical model of a system. The need for highly structured models will become clear when the bond graph modeling technique is introduced later on in this text.

3.1.1 Structure of the model

DYMOLA was designed with the need for structured models in mind and it enables the user to choose as many hierarchical levels for his models as he likes. This fact on its own

does not come as a big surprise though, because ACSL macros can call other macros as well in as many levels as desired [1]. The big difference between ACSL macros and DYMOLA submodels is that DYMOLA offers a data structure that is specially designed for hierarchical models. DYMOLA concatenates variable names automatically and without the restrictions applying for concatenation in ACSL making highly structured models impossible, namely only one concatenation level and symbol names no longer than six characters. In other words, while ACSL enables the user to hierarchically structure the program, it does not provide for hierarchically structured data and therefore it does not provide for a structured model.

3.1.2 Connection of submodels

After the possibility of splitting a large model is given, the single model parts have to be assembled in order to obtain the model of the whole system under consideration.

In fact, if submodels are chosen, several variables are chosen too, that relate the submodel with its environment; e.g. if the submodel is chosen to be an electrical resistor, these variables will be voltage and current at the system boundaries. In order to connect the submodel of the resistor to the submodel of a capacitor, one has to make sure that the voltages and the currents at the connection points are the same. In ACSL, the concatenation arguments for both boundaries to be connected have to be the same, as seen in a more complex example in the previous chapter. DYMOLA provides for a much more sophisticated and powerful system to connect subsystems.

DYMOLA provides for two types of variables, namely *terminal* or *local*. Terminals are supposed to be connected from outside the submodel and they can be grouped in arbitrary ways using so–called *cuts*. Locals are variables to be used only inside the submodel. Consider the part of a DYMOLA program shown in Fig.3 that is equivalent with the previously shown ACSL program segment shown in Fig.2.

The boundary variables that provide for the connection of the pump to the valve, p, $\dot{m}$, h, ρ and T are grouped in a cut and the submodels are connected explicitly using the connect statement rather than implicitly like in the ACSL program. This provides for much more readable model code.

Another feature of DYMOLA cuts is that it is possible to choose a *path* to connect the cuts through the submodel. The definition of a path is closely related to the definition of reference directions within a submodel. In the example of a valve or a pump, one assumes that a fluid enters the component from one end and leaves it at the other end, which is connected to another component and so on. The path definition provides the model with the information

```
model type pump
         terminal  pwe , wwe , hwe , rwe
         terminal  pwl , wwl , hwl , rwl , Twl
         cut  inwater  ( pwe , wwe , hwe , rwe , . )
         cut  outwater ( pwl , wwl , hwl , rwl , Twl )
         path water   < inwater - outwater >
         parameter kep = 0.85 , knp = 2. , kqr = 5000.
         local zdh , zqw , zep
         zqw = PH ( zdh )
         zdh = 144. * ( pwl - pwe ) / rwe
         wwe = 8.02 * knp * rwe * zqw
         zep = kep - (kep*( zqw-kqr )**2)/kqr**2
         hwl = hwe + 0.0013*zdh/zep
         wwe = wwl
         rwl = RLOFPH ( pwl,hwl )
         Twl = TLOFPH ( pwl,hwl )
end

                        .
                        .
                        .

submodel (pump)      pump  (kep = 0.85 , knp = 2. , kqr = 5000.)
submodel (valvei)    valv

         connect (water) pump to valv

                        .
                        .
                        .
```

Figure 3: DYMOLA program section

of the reference flow direction, namely water leaves the pump at cut `outwater` and enters the valve at cut `inwater`, if the cuts for the submodel of the valve are defined similarly.

Parameter values are passed to the submodel at its invocation and not by variable assignments like in ACSL. Default values can and should be assigned to all parameters declared for the submodel.

This example, however, is a very simple one and was only chosen to be consistent with the previous chosen ACSL example. DYMOLA offers much more features for connecting submodels like *nodes* or structured cuts and paths than are presented here on a first glance. More details are introduced if it is shown how bond graph modeling is done with DYMOLA. A complete language description is given in the DYMOLA User's Manual [2].

3.2 Reimplementation of a MMS program

In the last section it was shown that DYMOLA is a better tool to describe large continuous models than ACSL. It therefore is shown in this section how a given ACSL model is transferred to a DYMOLA model description.

As an example, the test example that is distributed with MMS was chosen. This is a

small example intended to simulate the water level in a special power plant component, a so called deaerator. Deaerators serve several purposes in a thermic power plant: they are used to degasify and pre–heat the condensate leaving a cool tower, they provide for feedwater storage and they provide sufficient NPSH to avoid cavitation in the feedwater pumps connected to them downstream. Fig.5 shows the system under consideration. The condensate leaving a (not modelled) cooling device, e.g a cool tower, is pumped by the condensate pumps, and then, controlled by a valve fed into the deaerator. Also fed into the deaerator is steam (superheated vapor) from an extraction steam line from a (not modelled) steam turbine. The model is intended for a deaerator water level control study, therefore there is a simple proportional and integral controller added to the model, which controls the fluid level in the deaerator dependent from the feedwater demand given as input to the system.

This very common low pressure feedwater train arrangement is transferred into a MMS model description shown in Fig.6. This procedure is explained in detail in the MMS User's Manual [5]. All models needed can be taken from the MMS library and are connected using the ACSL program shown in Fig.4. The equivalent DYMOLA model description is shown in Fig.7.

The difference is quite remarkable. The DYMOLA code is fairly self explanatory. The structure of the model can be recognized much more easily from the DYMOLA model description than from the ACSL program. The connections of the submodels are given explicitly and all submodels used are declared and parameters are defined with the declaration.

It can be noticed that there are not only syntactical changes, however, but also some parts of the ACSL program are missing in the DYMOLA model description. This is due to the fact that DYMOLA can be used in connection with various simulation languages that are all rather different from each other. The DYMOLA modeling language does not provide for simulation control statements like `TERMT` or for special ACSL control variables like `IALG`. DYMOLA also does not provide for a number of special simulation language features like `TABLE` for ACSL to define a function–like lookup table with linear interpolation.

If a DYMOLA model has been entered and processed properly, an *experiment* can be performed with the given model. The experiment is entered by an *experiment file* that contains all the special information concerning the simulation language to be used to run the experiment, like simulation time or integration algorithms. Special language features like tables can be added to the program using the experiment file as well. The experiment file used to specify a simulation run from the DYMOLA model description from Fig.7 is shown in Fig.8.

3.3 Reimplementation of MMS macros

In order to run the previously defined DYMOLA model, the submodels used in the deaerator level study model have to be redefined, too. Fig.9 and Fig.10 show a MMS macro and a DYMOLA submodel description of a valve for incompressible flow.

First of all, the variables used in the macro had to be sorted and declared as either *terminal* for interconnecting variables, *parameter* for parameter values to be constant and assigned from outside the model or *local* for local variables only used in a specific submodel. Fortunately some documentation for the MMS macros is available in the library file that has been deleted from the example to keep it small. This documentation helps to get around with this problem a little bit easier. The complete code of all reprogrammed ACSL macros can be found in the appendix A.

The most important thing a submodel consists of are the algebraic and differential equations modelling a real component like a pump or a valve. The algebraic equations used in an ACSL macro can be transferred as they are to the DYMOLA submodel description as can be seen from the example. For the differential equations, a first difference between ACSL and DYMOLA is noticed. DYMOLA does not use the integration (`INTEG`) operator to specify a differential equation. Instead of an `INTEG` operator, DYMOLA uses a derivative (`der`) operator to specify the derivative term[1]. According to this, all the differential equations used in integrated form for ACSL can be used in differential form directly for DYMOLA. Because DYMOLA offers formula manipulation, they even don't have to be solved for the derivative term by hand. For DYMOLA, it would be perfectly all right to type the differential equations in as they have been derived.

All function calls to FORTRAN intrinsic functions like `SIGN` and `SQRT` or to external functions like the steam property functions `RLOFPH` and `TLOFPH` remain the same in the equations. They are simply passed to the ACSL program by the interface.

Considering the example shows yet another difference. The MMS macros were designed to be as robust as possible. Therefore, a lot of validity checks are included in the macros. These validity checks are done using the ACSL `IF` statement. In case of a unrecoverable inconsistency like reverse flow in a component, the simulation run is aborted using a `TERMT` statement and an error message is printed out to inform the user of the problem.

Also, to make the macros more versatile, several different cases for certain component

[1]Consequently, the initial conditions for the integrators cannot be specified in connection with the differential equations. As can be seen from Fig.7, the initial conditions have to be specified with the invocation of the submodel containing the differential equations.

features were included in some models, one of them to be chosen by a switch variable, e.g. the valve characteristic[2] to be chosen by switch variable `KVA` in Fig.9. This is also done using the `IF` statement.

All these features like if–statements for branching a program or termination conditions for a simulation run are not offered by the DYMOLA language. Therefore, these features, that are clearly an advantage of the MMS macros coded in ACSL directly, are not available for the MMS submodels coded in DYMOLA[3] and therefore they are missing.

There would be a way to reintroduce these features using the experiment file. This, however, would result in non–modular code, because the model description and the validity check would be separated in different files. In addition, this would be a very cumbersome thing to do, because the person building a model had to file all that by himself to the experiment file, there is no way to connect this information somehow to the code of the submodel and to put it in its place upon compilation of the model.

3.4 Testing of the new library and performance comparison

It was possible to match the results of the old example with the newly written simulation code without any problems. This does not come as a big surprise, since the model equations were not altered at all. The ACSL code for the deaerator level study program, including the macros from the macro library, is printed in appendix A. The reimplemented code is printed in appendix B.

The new model is much more readable and fairly self–explanatory. However, the versatility of the new models is clearly restricted in comparison with the old macros, because the possibility of branching the program execution is not given if using DYMOLA. Also, there is a little bit of overhead created by the DYMOLA program, working like a preprocessor for ACSL.

The DYMOLA model also has no validity checks included, therefore program execution is not stopped in case of unphysical data, what makes the program harder to debug. The ACSL macros supply the user with some information about troubles occurring upon model execution. This helps the user to detect errors.

[2]The valve conductance varies with the valve stroke according to its inherent valve characteristic. The valve macro provides for one of three characteristics, equal percentage, linear and quick opening

[3]There is an if–statement available in the DYMOLA modelling language, but it is intended not for branching a model execution but for handling discontinuities in the model itself. Unfortunately, the ACSL interface is not able to process the DYMOLA if–statement anyway.

3.5 Enhancement suggestions for DYMOLA

Although DYMOLA is a very good tool for modeling purposes, some improvements should be made in order to turn it into a commonly useable software tool.

As far as the program execution and the user interface of DYMOLA is concerned, the program execution should be made much more robust. In some cases, the program execution is just aborted, e.g. if an ACSL simulation program cannot be generated upon request by the user because the experiment file has not been entered yet. Instead of aborting the program execution without any error messages, the user interface of DYMOLA should issue an error message and request the missing experiment file.

Some improvements should also be made for the modeling language DYMOLA itself. In the DYMOLA User Manual [2] it is stated that parameters can be passed through submodels. In fact, parameters can only be passed down to the next lower hierarchical level, and not to the levels hidden further down. This restricts the modularity of DYMOLA submodels containing more than one hierarchical sublevel, because the user of a submodel has to edit all the files included in the submodel and set the parameters on each level to the desired values individually. The same problem happens if initial conditions of state variables have to be specified differently from zero. Initial conditions are specified like parameter values upon invocation of a submodel. Also, DYMOLA should provide for global variables for a whole model or submodel, e.g. make a variable global for all submodels included in the submodel declaring a variable global.

DYMOLA should provide for some statements enabling validity checks for variables. Usually, simulation results get worthless if they override a well known range that can be specified before the simulation run, e.g. negative absolute temperatures. In this case, there should be a way to abort the simulation run and notify the user of the trouble, avoiding wasting computing power. This could be done using a special DYMOLA statement to be translated to the currently used simulation language, e.g. ACSL.

Another feature of the ACSL macros for MMS is branching of the model execution, e.g. the valve model shown in Fig.9 provides for three valve characteristics to be determined dependent from the parameter `KVA_ID` [3]. This valve characteristic is chosen once, and is not changed during execution of the model, therefore it is no discontinuity in the model, but its enhancing versatility of the model by providing several options for the user of the macro. This is especially helpful, if the modeler, who built a submodel, and the user of a model, are not the same people as for MMS. For the purpose of branching model execution upon compile time of the model in DYMOLA, a special case or branching statement could

be invented, causing only one case out of several choices to be compiled and translated to the actual simulation language to be used, e.g ACSL.

Upon simulation of thermodynamic systems as described later on in this thesis, it turned out that it might be necessary to access a lot of thermodynamic data like steam tables and nonlinear fluid properties like heat capacitances or heat conductivities. For that purpose, it would be very helpful, if DYMOLA could be linked with some data base, providing for the possibility of fast and efficient data access.

```
PROGRAM DEAERSTUDY
" THIS PROGRAM IS INTENDED FOR DEAERATOR LEVEL CONTROL STUDY  "
" "
DYNAMIC
     TERMT(T.GE.TSTOP)
     CONSTANT TSTOP=0.,IALG=2,OMASK=.FALSE.
DERIVATIVE
" "
     TABLE KHCCND,1,7/...
     690.,800.,880.,930.,960.,980.,1000.,...
     6000.,5000.,4000.,3000.,2000.,1000.,0./
     CONSTANT KCKCND=.FALSE.,KCKEXT=.FALSE.,KCKVLV=.FALSE.
     CONSTANT KDHVLV=0.0
PUMPMD("CND","COND","LCND")
     CONSTANT KEPCND=0.85,KNPCND=2.,KPRCND=3.4E-8,KQRCND=5000.
"          BOUNDARY CONDITIONS     "
     CONSTANT PCOND=1.42,HCOND=81.7,RCOND=61.8
" "
CONNI("C1","LCND","LC1")
     CONSTANT IPLC1=359.8
" "
VALVEI("VLV","LC1","LVLV")
     CONSTANT KCPVLV=4.403E4,KCVVLV=1.044E5,KVAVLV=1
" "
JUNC("J1","EDEA","LEXT","LVLV")
" "
PIPESR("EXT","STM","LEXT",0,0,0)
     CONSTANT KCFEXT=8.84E5,KDHEXT=0.
"          BOUNDARY CONDITIONS     "
     CONSTANT PSTM=142.2,HSTM=1353.2,RSTM=0.213
" "
DEAER("DEA","EDEA","FW")
     CONSTANT KDEDEA=0.,KDHDEA=144.,KVSDEA=8000.,KVTDEA=9980.
     CONSTANT ZIRDEA=36.99,ZIUDEA=325.4
"          BOUNDARY CONDITIONS     "
     CONSTANT WFW=4.5E6
" "
"          CONTROLS     "
DEMAND=K1*WFW+K2*(LSET-LDEA)
SUPPLY=K3*WCOND
     CONSTANT K1=1.E-6,K2=1.,K3=1.E-6,LSET=120.
PICONT("CNT",DEMAND,SUPPLY,YVLV,1)
     CONSTANT CPGCNT=0.1,CIGCNT=0.05,KLLCNT=0.,KHLCNT=1.
     CONSTANT ZICCNT=0.75
" "
INTEGER COUNT,KOUNT
PROCEDURAL(COUNT=)
     CONSTANT COUNT=0,KOUNT=1000
     COUNT=COUNT+1
     TERMT(COUNT.GE.KOUNT)
END $ " PROCEDURAL "
END $ " DERIVATIVE "
END $ " DYNAMIC "
END $ " PROGRAM "
```

Figure 4: ACSL program for deaerator level study

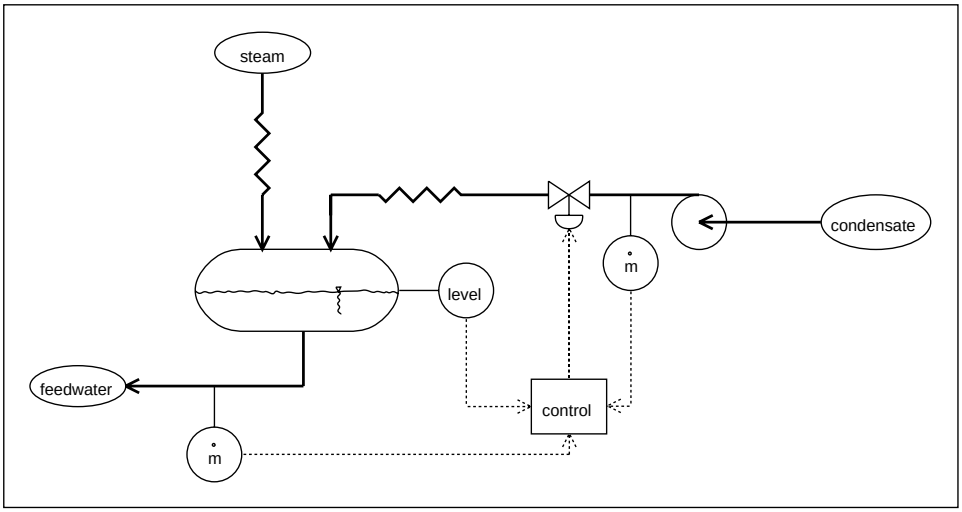

Figure 5: Common low pressure feedwater train

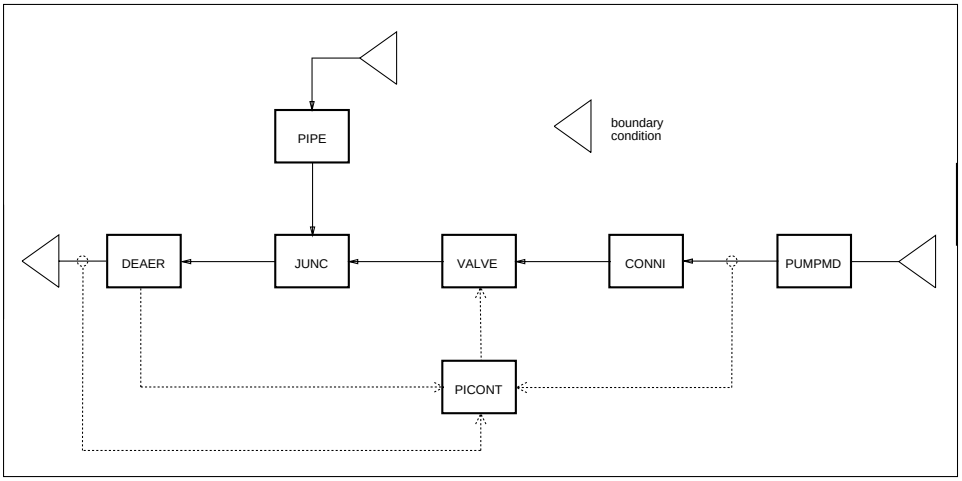

Figure 6: MMS model of deaerator level study system

```
@pump.dym
@conni.dym
@valvei.dym
@junc2.dym
@deaer.dym
@pipe.dym
@picont.dym

model power

submodel (pump)      pump
submodel (conni)     con   ( ic pwl=359.8 )
submodel (valvei)    valv
submodel (pipe)      pipe
submodel (picont)    ctrl
submodel (deaerator) dea1  ( ic zrh = 36.99 , zuh = 325.4 )
submodel (junction2) junc

constant lset = 120.
constant k1 = 0.000001 , k2 = 1.0 , k3 = 0.000001
constant wfw = 4.5E6
        local    putwe , pitwe

{ main line of water flow }

connect (water) pump to con to valv to junc to dea1
{ boundary conditions }
pump.pwe = 1.42
pump.hwe = 81.7
pump.rwe = 61.8
        putwe = TLOFPH(pump.pwe,pump.hwe)
dea1.wwl = wfw

{ extraction steam line }

connect (water) pipe to junc:inwater2
{ boundary conditions }
pipe.pwe = 142.2
pipe.hwe = 1353.2
pipe.rwe = 0.213
        pitwe = TVOFPH(pipe.pwe,pipe.hwe)

{ control system : P/I contrloller }

ctrl.csp  = k1*dea1.wwl + k2*(lset - dea1.lev)
ctrl.pvar = k3*pump.wwe
connect ctrl:control at valv:valve

end
```

Figure 7: DYMOLA model description for deaerator level study

```
cmodel

maxtime tmax = 100.
cinterval cint = 2.
termt ( t .GE. tmax )

initial
INTEGER d1xznn
CONSTANT ialg=2,nstp=10000
TABLE PH,1,7 /  690. ,800. ,880. ,930. ,960. ,980. ,1000.,  ...
               6000.,5000.,4000.,3000.,2000.,1000.,    0./
end

end
```

Figure 8: Experiment file for deaerator level study

```
MACRO VALVEI(ID,WE,WL)
     MACRO RELABEL L100,L200,L300
     LOGICAL OMASK,OWW_ID,KCK_ID
     INTEGER KVA_ID
     PROCEDURAL(ZCV_ID=Y_ID,KCV_ID,KVA_ID)
     IF(KVA_ID.EQ.1) ZCV_ID=ABS(Y_ID)**3*KCV_ID
     IF(KVA_ID.EQ.2) ZCV_ID=Y_ID*KCV_ID
     IF(KVA_ID.EQ.3) ZCV_ID=(1.-EXP(-10.*Y_ID))*KCV_ID
     END
     ZCQ_ID=KCP_ID*ZCV_ID/SQRT(ABS(KCP_ID*KCP_ID+ZCV_ID*ZCV_ID)+1.E-3)
     ZDP_ID=P_WE-P_WL+R_WE*KDH_ID/144.
     PROCEDURAL(W_WL=ZDP_ID,ZCQ_ID,R_WE)
     W_WL=ZCQ_ID*SIGN(SQRT(ABS(R_WE*ZDP_ID)),ZDP_ID)
     IF(ZZFRFL.AND.KCK_ID.AND.(ZDP_ID.LE.0.)) W_WL=0.
     END
     W_WE=W_WL
     H_WL=H_WE
     R_WL=RLOFPH(P_WL,H_WL)
     T_WL=TLOFPH(P_WL,H_WL)
'     VALIDITY CHECKS     '
     PROCEDURAL(OWW_ID=W_WE)
     CONSTANT OWW_ID=.FALSE.
     IF(OMASK.OR.(.NOT.ZZFRFL)) GO TO L300
     IF(OWW_ID) GO TO L200
     IF(W_WE.LT.0.) OWW_ID=.TRUE.
     IF(OWW_ID) PRINT L100
  L100..FORMAT(//,...
     ' ***EXECUTION TERMINATED; REVERSE FLOW IN VALVE _ID***',//)
  L200..CONTINUE
     TERMT(OWW_ID)
  L300..CONTINUE
     END
MACRO END
```

Figure 9: ACSL macro for a valve

```
model type valvei

        terminal  pwe , wwe , hwe , rwe
        terminal  pwl , wwl , hwl , rwl , Twl
        terminal  y
        cut  inwater  ( pwe , wwe , hwe , rwe , . )
        cut  outwater ( pwl , wwl , hwl , rwl , Twl )
        cut  valve    ( y )
        path water    < inwater - outwater >

        parameter kcp = 44030. , kcv = 104400. , kdh = 0.

        local zcv , zcq , zdp

        zcv = abs(y)**3 * kcv
        zdp = pwe - pwl + rwe*kdh/144.
        zcq = kcp*zcv / sqrt ( ABS(kcp*kcp + zcv*zcv) + 1./1000. )
        wwl = zcq * SIGN( SQRT(ABS(rwe*zdp)) , zdp )
        wwl = wwe
        hwl = hwe
        rwl = RLOFPH ( pwl,hwl )
        Twl = TLOFPH ( pwl,hwl )

end
```

Figure 10: DYMOLA submodel description for a valve model

4 Bond graph modeling with DYMOLA

The bond graph was introduced 1961 by Henry Paynter [6]. It is a graphical method of representing engineering systems, just like circuit diagrams are a graphical method of representing electrical circuits. Unlike circuit diagrams, the bond graph is much more powerful, because it applies to all kinds of engineering systems and because it shows not only the topological structure of a system like the circuit diagram, but also its computational structure.

In this chapter, a short introduction to the bond graph is given, more information can be found in the literature [7, 8, 9]. Furthermore, it is shown how bond graphs can be coded in the earlier introduced modeling language DYMOLA [2].

4.1 Bond graph fundamentals

The basic element of the bond graph is the *ideal energy bond*. The bond is regarded as a perfect conductor which transmits power instantaneously and without energy loss. In this respect it is an idealization similar to perfect electrical conductors or infinitely stiff rods. The bond, represented by a bold harpoon, connects two variables, one across variable, in bond graph terminology usually referred to as the *effort* e, and one through variable, the so called *flow* f. The bond is shown in Fig.11.

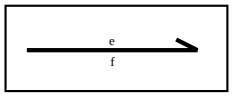

Figure 11: The bond

The terms across and through variables were coined by electrical engineers, measuring a voltage drop at a resistor using a device hooked up *across* the resistor, while measuring a current with a device hooked up such that the current has to go *through* the ammeter as well as through the resistor. Therefore they called the voltage an across variable and the current a through variable.

Also, the use of across and through variables can be related to Kirchhoff's laws known from electrical circuits theory. Considering a node in an electrical circuit, it is stated that all the voltages in all connected branches have to be equal, while all the currents have to add up to zero (Kirchhoff's current law). The variable that appears to have the same value at a junction is called across variable, while the variables to be summed up to zero are called through variables. This description is much more common, it is applicable for a lot of

different systems, e.g mechanical systems, where at a joint of several levers the velocity of the connected parts are equal and the forces add up to zero.

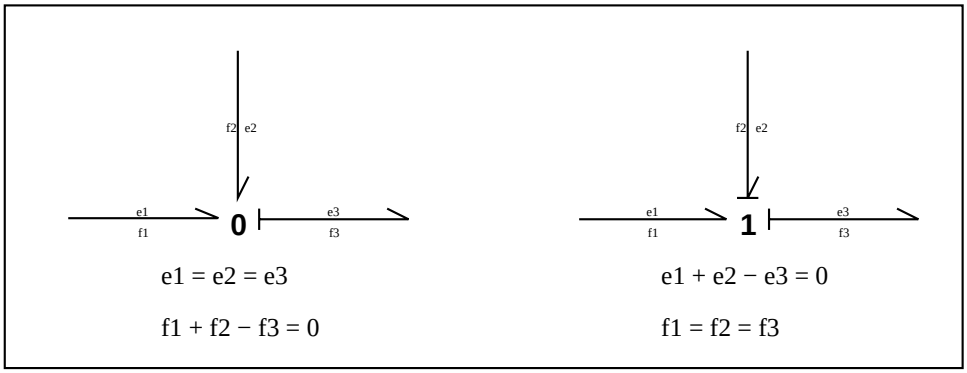

Figure 12: The bond graph junctions

The useful innovations introduced by the bond graph are the two junction types, the so called zero and one junctions. They are shown in Fig.12. The zero junction represents Kirchhoff's current law (i.e the across variables of all attached bonds have the same value, while the through variables add up to zero), whereas the one junction represents Kirchhoff's voltage law (i.e. the through variables of all attached bonds have the same value, while the across variables add up to zero). The zero junction therefore represents an electrical node and sometimes is called a parallel junction, while the one junction is sort of a serial junction.

If power bonds are used like in this thesis, yet another constraint applies to the flow and effort variables to be used in the bond graph. The product of the two variables has to be power, like the product of voltage and current in an electrical circuit is power.

4.2 A simple bond graph example

In order to demonstrate how the bond graph modeling technique is applied to technical systems, consider the simple example of a resistive heating element. Usually, these elements have a mostly ohmic resistance, therefore the resistance can be modeled as ohmic resistor to keep the example simple.

The electrical part of the system is shown in Fig.13. It consists of a power source connected to a resistor R and an isothermal environment for the resistor. The bond graph to be developed for this system is shown in Fig.14.

The power source is modelled using an effort source (SE) supplying an arbitrary current at a fixed voltage. The effort source is connected to the resistor (R). The equations describing

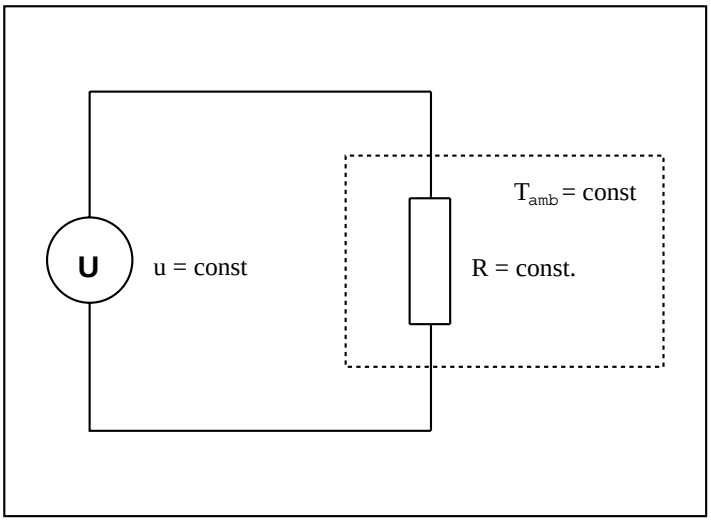

Figure 13: Resistive heating element

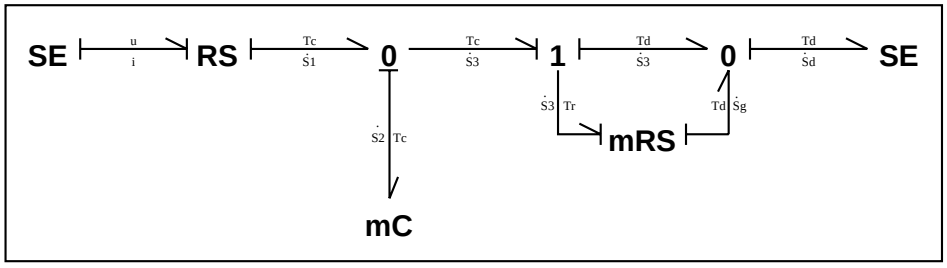

Figure 14: Bond graph for the resistive heating element

the system so far are just Ohms law $U = R \cdot I$ for the resistor and $U = \text{const.}$ for the power source.

All the power supplied to the resistor is dissipated in that resistor, that means it is no longer available in electrical form but that also it is not gone. So the resistor works as interface between power domains, namely the electrical power domain represented by the variables U and I and the thermodynamical power domain represented by the variables T and $\dot{S}$ [7]. According to this, a bond graph element called *resistive source* is introduced that has two bonds connected to it, namely the electrical and thermal bond. The thermal bond is to be connected further to some other elements as will be shown in the following.

If an adiabatic resistor is assumed for simplicity first, we have to deal with the heating process of a piece of matter. The first law of thermodynamics states for closed systems, that all the energy supplied from the outside of the system is used to change the internal energy

U of the system. In terms of thermodynamic variables, this is written as

$$dU = dQ. \tag{1}$$

Because the bond graph is a power graph, this equation is not that useful at all. The equation used in bond graph modeling is obtained by taking the derivative with respect to time from equation (1), yielding

$$\frac{dU}{dt} = \frac{dQ}{dt}. \tag{2}$$

The expression $dQ/dt = \dot{Q}$ stands for the thermal power supplied to the system under consideration.

From the thermodynamic relations, the caloric equation of state for incompressible matter that has been already multiplied with the mass m of the system is used to compute the change of internal energy dU. Again taking the derivative with respect to time, assuming constant mass m and heat capacity c, yields

$$\frac{dU}{dt} = m \cdot c \cdot \frac{dT}{dt}. \tag{3}$$

The thermal power supplied to the resistor is computed from the relation

$$\dot{Q} = \dot{S} \cdot T. \tag{4}$$

Equations (3) and (4) plugged into equation (1) and solved for $\dot{S}$ yields

$$\dot{S} = \frac{m \cdot c}{T} \frac{dT}{dt}. \tag{5}$$

This relation looks exactly the same way as for an electrical capacitor

$$i = C \cdot \frac{du}{dt} \tag{6}$$

except for the capacitance which is not a constant any more but modulated with the temperature. Therefore the bond graph element mC is assigned to this formula.

In a second step, heat transfer to the environment of the heated resistor will be modeled as well as heat storage in the resistor. To keep the model simple again, only one dimensional conductive heat transfer is taken into account. The model built so far works for that new task, it just has to be extended. To split the entropy flow generated in the resistor, a zero junction is introduced. This junction splits the entropy flow $\dot{S}1$ generated in the resistor at a certain temperature in a stored flow $\dot{S}2$ and a flow $\dot{S}3$ delivered to the environment. The bond connecting the resistor and the environment connects to a one junction to which

another resistive source element mRS is connected. This resistive source element accounts for the heat conduction resistance. This thermal resistor is connected to a one junction because it operates on a temperature difference just like the electrical resistor operates on a voltage difference.

One dimensional conductive heat transfer is described using Fourier's law [10]

$$q = k \cdot \frac{dT}{dx}. \tag{7}$$

This equation is transformed into

$$\dot{S} = \frac{k \cdot A}{L \cdot T}\Delta T = \frac{1}{R(T)} \cdot \Delta T. \tag{8}$$

for a linear temperature profile $T(x)$ over a given transport length L for a certain surface A. It indeed looks like Ohm's law for an electrical resistor. The value of the thermal resistance changes with temperature just like the thermal capacitance did as seen before.

The power dissipated in this thermal resistor is reintroduced using a zero junction, because it has to go somewhere, no power is lost to the infinite beyond. Energy is conserved all the time, it can be transformed, but can never be destroyed, as the law of energy conservation states. This reintroduced entropy flow has yet another thermodynamic meaning. As stated in many thermodynamic texts, heat transfer with temperature difference generates a certain amount of entropy. The entropy generation is modeled using the concept of the modulated resistive source introducing an entropy flow back into the system.

Finally, the environment is modeled by an effort source at constant temperature T_d. The whole bond graph model is shown in Fig.14.

This bond graph also exhibits another feature of the bond graph. Each bond in the graph has a vertical stroke attached to it at one end. It is called the *causal stroke* and it marks the side of the bond, at which the flow variable is to be calculated. Several rules apply for the causality of a bond graph, e.g. for a capacitor like the mC element, the causal stroke has to be placed away from the element at the junction, because a capacitor is supposed to calculate the effort variable of the attached bond. The causal bond graph helps to detect structural singularities and algebraic loops within a model [7] that otherwise will be noticed later upon compilation of the model.

4.3 Bond graph models in DYMOLA

In this section, it will be demonstrated how a given bond graph can be coded in DYMOLA. It is also shown, how DYMOLA processes the model description, and how a simulation program

for ACSL is generated. DYMOLA provides for hierarchically structured models, therefore the simplest elements are defined first, before they are assembled to a more complex model.

4.3.1 The effort source element SE

As a general matter of fact, each bond graph element accounts for as many relations as are bonds attached to it. The effort source is used to fix the value of the effort of the bond attached to it. Therefore, the effort source contains just the assignment

$$e = E0 \tag{9}$$

with the value $E0$ to be assigned from outside the model. The DYMOLA code for the effort source is shown in Fig.15.

4.3.2 The resistive source element

The resistive source element RS is a so called two port element, it has two bonds attached to it and therefore contains two relations to calculate variables of the attached bonds.

The input or primary side of the RS element is just modeled using Ohm's law

$$e1 = R \cdot f1 \tag{10}$$

while for the output or secondary side of this resistive element the missing variable is calculated from the relation $P1 = P2$ that is in the variables used in the bond graph

$$e1 \cdot f1 = e2 \cdot f2. \tag{11}$$

The DYMOLA code for the resistive source is shown in Fig.16.

```
{ Bond Graph effort source }

model type SE

  main cut A (e / .)
  terminal E0

  E0 = e

end
```

Figure 15: DYMOLA code for the effort source element

```
{ Bond Graph of a resistive source }

model type RS
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter R=1.0
  R*f1 = e1
  e1*f1 = e2*f2
end
```

Figure 16: DYMOLA code for the resistive source element

4.3.3 Modulated resistive source element

The modulated resistive source element mRS is very similar to the RS element. Rather than the resistance R being constant as for the RS element, it is modulated by the effort variable $e2$ of the output bond. The mRS element therefore contains one additional equation to determine the modulated resistance Rm

$$Rm = R \cdot e2. \tag{12}$$

The DYMOLA code for the modulated resistive source is shown in Fig.17.

```
{ Bond Graph of a modulated resistive source }

model type mRS
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter R=1.0
  local Rm
  Rm = R*e2
  Rm*f1 = e1
  e1*f1 = e2*f2
end
```

Figure 17: DYMOLA code for the modulated resistive source element

4.3.4 Modulated capacitance element

The modulated capacitance element mC is a very simple element, just like the other elements introduced so far. It is supposed to calculate the effort variable of the attached bond using the differential equation derived earlier in this section for a thermal capacitance

$$\frac{C}{e} \cdot \frac{de}{dt} = f. \tag{13}$$

The DYMOLA code for the modulated capacitance is shown in Fig.18.

4.3.5 The junctions and the bond submodel

The submodels or bond graph elements introduced so far are connected using the bonds and the two junction types shown in Fig.14. Fortunately, DYMOLA was designed with the concept of across and through variables in mind, what makes DYMOLA a perfect tool for bond graph modeling. In a DYMOLA cut, across and through variables are separated with a slash (`/`) as can be seen from all submodels introduced so far. If one or more cuts

are connected at a node, DYMOLA automatically sets all across or effort variables equal, while all through or flow variables are added up to zero. The zero junction therefore is easily represented by a node in DYMOLA, because the definition of a node in DYMOLA matches exactly the definition of the zero junction [2]. The one junction however cannot be represented that easily in DYMOLA.

In a DYMOLA node, all flow variables are added up to zero, while the effort variables are all set equal. For the one junction, the flows have to be set equal and the efforts have to be added up to zero. In order to emulate a one junction in DYMOLA, a bond submodel is introduced. All this submodel does is exchanging the flow and effort variables. A one junction then can be modeled using a DYMOLA node and as many bond models as are bonds attached to the one junction. This results in a bond graph in which all elements like R, RS, mRS or mC are connected to zero junctions exclusively. The DYMOLA code of the bond submodel is shown in Fig.19, while Fig.20 shows the DYMOLA expanded bond graph model.

```
{ Bond Graph modulated capacitor/compliance }

model type mC
  main cut A (e / f)
  parameter C = 413.
  C/e*der(e) = f
end
```

Figure 18: DYMOLA code for the modulated capacitance element

```
{ Bond Graph bond }

model type bond

  cut A (x / y) B (y / -x)
  main cut C [A B]
  main path P <A - B>

end
```

Figure 19: DYMOLA code for the bond submodel

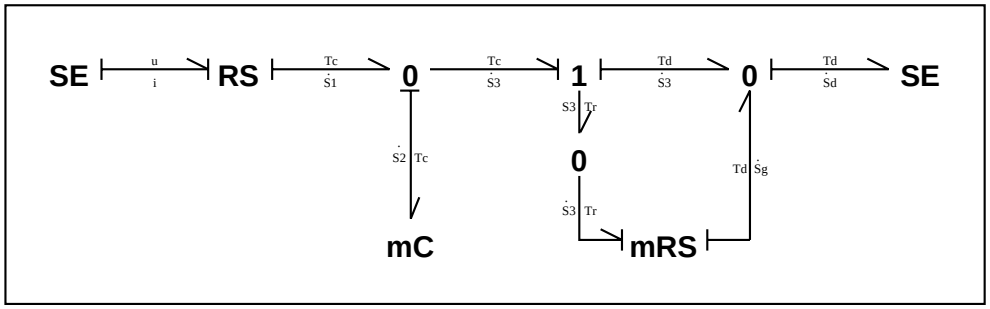

Figure 20: DYMOLA expanded bond graph for the resistive heating element

4.3.6 DYMOLA model description for the resistive heating element

Using the elements introduced in the previous sections, the bond graph shown in Fig.20 can be coded directly in DYMOLA. In order to keep the model as flexible as possible, and in order to demonstrate the modularity of DYMOLA models, the resistive heating element itself is modeled using a separate submodel. In doing this, the model of a very simple heating element is turned into the model of a resistor including thermal effects to be used in arbitrary bond graph models of electrical circuits. The electrical circuit shown in Fig.13 is just very simple. Fig.21 shows the code for the proposed submodel. The model of the entire circuit is actually built by connecting the power source to the resistor submodel as shown in Fig.22.

```
{bond graph model of a resistor including
 thermal effects}

model type resistor

submodel (bond) b1,b2,b3
submodel (RS)  res (R=1.)
submodel (mC)  cap (C=1.)
submodel (mRS) tra (R=1.)
submodel (SE)  env
main cut A(e/f)
node n1,n2,n3,n4

connect res from A to n1
connect cap at n1
connect b1 from n1 to n2
connect b2 from n2 to n3
connect b3 from n2 to n4
connect tra from n3 to n4
connect env at n4

end
```

Figure 21: DYMOLA code for the resistor submodel

```
{bond graph model for a resistive
 heating element}

@lib.bnd
@lib.rs
@lib.mrs
@lib.mc
@lib.se
@resistor.dym

model heater

    submodel (SE)  U0
    submodel (resistor) RT
    node  n

    connect U0 at n
    connect RT at n

    U0.E0  =  10.
    RT::env.E0 = 293.

end
```

Figure 22: DYMOLA model code for the resistive heating element

It should be noticed that all the bond graph submodels derived in this section can be used in any bond graph model of any type of system. They are not fixed to thermal or electrical systems. That is why the generic variable names e and f were used in the code to set up the derived equations. Whatever system is to be considered, one will find the same basic relations describing the system parts [7]. What changes are the meanings, dimensions and values of the parameters turning a generic submodel into a representation of a specific physical system. That is why bond graph models are much more versatile and modular than other types of models.

5 Reimplementation of a MMS example

As seen earlier in this text, the bond graph modeling approach offers a lot of advantages in modeling continuous systems with various interacting forms of energy. In a power plant, chemical, thermic, hydraulic, electric and even nuclear energy is involved in the design process. Therefore a unique modeling concept for all facets of the entire plant like the bond graph should be used to model a power plant.

The availability of the MMS macro library in source form as well as the complete MMS documentation offers the possibility of converting the given MMS models into bond graph models. Conversion of the given model means in particular, that the equations used to model the power plant components have to be rewritten in terms of adjugate power variables. These variables are voltage u and current i for the electrical case, pressure p and volume flow $\dot{V}$ for the hydraulic case and temperature T and entropy flow $\dot{S}$ for the thermal case [7]. Additionally, the structure of the model should be recognized and bond graphs should be developed from the model equations. Last, but not least a structurized bond graph model like the one developed in chapter 4 should be developed.

The example chosen to be reimplemented is the deaerator level study program already discussed in chapter 3.

5.1 The submodel PUMP

The MMS pump model simulates multiple parallel identical constant speed motor driven centrifugal pumps. It is based on the so called pump head curve

$$\dot{V} = f(\Delta H) \tag{14}$$

which is determined for each pump type individually by measurement. The head ΔH developed by the pump is calculated from the pressure difference over the pump using the simple relation for the static pressure in an incompressible fluid

$$\Delta H = \frac{p_{out} - p_{in}}{g\rho}. \tag{15}$$

The pressure difference has to be given from neighboring models or explicit boundary values. Equations (14) and (15) are in the above mentioned power variables p and $\dot{V}$, therefore no conversion is necessary for these equations.

No standard bond graph element is known so far to represent these equations. One could assign a pressure modulated flow source element to these two equations and connect it to a one junction calculating the pressure difference as shown in Fig.23.

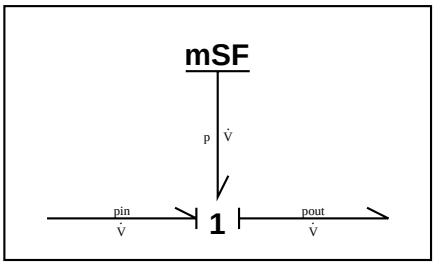

Figure 23: Hydraulic section of pump submodel

Because the pump is a non-dynamic model, the mass balance for MMS is simply

$$\dot{m}_{in} = \dot{m}_{out}. \tag{16}$$

For an incompressible fluid, the mass flow $\dot{m}$ can be replaced by the volume flow $\dot{V}$ if the density ρ of the fluid is known. Therefore equation (16) yields no further information besides

$$\dot{V}_{in} = \dot{V}_{out} \tag{17}$$

what validates the hydraulic model given by equations (15) and (14) and the corresponding bond graph of Fig.23.

MMS node equations	bond graph equations
$\Delta H = \frac{p_{out} - p_{in}}{g\rho}$	$\Delta H = \frac{p_{out} - p_{in}}{g\rho}$
$\dot{V} = f(\Delta H)$	$\dot{V} = f(\Delta H)$
$\dot{m}_{in} = \rho\dot{V}$, $\dot{m}_{in} = \dot{m}_{out}$	$\dot{V}_{in} = \dot{V}_{out} = \dot{V}$
$\eta = \eta_{max} - (\frac{\eta_{max}}{\dot{V}_d})^2(\dot{V}_d - \dot{V})^2$	$\eta = \eta_{max} - (\frac{\eta_{max}}{\dot{V}_d})^2(\dot{V}_d - \dot{V})^2$
$h_{out} = h_{in} + \frac{g\Delta H}{\eta}$	$\Delta T = \frac{1}{c\rho}(\frac{1}{\eta} - 1)(p_{out} - p_{in})$ $\Delta\dot{S} = \dot{V}\rho c \ln\frac{T_{out}}{T_{in}}$
$T_{out} = f(h_{out}, p_{out})$	
$\rho_{out} = f(h_{out}, p_{out})$	$\rho_{in} = \rho_{out}$

Table 1: Pump model equations

After the hydraulic section of the pump model could be transformed in a bond graph representation, consider the thermal part of the model. MMS uses the definition of the pump efficiency η to calculate the enthalpy h_{out} of the fluid leaving the pump. The efficiency of a pump is defined by

$$\eta = \frac{g \cdot \Delta H}{h_{out} - h_{in}} \tag{18}$$

according to the MMS Theory Manual [3]. The numerator is derived from the gain of potential energy $E = m \cdot g \cdot \Delta H$ of the pumped fluid. Taking the derivative with respect to time yields the hydraulic power imparted by the pump to the fluid. The denominator is derived from the total power $P = \dot{m}(h_{out} - h_{in})$ imparted by the pump.

The pump efficiency η can be approximated very well by an inverted parabola passing through zero at zero flow and a maximum value η_{max} at the pump design flow rate $\dot{V}_d$ [3].

$$\eta = \eta_{max} - (\frac{\eta_{max}}{\dot{V}_d})^2(\dot{V}_d - \dot{V})^2. \tag{19}$$

Equation (18) is solved for h_{out} and used in MMS to calculate the enthalpy h_{out} of the fluid leaving the pump. Here it becomes clear why startup and coastdown effects cannot be addressed by this model as stated in the MMS theory manual [3]. If the volume flow approaches zero, the efficiency does so, too, and therefore equation (18) becomes singular if solved for h_{out} because there is a head gain $\Delta H > 0$ even if there is no volume flow.

Equation (18) has to be replaced for the bond graph model because the bond graph doesn't operate on enthalpy. The definition of the enthalpy $h = u + pv$ is plugged into the equation. Sorting the variables u and pv and assuming an incompressible fluid yields

$$\Delta u = \frac{1}{\rho}(\frac{1}{\eta} - 1)(p_{out} - p_{in}). \tag{20}$$

Now a caloric equation of state for an incompressible fluid $du = cdT$ is used to calculate a temperature difference instead of an enthalpy difference. The obtained equation is :

$$\Delta T = \frac{1}{c\rho}(\frac{1}{\eta} - 1)(p_{out} - p_{in}). \tag{21}$$

The MMS model, that is clearly incompressible, uses two property functions to determine the outlet density and temperature as a function of enthalpy and pressure at the outlet. These two functions are eliminated from the new model, where the density of the fluid is taken to be constant and the temperature is calculated explicitly from the input temperature using equation (21), rather than implicitly using state functions or lookup tables.

Knowing the temperature difference ΔT and the heat capacitance c of the fluid, the entropy change can be calculated as well by integrating the caloric equation of state $ds = c\frac{dT}{T}$ yielding

$$\Delta \dot{S} = \dot{V}\rho c \ln \frac{T_{out}}{T_{in}}. \tag{22}$$

However, this entropy change only accounts for the entropy change due to the reversible heating of the fluid. Because the pumping of a fluid is a definetly irreversible process, there

should be some relationship to calculate the entropy generation by the pump. It is not clear how this can be done with the given model.

Nevertheless, it was possible to derive new equations compatible with the bond graph. Looking for bond graph elements, the same troubles as for the hydraulic part can be noticed. No bond graph element is known to represent equations (21) and (22) yet. More seriously, however, neither of the junction types allow the logarithm to be taken from a ratio of effort variables like required by equation (22). Therefore, the attempt to develop a bond graph from the equations given in MMS failed for the submodel PUMP. The old and newly derived model equations are summarized in table 1.

5.2 The submodel CONNI

MMS distinguishes between two types of submodels, namely resistive type submodels and storage type submodels. Resistive type submodels are intended to determine the flow through the submodel from the given pressure difference between in- and outlet of the modeled component as seen at the pump model. Storage type submodels are doing it the other way round. They calculate the pressure in the modeled component from the given flows at the submodel boundaries. In terms of bond graphs, storage type submodels are models containing only inertia and compliance effects, whereas resistive type submodels contain only resistive or dissipative effects. In order to avoid algebraic loops, one can only connect resistive type submodels to storage type ones and vice versa. The submodel CONNI is called an interconnective model in MMS and is used to connect two resistive type models which otherwise would create an algebraic loop. Therefore it is characterized as storage type model.

MMS node equations	bond graph equations
$\frac{dp_{out}}{dt} = K(\dot{m}_{in} - \dot{m}_{out})$	$\frac{dp_{out}}{dt} = \rho K(\dot{V}_{in} - \dot{V}_{out})$
$p_{in} = p_{out}$	$p_{in} = p_{out}$
$h_{in} = h_{out}$	$\dot{S}_{in} = \dot{S}_{out}$
$\rho_{out} = f(p_{out}, h_{out})$	$\rho_{in} = \rho_{out}$
$T_{out} = f(p_{out}, h_{out})$	$T_{in} = T_{out}$

Table 2: Connective node model equations

Unless it is called a storage type model, it is assumed that there is negligible mass and energy storage in the model and that the model has an undefined volume V [3]. Therefore

the mass conservation relation derived in the MMS Theory Manual [3] reduces to

$$\frac{dp_{out}}{dt} = K(\dot{m}_{in} - \dot{m}_{out}) \tag{23}$$

where K is an experimentally chosen constant. The MMS Theory Manual [3] gives a derivation for K. Because the model is taken incompressible it is no problem to convert this equation again using the constant density ρ :

$$\frac{dp_{out}}{dt} = \rho K(\dot{V}_{in} - \dot{V}_{out}) \tag{24}$$

The pressure variables at the in- and outlet are taken to be equal

$$p_{in} = p_{out}. \tag{25}$$

Comparing equation (24) with equation (6) describing the electrical capacitor shows that equation (24) indeed represents a capacitance, determining the effort variable p from the given flow. The hydraulic part of the connective modul therefore can be represented by the bond graph shown in Fig.24.

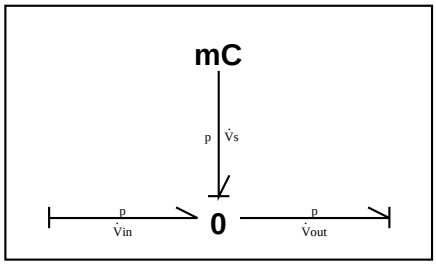

Figure 24: Hydraulic section of CONNI submodel

The thermal part of the model is converted easily. Because of the quasi steady state performance of the model, there is no change in enthalpy for the MMS-model and therefore there is no entropy or temperature change either. The MMS-equations

$$h_{in} = h_{out} \tag{26}$$

$$\rho_{out} = f(p_{out}, h_{out}) \tag{27}$$

$$T_{out} = f(p_{out}, h_{out}) \tag{28}$$

are turned into

$$\dot{S}_{in} = \dot{S}_{out} \tag{29}$$

$$\rho_{in} = \rho_{out} \tag{30}$$

$$T_{in} = T_{out}. \tag{31}$$

All model equations for the submodel CONNI, that is fully compatible with the bond graph approach are summarized in table 2.

5.3 The submodel VALVE

The submodel valve is used to simulate frictional pressure losses for single phase water flow due to a valve and associated piping. The valve conductance can be modulated, so that the flow through the valve is changed. VALVE is a resistive model in terms of MMS, and it is taken adiabatic and quasi steady state.

MMS node equations	bond graph equations
$\Delta p = p_{out} - p_{in} + \Delta h \rho g$	$\Delta p = p_{out} - p_{in}$
$\dot{m} = C_f \sqrt{\Delta p \rho}$	$\dot{V} = C_f \sqrt{\frac{\Delta p}{\rho}}$
$C_f = \frac{C_v C_p}{\sqrt{C_v^2 + C_p^2}}$	$C_f = \frac{C_v C_p}{\sqrt{C_v^2 + C_p^2}}$
$C_v = \lvert y \rvert^3 C_{v,max}$	$C_v = \lvert y \rvert^3 C_{v,max}$
$\dot{m}_{in} = \dot{m}_{out}$	$\dot{V}_{in} = \dot{V}_{out}$
$h_{in} = h_{out}$	$\Delta T = \frac{\Delta p}{\rho c}$
$T_{out} = f(h_{out}, p_{out})$	$\Delta \dot{S} = \dot{V} \rho c \ln \frac{T_{out}}{T_{in}}$
$\rho_{out} = f(h_{out}, p_{out})$	$\rho_{in} = \rho_{out}$

Table 3: Valve model equations

In MMS the mass flow through the valve is calculated using the incompressible relationship for the head loss in a pipe with fully developed turbulent flow $\Delta H = f \frac{l}{D} \frac{v^2}{2g}$. Assuming a constant piping diameter D and a constant density ρ, the velocity v can be replaced by $v = \frac{\dot{m}}{A\rho}$, while ΔH is replaced by the static pressure $\Delta p = \rho g \Delta H$. Solving for the mass flow yields

$$\dot{m} = C_f \sqrt{\Delta p \rho}. \tag{32}$$

The differential pressure Δp is calculated from the relationship

$$\Delta p = p_{out} - p_{in} + \rho g \Delta h \tag{33}$$

which provides for an additional head gain (static pressure) due to an elevation loss in the piping associated with the valve. The overall valve conductance C_f is determined from the associated pipe conductance C_p and the valve conductance C_v using the relationship

$$C_f = \frac{C_v C_p}{\sqrt{C_v^2 + C_p^2}}. \tag{34}$$

The valve conductance C_v is calculated from a function of the valve position y and the maximum valve conductance $C_{v,max}$

$$C_v = |y|^3 C_{v,max}. \tag{35}$$

Replacing the mass flow $\dot{m}$ in equation (32) gives an equation that calculates the volume flow $\dot{V}$ as a function of the differential pressure Δp for the incompressible case

$$\dot{V} = C_f \sqrt{\frac{\Delta p}{\rho}}. \tag{36}$$

The model is taken quasi steady state, there is no storage in the valve and therefore the mass flow $\dot{m}$ is constant in the valve yielding

$$\dot{m}_{in} = \dot{m}_{out} \tag{37}$$

for MMS. This is rewritten as constant volume flow $\dot{V}$

$$\dot{V}_{in} = \dot{V}_{out} \tag{38}$$

for the bond graph since the density is taken constant, because as seen in the submodel PUMP before, the density at the valve outlet will not be determined any more from a state function

$$\rho_{out} = f(h_{out}, p_{out}) \tag{39}$$

of the fluid but will be taken to be a constant.

Looking for bond graph elements to represent these equations, it is seen first that the additional head gain provided by the model in equation (33) is easily represented by an effort source attached to a one junction. Equation (36) is represented by a modulated resistive element similar to the one introduced for equation (8) in chapter 4. The only difference is that a relation of the form

$$\Delta p = R(\dot{V}) \cdot \dot{V} \tag{40}$$

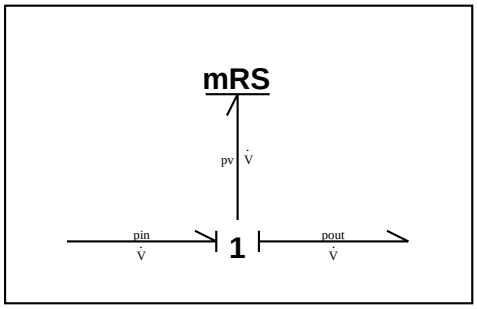

Figure 25: Hydraulic section of VALVE submodel

is obtained from equation (36) that has the resistance modulated by the flow variable rather than by the effort variable. The hydraulic part of the MMS valve model therefore can be represented by the bond graph shown in Fig.25.

Considering again the thermal part of the model, it turns out that the model for VALVE is an adiabatic and quasi steady state model and that therefore the enthalpy is constant over the valve. MMS uses the energy balance

$$h_{in} = h_{out}. \tag{41}$$

If again the definition of enthalpy $h = u + pv$ is plugged in and the variables u and $pv = p/\rho$ are sorted together, this equation turns into

$$\Delta u = \frac{\Delta p}{\rho} \tag{42}$$

where a caloric equation of state $du = cdT$ can be used to calculate the temperature difference from inlet to the outlet of the valve, as has been seen already for the submodel PUMP.

$$\Delta T = \frac{\Delta p}{\rho c} \tag{43}$$

The amount of entropy added to the fluid is determined from a caloric equation of state $ds = c\frac{dT}{T}$ as well :

$$\Delta \dot{S} = \dot{V} \rho c \ln \frac{T_{out}}{T_{in}}. \tag{44}$$

In the MMS model the temperature of the fluid at the valve outlet is determined as usual from a state function

$$T_{out} = f(h_{out}, p_{out}) \tag{45}$$

that is no longer used for the bond graph model.

The equations derived for the thermal part of the submodel VALVE are very similar to the equations derived for the thermal part of the submodel PUMP. Consequently, no

bond graph representation can be found for these equations either. The old and new model equations are summarized in table 3.

5.4 The submodel PIPE

The submodel PIPE can be used to model the pressure loss in a certain pipe. PIPE is a resistive model like VALVE. It offers many options like inertia and thermal effects, but none of them are used in the example to be reimplemented. As like VALVE, PIPE is a model for single phase flow only.

MMS node equations	bond graph equations
$\Delta p = p_{out} - p_{in} + \rho_{out} g \Delta h$	$\Delta p = p_{out} - p_{in}$
$\rho_{avg} = \frac{(\rho_{in} + \rho_{out}}{2})$	
$\dot{m}_{avg} = C_f \sqrt{\Delta p \rho_{avg}}$	$\dot{V} = C_f \sqrt{\frac{\Delta p}{\rho}}$
$\dot{m}_{in} = \dot{m}_{out} = \dot{m}_{avg}$	$\dot{V}_{in} = \dot{V}_{out}$
$h_{in} = h_{out}$ $T_{out} = f(h_{out}, p_{out})$	$\Delta T = \frac{\Delta p}{\rho c}$ $\Delta \dot{S} = \dot{V} \rho c \ln \frac{T_{out}}{T_{in}}$
$\rho_{out} = f(h_{out}, p_{out})$	$\rho_{in} = \rho_{out}$

Table 4: Pipe model equations

As in the submodel VALVE, the mass flow is calculated from the incompressible relationship for the head loss in a pipe with fully developed turbulent flow $\Delta H = f \frac{l}{D} \frac{v^2}{2g}$, which yields

$$\dot{m} = C_f \sqrt{\Delta p \rho_{avg}}. \tag{46}$$

The average density ρ_{avg} given by

$$\rho_{avg} = \frac{\rho_{in} + \rho_{out}}{2} \tag{47}$$

is used in connection with that relationship here, because the submodel PIPE is intended to be used with compressible fluids as well. However, the relations for incompressible flow can be used with decent accuracy as long as the pressure drop over the modeled flow resistance is small (less than 10%) compared to the upstream pressure as stated in the MMS Theory Manual [5]. The downstream density ρ_{out} is determined as in the other modules from a fluid state function

$$\rho_{out} = f(p_{out}, h_{out}). \tag{48}$$

The conversion of the equations works just the same way as in submodel VALVE. The differential pressure Δp is given by

$$\Delta p = p_{out} - p_{in} + \Delta h \rho g \tag{49}$$

from neighboring models or boundary conditions, providing for an additional head gain due to elevation loss of the pipe. This equation remains the same in both model descriptions. The average volume flow instead of the mass flow is calculated using

$$\dot{V}_{avg} = C_f \sqrt{\frac{\Delta p}{\rho}}. \tag{50}$$

The bond graph for the hydraulic part of the submodel PIPE is just the same as for the submodel VALVE. However, this bond graph shown again in Fig.26 is not capable of representing compressibility effects at all.

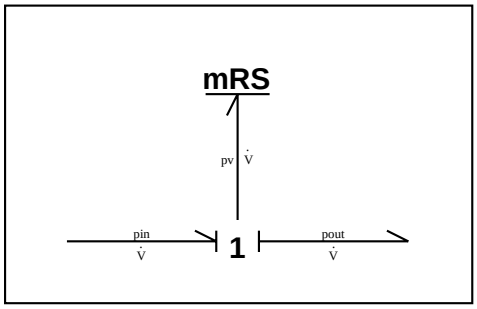

Figure 26: Hydraulic section of submodel PIPE

As for the submodel VALVE, the pipe is taken to be adiabatic and therefore

$$h_{in} = h_{out} \tag{51}$$

which is transformed into

$$\Delta T = \frac{\Delta p}{\rho_{avg} c} \tag{52}$$

using the same technique as in submodel VALVE. Again this relationship should be used rather than a state function

$$T_{out} = f(p_{out}, h_{out}). \tag{53}$$

The amount of entropy added to the fluid is determined from a caloric equation of state $ds = c\frac{dT}{T}$ as well

$$\Delta \dot{S} = \dot{V} \rho c \ln \frac{T_{out}}{T_{in}}. \tag{54}$$

Of course no bond graph representation can be found for these equations either. The old and new model equations are summarized in table 4.

5.5 The submodel JUNC

The submodel JUNC is used to merge several fluid flows into one as done by manifolds or tees. It is adiabatic and assumes quasi steady state performance just like all the other modules considered so far.

MMS node equations	bond graph equations
$p_{out} = p_{in,1} = p_{in,2}$	$p_{out} = p_{in,1} = p_{in,2}$
$\dot{m}_{out} = \dot{m}_{in,1} + \dot{m}_{in,2}$	$\dot{V}_{out} = \dot{V}_{in,1} + \dot{V}_{in,2}$
$\rho_{out} = \frac{\dot{m}_{in,1}\rho_{in,1} + \dot{m}_{in,2}\rho_{in,2}}{\dot{m}_{out}}$	
$h_{out} = \frac{\dot{m}_{in,1}h_{in,1} + \dot{m}_{in,2}h_{in,2}}{\dot{m}_{out}}$	
$T_{out} = \frac{\dot{m}_{in,1}T_{in,1} + \dot{m}_{in,2}T_{in,2}}{\dot{m}_{out}}$	

Table 5: Junction model equations

At a junction, all pressures at all inlets and outlets are equal, resulting in the following equation for both, the MMS and the bond graph description

$$p_{out} = p_{in,1} = p_{in,2}. \tag{55}$$

While the pressure is the same at all combined components, all incoming mass flows add up resulting in the leaving mass flow if no mass is stored in the junction. Therefore the equation

$$\dot{m}_{out} = \dot{m}_{in,1} + \dot{m}_{in,2} \tag{56}$$

is used in the MMS module JUNC to determine the leaving mass flow. This equation can be rewritten using the densities ρ of the fluids :

$$\rho_{out}\dot{V}_{out} = \rho_{in,1}\dot{V}_{in,1} + \rho_{in,2}\dot{V}_{in,2}. \tag{57}$$

The outlet density ρ_{out} for the MMS modul is calculated as mixed bulk density of the entering flow streams

$$\rho_{out} = \frac{\dot{m}_{in,1}\rho_{in,1} + \dot{m}_{in,2}\rho_{in,2}}{\dot{m}_{out}}. \tag{58}$$

Replacing $\dot{m}$ again with $\rho\dot{V}$ and solving for ρ_{out} yields

$$\rho_{out}^2\dot{V}_{out} = \rho_{in,1}^2\dot{V}_{in,1} + \rho_{in,2}^2\dot{V}_{in,2} \tag{59}$$

that is clearly in contradiction with equation (57). Both equations (57) and (59) are the same only if the density of the fluids to be merged are the same. In that case, equation (58) is useless and equation (57) yields

$$\dot{V}_{out} = \dot{V}_{in1} + \dot{V}_{in2}\,. \tag{60}$$

A bond graph is easily found for a manifold if the combined flow streams have the same density, e.g. for water. In this case, the flows are added using equation (60), while the pressures are forced to be the same by equation (55) resulting in a simple zero junction as shown in Fig.27.

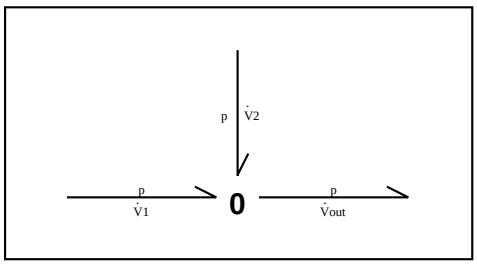

Figure 27: Hydraulic section of submodel JUNC

Looking at the example to be reimplemented once more shows that the modul JUNC is supposed to merge a steam and a water flow. This cannot be addressed with the model given in MMS. If compressibility effects are to be addressed in a manifold, a much more sophisticated model has to be developed. Obviously, the attempt to calculate average densities like statistical mean values fails to model the physical reality correctly.

After the hydraulic model implemented in the modul JUNC has proven not to be applicable for compressible fluids, consider the thermal model implemented in the MMS modul. Because no heat transfer to the environment is considered, temperature and entropy of the fluid leaving the manifold is calculated as the mixed bulk temperature and enthalpy of the entering fluid temperatures and enthalpies as well using the equations

$$T_{out} = \frac{\dot{m}_{in,1}T_{in,1} + \dot{m}_{in,2}T_{in,2}}{\dot{m}_{out}} \tag{61}$$

$$h_{out} = \frac{\dot{m}_{in,1}h_{in,1} + \dot{m}_{in,2}h_{in,2}}{\dot{m}_{out}}. \tag{62}$$

These equations also are applicable only if the fluids to be merged have at least the same aggregate state. While equation (62) doesn't account for the heat of vaporization, equation (61) is only applicable if the fluids to be merged have the same heat capacitance which changes drastically with the aggregate state of a substance.

According to this, the attempt to develop a bond graph from this modul doesn't make any sense at all. A much more sophisticated model has to be used if the effects occurring in a junction of that type have to be addressed properly. The old and new model equations for incompressible fluids are summarized in table 5.

5.6 The submodel DEAER

The submodel DEAER simulates a deaerator or open feedwater heater. The model is applicable for all conditions in which the steam and water in the deaerator are in phase equilibrium. The MMS model includes only one single port for deaerating steam and feedwater inlet. The MMS Theory Manual [3] advises the user to use the MMS junction module to combine the incoming feedwater and steam flows. This is a big problem of the model, because as shown earlier, the junction model is not able to do this correctly.

MMS node equations	bond graph equations
$p_{out} = p_{in} + \rho_{out} g \Delta H$	$p_{out} = p_{in}$
$p_{in} = f(\bar{u}, \bar{\rho})$	$p_{in} = f(\bar{u}, \bar{\rho})$
$\frac{d\bar{\rho}}{dt} = \frac{1}{V_t}(\dot{m}_{in} - \dot{m}_{out})$	
$\frac{d\bar{u}}{dt} = \frac{\dot{m}_{in} h_{in} - \dot{m}_{out} h_{out} - V_t \bar{u} \frac{d\bar{\rho}}{dt}}{\bar{\rho} V_t}$	
$h_{out} = f(p_{in})$	
$T_{out} = f(p_{in}, h_{out})$	
$L = D \frac{V_t}{V_s} (\frac{\bar{\rho} - \rho_{vap}}{\rho_{liq} - \rho_{vap}})$	
$\rho_{liq} = f(p_{in}, h_{liq})$	
$\rho_{vap} = f(p_{in}, h_{vap})$	
$h_{liq} = f(p_{in})$	
$h_{vap} = f(p_{in})$	

Table 6: Deaerator model equations

In order to derive the model equations for the hydraulic part of the model, it is assumed that the pressure is the same everywhere in the deaerator vessel. However, the model includes an additional elevation head gain due to associated downstream piping to provide for a certain NPSH ΔH (net positive suction head) needed to avoid cavitation in a downstream feedwater

pump. The pressure at the outlet of the deaerator is therefore determined from the equation

$$p_{out} = p_{in} + \rho_{out} g \Delta H. \tag{63}$$

The pressure p_{in} inside the deaerator is determined from the vapor pressure curve for water given as a function of the bulk specific internal energy $\bar{u}$ and the bulk density $\bar{\rho}$ of the fluid inside the vessel

$$p_{in} = f(\bar{u}, \bar{\rho}) \tag{64}$$

which are determined from a mass and an energy balance respectively.

The bulk internal specific energy $\bar{u}$ in the deaerator is calculated from the energy balance

$$\frac{d\bar{u}}{dt} = \frac{\dot{m}_{in} h_{in} - \dot{m}_{out} h_{out} - V_t \bar{u} \frac{d\bar{\rho}}{dt}}{\bar{\rho} V_t} \tag{65}$$

by integration. This energy balance is derived from the extensive energy balance

$$\frac{dU}{dt} = \dot{m}_{in}(u_{in} + p_{in} v_{in}) - \dot{m}_{out}(u_{out} + p_{out} v_{out}) \tag{66}$$

by plugging in the derivative of the extensive internal energy $U = mu = \rho V u$. The enthalpy of the fluid leaving the deaerator is calculated from another state function

$$h_{out} = f(p_{in}). \tag{67}$$

The mass balance for the deaerator in MMS is given by

$$\frac{d\bar{\rho}}{dt} = \frac{1}{V_t}(\dot{m}_{in} - \dot{m}_{out}). \tag{68}$$

The mass balance equation is also integrated over time to obtain the bulk density $\bar{\rho}$ of the fluid in the vessel.

This set of equations cannot be rewritten like the equations of the modules PIPE, VALVE and PUMP, because the u and p variables cannot be sorted together like seen for the other modules. This is due to the fact that the mass flow cannot be eliminated from these equations. Consider equation (68). Assuming quasi steady state performance like for modules PIPE, PUMP or VALVE would result in $\dot{m}_{in} = \dot{m}_{out}$ eliminating the whole equation from the model. Obviously, the bond graphs for incompressible and quasi steady state models applied so far are not able to deal with the mass balance needed for this model.

Yet another problem that comes to mind is, that if dealing with dynamic processes, dynamic equations should be used. Unfortunately, state equations like $du = cdT$ or a vapor pressure curve used for conversion of the modules PIPE or VALVE are not dynamic

equations at all. As stated in thermodynamic text books, e.g. [15], these equations apply only for ideal equilibrium conditions, what means that infinitely large amount of time is taken to accomplish a change of state, so that the system under consideration is in equilibrium all the time.

Because the attempt to separate the hydraulic and the thermal parts of the model used in MMS failed, the equation

$$T_{out} = f(p_{in}, h_{out}) \tag{69}$$

to calculate the temperature inside the deaerator vessel, that is yet another state function, could not be replaced either.

Geometry calculations are used to calculate the water level L in the deaerator storage tank. If the water level varies linearly with volume (uniform cross section of the tank) the level is given by

$$L = D\frac{V_t}{V_s}\left(\frac{\bar{\rho} - \rho_{vap}}{\rho_{liq} - \rho_{vap}}\right) \tag{70}$$

where D denotes the tank diameter and V_s the volume of the storage tank. A derivation for this equation is also given in the MMS Theory Manual [3]. The required densities of the liquid and vapor phases are determined as functions of pressure p and enthalpy h

$$\rho_{liq} = f(p_{in}, h_{liq}) \tag{71}$$

$$\rho_{vap} = f(p_{in}, h_{vap}). \tag{72}$$

The enthalpies themselves are determined as functions of the pressure

$$h_{liq} = f(p_{in}) \tag{73}$$

$$h_{vap} = f(p_{in}). \tag{74}$$

Level calculations for the bond graph have to be done with basically the same relationship (70). However, the densities should be calculated not from state functions, but from mass and volume, according to the definition of the density.

Unfortunately, nothing could be done to this model to fit it in a bond graph representation. However, a new model should be developed anyway, because the junction module is needed to merge the incoming flows prior to enter the deaerator. As seen in the previous section, the MMS junction module is not able to do this correctly. The new model should provide for two inlet ports for water and deaerating steam and one outlet port for the heated water. The old and new model equations are summarized in table 6.

5.7 The submodel PICONT

The modul PICONT simulates a simple proportional and integral controller including antireset windup. Of course, this modul is a control component and therefore no bond graph representation has to be found for it, because the controller deals with powerless signals exclusively. The controller used for the reformulated example of chapter 3 would be fine with a bond graph model as well as with any other type of model.

6 A new model

In the last chapter, it has been seen that the MMS model cannot be translated completely into a bond graph representation. While the hydraulics of incompressible fluids can be modeled using bond graphs fairly easy, major problems have to be faced if compressible fluids and even phase changes have to be considered. In this chapter, a new model, capable of dealing with compressibility effects as well as with phase changes is proposed.

As has seen from the discussion of the modules VALVE and PIPE, incompressible flow can be described quite well by the bond graph. Therefore, new models for PIPE, VALVE and PUMP will be derived first, before an attempt is made to develop a compressible model needed for the deaerator simulation.

6.1 Bond graph model for a pipe

The hydraulic model given in MMS for a pipe is enhanced a little bit in order to achieve a better model compatibility. Consider the MMS model of the entire feedwater train given in chapter 3 again. A so called connective node was inserted between the valve and the pump to avoid an algebraic loop. This modul has to be inserted between two modules characterized as storage type moduls in the MMS terminology [5, 3]. All the connective node modul contains is a capacitance element in terms of the bond graph. This element is equipped with a non–physically derived capacitance K as can be seen from the MMS Theory Manual [3].

Bond graph models for incompressible flow in pipes found in the literature [7, 11] include capacitances as well as inertance effects occurring in a pipe. Therefore, the new models are given a more detailed hydraulic part, resulting in a better interconnectability without non–physical interconnective nodes. The new hydraulic model of the incompressible fluid flow in a pipe is shown in Fig.28. It contains resistive, capacitive and inertia effects.

The thermal model used in MMS is not useable at all with the bond graph, as has been shown in chapter 5. Therefore a new model is proposed to model the thermal effects in a pipe. The effect of heat conduction was introduced in chapter 4. Heat conduction occurs in a pipe or in any fluid if the temperature is not homogen in the fluid. Therefore, heat conduction is addressed by the new model. However, the conduction process is altered with the movement of the fluid. Consequently, the heat conduction resistance is modulated with the fluid velocity, as proposed in [7]. The new bond graph element is given the name conductive source mGS and it is also applicable for convection. Another thermal effect occurring is the storage of heat in a piece of matter, already addressed in chapter 4 as well

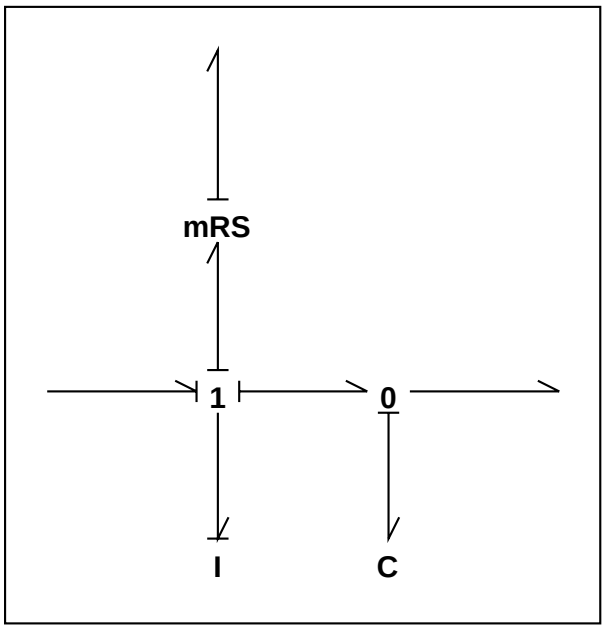

Figure 28: Hydraulic part of the incompressible fluid flow model

as the conduction. The modulated capacitance element is used here in the same way as seen in chapter 4. A third effect to overcome in the new model is the entropy flow from one end of the pipe to the other due to the movement of the fluid. This effect is modeled by introducing a so called modulated flow source SFS. The entropy flow generated by this source is modulated with the volume flow taken from the hydraulic model part and it changes with the specific entropy of the moving fluid as well. The specific entropy of the fluid has to be determined from a state function $s = f(p, T)$. The bond graph for the thermal part of the model is shown in Fig.29.

The two model parts, hydraulic an thermal, are connected by a bond leaving the hydraulic resistance. As has been seen also in chapter 4, a resistor dissipates power, but the dissipated power usually appears as heat output. This is the case for the hydraulic resistor as well as for an electrical resistor. Therefore, the hydraulic resistor is modeled as resistive source producing an entropy flow at a certain temperature to be introduced to the thermal subsystem, just as has been demonstrated in the example presented in chapter 4. The complete model consisting of thermal and hydraulic model is shown in Fig.30.

If a long pipe is to be modeled, several of these models can be put in a line to account for the dynamics occurring in the pipe itself, like oscillations and pressure waves occurring in a pipe system upon fast valve actuation or other fast pressure alterations. As shown in [7], the proposed L–C–chain represents the lossless wave equation, as well as a chain of thermal mC

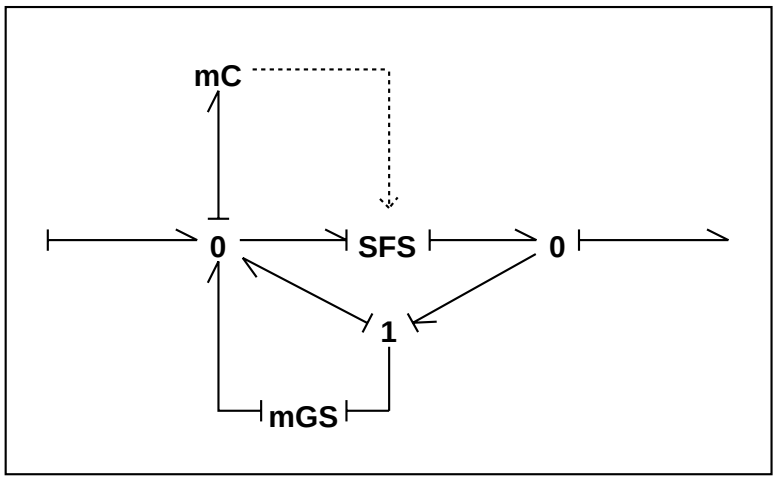

Figure 29: Thermal part of the incompressible fluid flow model

and mRS elements without the modulated flow source introduced to model fluid movements represents the heat conduction equation.

Based on the principles introduced in chapter 4, these bond graphs are translated to a DYMOLA model description. This is yet another excellent example of how structurized and modular models are built using the DYMOLA modeling language.

The hydraulic part of the model, according to the bond graph shown in Fig.28, is coded in DYMOLA as shown in Fig.31. The submodel contains three cuts, two hydraulic cuts to be combined in a path to faciliate the interconnectability with other hydraulic submodels, and the thermal cut to be connected to a thermal submodel.

The thermal submodel shown in Fig.32 is based on the bond graph shown in Fig.29. It contains only the two thermal cuts connected with a path–statement. Thermal inputs like the dissipated power input from the hydraulic submodel can be connected to either of the thermal cuts directly.

This can be seen in the submodel of the next higher hierarchical level. The two submodels coded so far are combined into the model of a pipe, or a segment of a pipe only. On this level, the cuts for thermal and hydraulic power input are combined into a more powerful cut, demonstrating the ability of DYMOLA to structure cuts as well as models and data. The DYMOLA code is printed in Fig.33.

The submodel for a segment of a pipe may be used to model pipes with incompressible fluid flow and an arbitrary number of segments. In Fig.34, the code for a pipe assembled from 4 segments is printed as example.

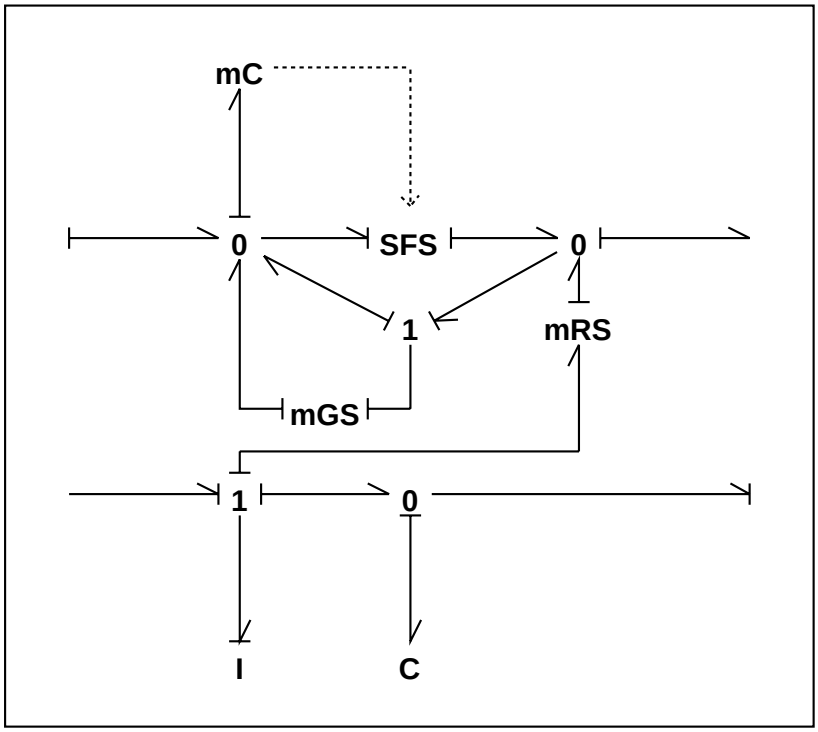

Figure 30: Bond graph for submodel pipe

6.2 Bond graph model for a valve

After the model for a pipe has been built, the model for a valve is very easy to obtain. Just as has been seen from the MMS model, the valve is basically a pipe with a variable resistance that can be controlled from the outside of the model. In contrast to the pipe, the valve is not modeled as distributed system like the pipe, therefore only one convective and one hydraulic segment is needed. The hydraulic segment itself appears to be slightly altered. In order to enable the valve to operate fully closed, i.e. with zero flow, the resistive element was modeled as conductive element to avoid a singularity for infinitely large resistances. Also, an additional terminal y was added to the resistive source mRSv of the valve, that changes the valve conductance according to the valve position to be simulated. The inductive element was eliminated from the model, otherwise the causality structure would not be consistent any more, because the causality of the mRSv element is different from the causality of the mRS element used for the pipe model. The bond graph for the valve model is shown in Fig.35. The code for the convective source element of the valve mRSv is printed in Fig.36, the code for the hydraulic segment of the valve is printed in Fig.37 and the code for the valve submodel is shown in Fig.38. The thermal segment used for the valve is the same as

```
{ bond graph model of hydraulic segment
 of a pipe }

model type HydSeg

submodel(I) ind (I=17.) (ic f=0.)
submodel(C) cp (C=8.93E-7) (ic e=0.98E6)
submodel(mRS) res (R=7.5)
submodel(bond) B1,B2,B3,B4
node n1,n2,n3

cut A(e1/f1) , B(e2/-f2) , ThOut(e3/-f3)
main cut C[A B]
main path  PH<A - B>

connect B1 from A to n1
connect B2 from n1 to n2
connect res from n2 to ThOut
connect B3 from n1 to B
connect B4 from n1 to n3
connect ind  at n3
connect B at cp

end
```

Figure 31: DYMOLA code for the hydraulic part of the pipe model

```
{ bond graph model for a convective segment
  of a pipe }

model type CvSeg

submodel mC (C=9.2E3) (ic e=545.)
submodel mGS (b=17.2)
submodel SFS (s0=7033.,m=3.73,T0=545,rho=3.4)
submodel (bond) B1,B2,B3
node n1,n2

cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
main path PT<InTherm - OutTherm>

parameter Area=0.1
terminal Vd

{ main convection line }
connect InTherm at mC
connect SFS from InTherm to OutTherm

{ reverse conduction effect }
connect B1 from OutTherm to n1
connect B2 from n1 to InTherm
connect B3 from n1 to n2
connect mGS from n2 to InTherm
mGS.vwater = Vd / Area

{ modulation inputs for SFS element }
SFS.Vd = Vd

end
```

Figure 32: DYMOLA code for the thermal part of the pipe model

used for the pipe, therefore refer to Fig.32 for the code.

6.3 Bond graph model for a pump

The last incompressible model to build is the model for the constant speed motor driven pump. Just like the valve, the pump is not modeled as distributed system, therefore only one hydraulic and one thermal segment is used to model the pump.

The thermal part of the pump is just the same as for the models valve and pipe. The hydraulic part of the pump could be modeled using basically the bond graph proposed in Fig.23. The pressure modulated flow source mSF could be extended with a thermal bond connecting to the thermal subsystem. The power delivered to the fluid then could be split between the thermal and hydraulic subsystems, using the efficiency given in the MMS model as a function of volume flow.

For the hydraulic part, it was decided not to use the bond graph derived from the MMS model, because this pump model is not able to address startup or shutdown of the pump as has been seen in chapter 5. Instead, a bond graph derived for a hydraulic motor in [7] is

```
{ bond graph model of a pipe segment }

model type PipeSeg

submodel HydSeg
submodel CvSeg (Area=0.5)

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

connect HydSeg from InHyd to OutHyd
connect CvSeg from InTherm to OutTherm
connect HydSeg:ThOut at OutTherm

CvSeg.Vd = fh1

end
```

Figure 33: DYMOLA code for a segment of the pipe model

```
{ bond graph model of a pipe, including thermal and flow effects }

model type Pipe

submodel (PipeSeg) PipeSeg1,PipeSeg2,PipeSeg3,PipeSeg4

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

connect InFluid - PipeSeg1 - PipeSeg2 - PipeSeg3 - PipeSeg4 - OutFluid

end
```

Figure 34: DYMOLA code for a pipe

used to model the pump. The bond graph is shown in Fig.39.

The pump is a constant speed motor driven pump, therefore a flow source, supplying a arbitrary torque at a fixed angular velocity ω is used as unmodulated power source. An also unmodulated resistive source element attached to a one junction accounts for the frictional losses of the mechanical components of the pump. The remaining power is transformed from the rotational to the hydraulic power domain, using a transformer TF. In the hydraulic domain, another resistive source allows a back flow in the pump dependent on the pressure built up due to the spinning pump wheel. A capacitance is connected to the outlet of the pump in order to keep consistency with the already derived bond graph models to provide for an arbitrary connectability of the hydraulic submodels.

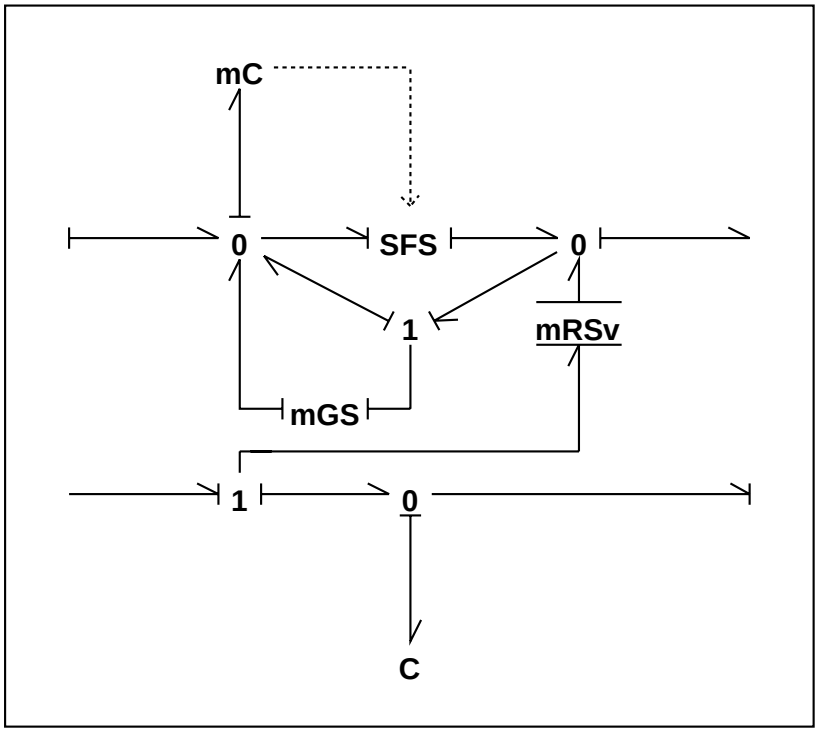

Figure 35: Bond graph for the submodel valve

The hydraulic and the thermal submodels are connected again with the bonds leaving the resistive elements. The entropy generated by frictional losses is introduced at the outlet cut of the thermal submodel, as this was done for the VALVE and PIPE models, too. The bond graph for the pump submodel is shown in Fig.40.

The code development from the graph again is relatively straight forward. Notice that the model was split in three submodels, one for the thermal part, shown in Fig32, one for the hydraulic part, shown in Fig.42, and one mechanical part, shown in Fig.41. This structure was chosen to enable a further extension and detailing of the model parts. It might be interesting for example, to enhance the mechanical model part by adding inertia elements for the pump wheel or add an electrical part accounting for an electric motor to drive the pump later on. In that case, just the mechanical model has to be replaced by a more sophisticated one without altering the rest of the model. The code for the entire submodel pump is shown in Fig.43.

```
{ Bond Graph of a modulated
  resistive source }

model type mRSv
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  terminal y
  parameter R0 = 10.
  local R

  R=R0*ABS(y)**3
  f1 = SQRT(ABS(e1*R))
  e1*f1 = e2*f2

end
```

Figure 36: DYMOLA code for the modulated convective source mRSv

```
{ bond graph model of hydraulic segment
  of a valve }

model type HydSegV

submodel(C) cp (C=1.2E-11) (ic e=2.48E6)
submodel(mRSv) vres (R0=5.27E-7)
submodel(bond) B1,B2,B3
node n1,n2

cut A(e1/f1) , B(e2/-f2) , ThOut(e3/-f3)
main cut C[A B]
main path  PH<A - B>

connect B1 from A to n1
connect B2 from n1 to n2
connect vres from n2 to ThOut
connect B3 from n1 to B
connect B at cp

end
```

Figure 37: DYMOLA code for the hydraulic part of the valve model

6.4 Bond graph model for the deaerator

For the deaerator modul in MMS, no bond graph representation could be found from the model equations given in the MMS macro. Furthermore, it was recognized that the deaerator module in MMS should provide for different inlet ports for steam and water and should not depend on the junction module to merge steam and water flows.

Consequently, a completely new model has to be derived. Most of this new model was taken from the models describing chemical reaction dynamics, discussed thoroughly in [7]. As a first step, a model is to be built for the phase equilibrium of water and steam in a closed system. Once this is done, it is no problem to make this system an open one, just by attaching bonds for mass and entropy flow supplying power in various forms from the outside. As an additional constraint, a process is considered with water exclusively, that means in particular that no air is present in the closed vessel where the process takes place. This helps to make the model a little bit easier, because no partial pressures have to be considered.

The basic idea is to model the equilibrium of water and steam as a type of chemical reaction that turns water into steam (referred to as the vaporization reaction) and steam into water (referred to as the condensation reaction). As for chemical reactions, a dynamical equilibrium is reached if the same amount of water vaporized is condensed in a specific time interval.

The two species involved in the reaction are liquid and gaseous water. Each of these is

```
{ bond graph model of a valve }

model type Valve

submodel HydSegV
submodel CvSeg (Area=0.1)

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

terminal y

connect HydSegV from InHyd to OutHyd
connect CvSeg from InTherm to OutTherm
connect HydSegV:ThOut at OutTherm

CvSeg.Vd = fh1
CvSeg.y = y
HydSegV::vres.y = y

end
```

Figure 38: DYMOLA code for the valve model

represented by a zero junction. The two reactions are represented by two one junctions which balance the chemical power into and out of the chemical reaction. The reaction described by the bond graph shown in Fig.44 dies out if the chemical potentials of steam and liquid are the same or in other words, if equilibrium is reached[4].

Connected to the one junctions is a new bond graph element, the so called *chemical reactor* ChR controlling the chemical reaction and balancing the power in and out of the reaction. If liquid is vaporized for example, the power has to be delivered from somewhere, usually from the hydraulic or thermal power domains. The DYMOLA code for the chemical reactor ChR is shown in Fig.45.

The power balance equation used in the ChR element is the Gibbs Equation and it computes the entropy flow $\dot{S}$ generated or used by the reaction.

The volume flow generated or used by the reaction is computed from a derivative of a equation of state $V = \rho \cdot n$ of the liquid, assuming a vessel with a constant volume and an incompressible liquid.

Last but not least the mass flow between the reaction partners is computed from the reaction rate equation of the form $\nu = k \cdot n$, proposing that the amount of mass transferred by the reaction is dependent from a reaction rate constant k and proportional to the total

[4]The chemical potentials of water and steam are equal if both are saturated at the same temperature and vapor pressure.

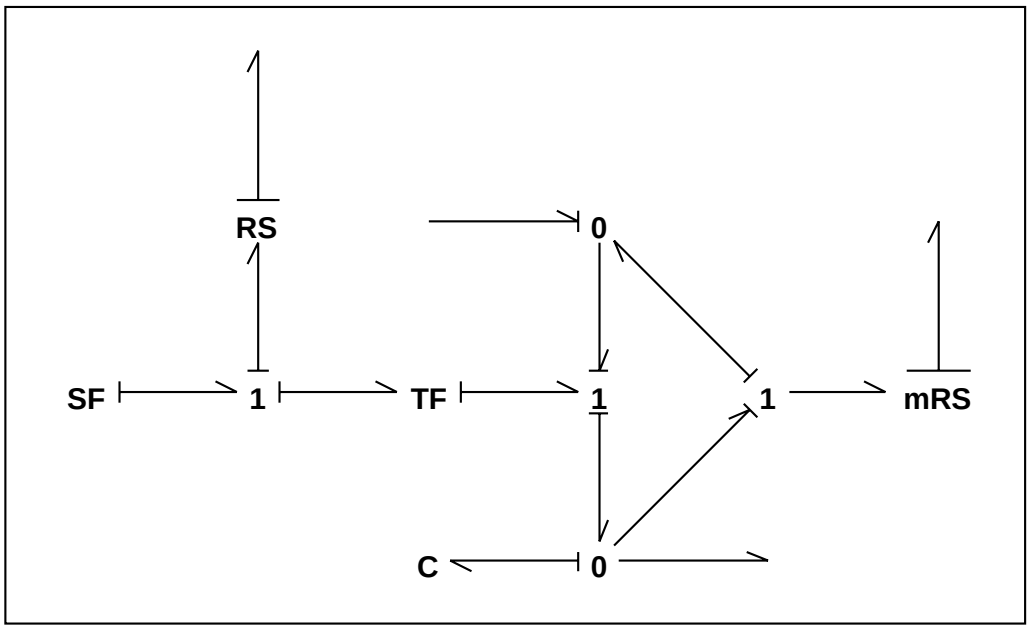

Figure 39: Hydraulic part of the pump model

mass available for the reaction. The computation of the reaction rate constant k is due to the fact that the reaction is in equilibrium if the three across variables describing the state of the two substances, T, p, μ are equal[5]. However, the pressure in a closed vessel containing a gas has to be the same everywhere. Therefore no physical pressure difference can occur between liquid and gas. There is yet another pressure that can be different from the real gas pressure in the vessel, namely the vapor pressure of the liquid if saturation is reached. The liquid starts to boil if the vapor pressure of the liquid is higher than the gas pressure and vapor condenses if the gas pressure gets higher than the vapor pressure of the liquid. The vapor pressure is a function of temperature only. Therefore, the reaction rate constant k is computed as a function of the chemical potential difference computed at the one junction of the corresponding bond graph and a theoretical pressure difference between the vapor pressure of the liquid and the actual pressure in the vessel.

Connected to the zero junctions is a energy storage element CF in which mass, volume and entropy is accumulated by integration of the related derivative terms $\dot{n}$, $\dot{V}$ and $\dot{S}$. The distribution of the energy among the three separate energy storage types chemical, thermal and hydraulic energy storage is determined from the Gibbs–Duhem equation.

Note that there are different submodels for the liquid and the vapor phase. The submodel

[5]Notice that one of the three variables is redundant. Usually, the two state variables p and T are sufficient to describe the state of a substance. The chemical potential is then computed using the formula $\mu = \frac{H-TS}{n}$.

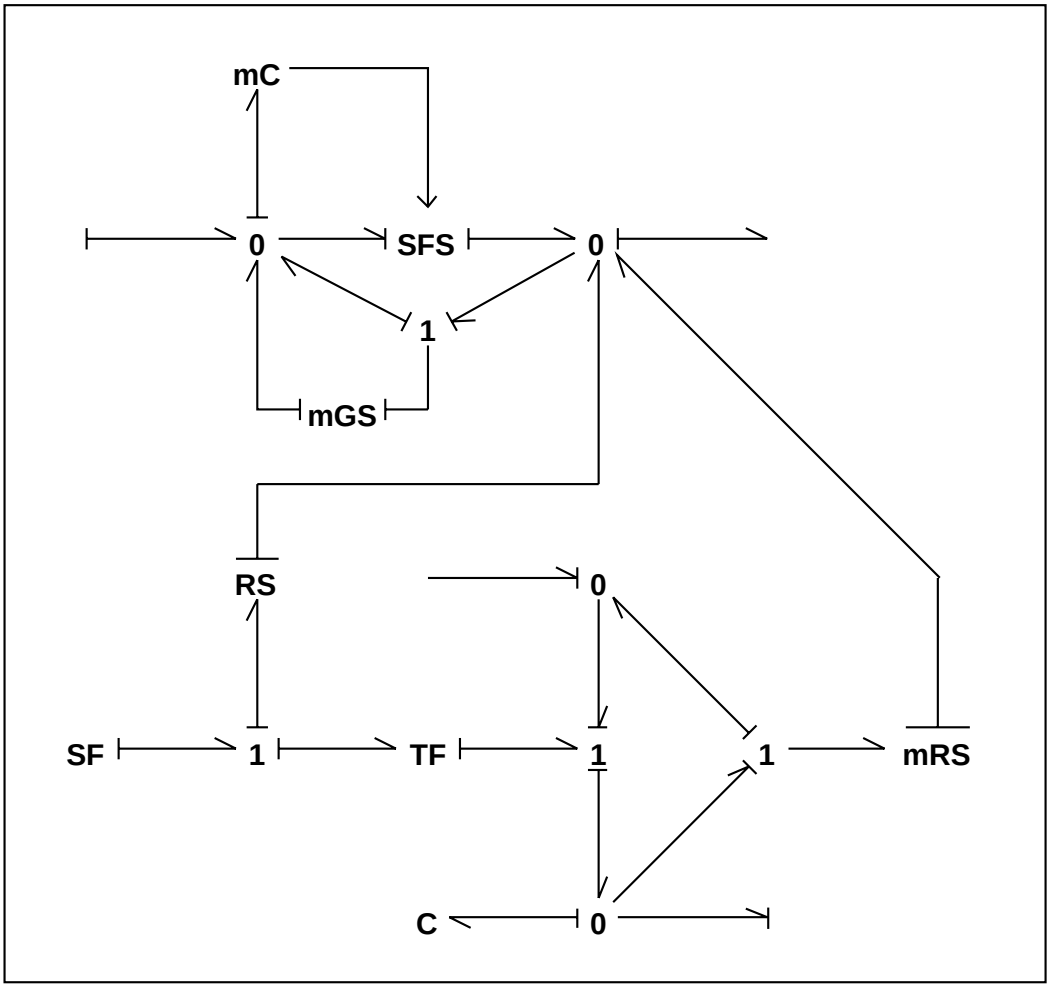

Figure 40: Bond graph for the submodel pump

attached to the liquid node CFl is shown in Fig.46. Note that the theoretical vapor pressure is used for the hydraulic cut, while the real pressure p is computed only in the submodel attached to the vapor node. The pressure change $\dot{p}$ used in the Gibbs–Duhem equations are linked with a terminal. No pressure is calculated in the CFl model.

Now compare the submodel attached to the vapor node CFg shown in Fig.47. While the CFl model pressure is taken from the CFg model, the volume for the CFg model is taken from the CFl model. This was done because the equation of state for an incompressible liquid is considerably easier to handle than for a gas.

So far, the chemical reaction bond graph of the steam/water system was considered. However, the thermal and hydraulic bonds attached to the ChR and CF elements in the bond graph shown in Fig44 have to be connected to something else.

Mass is not the only thing to be exchanged between the liquid and vapor subsystems.

```
{ mechanical section of a constant
  speed motor driven pump }

model type MechSec

submodel(RS) frmech (R=0.001)
submodel(bond) B1,B2,B3
node n1,n2

cut MechOut(em/-fm) , MechIn(ei / fi )
cut ThOut(et/-ft)
main cut C[MechIn MechOut]
main path P<MechIn - MechOut>

connect B1 from MechIn to n1
connect B2 from MechOut to n1
connect B3 from n1 to n2
connect frmech from n2 to ThOut

end
```

Figure 41: DYMOLA code for the mechanical part of the pump submodel

```
{ fluid flow section of a constant speed
  motor driven pump }

model type FlowSec

submodel(C)  capout (C=1.2E-11) (ic e=9.79E3)
submodel(mRS) frflow (R0=5.27E-7)
submodel(bond) B1,B2,B3,B4,B5,B6
node n1,n2,n3

cut InHyd(ei / fi) , OutHyd(eo/-fo)
cut MechIn(em/fm) , ThOut(et/-ft)
main cut C[InHyd OutHyd]
main path P<InHyd - OutHyd>

connect B1 from n1 to MechIn
connect B2 from InHyd to n1
connect B3 from n1 to OutHyd

connect B4 from OutHyd to n2
connect B5 from n2 to InHyd
connect B6 from n2 to n3

connect frflow from n3 to ThOut
connect capout at OutHyd

end
```

Figure 42: DYMOLA code for the hydraulic part of the pump submodel

Entropy and volume are exchanged as well during a reaction. Therefore exchange systems similar to the mass transfer system were built for entropy and volume flow. They exhibit the same basic structure as the mass transfer system and are shown in Fig.48 and Fig.49 respectively. The coding of these bond graphs in DYMOLA is again relatively straight forward, therefore the DYMOLA code for these bond graphs is printed only in the appendix.

If all three subsystems are assembled, the resulting bond graph no longer is a planar graph, but it bends around to form a sphere. Therefore no bond graph is presented here for the complete system. The graphs are connected at the CF and ChR elements respectively.

In the appendix, a complete model intended to model a closed water steam system is printed as DYMOLA code. However, this model was not executed successful so far. There are quite a few problems yet to be overcome.

Consider the equation used to calculate the pressure derivative in submodel CFg stating the pressure derivative to be a function of volume change exclusively.

$$\dot{p} = \frac{1}{C_{hy}} \cdot \dot{V}. \tag{75}$$

This is only correct for a closed system. Unfortunately, the vapor phase to be modeled with this equation is not a closed system. The pressure derivative has to be a function of mass

```
{ bond graph model of a pump, including thermal
  and flow effects }

model type Pump

submodel(SF) sf
submodel(TF) tf (m=1.7E-4)
submodel(MechSec) msec
submodel(FlowSec) fsec
submodel(CvSeg) cseg (Area=0.1)

node n1,n2,n3,n4,n5,n6

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

terminal rpm

connect sf to msec to tf to fsec:MechIn
connect fsec from InHyd to OutHyd
sf.F0 = rpm*2.*3.14

connect cseg from InTherm to OutTherm
connect msec:ThOut at OutTherm
connect fsec:ThOut at OutTherm
cseg.Vd = fh1

end
```

Figure 43: DYMOLA code for the pump submodel

flow and temperature change as well, as can be seen from the equation of state $p \cdot V = n \cdot R \cdot T$ that should be obtained upon integration of equation (75).

Another problem comes to mind if the calculation of the entropy stored in the CF elements is considered. From the thermal bond graph shown in Fig.49 the entropy to be stored in the CF elements is computed from the sum of the attached bonds. However, mass to be stored in the CF element carries entropy with it, too, that is not yet taken into account by the model.

6.5 Parameterization and test of the new model

While the previous section dealt with the physical structure of the deaerator level study model, this section expands on the computational problems of the new model. In order to obtain meaningful results from the proposed models, values for the resistances, inductances and capacitances as well as for the initial conditions and the boundary conditions of the submodels and the entire model have to be found. Due to the fact that a different model

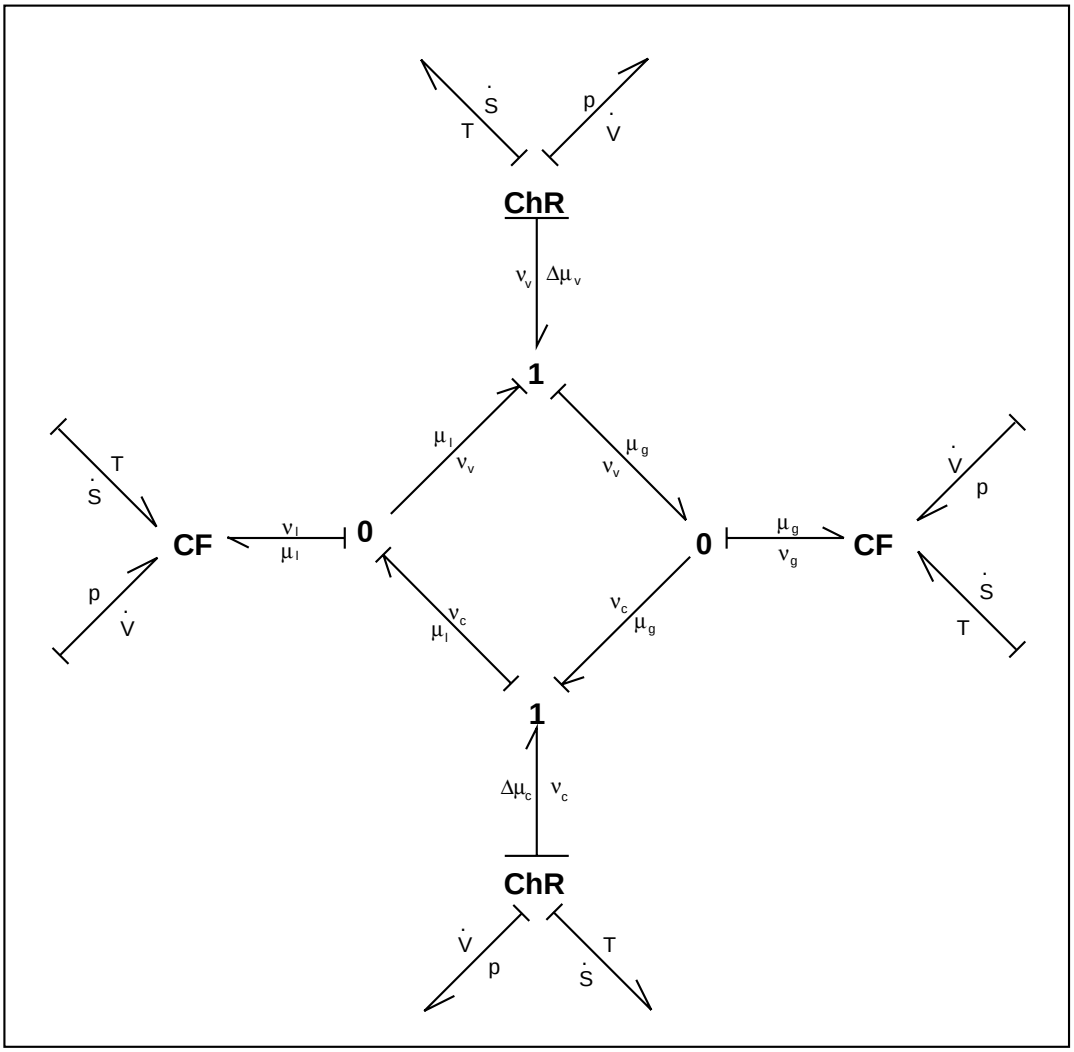

Figure 44: Chemical reaction bond graph for water/steam system

than the MMS reference model had been developed, a new set of parameters has to be found. The steady state boundary values as shown in Fig.50 of course remain the same for the new model. This enables us to compare the performance and the results of the different models.

6.5.1 Derivation of common parameters

A couple of elements are used in all submodels like capacitive, inductive and resistive elements or the modulated entropy source element (SFS) for the entropy convection model. For these elements, common formulas for the computation of the physical parameter values are derived in advance.

```
{ bond graph model for chemical reactor for mass
  transfer reaction}

model type ChRT

cut chem (mu/-nu)
cut hyd (p/-q)
cut therm(T/-Sd)
terminal n
parameter R=8.314
local k
constant eps=1.E-4 , rho=1.8E-5
constant epst=10. , one=1.0

{reaction rate : get nu}
k = 1. / ( 1. + EXP(BOUND(-20.,20.,mu/1000.+p/1000.))) -0.5
nu = k * n / 50000.

{derivative of equation of state : get q}
q = rho * nu

{Gibbs equation : get Sd}
0 = -p*q + (T+epst*SIGN(one,T))*Sd + mu*nu

end
```

Figure 45: DYMOLA code for the chemical reactor

Hydraulic flow inertance The derivation of the hydraulic flow inertance coefficient is due to [11]. Consider a moving fluid, e.g. in a pipe of length l and cross section A. The motion of the fluid is described by Newton's Law $F = ma$. The mass of the fluid in the pipe is $m = \rho Al$, the force accelerating the fluid is $F = A(p_1 - p_2)$ and the velocity of the fluid is $v = q/A$ where ρ denotes the mass density of the fluid, p the pressure and q the volume flow in the pipe. All plugged in yields

$$\rho Al \frac{d}{dt}\frac{q}{A} = A(p_1 - p_2) \tag{76}$$

which gives the differential equation for an inductive element

$$p = \frac{\rho l}{A}\frac{dq}{dt}. \tag{77}$$

Therefor, the inertia coefficient L, e.g for a pipe segment, is evaluated using the formula

$$L = \frac{\rho l}{A}. \tag{78}$$

Hydraulic flow capacitance In order to derive a capacitance coefficient for a fluid, consider the definition of the bulk modulus E of a substance as given in [12]

$$E = \frac{dp}{dv/v}. \tag{79}$$

```
{ bond graph model for capacitive field }

model type CFl

cut chem (mu/nu)
cut hyd (pvap/q)
cut therm(T/Sd)

parameter Cth = 75.906    { J/mol K }
parameter Chy = 1.2E-11   { m**5/N }
constant  rho = 1.8E-5    { m**3/mol }

terminal V,pd

constant Tc=373.15,pc=101330.,rr=4883.
local pd,Td,S,mud,n

{thermal capacitance : get Td}
Td = T*Sd/(n*Cth)
der(T) = Td

{terminal from CFl : get pd}
pd = pd

{Gibbs Duhem equation : get mud}
0 = -pd*V + Td*S + mud*n
der(mu) = mud

{vapor pressure curve : get pvap as theoretical pressure to
 determine equilibrium}
pvap = pc * EXP( rr*(T-Tc)/(T*Tc) )

{accumulation of through variables}
der(S) = Sd
der(n) = nu
V=rho*n        {might be der(V) = q as well}

end
```

Figure 46: DYMOLA code for the capacitive field CFl for the liquid node

This is transformed easily into the form $q = C\frac{dp}{dt}$ for a constant specific volume v of the fluid, yielding

$$q = \frac{V}{E}\frac{dp}{dt} \tag{80}$$

that gives the explicit formula to compute the capacitance coefficient as

$$C = \frac{V}{E}. \tag{81}$$

The bulkmoduli of various substances are tabulated, e.g in [13]. For ideal gases, however, they are a function of pressure only, e.g. $E = \kappa p$ for an adiabatic gas, as stated in [12].

```
{ bond graph model for capacitive field }

model type CFg

cut chem (mu/nu)
cut hyd (p/q)
cut therm(T/Sd)

parameter   Cth   = 36.306  { J/mol K }
constant    R     = 8.314 { J/mol K }
constant    kappa = 1.4    { dimensionless }

terminal V,pd

local pd,Td,S,mud,n,Chy

{thermal capacitance : get Td}
Td = T*Sd/(n*Cth)
der(T) = Td

{hydraulic capacitance : get pd}
Chy = V/(kappa*p)
pd = q/Chy

{Gibbs Duhem equation : get mud}
0 = -pd*V + Td*S + mud*n
der(mu) = mud

{equation of state : get p}
p*V=n*R*T

{related integrators}
der(S) = Sd
der(n) = nu

end
```

Figure 47: DYMOLA code for the capacitive field CFg for the vapor node

Hydraulic flow resistance For the steady motion of incompressible viscous fluids, it is customary to express the pressure loss in a pipe as

$$\Delta p = \frac{f l \rho}{2D} v^2 \tag{82}$$

what means that the pressure loss is proportional to the velocity squared in the pipe. This is transformed into

$$\Delta p = \frac{f l \rho}{2DA^2} q^2 \tag{83}$$

in order to match the needs of the bond graph to express the resistance in terms of pressure p and volume flow q. The resistance coefficient used in the models therefore can be computed using the formula

$$R = \frac{f l \rho}{2DA^2}. \tag{84}$$

Unfortunately, these equations are subject to large uncertainties. Especially the friction factor f is a very complex function of the Reynolds Number of the flow and of the roughness

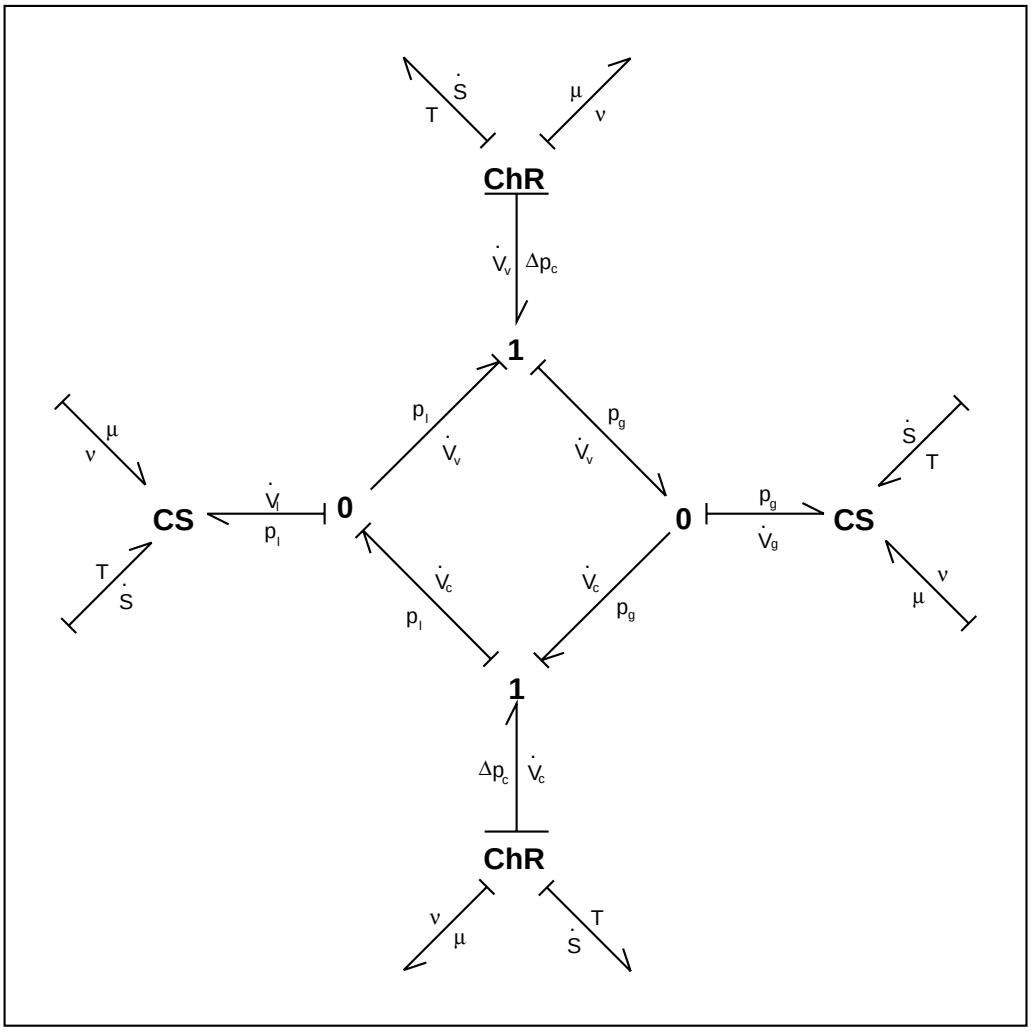

Figure 48: Hydraulic reaction bond graph for water/steam system

of the pipe, that is rather difficult to obtain. Rather than determining the pipe conductance by calculating the friction factor of a pipe, etc., equation (83) is transposed to solve for the resistance off-line using steady state operating point data. Since the form of the equation is valid over a wide operating range, the resistance R needs only to be calculated at one operating point. Using this value in a simulation should yield an accurate prediction over a wide range, as stated in the MMS User's Manual [5].

Thermal capacitance and conductance For the definition of a thermal capacitance of a piece of matter, a caloric equation of state is used, namely

$$\dot{S} = \frac{cm}{T}\frac{dT}{dt} \tag{85}$$

where the product of mass m and specific heat c denote the thermal capacitance C that is used in the bond graphs. The dissipative heat conduction process in a piece of matter can be described by the relation

$$\Delta T = \theta T \dot{S} \tag{86}$$

where θ denotes the thermal resistance of the body. This resistance is computed from the tabulated values for the thermal conductivity k using the formula

$$\theta = \frac{l}{kA} \tag{87}$$

with the length l and the area A of the body under investigation.

Convective source In order to model the convective entropy flow correctly, a temperature modulated entropy flow source was proposed. The convective entropy flow is determined from the product of specific entropy s and mass flow $\dot{m}$ of the fluid. Since the model is not capable to calculate a local density, the mass flow has to be computed from the product of a constant average density ρ and the local volume flow $\dot{V}$. The specific entropy of the fluid is a rather complex function of pressure and temperature. However, in case of neglectable pressure drop, like in a steam pipe, or in case of saturated liquids, like for the valve, the entropy might be approximated by a function of temperature only:

$$s = s_o + \frac{\Delta s}{\Delta T}(T - T_o) \tag{88}$$

where the coefficients s_o, Δs and ΔT are determined from the steady state off-line operating point data of the pipe, like this was done for the flow resistance in the pipe.

6.5.2 Parameters for submodel PIPE

diameter	$D = 0.8$ m	number of segments	4
cross section area	$A = 0.5$ m^2	volume of each segment	$V = 1.25$ m^3
length of pipe	$l = 10$ m	mass in each segment	$m = 4.25$ kg
fluid in pipe	superheated vapor	specific heat	$c_p = 2.16 \cdot 10^3$ J/kg K
		thermal conductivity	$\bar{k} = 86.15$ W/m K

Table 7: PIPE dimensions

The bond graph for a segment of a pipe is depicted in Fig.30. In determining the parameters for the model, another problem of the incompressible bond graph can be noticed.

Inlet boundary values	Outlet boundary values
$T = 619.3$ K	$T = 619.0$ K
$p = 0.98$ MPa	$p = 0.95$ MPa
$\rho = 3.41$ kg/m^3	$\rho = 3.38$ kg/m^3
$h = 3.148 \cdot 10^3$ J/kg	$h = 3.148 \cdot 10^3$ J/kg
$s = 7.2958 \cdot 10^3$ J/kg K	$s = 7.3131 \cdot 10^3$ J/kg K
$\dot{m} = 107.5$ kg/s	$\dot{m} = 107.5$ kg/s
$q = 31.5$ m^3/s	$q = 31.8$ m^3/s

Table 8: PIPE boundary values (steady state)

integrator initial conditions	parameter values
$T = 619$ K	$C_{th} = 9.2 \cdot 10^3$ J/K
$p = 0.98$ MPa	$C_{hy} = 8.93 \cdot 10^{-7}$ m^5/N
$q = 31.7$ m^3/s	$L = 17$ kg/m^4
—	$R = 7.6$ N s^2/m^8
—	$b = \frac{1}{\theta} = 17.2$ W/K

Table 9: PIPE integrator initial conditions and parameter values

Most of the parameters used in the model are not strictly constant, all of them depend in one way or the other from some changing conditions inside the model, e.g. the inertance of a fluid flowing in a pipe depends not only on the constant pipe dimensions, but also on the density of the fluid under consideration, that is object to changes if any type of compressible fluid is considered. However, for all varying parameters like density, bulk modulus, thermal conductivity or heat capacity, average values may be used as parameters, because the model is taken to be clearly incompressible and therefore local densities cannot be calculated with

$s_o = 7.2958 \cdot 10^3$ J/kg K	$T_o = 619$ K
$\Delta s = 170$ J/kg K	$\Delta T = 50$ K

Table 10: PIPE SFS interpolation coefficients

cross section area	$A = 0.1$ m^2	volume in valve	$V = 0.025$ m^3
fluid in valve	water	specific heat	$c_p = 4.18 \cdot 10^3$ J/kg K
thermal conductivity	$k = 640$ W/K m	bulk modulus	$E = 2100$ N/mm^2

Table 11: VALVE dimensions

the given model.

The boundary conditions are chosen such that they match the boundary conditions chosen in the MMS reference model. Some assumptions have to be made about the physical dimensions of the pipe, because the MMS User's Manual [5] operates on operating point data for the example exclusively, so this information is missing for the given model. For the new model we assume the values summarized in table 7. The complete boundary conditions for the model including all important thermodynamic fluid properties are summarized in table 8. Please note that the given values denote the off-line steady state boundary values of the MMS reference model and that not all of these are input to the model directly. Some of the boundary values are calculated by the model, e.g it is sufficient to supply the pressure at the inlet of the pipe, whereas the outlet pressure is calculated by the model. The initial conditions for the integrators are chosen such that they are close to the expected steady state operating conditions of the pipe. The initial conditions and the parameters for the integrators are summarized in table 9. The interpolation coefficients for the SFS element, that are taken from a steam table [14] are summarized in table 10.

6.5.3 Parameters for submodel VALVE

The bond graph for the submodel VALVE is depicted in Fig. 35. The boundary conditions of the MMS reference model needed for the parameter calculations are summarized in table 12. The parameter evaluation is straight forward, and very similar to the calculations necessary for the submodel PIPE, since the models are almost identical. Due to the fact that liquid water is an almost incompressible fluid, the chosen parameters summarized in table 13 are no average values as for the pipe model. Note that the flow resistance element mRS is modeled with its conductance rather than its resistance as parameter, since DYMOLA is not able to handel the related quadratic equation correctly. As in the pipe model, some assumptions have to be made about the physical dimensions of the valve that are summarized in table 11. The parameters for the SFS element again are taken from a steam table [14] and summarized in table 14.

Inlet boundary values	Outlet boundary values
$T = 319.2$ K	$T = 319.2$ K
$p = 2.48$ MPa	$p = 0.95$ MPa
$\rho = 999.1$ kg/m^3	$\rho = 999.1$ kg/m^3
$h = 194 \cdot 10^3$ J/kg	$h = 194 \cdot 10^3$ J/kg
$s = 760 \cdot 10^3$ J/kg K	$s = 760 \cdot 10^3$ J/kg K
$\dot{m} = 459.5$ kg/s	$\dot{m} = 459.5$ kg/s
$q = 0.46$ m^3/s	$q = 0.46$ m^3/s
$y = 0.64$	

Table 12: VALVE boundary values (steady state)

integrator initial conditions	parameter values
$T = 319$ K	$C_{th} = 104 \cdot 10^3$ J/K
$p = 2.48$ MPa	$C_{hy} = 1.2 \cdot 10^{-11}$ m^5/N
—	$b = \frac{1}{\theta} = 640$ W/K
—	$R_o = 5.27 \cdot 10^{-7}$ m^8/N s^2

Table 13: VALVE integrator initial conditions and parameter values

6.5.4 Parameters for submodel PUMP

The bond graph for the submodel PUMP is depicted in Fig.40. The boundary conditions for the model are summarized in table 16 as well as the assumptions for the pump dimensions are summarized in table 15. For the parameters of the SFS element and the hydraulic and thermal capacitances of the pump, the same values as in the valve model are chosen and summarized in table 18 and table 17, respectively.

$s_o = 760$ J/kg K	$T_o = 319$ K
$\Delta s = 259$ J/kg K	$\Delta T = 25$ K

Table 14: VALVE SFS interpolation coefficients

cross section area	$A = 0.1$ m^2	volume in pump	$V = 0.5$ m^3
fluid in pump	water	specific heat	$c_p = 4.18 \cdot 10^3$ J/kg K

Table 15: PUMP dimensions

Inlet boundary values	Outlet boundary values
$T = 318.4$ K	$T = 319.2$ K
$p = 9.79 \cdot 10^{-3}$ MPa	$p = 2.48$ MPa
$\rho = 999.1$ kg/m^3	$\rho = 999.1$ kg/m^3
$h = 190 \cdot 10^3$ J/kg	$h = 194 \cdot 10^3$ J/kg
$s = 760 \cdot 10^3$ J/kg K	$s = 760 \cdot 10^3$ J/kg K
$\dot{m} = 459.5$ kg/s	$\dot{m} = 459.5$ kg/s
$q = 0.46$ m^3/s	$q = 0.46$ m^3/s

Table 16: PUMP boundary values

In addition to these, parameter values for the transformation ratio m of the transformer and the mechanical resistance (friction in bearings, etc.) have to be chosen. For the mechanical friction R_{mech} in the pump bearings, it is assumed that it is not exceeding 1/1000 of the input torque. The hydraulic conductance R_{hyd} and the transformation ratio m are the two parameters that determine the pump characteristic given in a so called *head curve*[6] for each pump type explicitly. For the MMS-macro, a head curve of a special pump can be supplied

[6]The head curve is meant to be the head developed by the pump as a function of volume flow q. For further information, see [12].

integrator initial conditions	parameter values
$T = 318.4$ K	$C_{th} = 104 \cdot 10^3$ J/K
$p = 2.48$ MPa	$C_{hy} = 1.2 \cdot 10^{-11}$ m^5/N
—	$m = 1.7 \cdot 10^{-4}$
—	$R_{mech} = 0.001$
—	$R_{hyd} = 5.27 \cdot 10^{-7}$ m^8/N s^2

Table 17: PUMP integrator initial conditions and parameter values

$s_o = 760$ J/kg K	$T_o = 319$ K
$\Delta s = 259$ J/kg K	$\Delta T = 25$ K

Table 18: PUMP SFS interpolation coefficients

as input data. It is not clear to the author how the head curve can be matched in any way by the new model. The two parameters were chosen such, that the developed heads of the pump models are matched at the given flow of $q = 0.46$ m^3/s. However, the proper parameterization of this model is open to question. A simple trial shows that the performance characteristic of the reference pump, given by a head curve directly, is not matched at all. Obviously, the transformation ratio m is not a constant over the whole range of operation of the pump. In order to obtain better results, it should be modulated in some fashion with the flow through the pump.

6.5.5 Parameters for the submodel DEAER

For the deaerator, a completely new model was built which is shown in Fig.44, 48 and 49. To test this model, a closed system is considered first. The system is chosen as a closed container with a total volume of 2 liters and a cross sectional area of 10 cm^2. In the beginning of the experiment, there is one liter of liquid water at a temperature of 375 K and one liter of vaporized water at a temperature of 370 K. The pressure inside the container is 0.9 bar which is the vapor pressure of water for 370 K. This means that a vaporization reaction should take place until the pressure in the container has risen to the vapor pressure of water at a temperature slightly below 375 K, because the liquid water will cool down just a little bit during the reaction. The initial fluid properties as well as the heat capacitances needed for model parameterization of the two fluids in the container are summarized in table 19. The hydraulic capacitance needed for the incompressible liquid can be found in table 17 or 13.

Because a closed system is considered, no boundary values have to be listed. All flows in and out of the system simply have to be equal to zero.

liquid phase	gaseous phase
$T = 375$ K	$T = 370$ K
$S = 1.271 \cdot 10^3$ J/K	$S = 4.699 \cdot 10^3$ J/K
$p_{vap} = 1.082$ MPa	$p_{vap} = 0.9047$ MPa
$V = 0.001$ m^3	$V = 0.001$ m^3
$\mu = -1583.8$ J/mol	$\mu = -1019$ J/mol
$n = 53.2$ mol	$n = 0.035346$ mol
$s = 1.3288 \cdot 10^3$ J/kg K	$s = 7.392 \cdot 10^3$ J/kg K
$v = 1.045 \cdot 10^{-3}$ m^3/kg	$v = 1.573$ m^3/kg
$h = 406.3 \cdot 10^3$ J/kg	$h = 2678.5 \cdot 10^3$ J/kg
$c_p = 4.220$ kJ/kg K	$c_p = 2.017$ kJ/kg K
$k = 681 \cdot 10^3$ W/m K	$k = 681 \cdot 10^3$ W/m K

Table 19: Fluid initial properties for the deaerator test

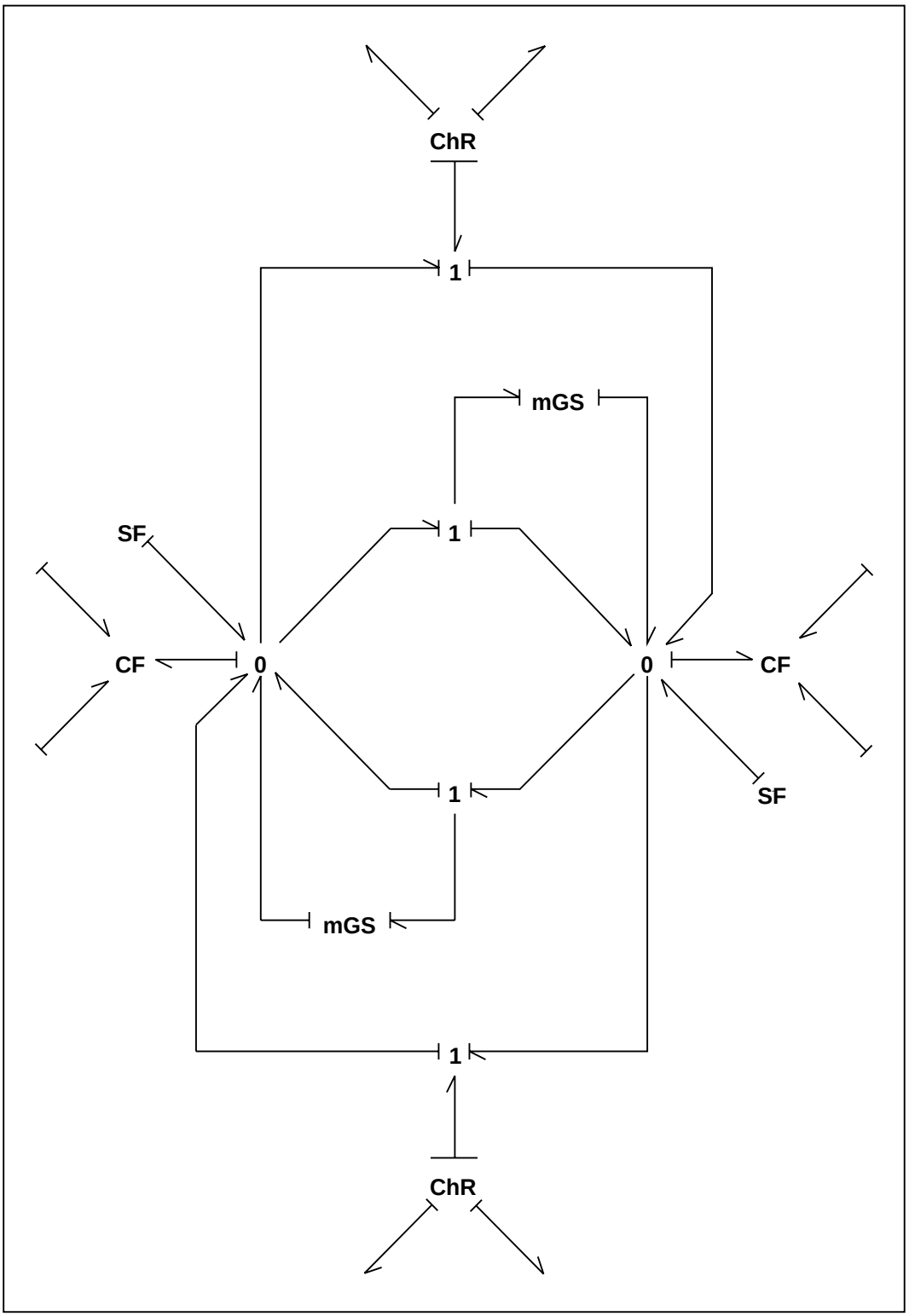

Figure 49: Thermal reaction bond graph for water/steam system

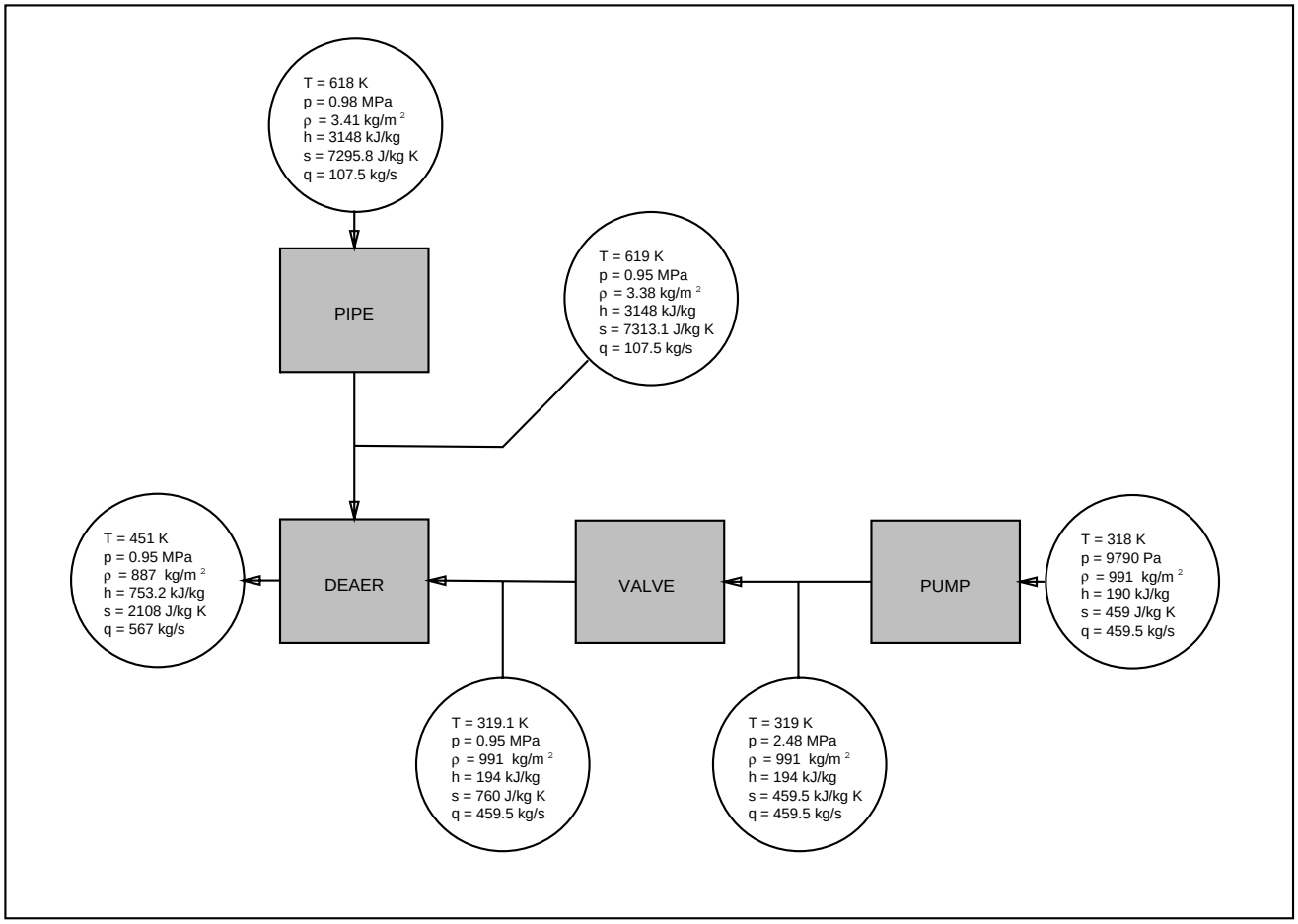

Figure 50: Boundary values of the steady state of the deaerator model

7 Performance of the new models

The goal of this thesis was not only to come up with new models, but also to verify the newly derived models by comparing the simulation results of the old and the new model. Because the kernel of the deaerator level study model formulated as MMS macros, the deaerator model itself, could not be successfully reimplemented, only single model tests were performed for the new bond graph models. Yet another problem in comparing the old and new model performance is that the old models do not exhibit any dynamic performance for a single component. Therefore the dynamic performance, e.g. obtained for the model pipe, could not be compared with the old MMS model.

The DYMOLA code used to test the single reimplemented submodels is listed in the appendix. The boundary values and parameters derived in the previous chapter are inserted in the models and a simulation run is performed as long as equilibrium is reached. With all models, except the deaerator model, the steady state values obtained from the original MMS deaerator level study simulation program could be matched.

7.1 The submodel PIPE

Consider a pipe containing a fluid at a homogeneous temperature and pressure that is initially not moving. At time $t = 0$ s the flow at the outlet is set to a specific value greater than zero. The dynamic response of the incompressible fluid is simulated with the new pipe model. The results are shown in Fig. 51. The pipe was modeled with four segments, therefore the state variables p, $\dot{V}$ and T can be determined at five positions (inlet, outlet and three internal segment boundaries), referred to as nodes. Node number one is the inlet side of the pipe, while node number five represents the outlet side of the pipe.

Note the decay of the pressure, flow and temperature waves according to the flow resistance in the pipe until steady state is reached. The steady state performance of the old model is matched exactly for the hydraulic model part as far as volume flow and pressure profile are concerned. The outlet values for temperature and density, which are determined from state functions in the MMS model, cannot be matched with the new model, because the density is a constant in the new model and therefore a density change as well as a temperature change due to the fluid expansion cannot be determined using an incompressible model.

Some other problems in the new model should be mentioned here as well. Consider the high amplitudes of the temperature waves obtained from the simulation run, which don't

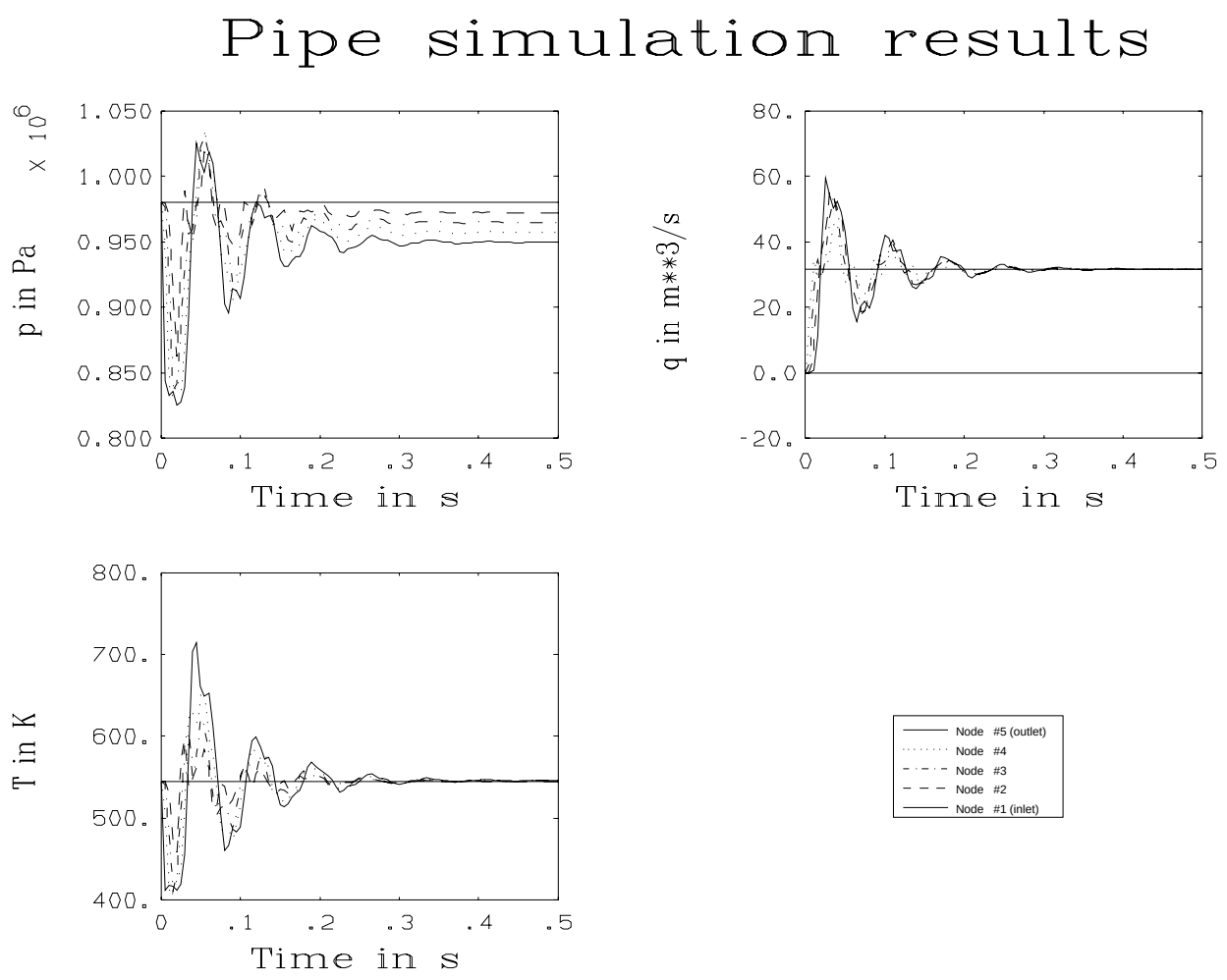

Figure 51: PIPE simulation results

look very realistic at all. These waves are due to the fact, that the entropy flow in the model is mainly dependent from the mass or volume flow, respectively. Therefore it happens, that a large entropy flow due to mass transfer is taken out of or into the capacitances determining the temperature of the node under consideration, causing a large temperature drop or rise. These temperature changes computed by the model are not correct at all because if entropy connected to mass flows into or out of an open system the temperature doesn't change necessarily.

This shows that the simple thermal capacitance elements derived for the heat diffusion in [7] are not able to model the incompressible fluid flow process correctly.

7.2 The submodels VALVE and PUMP

The valve and pump models were not modeled as distributed system like the pipe, therefore they don't exhibit dynamic behaviour like the pipe. The same type of experiment as performed with the pipe model was performed with the valve and pump models, namely initially stationary fluid is accelerated by applying a volume flow from the outlet side of the component. The model then determines the pressure drop over the component as well as the volume flows at the inlet side of the component. The simulation results obtained upon execution of the valve test model printed in the appendix are shown in Fig.52. The pump model exhibits basically the same nondynamic behaviour as the valve model, and the steady state boundary values of the MMS model used as reference are matched for both new models. The pump model test code is also printed in the appendix.

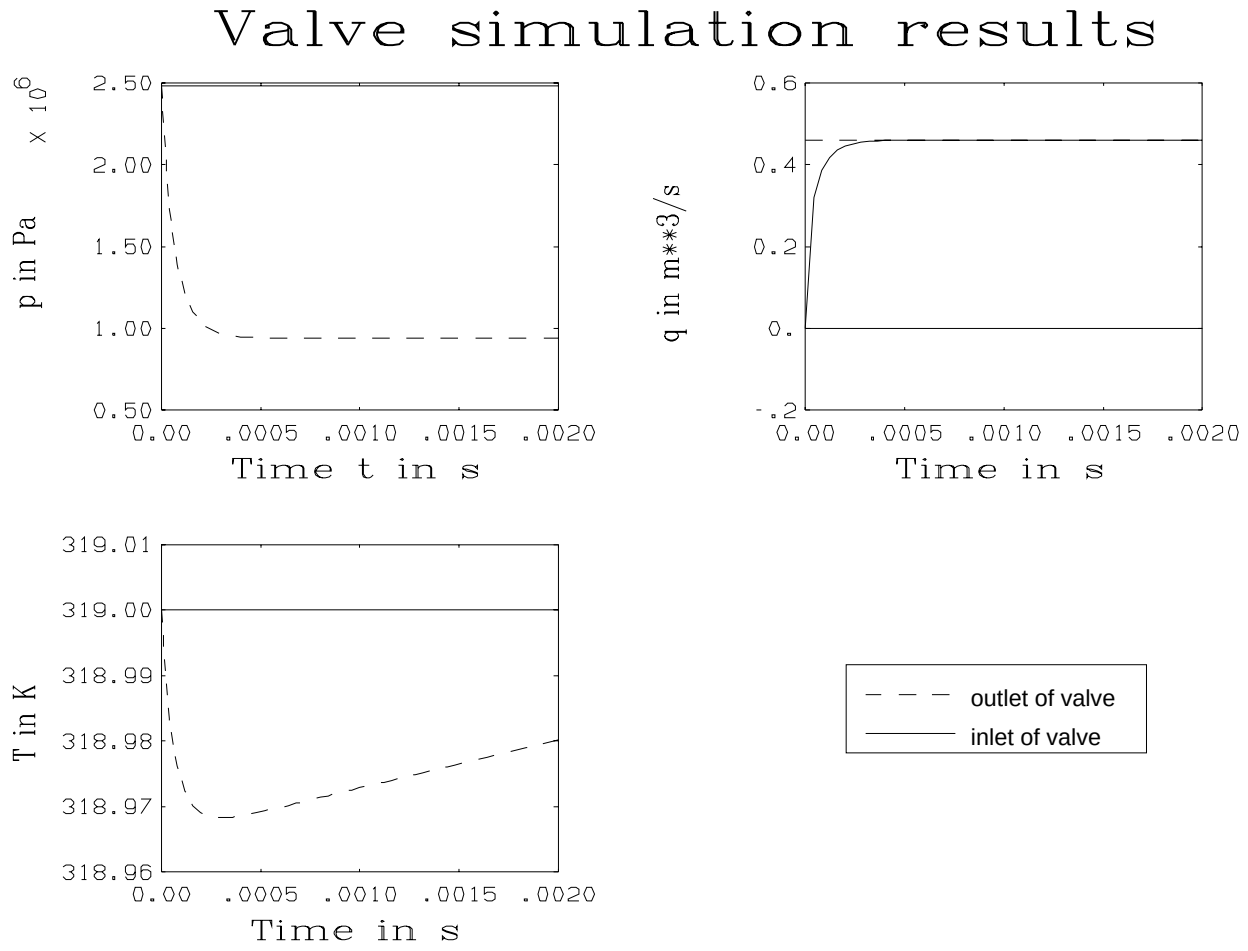

Figure 52: VALVE simulation results

7.3 The submodel DEAER

The code for a deaerator test model is printed in the appendix together with the previously discussed test models. However, this model never was executed successfully. While there are some problems as mentioned for the incompressible models, the deaerator model which has to be incompressible includes so many problems, that a stable model could not be obtained from the bond graph derived in chapter 6.

8 Conclusion

The Modular Modeling System (MMS) is a simulation code for modeling nuclear and fossil–fueled power plant dynamics developed by the Electric Power Research Institute (EPRI), Palo Alto, California. It is intended for use during both plant design and plant operation. The MMS code permits simulation of a wide range of transients that may occur with a specific plant configuration. A model of a power plant of arbitrary configuration may be assembled from preprogrammed generic MMS component models, called *modules* in MMS. Each significant physical component of the plant is represented by a module or modules. The user defines their interconnections and provides sufficient data to describe the physical components, that way turning the preprogrammed generic modules into actual models of power plant components. The available module library on which the reimplementation was based is the ACSL (Advance Continuous Simulation Language) based version of MMS (V1.0). The code was properly verified and validated before it was released in 1983.

The purpose of the reimplementation was to make use of the advantages which are offered by DYMOLA. DYMOLA was developed by Hilding Elmquist, Lund, Sweden. It is a modeling language that supports the user in coding more readable and better modularized hierarchical model descriptions than ACSL. However, DYMOLA does not provide for the numerical solution of the created model itself but generates code for the simulation languages DESIRE, SIMNON and ACSL and also plain FORTRAN. That means that DYMOLA can be regarded as a powerful *macro handler* for several simulation languages. DYMOLA is an object oriented language and DYMOLA models are much more modular than ACSL macros since equations are automatically solved during model expansion for the variable appropriate in the context of the model call environment. For this reason, DYMOLA models are better reusable and of higher quality than others, which provides for powerful model libraries. The general modeling capabilities of DYMOLA are widespread, they cover modeling of electrical, mechanical, thermo-dynamical, chemical and other systems as well as combinations of them.

The first step for the reimplementation of the MMS macro library was a translation of a part of MMS macros to reproduce the results of a given MMS example. All basic concepts of parameterization and connection of the MMS macros as well as the equations describing the plant components remained the same as described in the MMS Users Manual for this first step. Some minor inconveniences were noticed in rewriting the models. These are due to the fact that DYMOLA is a relatively new tool and not as advanced as ACSL, which is a state of the art software product.

In a second step the principles of bond graph modeling were used to build a new set of submodels for the same model example mentioned above. The bond graph modeling technique offers a much more straight forward way of deriving models than a simple accumulation of modeling equations. This is due to the fact, that bond graph modeling mainly operates on a set of well defined standard elements like inertances, capacitances and resistors that basically remain the same for various different types of systems like thermal, hydraulic or mechanical systems.

This resulted in a completely different way of modeling the plant components, because a different set of physical variables is used to model a system using bond graphs. For example, in common systems the temperature T, entalpy H and mass flowrate $\frac{dm}{dt}$ are used to model thermodynamic processes, while for bond graph modeling temperature T, entropy flow $\frac{dS}{dt}$, chemical potential μ and molar mass flow $\dot{n}$ are used.

While the models dealing with incompressible fluid motion could be properly translated and executed, the models dealing with compressible fluid dynamics and steam/water equilibrium are too complicated to put them into a bond graph representation yet.

While the fluid dynamics for incompressible substances are addressed properly by the bond graph, the thermal part of the incompressible fluid motion model doesn't work correctly yet. Major problems have to be faced in order to express the energy storage in an opened system in terms of the bond graph. This problem occurs with all models described in this thesis and has to be solved first. An attempt to overcome the storage problem was made in defining a capacitive field for the deaerator model, however, the problem could not be solved yet.

A MMS deaerator example macros and program

```
''                                                                    26610000
''                                                                    26620000
'          DESCRIPTION OF MODULE PICONT     '                        26630000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                      26640000
'     THIS MODULE REPRESENTS A PROPORTIONAL PLUS INTEGRAL     '       26650000
'     CONTROLLER WITH ANTI-RESET-WINDUP.     '                      26660000
'          INTERCONNECTIONS     '                                    26670000
'     CSP : CONTROLLER SET POINT VARIABLE     '                     26680000
'     PVAR : FEEDBACK VARIABLE     '                                26690000
'     OUTVAR : OUTPUT VARIABLE     '                                26700000
'          TRANSLATION PARAMETERS     '                             26710000
'     ACTION : PARAMETER SPECIFYING DIRECT OR REVERSE ACTION     '  26720000
'               (1:DIRECT;2:REVERSE)     '                          26730000
'          NOMENCLATURE     '                                       26740000
'     INTERNAL VARIABLES     '                                      26750000
'     ZOI : INTEGRATOR OUTPUT (FRACTION)     '                      26760000
'     ZOP : PROPORTIONAL OUTPUT (FRACTION)     '                    26770000
'     ZDC : INTEGRATOR DERIVATIVE (FRACTION/SEC)     '              26780000
'     ZOT : UNBOUNDED OUTPUT     '                                  26790000
'     ZER : CONTROL ERROR SIGNAL     '                             26800000
'     PARAMETERS     '                                              26810000
'     CIG : INTEGRATOR GAIN     '                                   26820000
'     CPG : PROPORTIONAL GAIN     '                                 26830000
'     KLL : LOW LIMIT OF CONTROLLER OUTPUT     '                    26840000
'     KHL : HIGH LIMIT OF CONTROLLER OUTPUT     '                   26850000
'          INITIAL CONDITIONS     '                                 26860000
'     ZIC : CONTROLLER OUTPUT     '                                 26870000
MACRO PICONT(ID,CSP,PVAR,OUTVAR,ACTION)                               26880000
     MACRO RELABEL L100,L200,L300                                     26890000
     LOGICAL ZLM_ID                                                   26900000
     MACRO IF(ACTION=2) L1                                            26910000
     ZER_ID=CSP-PVAR                                                  26920000
     MACRO GO TO L2                                                   26930000
     MACRO L1..CONTINUE                                               26940000
     ZER_ID=PVAR-CSP                                                  26950000
     MACRO L2..CONTINUE                                               26960000
     OUTVAR=BOUND(KLL_ID,KHL_ID,ZOT_ID)                               26970000
     ZOP_ID=CPG_ID*ZER_ID                                             26980000
     ZOT_ID=ZOI_ID+ZOP_ID                                             26990000
     PROCEDURAL(ZOC_ID=ZOT_ID,ZER_ID)                                 27000000
     CONSTANT ZLM_ID=.FALSE.                                          27010000
     IF(ZZFRFL) ZLM_ID= ...                                           27020000
       (ZOT_ID.GE.KHL_ID.AND.ZER_ID.GE.0.) .OR. ...                   27030000
       (ZOT_ID.LE.KLL_ID.AND.ZER_ID.LE.0.)                            27040000
     ZOC_ID=ZER_ID                                                    27050000
     IF(ZLM_ID) ZOC_ID=BOUND(0.,0.,ZER_ID)                            27060000
     END                                                              27070000
     ZDC_ID=CIG_ID*ZOC_ID                                             27080000
     ZOI_ID=INTVC(ZDC_ID,ZIC_ID)                                      27090000
MACRO END                                                             27100000
''                                                                    18500000
''                                                                    18510000
'          DESCRIPTION OF MODULE PUMPMD      '                       18520000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                      18530000
'     THIS MODULE REPRESENTS ANY NUMBER OF IDENTICAL CONSTANT    '    18540000
'     SPEED CENTRIFUGAL PUMPS IN PARALLEL.     '                    18550000
'     INTERCONNECTIONS     '                                        18560000
'     WE : FLOW ENTERING     '                                      18570000
'     WL : FLOW LEAVING     '                                       18580000
'          INTERCONNECTING VARIABLES     '                          18590000
'     H : ENTHALPY(BTU/LBM)     '                                   18600000
'     P : PRESSURE(PSIA)     '                                      18610000
'     R : DENSITY(LBM/FT3)     '                                    18620000
'     W : FLOWRATE(LBM/HR)     '                                    18630000
'          INTERNAL VARIABLES     '                                 18640000
'     ZDH : PUMP DIFFERNTIAL HEAD(FT OF H2O)     '                  18650000
'     ZQW : VOLUME FLOWRATE(GPM)      '                             18660000
'     ZWW : FLOWRATE(LBM/HR)     '                                  18670000
```

```
'     ZEP : PUMP EFFICIENCY(FRACTION)     '                              18680000
'          PARAMETERS     '                                             18690000
'     KCK : LOGICAL CHECK VALVE SWITCH     '                           18700000
'     KEP : MAXIMUM PUMP EFFICIENCY(FRACTION)     '                    18710000
'     KNP : NUMBER OF OPERATING PUMPS     '                            18720000
'     KPR : KEP/KQR**2     '                                           18730000
'     KQR : VOLUME FLOWRATE AT KEP(GPM)     '                         18740000
'     KHC(TABLE) : PUMP HEAD CURVE(GPM VS. FT H2O)     '              18750000
MACRO PUMPMD(ID,WE,WL)                                                 18760000
     MACRO RELABEL L100,L200,L300                                      18770000
     LOGICAL OMASK,OWW_ID,KCK_ID                                       18780000
     PROCEDURAL(ZQW_ID=ZDH_ID)                                         18790000
     ZQW_ID=KHC_ID(ZDH_ID)                                             18800000
     IF(ZZFRFL.AND.KCK_ID.AND.(ZQW_ID.LE.0.)) ZQW_ID=0.                18810000
     END                                                               18820000
     ZDH_ID=144.*(P_WL-P_WE)/R_WE                                      18830000
     W_WE=8.02*KNP_ID*R_WE*ZQW_ID                                      18840000
     ZEP_ID=KEP_ID-KPR_ID*(ZQW_ID-KQR_ID)**2                           18850000
     H_WL=H_WE+.0013*ZDH_ID/ZEP_ID                                     18860000
     W_WL=W_WE                                                         18870000
     R_WL=RLOFPH(P_WL,H_WL)                                            18880000
     T_WL=TLOFPH(P_WL,H_WL)                                            18890000
'     VALIDITY CHECKS     '                                            18900000
     PROCEDURAL(OWW_ID=W_WE)                                           18910000
     CONSTANT OWW_ID=.FALSE.                                           18920000
     IF(OMASK.OR.(.NOT.ZZFRFL)) GO TO L300                             18930000
     IF(OWW_ID) GO TO L200                                             18940000
     IF(W_WE.LT.0.) OWW_ID=.TRUE.                                      18950000
     IF(OWW_ID) PRINT L100                                             18960000
  L100..FORMAT(//,...                                                  18970000
     ' ***EXECUTION TERMINATED; REVERSE FLOW IN PUMP _ID***',//)       18980000
  L200..CONTINUE                                                       18990000
     TERMT(OWW_ID)                                                     19000000
  L300..CONTINUE                                                       19010000
     END                                                               19020000
MACRO END                                                              19030000
''                                                                     72520000
''                                                                     72530000
'          DESCRIPTION OF MODULE CONNI     '                           72540000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                       72550000
'     THIS MODULE ALLOWS CONNECTION OF TWO RESISTIVE TYPE MODULES     ' 72560000
'     OPERATING WITH AN INCOMPRESSIBLE FLOW.  THE MODULE PROVIDES A '   72570000
'     NON-PHYSICAL REPRESENTATION VARYING THE MODULE PRESSURE,       '  72580000
'     PWL, TO EQUALIZE THE ENTERING AND LEAVING FLOWS.     '            72590000
'          INTERCONNECTIONS     '                                       72600000
'     WE : WATER ENTERING     '                                        72610000
'     WL : WATER LEAVING     '                                         72620000
'          NOMENCLATURE     '                                         72630000
'     INTERCONNECTING VARIABLES     '                                  72640000
'     H : ENTHALPY (BTU/LBM)     '                                     72650000
'     P : PRESSURE (PSIA)     '                                        72660000
'     R : DENSITY (LBM/FT3)     '                                      72670000
'     T : TEMPERATURE (F)     '                                        72680000
'     W : FLOW RATE (LBM/HR)     '                                     72690000
MACRO CONNI(ID,WE,WL)                                                  72700000
     CONSTANT KDP_ID=1000.                                             72710000
     DP_WL=(W_WE-W_WL)/KDP_ID                                          72720000
     P_WL=INTVC(DP_WL,IP_WL)                                           72730000
     P_WE=P_WL                                                         72740000
     H_WL=H_WE                                                         72750000
     T_WL=TLOFPH(P_WL,H_WL)                                            72760000
     R_WL=RLOFPH(P_WL,H_WL)                                            72770000
MACRO END                                                              72780000
''                                                                     25260000
''                                                                     25270000
'          DESCRIPTION OF MODULE VALVEI     '                         25280000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                       25290000
'     THIS MODULE REPRESENTS A VALVE AND A PIPE IN SERIES     '        25300000
'     CARRYING AN INCOMPRESSIBLE FLUID.     '                        25310000
'          INTERCONNECTIONS     '                                     25320000
'     WE : FLOW ENTERING     '                                        25330000
```

```
'     WL : FLOW LEAVING     '                                          25340000
'     Y : VALVE POSITION(FRACTION)     '                               25350000
'          INTERCONNECTING VARIABLES      '                           25360000
'     H : ENTHALPY(BTU/LBM)     '                                      25370000
'     P : PRESSURE(PSIA)     '                                         25380000
'     R : DENSITY(LBM/FT3)     '                                       25390000
'     W : FLOWRATE(LBM/HR)     '                                       25400000
'          INTERNAL VARIABLES     '                                   25410000
'     ZCV : VALVE CONDUCTANCE     '                                   25420000
'     ZCQ : COMBINED PIPE AND VALVE CONDUCTANCE    '                  25430000
'     ZDP : DIFFERENTIAL PRESSURE (PSI)     '                         25440000
'          PARAMETERS     '                                           25450000
'     KCK : LOGICAL CHECK VALVE SWITCH     '                          25460000
'     KCP : PIPE CONDUCTANCE      '                                   25470000
'     KCV : VALVE MAXIMUM CONDUCTANCE     '                          25480000
'     KVA : VALVE CHARACTERISTIC PARAMETER     '                      25490000
'          (1:EQUAL PERCENT;2:LINEAR;3:QUICK OPEN)     '             25500000
'          VALIDITY FLAGS     '                                       25510000
'     OWW : REVERSE PRESSURE (TERMINATE)     '                       25520000
MACRO VALVEI(ID,WE,WL)                                                25530000
     MACRO RELABEL L100,L200,L300                                     25540000
     LOGICAL OMASK,OWW_ID,KCK_ID                                      25550000
     INTEGER KVA_ID                                                   25560000
     PROCEDURAL(ZCV_ID=Y_ID,KCV_ID,KVA_ID)                            25570000
     IF(KVA_ID.EQ.1) ZCV_ID=ABS(Y_ID)**3*KCV_ID                       25580000
     IF(KVA_ID.EQ.2) ZCV_ID=Y_ID*KCV_ID                               25590000
     IF(KVA_ID.EQ.3) ZCV_ID=(1.-EXP(-10.*Y_ID))*KCV_ID                25600000
     END                                                              25610000
     ZCQ_ID=KCP_ID*ZCV_ID/SQRT(ABS(KCP_ID*KCP_ID+ZCV_ID*ZCV_ID)+1.E-3) 25620000
     ZDP_ID=P_WE-P_WL+R_WE*KDH_ID/144.                                25630000
     PROCEDURAL(W_WL=ZDP_ID,ZCQ_ID,R_WE)                              25640000
     W_WL=ZCQ_ID*SIGN(SQRT(ABS(R_WE*ZDP_ID)),ZDP_ID)                  25650000
     IF(ZZFRFL.AND.KCK_ID.AND.(ZDP_ID.LE.0.)) W_WL=0.                 25660000
     END                                                              25670000
     W_WE=W_WL                                                        25680000
     H_WL=H_WE                                                        25690000
     R_WL=RLOFPH(P_WL,H_WL)                                           25700000
     T_WL=TLOFPH(P_WL,H_WL)                                           25710000
'     VALIDITY CHECKS     '                                          25720000
     PROCEDURAL(OWW_ID=W_WE)                                          25730000
     CONSTANT OWW_ID=.FALSE.                                          25740000
     IF(OMASK.OR.(.NOT.ZZFRFL)) GO TO L300                            25750000
     IF(OWW_ID) GO TO L200                                            25760000
     IF(W_WE.LT.0.) OWW_ID=.TRUE.                                     25770000
     IF(OWW_ID) PRINT L100                                            25780000
  L100..FORMAT(//,...                                                 25790000
     ' ***EXECUTION TERMINATED; REVERSE FLOW IN VALVE _ID***',//)     25800000
  L200..CONTINUE                                                      25810000
     TERMT(OWW_ID)                                                    25820000
  L300..CONTINUE                                                      25830000
     END                                                              25840000
MACRO END                                                             25850000
''                                                                    73710000
''                                                                    73720000
'     FLOW JUNCTION MODULE JUNC     '                                 73730000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                       73740000
'     THIS MODULE PROVIDES FOR UP TO 9 FLOW JUNCTIONS.     '          73750000
'          INTERCONNECTIONS     '                                      73760000
'     WE-I : I-TH FLOW STREAM ENTERING     '                          73770000
'     WL : FLOW STREAM LEAVING     '                                  73780000
'          INTERCONNECTING VARIABLES     '                           73790000
'     P : PRESSURE (PSIA)     '                                       73800000
'     W : FLOWRATE (LBM/HR)     '                                     73810000
'     H : ENTHALPY (BTU/LBM)     '                                    73820000
'     R : DENSITY (LBM/FT3)     '                                     73830000
'     T : TEMPERATURE (F)     '                                       73840000
MACRO JUNC(ID,WL,WE1,WE2,WE3,WE4,WE5,WE6,WE7,WE8,WE9)                 73850000
     MACRO ASSIGN N                                                   73860000
     MACRO DECREMENT 2                                                73870000
     P_WE1=P_WL                                                       73880000
     P_WE2=P_WL                                                       73890000
```

```
MACRO IF(N=2) L2                                                    73900000
P_WE3=P_WL                                                          73910000
MACRO IF(N=3) L3                                                    73920000
P_WE4=P_WL                                                          73930000
MACRO IF(N=4) L4                                                    73940000
P_WE5=P_WL                                                          73950000
MACRO IF(N=5) L5                                                    73960000
P_WE6=P_WL                                                          73970000
MACRO IF(N=6) L6                                                    73980000
P_WE7=P_WL                                                          73990000
MACRO IF(N=7) L7                                                    74000000
P_WE8=P_WL                                                          74010000
MACRO IF(N=8) L8                                                    74020000
P_WE9=P_WL                                                          74030000
W_WL=W_WE1+W_WE2+W_WE3+W_WE4+W_WE5...                               74040000
     +W_WE6+W_WE7+W_WE8+W_WE9                                       74050000
H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3+H_WE4*W_WE4...    74060000
     +H_WE5*W_WE5+H_WE6*W_WE6+H_WE7*W_WE7+H_WE8*W_WE8...            74070000
     +H_WE9*W_WE9)/BOUND(1.E-3,1.E38,W_WL)                          74080000
R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3+R_WE4*W_WE4...    74090000
     +R_WE5*W_WE5+R_WE6*W_WE6+R_WE7*W_WE7+R_WE8*W_WE8...            74100000
     +R_WE9*W_WE9)/BOUND(1.E-3,1.E38,W_WL)                          74110000
T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3+T_WE4*W_WE4...    74120000
     +T_WE5*W_WE5+T_WE6*W_WE6+T_WE7*W_WE7+T_WE8*W_WE8...            74130000
     +T_WE9*W_WE9)/BOUND(1.E-3,1.E38,W_WL)                          74140000
MACRO GO TO L9                                                      74150000
MACRO L2..CONTINUE                                                  74160000
W_WL=W_WE1+W_WE2                                                    74170000
H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2)/BOUND(1.E-3,1.E38,W_WL)      74180000
R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2)/BOUND(1.E-3,1.E38,W_WL)      74190000
T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2)/BOUND(1.E-3,1.E38,W_WL)      74200000
MACRO GO TO L9                                                      74210000
MACRO L3..CONTINUE                                                  74220000
W_WL=W_WE1+W_WE2+W_WE3                                              74230000
H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3)...               74240000
     /BOUND(1.E-3,1.E38,W_WL)                                       74250000
R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3)...               74260000
     /BOUND(1.E-3,1.E38,W_WL)                                       74270000
T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3)...               74280000
     /BOUND(1.E-3,1.E38,W_WL)                                       74290000
MACRO GO TO L9                                                      74300000
MACRO L4..CONTINUE                                                  74310000
W_WL=W_WE1+W_WE2+W_WE3+W_WE4                                        74320000
H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3+H_WE4*W_WE4)/...  74330000
     BOUND(1.E-3,1.E38,W_WL)                                        74340000
R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3+R_WE4*W_WE4)/...  74350000
     BOUND(1.E-3,1.E38,W_WL)                                        74360000
T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3+T_WE4*W_WE4)/...  74370000
     BOUND(1.E-3,1.E38,W_WL)                                        74380000
MACRO GO TO L9                                                      74390000
MACRO L5..CONTINUE                                                  74400000
W_WL=W_WE1+W_WE2+W_WE3+W_WE4+W_WE5                                  74410000
H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3+H_WE4*W_WE4...    74420000
     +H_WE5*W_WE5)/BOUND(1.E-3,1.E38,W_WL)                          74430000
R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3+R_WE4*W_WE4...    74440000
     +R_WE5*W_WE5)/BOUND(1.E-3,1.E38,W_WL)                          74450000
T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3+T_WE4*W_WE4...    74460000
     +T_WE5*W_WE5)/BOUND(1.E-3,1.E38,W_WL)                          74470000
MACRO GO TO L9                                                      74480000
MACRO L6..CONTINUE                                                  74490000
W_WL=W_WE1+W_WE2+W_WE3+W_WE4+W_WE5...                               74500000
     +W_WE6                                                         74510000
H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3+H_WE4*W_WE4...    74520000
     +H_WE5*W_WE5+H_WE6*W_WE6)/BOUND(1.E-3,1.E38,W_WL)              74530000
R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3+R_WE4*W_WE4...    74540000
     +R_WE5*W_WE5+R_WE6*W_WE6)/BOUND(1.E-3,1.E38,W_WL)              74550000
T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3+T_WE4*W_WE4...    74560000
     +T_WE5*W_WE5+T_WE6*W_WE6)/BOUND(1.E-3,1.E38,W_WL)              74570000
MACRO GO TO L9                                                      74580000
MACRO L7..CONTINUE                                                  74590000
W_WL=W_WE1+W_WE2+W_WE3+W_WE4+W_WE5...                               74600000
```

```
          +W_WE6+W_WE7                                                    74610000
     H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3+H_WE4*W_WE4...     74620000
          +H_WE5*W_WE5+H_WE6*W_WE6+H_WE7*W_WE7)/BOUND(1.E-3,1.E38,W_WL) 74630000
     R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3+R_WE4*W_WE4...     74640000
          +R_WE5*W_WE5+R_WE6*W_WE6+R_WE7*W_WE7)/BOUND(1.E-3,1.E38,W_WL) 74650000
     T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3+T_WE4*W_WE4...     74660000
          +T_WE5*W_WE5+T_WE6*W_WE6+T_WE7*W_WE7)/BOUND(1.E-3,1.E38,W_WL) 74670000
     MACRO GO TO L9                                                       74680000
     MACRO L8..CONTINUE                                                   74690000
     W_WL=W_WE1+W_WE2+W_WE3+W_WE4+W_WE5...                                74700000
          +W_WE6+W_WE7+W_WE8                                              74710000
     H_WL=(H_WE1*(W_WE1+1.E-3)+H_WE2*W_WE2+H_WE3*W_WE3+H_WE4*W_WE4...     74720000
          +H_WE5*W_WE5+H_WE6*W_WE6+H_WE7*W_WE7+H_WE8*W_WE8)/...           74730000
          BOUND(1.E-3,1.E38,W_WL)                                         74740000
     R_WL=(R_WE1*(W_WE1+1.E-3)+R_WE2*W_WE2+R_WE3*W_WE3+R_WE4*W_WE4...     74750000
          +R_WE5*W_WE5+R_WE6*W_WE6+R_WE7*W_WE7+R_WE8*W_WE8)/...           74760000
          BOUND(1.E-3,1.E38,W_WL)                                         74770000
     T_WL=(T_WE1*(W_WE1+1.E-3)+T_WE2*W_WE2+T_WE3*W_WE3+T_WE4*W_WE4...     74780000
          +T_WE5*W_WE5+T_WE6*W_WE6+T_WE7*W_WE7+T_WE8*W_WE8)/...           74790000
          BOUND(1.E-3,1.E38,W_WL)                                         74800000
     MACRO L9..CONTINUE                                                   74810000
MACRO END                                                                 74820000
''                                                                        12120000
''                                                                        12130000
'          DESCRIPTION OF MODULE PIPESR     '                            12140000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                         12150000
'     THIS MODULE REPRESENTS THE FRICTIONAL AND ELEVATION     '          12160000
'     HEAD LOSSES,MASS INERTIA, THERMAL INERTIA AND HEAT LOSS TO      ' 12170000
'     THE ATMOSPHERE,AND TRANSPORT DELAY IN LONG     '                  12180000
'     PIPES. PIPESR IS A RESISTIVE TYPE      '                          12190000
'     COMPONENT FLOWING SUPERHEATED STEAM.     '                       12200000
'     ALL OPTIONS PROVIDE FOR FRICTIONAL AND ELEVATION PRESSURE LOSS.  '12210000
'     ADDITIONAL EFFECTS ARE PROVEDED BY TRANSLATION PARMS.     '       12220000
'          INTERCONNECTIONS     '                                       12230000
'     WE : FLOW ENTERING     '                                          12240000
'     WL : FLOW LEAVING     '                                          12250000
'          TRANSLATION PARAMETERS     '                                12260000
'     INERT : PROVIDES MASS INERTIA EFFECT (0: NO,1: YES)     '         12270000
'     THERM : PROVIDES THERMAL INERTIA AND HEAT LOSS     '              12280000
'               (0: NO EFFECT, N: CREATES N NODE PIPE)     '           12290000
'             NOTE : INVOCATION OF THERM PRECLUDES DELAY     '         12300000
'     DLAY : PROVIDES FOR TRANSPORT DELAY (0: NO,1: YES)     '         12310000
'     INTERCONNECTING VARIABLES     '                                   12320000
'     H : ENTHALPY (BTU/LBM)     '                                      12330000
'     P : PRESSURE (PSIA)     '                                         12340000
'     R : DENSITY (LBM/FT3)     '                                       12350000
'     T : TEMPERATURE (F)     '                                         12360000
'     W : FLOW RATE (LBM/HR)     '                                      12370000
'     INTERNAL VARIABLES     '                                          12380000
'     ZDD : PRESSURE PROFILE INCREMENT (PSI)     '                      12390000
'     ZDP : DRIVING PRESSURE DIFFERENTIAL (PSI)     '                   12400000
'     ZDW : EXPANSION FLOW INCREMENT (LBM/HR)     '                     12410000
'     ZMF : PRESSURE DIFFERENTIAL PARAMETER (PSI)     '                 12420000
'     ZHW : INTERMEDIATE ENTHALPY (BTU/LBM)     '                       12430000
'     ZPN : PRESSURE LEAVING THERM NODES (PSI)     '                    12440000
'     ZPA : ACCUMULATED ZP1     '                                       12450000
'     ZPB : ACCUMULATED ZP2     '                                       12460000
'     ZP1 : PARTIAL OF DENSITY W/R PRESSURE (LBM/PSI-FT3)     '         12470000
'     ZP2 : PARTIAL OF DENSITY W/R ENTHALPY (LBM2/FT3-BTU)     '        12480000
'     ZQN : HEAT LOSS PER THERM NODE (BTU/HR)     '                     12490000
'     ZRE : EFFECTIVE DENSITY (LBM/FT3)     '                         12500000
'     ZRN : DENSITY LEAVING THERM NODES (LBM/FT3)     '               12510000
'     ZTD : TRANSPORT DELAY (SEC)     '                               12520000
'     ZXN : NUMBER OF THERM NODES (=THERM)     '                      12530000
'     ZVN : PIPE VOLUME PER THERM NODE (FT3)     '                    12540000
'     ZWW : FLOWRATE LEAVING THERM NODES (LBM/HR)     '               12550000
'     PARAMETERS                                INERT THERM DLAY  '   12560000
'     KAF : FLOW AREA (FT2)                       X     X     X   '   12570000
'     KCF : FLOW CONDUCTANCE                      X     X     X     '12580000
'     KCK : CHECK VALVE LOGICAL SWITCH            X     X     X     '12590000
'     KDH : ELEVATION LOSS INLET TO OUTLET (FT)   X     X     X     '12600000
```

```
'     KLP : PIPE LENGTH (FT)                          X     X     X     '12610000
'     KMM : MASS OF PIPE METAL (LBM)                        X          ' 12620000
'     KTA : AMBIENT TEMPERATURE (F)                         X          ' 12630000
'     KUL : HEAT LOSS CONDUCTANCE (BTU/F-HR)                X        '   12640000
'     KVP : VOLUME OF PIPE (FT3) (KVP=KLP*KAF)              X          ' 12650000
'          NOTE : ONLY KCF AND KDH ARE REQUIRED IF NO      '            12660000
'                 OPTIONAL PARAMETERS ARE SPECIFIED.     '             12670000
'          INITIAL CONDITIONS     '                                    12680000
'     ZIH : ENTHALPY LEAVING THE THERM NODES (BTU/LBM) (THERM)     '    12690000
'     IHWL : ENTHALPY OF FLOW LEAVING (BTU/LBM) (DELAY)     '          12700000
'     IWWE : FLOW RATE ENTERING (LBM/HR) (INERT)     '                 12710000
'          VALIDITY FLAGS     '                                       12720000
'     OWW : REVERSE FLOW     '                                        12730000
MACRO PIPESR(ID,WE,WL,INERT,THERM,DLAY)                               12740000
     MACRO RELABEL L100,L200,L300                                     12750000
     LOGICAL OMASK,OWW_ID,KCK_ID                                      12760000
     CONSTANT KFM_ID=0.5                                              12770000
     MACRO IF(INERT=1) L1                                             12780000
'          INERTIAL CALCULATIONS     '                                12790000
     PROCEDURAL(W_WE=P_WE,P_WL,ZRE_ID)                                12800000
     ZDP_ID=P_WE-P_WL+ZRE_ID*KDH_ID/144.                              12810000
     ZD2_ID=ZDP_ID*ZDP_ID                                             12820000
     W_WE=(ZD2_ID/(ZD2_ID+KFM_ID))*KCF_ID...                          12830000
     *SIGN(SQRT(ABS(ZRE_ID*ZDP_ID)),ZDP_ID)                           12840000
     IF(ZDP_ID.LE.0..AND.KCK_ID.AND.ZZFRFL) W_WE=0.                   12850000
     END                                                              12860000
     MACRO GO TO L2                                                   12870000
   MACRO L1..CONTINUE                                                 12880000
     ZDP_ID=P_WE-P_WL+ZRE_ID*KDH_ID/144.                              12890000
     ZD2_ID=ZDP_ID*ZDP_ID                                             12900000
     DW_WE=3600.*32.2*144.*(KAF_ID/KLP_ID)*(ZDP_ID...                 12910000
     -ZD2_ID/(ZD2_ID+KFM_ID)*ABS(W_WE/KCF_ID)*(W_WE/KCF_ID)/ZRE_ID)   12920000
     W_WE=INTVC(DW_WE,IW_WE)                                          12930000
   MACRO L2..CONTINUE                                                 12940000
     MACRO IF(THERM=0) L4                                             12950000
'          THERMAL CALCULATIONS     '                                12960000
     ARRAY ZWW_ID(THERM),ZHW_ID(THERM),ZP1_ID(THERM),ZP2_ID(THERM),... 12970000
     ZRN_ID(THERM),ZPN_ID(THERM),ZDH_ID(THERM),ZIH_ID(THERM)          12980000
     PROCEDURAL(ZRE_ID,ZRN_ID,ZP1_ID,ZP2_ID,ZPA_ID,ZPB_ID...          12990000
     =P_WE,P_WL,ZHW_ID,R_WE)                                          13000000
     ZXN_ID=THERM                                                     13010000
     ZDD_ID=(P_WE-P_WL)/ZXN_ID                                        13020000
     ZPN_ID(1)=P_WE-ZDD_ID                                            13030000
     ZRN_ID(1)=RVOFPH(ZPN_ID(1),ZHW_ID(1))                            13040000
     ZP1_ID(1)=5.4*DRVDPH(ZPN_ID(1),ZHW_ID(1))                        13050000
     ZP2_ID(1)=DRVDHP(ZPN_ID(1),ZHW_ID(1))                            13060000
     ZPA_ID=ZP1_ID(1)/ZXN_ID                                          13070000
     ZPB_ID=ZP2_ID(1)/ZXN_ID                                          13080000
     ZRE_ID=(R_WE+ZRN_ID(1))/(2.*ZXN_ID)                              13090000
     MACRO IF(THERM=1) L3                                             13100000
     INTEGER I                                                        13110000
     DO ID_L3 I=2,THERM                                               13120000
     ZPN_ID(I)=ZPN_ID(I-1)-ZDD_ID                                     13130000
     ZRN_ID(I)=RVOFPH(ZPN_ID(I),ZHW_ID(I))                            13140000
     ZP1_ID(I)=5.4*DRVDPH(ZPN_ID(I),ZHW_ID(I))                        13150000
     ZP2_ID(I)=DRVDHP(ZPN_ID(I),ZHW_ID(I))                            13160000
     ZPA_ID=ZPA_ID+ZP1_ID(I)/ZXN_ID                                   13170000
     ZPB_ID=ZPB_ID+ZP2_ID(I)/ZXN_ID                                   13180000
     ZRE_ID=ZRE_ID+(ZRN_ID(I)+ZRN_ID(I-1))/(2.*ZXN_ID)                13190000
     ID_L3..CONTINUE                                                  13200000
   MACRO L3..CONTINUE                                                 13210000
     END                                                              13220000
     PROCEDURAL(ZDH_ID,W_WL=W_WE,H_WE,T_WL,ZRN_ID,ZPN_ID,...          13230000
     ZP1_ID,ZP2_ID,ZPA_ID,ZPB_ID,ZRE_ID)                              13240000
     ZXN_ID=THERM                                                     13250000
     ZQN_ID=KUL_ID*(T_WL-KTA_ID)/ZXN_ID                               13260000
     ZVN_ID=KVP_ID/ZXN_ID                                             13270000
     ZMN_ID=KMM_ID/ZXN_ID                                             13280000
     ZWW_ID(1)=((ZRN_ID(1)+ZP2_ID(1)*(ZHW_ID(1)-H_WE)...              13290000
     +0.25*ZMN_ID/ZVN_ID)...                                          13300000
     *W_WE+ZP2_ID(1)*ZQN_ID)/(ZRN_ID(1)+0.25*ZMN_ID/ZVN_ID)           13310000
```

```
     ZDH_ID(1)=(W_WE*H_WE-ZWW_ID(1)*ZHW_ID(1)-ZQN_ID)...                13320000
     /(3600.*ZVN_ID*((ZRN_ID(1)+ZHW_ID(1)*ZP2_ID(1))...                 13330000
     +0.25*ZMN_ID/ZVN_ID))                                              13340000
     MACRO IF(THERM=1) L8                                               13350000
     DO ID_L2 I=2,THERM                                                 13360000
     ZWW_ID(I)=((ZRN_ID(I)+ZP2_ID(I)*(ZHW_ID(I)-ZHW_ID(I-1))...         13370000
     +0.25*ZMN_ID/ZVN_ID)...                                            13380000
     *ZWW_ID(I-1)-ZP2_ID(I)*ZQN_ID)/(ZRN_ID(I)+0.25*ZMN_ID/ZVN_ID)      13390000
     ZDH_ID(I)=(ZWW_ID(I-1)*ZHW_ID(I-1)-ZWW_ID(I)*ZHW_ID(I)-ZQN_ID)...  13400000
     /(3600.*ZVN_ID*((ZRN_ID(I)+ZHW_ID(I)*ZP2_ID(I))...                 13410000
     +0.25*ZMN_ID/ZVN_ID))                                              13420000
     ID_L2..CONTINUE                                                    13430000
   MACRO L8..CONTINUE                                                   13440000
     W_WL=ZWW_ID(THERM)                                                 13450000
     END                                                                13460000
     ZHW_ID=INTVC(ZDH_ID,ZIH_ID)                                        13470000
     H_WL=ZHW_ID(THERM)                                                 13480000
     MACRO GO TO L5                                                     13490000
   MACRO L4..CONTINUE                                                   13500000
     ZHW_ID=H_WE                                                        13510000
     ZP2_ID=DRVDHP(P_WL,H_WL)                                           13520000
     ZRE_ID=(R_WE+R_WL)/2.                                              13530000
     W_WL=(ZRE_ID+ZP2_ID*(H_WE-H_WL))*W_WE/ZRE_ID                       13540000
     MACRO IF(DLAY=1) L7                                                13550000
'          TRANSPORT ELAY CALCULATIONS     '                           13560000
     H_WL=ZHW_ID                                                        13570000
     MACRO GO TO L5                                                     13580000
   MACRO L7..CONTINUE                                                   13590000
     ZTD_ID=3600.*ZRE_ID*KVP_ID/(ABS(W_WE)+1.E-6)                       13600000
     H_WL=DELAY(ZHW_ID,IH_WL,ZTD_ID,1000)                               13610000
   MACRO L5..CONTINUE                                                   13620000
     T_WL=TVOFPH(P_WL,H_WL)                                             13630000
     R_WL=RVOFPH(P_WL,H_WL)                                             13640000
'          VALIDITY CHECKS     '                                       13650000
     PROCEDURAL(OWW_ID=W_WE)                                            13660000
     CONSTANT OWW_ID=.FALSE.                                            13670000
     IF(OMASK.OR.(.NOT.ZZFRFL)) GO TO L300                              13680000
     IF(OWW_ID) GO TO L200                                              13690000
     IF(W_WE.LT.0.) OWW_ID=.TRUE.                                       13700000
     IF(OWW_ID) PRINT L100                                              13710000
  L100..FORMAT(//,...                                                   13720000
     ' ***EXECUTION TERMINATED; REVERSE FLOW IN PIPE _ID***',//)        13730000
  L200..CONTINUE                                                        13740000
     TERMT(OWW_ID)                                                      13750000
  L300..CONTINUE                                                        13760000
     END                                                                13770000
MACRO END                                                               13780000
''                                                                      06000000
''                                                                      06010000
'          DESCRIPTION OF MODULE DEAER     '                           06020000
'     CYBER VERSION, REVISION 0 (4/22/83)     '                       06030000
'     THIS MODULE REPRESENTS A TRAY TYPE OPEN FEEDWATER HEATER,     '  06040000
'     DEAERATOR.  THE VESSEL IS MODELED AS AN EQUILIBRIUM     '       06050000
'     TWO-PHASE MIXTURE; THE EFFECTS OF NON-CONDENSEABLE     '        06060000
'     GASES ARE NOT ACCOUNTED FOR.     '                              06070000
'          INTERCONNECTIONS     '                                     06080000
'     WE : FLOW ENTERING     '                                        06090000
'     WL : FLOW LEAVING     '                                         06100000
'          INTERCONNECTING VARIABLES     '                           06110000
'     H : ENTHALPY (BTU/LBM)     '                                   06120000
'     L : LEVEL (IN)     '                                           06130000
'     P : PRESSURE(PSIA)     '                                       06140000
'     R : DENSITY(LBM/FT3)     '                                     06150000
'     T : TEMPERATURE(F)     '                                       06160000
'     W : FLOWRATE(LBM/HR)     '                                     06170000
'          INTERNAL VARIABLES     '                                  06180000
'     ZRH(ZDR) : BULK DENSITY IN VESSEL(LBM/FT3)     '              06190000
'     ZRF : DENSITY OF SAT. LIQUID (LBM/FT3)     '                  06200000
'     ZRG : DENSITY OF SAT. VAPOR (LBM/FT3)     '                   06210000
'     ZUH(ZDU) : BULK SPECIFIC INTERNAL ENERGY IN VESSEL(BTU/LBM)     ' 06220000
'          PARAMETERS     '                                          06230000
```

```
'     KDH : STORAGE TANK DIAMETER (IN)     '                          06240000
'     KDE : ELEVATION DROP TO PUMP SUCTION (FT)     '                 06250000
'     KVS : VOLUME OF STORAGE TANK(FT3)     '                         06260000
'     KVT : TOTAL VOLUME OF STORAGE AND DEARATING TANKS(FT3)     '    06270000
'          INITIAL CONDITIONS     '                                   06280000
'     ZIR(ZRH) : BULK DENSITY(LBM/FT3)     '                         06290000
'     ZIU(ZUH) : BULK INTERNAL ENERGY(BTU/LBM)     '                 06300000
'          VALIDITY FLAGS     '                                      06310000
'     ODL : LEVEL OUT OF LIMITS(TERMINATE)     '                     06320000
MACRO DEAER(ID,WE,WL)                                                06330000
     MACRO RELABEL L100,L200,L300                                    06340000
     INTEGER ZNN_ID                                                  06350000
     CONSTANT ZNN_ID=-1                                              06360000
     LOGICAL OMASK,ODL_ID                                            06370000
     ZDU_ID=(W_WE*H_WE-W_WL*H_WL-3600.*KVT_ID*ZUH_ID*ZDR_ID)...      06380000
     /(3600.*KVT_ID*ZRH_ID)                                          06390000
     ZUH_ID=INTVC(ZDU_ID,ZIU_ID)                                     06400000
     ZDR_ID=(W_WE-W_WL)/(3600.*KVT_ID)                               06410000
     ZRH_ID=INTVC(ZDR_ID,ZIR_ID)                                     06420000
     P_WE=POFUR(ZUH_ID,ZRH_ID,ZNN_ID)                                06430000
     P_WL=P_WE+KDE_ID*R_WL/144.                                      06440000
     ZHF_ID=HFOFP(P_WE)                                              06450000
     ZHG_ID=HGOFP(P_WE)                                              06460000
     H_WL=ZHF_ID                                                     06470000
     R_WL=ZRF_ID                                                     06480000
     T_WL=TLOFPH(P_WE,ZHF_ID)                                        06490000
     ZRF_ID=RLOFPH(P_WE,ZHF_ID)                                      06500000
     ZRG_ID=RVOFPH(P_WE,ZHG_ID)                                      06510000
     L_ID=KDH_ID*(KVT_ID/KVS_ID)*(ZRH_ID-ZRG_ID)/(ZRF_ID-ZRG_ID)     06520000
'          VALIDITY CHECK     '                                      06530000
     PROCEDURAL (ODL_ID=L_ID)                                        06540000
     CONSTANT ODL_ID=.FALSE.                                         06550000
     IF(OMASK.OR.(.NOT.ZZFRFL)) GO TO L300                           06560000
     IF(ODL_ID) GO TO L200                                           06570000
     IF(L_ID.LE.0..OR.L_ID.GE.KDH_ID) ODL_ID=.TRUE.                  06580000
     IF(ODL_ID) PRINT L100                                           06590000
   L100..FORMAT(//,...                                               06600000
     '***EXECUTION TERMINATED; LEVEL OUT OF BOUNDS DEAER _ID ***',//)06610000
   L200..CONTINUE                                                    06620000
     TERMT(ODL_ID)                                                   06630000
   L300..CONTINUE                                                    06640000
     END                                                             06650000
MACRO END                                                            06660000
''
''
''
PROGRAM DEAERSTUDY
" THIS PROGRAM IS INTENDED FOR DEAERATOR LEVEL CONTROL STUDY  "
" "
DYNAMIC
     TERMT(T.GE.TSTOP)
     CONSTANT TSTOP=0.,IALG=2,OMASK=.FALSE.
DERIVATIVE
" "
     TABLE KHCCND,1,7/...
     690.,800.,880.,930.,960.,980.,1000.,...
     6000.,5000.,4000.,3000.,2000.,1000.,0./
     CONSTANT KCKCND=.FALSE.,KCKEXT=.FALSE.,KCKVLV=.FALSE.
     CONSTANT KDHVLV=0.0
PUMPMD("CND","COND","LCND")
     CONSTANT KEPCND=0.85,KNPCND=2.,KPRCND=3.4E-8,KQRCND=5000.
"          BOUNDARY CONDITIONS     "
     CONSTANT PCOND=1.42,HCOND=81.7,RCOND=61.8
" "
CONNI("C1","LCND","LC1")
     CONSTANT IPLC1=359.8
" "
VALVEI("VLV","LC1","LVLV")
     CONSTANT KCPVLV=4.403E4,KCVVLV=1.044E5,KVAVLV=1
" "
JUNC("J1","EDEA","LEXT","LVLV")
```

```
" "
PIPESR("EXT","STM","LEXT",0,0,0)
     CONSTANT KCFEXT=8.84E5,KDHEXT=0.
"          BOUNDARY CONDITIONS     "
     CONSTANT PSTM=142.2,HSTM=1353.2,RSTM=0.213
" "
DEAER("DEA","EDEA","FW")
     CONSTANT KDEDEA=0.,KDHDEA=144.,KVSDEA=8000.,KVTDEA=9980.
     CONSTANT ZIRDEA=36.99,ZIUDEA=325.4
"          BOUNDARY CONDITIONS     "
     CONSTANT WFW=4.5E6
" "
"          CONTROLS     "
DEMAND=K1*WFW+K2*(LSET-LDEA)
SUPPLY=K3*WCOND
     CONSTANT K1=1.E-6,K2=1.,K3=1.E-6,LSET=120.
PICONT("CNT",DEMAND,SUPPLY,YVLV,1)
     CONSTANT CPGCNT=0.1,CIGCNT=0.05,KLLCNT=0.,KHLCNT=1.
     CONSTANT ZICCNT=0.75
" "
INTEGER COUNT,KOUNT
PROCEDURAL(COUNT=)
     CONSTANT COUNT=0,KOUNT=1000
     COUNT=COUNT+1
     TERMT(COUNT.GE.KOUNT)
END $ " PROCEDURAL "
END $ " DERIVATIVE "
END $ " DYNAMIC "
END $ " PROGRAM "
```

B Recoded deaerator example program

```
{ ********************************************************************** }
{ @pump.dym }

model type pump

{ interface to model :
  pwe  : pressure of water entering (psia)
  pwl  : pressure of water leaving (psia)
  wwe  : flowrate of water entering (lbm/hr)
  wwl  : flowrate of water leaving (lbm/hr)
  hwe  : entalpie of water entering (BTU/lbm)
  hwl  : entalpie of water leaving (BTU/lbm)
  rwe  : density of water entering (lbm/ft3)
  rwl  : density of water leaving (lbm/ft3)
  Twe  : temperature of water entering (F)
  Twl  : temperature of water leaving (F)

  parameters for model
  kep  : maximum pump efficiency in %
  knp  : number of operating pumps
  kqr  : volume flow rate at kep (gpm)
       : pump head curve }

terminal  pwe , wwe , hwe , rwe
terminal  pwl , wwl , hwl , rwl , Twl

cut  inwater  ( pwe , wwe , hwe , rwe , . )
cut  outwater ( pwl , wwl , hwl , rwl , Twl )

path water   < inwater - outwater >

        parameter kep = 0.85 , knp = 2. , kqr = 5000.

{ local variables and constants
  zdh : pump differential head ( ft of H2O )
```

```
  zqw : volume flow rate (gpm)
  zep : pump efficiency in percent }

local zdh , zqw , zep

{ fluid dymamics }

zdh = 144. * ( pwl - pwe ) / rwe

zqw = PH ( zdh )

wwe = 8.02 * knp * rwe * zqw

{ pump efficiency }

zep = kep - (kep*( zqw-kqr )**2)/kqr**2

{ conservation of energy }

hwl = hwe + 0.0013*zdh/zep

{ conservation of mass / no storage in pump }

wwe = wwl

{ fluid properties }

rwl = RLOFPH ( pwl,hwl )
Twl = TLOFPH ( pwl,hwl )

end


{ ********************************************************************** } {
{ @conni.dym }

model type conni

{ interface to model :
  pwe  : pressure of water entering (psia)
  pwl  : pressure of water leaving (psia)
  wwe  : flowrate of water entering (lbm/hr)
  wwl  : flowrate of water leaving (lbm/hr)
  hwe  : entalpie of water entering (BTU/lbm)
  hwl  : entalpie of water leaving (BTU/lbm)
  rwe  : density of water entering (lbm/ft3)
  rwl  : density of water leaving (lbm/ft3)
  Twe  : temperature of water entering (F)
  Twl  : temperature of water leaving (F)

  parameters for model
  no parameters used }

terminal  pwe , wwe , hwe , rwe
terminal  pwl , wwl , hwl , rwl , Twl
cut  inwater  ( pwe , wwe , hwe , rwe , . )
cut  outwater ( pwl , wwl , hwl , rwl , Twl )
path water   < inwater - outwater >


{ local variables and constants }

local    dp
constant kdp = 1000.

dp = ( wwe - wwl ) / kdp
der(pwl) = dp
pwe = pwl
hwl = hwe
Twl = TLOFPH ( pwl,hwl )
rwl = RLOFPH ( pwl,hwl )
```

```
end


{ ********************************************************************* }
{ @valvei.dym }

model type valvei

{ interface to model :
  pwe  : pressure of water entering (psia)
  pwl  : pressure of water leaving (psia)
  wwe  : flowrate of water entering (lbm/hr)
  wwl  : flowrate of water leaving (lbm/hr)
  hwe  : entalpie of water entering (BTU/lbm)
  hwl  : entalpie of water leaving (BTU/lbm)
  rwe  : density of water entering (lbm/ft3)
  rwl  : density of water leaving (lbm/ft3)
  Twe  : temperature of water entering (F)
  Twl  : temperature of water leaving (F)
  y    : valve position (fraction)

  parameters for model
  kcp  : associated pipe conductance
  kcv  : maximum valve conductance
  kdh  : elevation drop inlet to outlet (ft) }

terminal  pwe , wwe , hwe , rwe
terminal  pwl , wwl , hwl , rwl , Twl
terminal  y
cut  inwater  ( pwe , wwe , hwe , rwe , . )
cut  outwater ( pwl , wwl , hwl , rwl , Twl )
cut  valve   ( y )
path water   < inwater - outwater >

        parameter kcp = 44030. , kcv = 104400. , kdh = 0.

{ local variables and constants
  zcv  : valve conductance
  zcq  : combined pipe ans valve conductance
  zdp  : differential pressure (psi) }

local zcv , zcq , zdp

{ valve conductance as function of y }

zcv = abs(y)**3 * kcv

{ differential pressure }

zdp = pwe - pwl + rwe*kdh/144.

{ overall conductance of valve and associated pipe }

zcq = kcp*zcv / sqrt ( ABS(kcp*kcp + zcv*zcv) + 1./1000. )

{......}

wwl = zcq * SIGN( SQRT(ABS(rwe*zdp)) , zdp )

{ quasi steady state and adiabatic performance wilkl result in }

wwl = wwe
hwl = hwe

{ fluid properties }

rwl = RLOFPH ( pwl,hwl )
Twl = TLOFPH ( pwl,hwl )
```

```
end


{ ********************************************************************* }
@junc2.dym

model type junction2

{ interface to model :
  pwe1 : pressure of water entering (psia)
  pwe2 : pressure of water entering (psia)
  pwl  : pressure of water leaving (psia)
  wwe1 : flowrate of water entering (lbm/hr)
  wwe2 : flowrate of water entering (lbm/hr)
  wwl  : flowrate of water leaving (lbm/hr)
  hwe1 : entalpie of water entering (BTU/lbm)
  hwe2 : entalpie of water entering (BTU/lbm)
  hwl  : entalpie of water leaving (BTU/lbm)
  rwe1 : density of water entering (lbm/ft3)
  rwe2 : density of water entering (lbm/ft3)
  rwl  : density of water leaving (lbm/ft3)
  Twe1 : temperature of water entering (F)
  Twe2 : temperature of water entering (F)
  Twl  : temperature of water leaving (F)

  parameters for model
  no parameters required  }

terminal  pwe1 , wwe1 , hwe1 , rwe1 , Twe1
terminal  pwe2 , wwe2 , hwe2 , rwe2 , Twe2
terminal  pwl  , wwl  , hwl  , rwl  , Twl
cut  inwater1  ( pwe1 , wwe1 , hwe1 , rwe1 , Twe1 )
cut  inwater2  ( pwe2 , wwe2 , hwe2 , rwe2 , Twe2 )
cut  outwater ( pwl , wwl , hwl , rwl , Twl )
path water   < inwater1 - outwater >


{ local variables and constants
  no local constants required }
local Tcheck

{ mass balance }

wwl = wwe1 + wwe2

{ energy balance }

hwl = ( (wwe1+1./1000.)*hwe1 + wwe2*hwe2 ) / ->
              BOUND(1./1000.,1.E38,wwl)

{ momentum balance

not considered }

{ heat transfer

not considered }

{ fluid mechanics }

pwe1 = pwl
pwe2 = pwl

{ fluid properties }

Twl = ( (wwe1+1./1000.)*Twe1 + wwe2*Twe2 ) / ->
              BOUND(1./1000.,1.E38,wwl)
        Tcheck = TLOFPH( pwl,hwl )
rwl = ( (wwe1+1./1000.)*rwe1 + wwe2*rwe2 ) / ->
              BOUND(1./1000.,1.E38,wwl)
```

```
{ geometry calculations

not considered }

end


{ ********************************************************************** }
@deaer.dym

model type deaerator

{ interface to model :
  pwe  : pressure of water entering (psia)
  pwl  : pressure of water leaving (psia)
  wwe  : flowrate of water entering (lbm/hr)
  wwl  : flowrate of water leaving (lbm/hr)
  hwe  : entalpie of water entering (BTU/lbm)
  hwl  : entalpie of water leaving (BTU/lbm)
  rwe  : density of water entering (lbm/ft3)
  rwl  : density of water leaving (lbm/ft3)
  Twe  : temperature of water entering (F)
  Twl  : temperature of water leaving (F)
  lev : water level in deaerator

  parameters for model
  kde : elevation drop to pump suction (ft)
  kdh : diameter storage tank
  kvs : volume of storage tank
  kvt : total volume of storage and deaeration tank }

terminal  pwe , wwe , hwe , rwe
terminal  pwl , wwl , hwl , rwl , Twl
terminal  lev
cut  inwater  ( pwe , wwe , hwe , rwe , . )
cut  outwater ( pwl , wwl , hwl , rwl , Twl )
cut  level   ( lev )
path water   < inwater - outwater >

        parameter kde = 0. , kdh = 144.0 , kvs = 8000.0 , kvt = 9980.0

{ local variables and constants
  zrh  : bulk density in vessel (lbm/ft3)
  zdr  : derivative of zrh
  zuh  : bulk specific internal energy in vessel (BTU/lbm)
  zrf  : density of saturied liquid (lbm/ft3) }

local    zrh , zdr , zuh , zrf , zrg , zhf , zhg
constant znn = -1

{ mass balance }

zdr        = ( wwe - wwl ) / (3600.0*kvt)
        der( zrh ) = zdr

{ energy balance }

der( zuh ) = ( wwe*hwe - wwl*hwl - 3600.0*kvt*zuh*zdr )   ->
                     / ( 3600.0*zrh*kvt )

{ momentum balance

not considered }

{ heat transfer

not considered }

{ fluid mechanics }

pwl = pwe + rwl*kde/144.0
```

```
{ fluid properties }

pwe = POFUR ( zuh,zrh,znn)
zhf = HFOFP ( pwe )
zhg = HGOFP ( pwe )
Twl = TLOFPH ( pwe,zhf )
zrf = RLOFPH ( pwe,zhf )
zrg = RVOFPH ( pwe,zhg )
hwl = zhf
rwl = zrf

{ geometry calculations }

lev = kdh*( kvt/kvs )*( zrh - zrg )/( zrf - zrg )

end


{ ********************************************************************** }
@pipe.dym

model type pipe

{ interface to model :
  pwe  : pressure of water entering (psia)
  pwl  : pressure of water leaving (psia)
  wwe  : flowrate of water entering (lbm/hr)
  wwl  : flowrate of water leaving (lbm/hr)
  hwe  : entalpie of water entering (BTU/lbm)
  hwl  : entalpie of water leaving (BTU/lbm)
  rwe  : density of water entering (lbm/ft3)
  rwl  : density of water leaving (lbm/ft3)
  Twe  : temperature of water entering (F)
  Twl  : temperature of water leaving (F)

  parameters for model
  kdh : elevation drop inlet to outlet (ft)
  kcf : flow conductance ( (lbm*ft3)/(hr**2*psi) }

terminal  pwe , wwe , hwe , rwe
terminal  pwl , wwl , hwl , rwl , Twl

cut  inwater  ( pwe , wwe , hwe , rwe , . )
cut  outwater ( pwl , wwl , hwl , rwl , Twl )

path water   < inwater - outwater >

        parameter kdh = 0. , kcf = 884000.

{ local variables and constants
  zdp : driving pressure differential (psi)
  zp2 : square driving pressure differential (psi**2)
  zhw : intermediate enthalpy (BTU/lbm)
  zre : effective density (lbm/ft3) }

local    zdp , zd2 , zp2 , zhw , zre
constant kfm = 0.5

{ driving pressure : head loss in pipe due to friction }

zdp = pwe - pwl + zre*kdh/144.

{ steady state momentum equation }

zd2 = zdp * zdp
wwe =  ( zd2/(zd2+kfm) ) * kcf * sign( sqrt(abs(zre*zdp)) , zdp)

{ thermodynamics }

zhw = hwe
```

```
zp2 = DRVDHP ( pwl,hwl )
zre = (rwe + rwl ) / 2.
wwl = ( zre + zp2*(hwl - hwe) ) * wwe / zre
hwl = zhw

Twl = TVOFPH ( pwl,hwl )
rwl = RVOFPH ( pwl,hwl )

end


{ ****************************************************************** }
@picont.dym
model type picont

{ interface to model :
  csp   : controller set point
  pvar  : process variable to be controlled
  out   : process parameter being changed by controller

  parameter for model
  cpg  : proportional gain
  cig  : integral gain
  kll  : low limit of contoller output
  khl  : high limit of controller output }

terminal csp , pvar , out

cut setpoint (csp)
cut input    (pvar)
cut control  (out)

parameter cpg = 0.1 , cig = 0.05 , kll = 0. , ->
                  khl = 1. , zoiic = 0.75

{ local variables and constants
  zoi : integrator output (fraction)
  zop : proportional output (fraction)
  zdc : integrator derivative (fraction/sec)
  zot : unbounded output
  zer : control error signal }

local zoi , zop , zdc , zot , zer

{ calculate error : direct or reverse acting }

zer = (csp-pvar)

{ calculate proportional output  z o p }

zop = cpg * zer

{ calculate integral output  z o i

prevent windup if output out of bounds }
zdc      = cig * zer
zoi      = LIMINT ( zdc , zoiic , kll , khl )

{ calculate overall output  z o t  / limit the output  o u t }

zot = zoi + zop
out = BOUND ( kll , khl , zot )

end


{ ****************************************************************** }

model power

submodel (pump)      pump
```

```
submodel (conni)     con   ( ic pwl=359.8 )
submodel (valvei)    valv
submodel (pipe)      pipe
submodel (picont)    ctrl
submodel (deaerator) dea1  ( ic zrh = 36.99 , zuh = 325.4 )
submodel (junction2) junc

constant lset = 120.
constant k1 = 0.000001 , k2 = 1.0 , k3 = 0.000001
constant wfw = 4.5E6
        local    putwe , pitwe

{ main line of water flow }

connect (water) pump to con to valv to junc to dea1
{ boundary conditions }
pump.pwe = 1.42
pump.hwe = 81.7
pump.rwe = 61.8
        putwe = TLOFPH(pump.pwe,pump.hwe)
dea1.wwl = wfw

{ extraction steam line }

connect (water) pipe to junc:inwater2
{ boundary conditions }
pipe.pwe = 142.2
pipe.hwe = 1353.2
pipe.rwe = 0.213
        pitwe = TVOFPH(pipe.pwe,pipe.hwe)

{ control system : P/I contrloller }

ctrl.csp  = k1*dea1.wwl + k2*(lset - dea1.lev)
ctrl.pvar = k3*pump.wwe
connect ctrl:control at valv:valve

end
```

C DYMOLA test programs of the new submodels

C.1 PIPE test program

```
{ ************************************************************************ }
{ @lib.bnd }
{ Bond Graph bond }

model type bond
  cut A (x / y) B (y / -x)
  main cut C [A B]
  main path P <A - B>
end


{ ************************************************************************ }
{ @lib.c }
{ Bond Graph capacitor/compliance }

model type C
  main cut A (e / f)
  parameter C=1.0
  C*der(e) = f
end


{ ************************************************************************ }
```

```
{ @lib.i }
{ Bond Graph inductor/inertia }

model type I
  main cut A (e / f)
  parameter I=1.0
  I*der(f) = e
end


{ ********************************************************************** }
{ @lib.mrs }
{ Bond Graph of a modulated resistive source }

model type mRS
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter R=1.0
  R*f1*f1 = e1
  e1*f1 = e2*f2
end


{ ********************************************************************** }
{ @lib.mgs }
{bond graph conductance source for one dimensional cell}

model type mGS
  cut A(e1/f1),B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter b=1.0
  terminal vwater
  local G1,G
  G1 = b / EXP(BOUND(0.,20.,vwater))
  G  = G1/e2
  G*e1  = f1
  f1*e1 = f2*e2
end


{ ********************************************************************** }
{ @lib.mc }
{ Bond Graph modulated capacitor/compliance }

model type mC
  main cut A (e / f)
  parameter C = 413.
  C/e*der(e) = f
end


{ ********************************************************************** }
{ @lib.sfs }
{ bond graph model for flow source with input SFS }

model type SFS
  cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
  main path P<InTherm - OutTherm>
  terminal Vd
  local s
  parameter s0 = 7033.0 , m = 2.562 , T0 = 545.6 , rho = 3.41
  s = s0 + m*(Ti-T0)
  Sdi = s * rho * Vd
  Ti*Sdi = To*Sdo
end


{ ********************************************************************** }
{ @lib.sfc }
```

```
{ bond graph model of a flow source }

model type SFc
  main cut A(Ti/-Sdi)
  parameter s0 = 7033.0 , m = 2.562 , T0 = 545.6 , rho = 3.41
  terminal Vd
  local s
  s = s0 + m*(Ti-T0)
  Sdi = s * rho * Vd
end


{ ************************************************************************ }
{ @lib.SE }
{ Bond Graph effort source }

model type SE
  main cut A (e / .)
  terminal E0
  E0 = e
end


{ ************************************************************************ }
{ @lib.SiF }
{ bond graph model of a flow source }

model type SiF
  main cut A(./ f)
  terminal F0
  F0=f
end


{ ************************************************************************ }
{ @cvseg.dym }
{ bond graph model for a convective segment of a pipe }

model type CvSeg

submodel mC (C=9.2E3) (ic e=545.)
submodel mGS (b=17.2)
submodel SFS (s0=7033.,m=3.73,T0=545,rho=3.4)
submodel (bond) B1,B2,B3
node n1,n2

cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
main path PT<InTherm - OutTherm>

parameter Area=0.1
terminal Vd

{ main convection line }
connect InTherm at mC
connect SFS from InTherm to OutTherm

{ reverse conduction effect }
connect B1 from OutTherm to n1
connect B2 from n1 to InTherm
connect B3 from n1 to n2
connect mGS from n2 to InTherm
mGS.vwater = Vd / Area

{ modulation inputs for SFS element }
SFS.Vd = Vd

end


{ ************************************************************************ }
{ @hydseg.dym }
```

```
{ bond graph model of hydraulic segment of a pipe }

model type HydSeg

submodel(I) ind (I=17.) (ic f=0.)
submodel(C) cp (C=8.93E-7) (ic e=0.98E6)
submodel(mRS) res (R=7.5)
submodel(bond) B1,B2,B3,B4
node n1,n2,n3

cut A(e1/f1) , B(e2/-f2) , ThOut(e3/-f3)
main cut C[A B]
main path  PH<A - B>

connect B1 from A to n1
connect B2 from n1 to n2
connect res from n2 to ThOut
connect B3 from n1 to B
connect B4 from n1 to n3
connect ind  at n3
connect B at cp

end


{ ********************************************************************** }
{ @pipeseg.dym }
{ bond graph model of a pipe segment }

model type PipeSeg

submodel HydSeg
submodel CvSeg (Area=0.5)

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

connect HydSeg from InHyd to OutHyd
connect CvSeg from InTherm to OutTherm
connect HydSeg:ThOut at OutTherm

CvSeg.Vd = fh1

end


{ ********************************************************************** }
{ @pipe.dym }
{ bond graph model of a pipe, including thermal and flow effects }

model type Pipe

submodel (PipeSeg) PipeSeg1,PipeSeg2,PipeSeg3,PipeSeg4

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

connect InFluid - PipeSeg1 - PipeSeg2 - PipeSeg3 - PipeSeg4 - OutFluid

end


{ ********************************************************************** }
{ test of submodel Pipe with specific boundary values }
model pipetest
```

```
submodel Pipe
submodel SE
submodel SiF
submodel (mC) exC (C=9.2E3) (ic e=545.)
submodel (SFc) SFCi (s0=7033.,m=3.73,T0=545,rho=3.4)
submodel (SFc) SFCo (s0=7033.,m=3.73,T0=545,rho=3.4)

connect SE at Pipe:InHyd
connect SiF at Pipe:OutHyd
SE.E0 = 0.98E6  { N/m**2 (Pa) }
SiF.F0 = 31.7  { m**3/sec }

connect SFCi at Pipe:InTherm
connect SFCo at Pipe:OutTherm
SFCi.Vd = Pipe.fh1
SFCo.Vd = -Pipe.fh2

connect exC  at Pipe:OutTherm

end
```

C.2 PUMP test program

```
{ ************************************************************************ }
{ @lib.bnd }
{ Bond Graph bond }

model type bond
  cut A (x / y) B (y / -x)
  main cut C [A B]
  main path P <A - B>
end


{ ************************************************************************ }
{ @lib.c }
{ Bond Graph capacitor/compliance }

model type C
  main cut A (e / f)
  parameter C=1.0
  C*der(e) = f
end


{ ************************************************************************ }
{ @lib.mgs }
{ bond graph conductance source for one dimensional cell }

model type mGS
  cut A(e1/f1),B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter b=1.0
  terminal vwater
  local G1,G
  G1 = b / EXP(BOUND(0.,20.,vwater))
  G  = G1/e2
  G*e1  = f1
  f1*e1 = f2*e2
end


{ ************************************************************************ }
{ @lib.mc }
{ Bond Graph modulated capacitor/compliance }
```

```
model type mC
  main cut A (e / f)
  parameter C = 413.
  C/e*der(e) = f
end

{ ************************************************************************ }
{ @lib.sfs }
{ bond graph model for flow source with input SFS }

model type SFS
  cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
  main path P<InTherm - OutTherm>
  terminal Vd
  local s
  parameter s0 = 7033.0 , m = 2.562 , T0 = 545.6 , rho = 3.41
  s = s0 + m*(Ti-T0)
  Sdi = s * rho * Vd
  Ti*Sdi = To*Sdo
end


{ ************************************************************************ }
{ @lib.sfc }
{ bond graph model of a flow source }

model type SFc
  main cut A(Ti/-Sdi)
  parameter s0 = 7033.0 , m = 2.562 , T0 = 545.6 , rho = 3.41
  terminal Vd
  local s
  s = s0 + m*(Ti-T0)
  Sdi = s * rho * Vd
end


{ ************************************************************************ }
{ @lib.SE }
{ Bond Graph effort source }

model type SE
  main cut A (e / .)
  terminal E0
  E0 = e
end


{ ************************************************************************ }
{ @lib.SiF }
{ bond graph model of a flow sink }

model type SiF
  main cut A(./ f)
  terminal F0
  F0=f
end


{ ************************************************************************ }
{ @lib.mrs }
{ Bond Graph of a modulated resistive source }

model type mRS
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter R0 = 10.
  f1 = SQRT(ABS(e1*R0))
  e1*f1 = e2*f2
end
```

```
{ ********************************************************************** }
{ @lib.tf }
{ bond graph model of a transformer }
model type TF
  cut A(e1/f1) B(e2/ -f2)
  main cut C[A B]
  main path P<A - B>
  parameter m=1.0
  e1=m*e2
  f2=m*f1
end


{ ********************************************************************** }
{ @lib.sf }
{ bond graph model of a flow source }

model type SF
  main cut A(./ -f)
  terminal F0
  F0=f
end


{ ********************************************************************** }
{ @lib.rs }
{ bond Graph of a resistive source }

model type RS
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter R=1.0
  R*f1 = e1
  e1*f1 = e2*f2
end


{ ********************************************************************** }
{ @flowsec.dym }
{ fluid flow section of a constant speed motor driven pump }

model type FlowSec

submodel(C)  capout (C=1.2E-11) (ic e=9.79E3)
submodel(mRS) frflow (R0=5.27E-7)
submodel(bond) B1,B2,B3,B4,B5,B6
node n1,n2,n3

cut InHyd(ei / fi) , OutHyd(eo/-fo) , MechIn(em/fm) , ThOut(et/-ft)
main cut C[InHyd OutHyd]
main path P<InHyd - OutHyd>

connect B1 from n1 to MechIn
connect B2 from InHyd to n1
connect B3 from n1 to OutHyd

connect B4 from OutHyd to n2
connect B5 from n2 to InHyd
connect B6 from n2 to n3

connect frflow from n3 to ThOut
connect capout at OutHyd

end


{ ********************************************************************** }
{ @mechsec.dym }
{ mechanical section of a constant speed motor driven pump }
```

```
model type MechSec

submodel(RS) frmech (R=0.001)
submodel(bond) B1,B2,B3
node n1,n2

cut MechOut(em/-fm) , MechIn(ei / fi ) , ThOut(et/-ft)
main cut C[MechIn MechOut]
main path P<MechIn - MechOut>

connect B1 from MechIn to n1
connect B2 from MechOut to n1
connect B3 from n1 to n2
connect frmech from n2 to ThOut

end


{ ********************************************************************** }
{ @cvseg.dym }
{ bond graph model for a convective segment of a pipe }

model type CvSeg

submodel mC (C=104.E3) (ic e=318.)
submodel mGS (b=640.)
submodel SFS (s0=600. , m=12.5 , T0=319. rho=999.1)
submodel (bond) B1,B2,B3
node n1,n2

cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
main path PT<InTherm - OutTherm>

parameter Area=0.1
terminal Vd

{ main convection line }
connect InTherm at mC
connect SFS from InTherm to OutTherm

{ reverse conduction effect }
connect B1 from OutTherm to n1
connect B2 from n1 to InTherm
connect B3 from n1 to n2
connect mGS from n2 to InTherm
mGS.vwater = Vd / Area

{ modulation inputs for SFS element }
SFS.Vd = Vd

end


{ ********************************************************************** }
{ @pump.dym }
{ bond graph model of a pump, including thermal and flow effects }

model type Pump

submodel(SF) sf
submodel(TF) tf (m=1.7E-4)
submodel(MechSec) msec
submodel(FlowSec) fsec
submodel(CvSeg) cseg (Area=0.1)

node n1,n2,n3,n4,n5,n6

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
```

```
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

terminal rpm

connect sf to msec to tf to fsec:MechIn
connect fsec from InHyd to OutHyd
sf.F0 = rpm*2.*3.14

connect cseg from InTherm to OutTherm
connect msec:ThOut at OutTherm
connect fsec:ThOut at OutTherm
cseg.Vd = fh1

end


{ ********************************************************************** }

model pumptest

submodel Pump
submodel (SE) SE
submodel SiF
submodel (SFc) SFCi (s0=600. , m=12.5 , T0=319. , rho=999.1)
submodel (SFc) SFCo (s0=600. , m=12.5 , T0=319. , rho=999.1)
submodel(mC) exC (C=104.E3) (ic e=318.)

connect SE at Pump:InHyd
connect SiF at Pump:OutHyd
SE.E0 = 9.79E3
SiF.F0 = 0.46

connect SFCi at Pump:InTherm
connect SFCo at Pump:OutTherm
SFCi.Vd = Pump.fh1
SFCo.Vd = -Pump.fh2

connect Pump:OutTherm at exC

Pump.rpm = 1500.

end
```

C.3 VALVE test program

```
{ ********************************************************************** }
{ @lib.bnd }
{ bond Graph bond }

model type bond

  cut A (x / y) B (y / -x)
  main cut C [A B]
  main path P <A - B>

end


{ ********************************************************************** }
{ @lib.c }
{ Bond Graph capacitor/compliance }

model type C
  main cut A (e / f)
  parameter C=1.0
  C*der(e) = f
end
```

```
{ ********************************************************************** }
{ @lib.rev }
{ Bond Graph of a modulated resistive source }

model type mRSv
  cut A(e1/f1), B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  terminal y
  parameter R0 = 10.
  local R
  R=R0*ABS(y)**3
  f1 = SQRT(ABS(e1*R))
  e1*f1 = e2*f2
end


{ ********************************************************************** }
{ @lib.mgs }
{ bond graph conductance source for one dimensional cell }

model type mGS
  cut A(e1/f1),B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter b=1.0
  terminal vwater
  local G1,G
  G1 = b / EXP(BOUND(0.,20.,vwater))
  G  = G1/e2
  G*e1  = f1
  f1*e1 = f2*e2
end


{ ********************************************************************** }
{ @lib.mc }
{ Bond Graph modulated capacitor/compliance }

model type mC
  main cut A (e / f)
  parameter C = 413.
  C/e*der(e) = f
end


{ ********************************************************************** }
{ @lib.sfs }
{ bond graph model for flow source with input SFS }

model type SFS
  cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
  main path P<InTherm - OutTherm>
  terminal Vd
  local s
  parameter s0 = 7033.0 , m = 2.562 , T0 = 545.6 , rho = 3.41
  s = s0 + m*(Ti-T0)
  Sdi = s * rho * Vd
  Ti*Sdi = To*Sdo
end


{ ********************************************************************** }
{ @lib.sfc }
{ bond graph model of a flow source }

model type SFc
  main cut A(Ti/-Sdi)
  parameter s0 = 7033.0 , m = 2.562 , T0 = 545.6 , rho = 3.41
```

```
  terminal Vd
  local s
  s = s0 + m*(Ti-T0)
  Sdi = s * rho * Vd
end


{ ********************************************************************** }
{ @lib.SE }
{ Bond Graph effort source }

model type SE
  main cut A (e / .)
  terminal E0
  E0 = e
end


{ ********************************************************************** }
{ @lib.SiF }
{ bond graph model of a flow sink }

model type SiF
  main cut A(./ f)
  terminal F0
  F0=f
end


{ ********************************************************************** }
{ @cvseg.dym }
{ bond graph model for a convective segment of a pipe }

model type CvSeg

submodel mC (C=104.E3) (ic e=319.)
submodel mGS (b=640.)
submodel SFS (s0=600. , m=12.5 , T0=319. rho=999.1)
submodel (bond) B1,B2,B3
node n1,n2

cut InTherm(Ti/Sdi) , OutTherm(To/-Sdo)
main path PT<InTherm - OutTherm>

parameter Area=0.1
terminal Vd,y

{ main convection line }
connect InTherm at mC
connect SFS from InTherm to OutTherm

{ reverse conduction effect }
connect B1 from OutTherm to n1
connect B2 from n1 to InTherm
connect B3 from n1 to n2
connect mGS from n2 to InTherm
mGS.vwater = Vd / (Area*(y+1.0E-10))

{ modulation inputs for SFS element }
SFS.Vd = Vd

end


{ ********************************************************************** }
{ @hydsegv.dym }
{ bond graph model of hydraulic segment of a valve }

model type HydSegV

submodel(C) cp (C=1.2E-11) (ic e=2.48E6)
```

```
submodel(mRSv) vres (R0=5.27E-7)
submodel(bond) B1,B2,B3
node n1,n2

cut A(e1/f1) , B(e2/-f2) , ThOut(e3/-f3)
main cut C[A B]
main path  PH<A - B>

connect B1 from A to n1
connect B2 from n1 to n2
connect vres from n2 to ThOut
connect B3 from n1 to B
connect B at cp

end


{ ********************************************************************** }
{ @valve.dym }
{ bond graph model of a valve }

model type Valve

submodel HydSegV
submodel CvSeg (Area=0.1)

cut InTherm(et1 / ft1) , OutTherm(et2 / -ft2)
cut InHyd(eh1 / fh1) , OutHyd(eh2 / -fh2)
cut InFluid[InHyd InTherm], OutFluid[OutHyd OutTherm]
main cut D[InFluid OutFluid]
main path P<InFluid - OutFluid>

terminal y

connect HydSegV from InHyd to OutHyd
connect CvSeg from InTherm to OutTherm
connect HydSegV:ThOut at OutTherm

CvSeg.Vd = fh1
CvSeg.y = y
HydSegV::vres.y = y

end


{ ********************************************************************** }
{ bond graph model of a valve, including thermal and flow effects }

model valvetest

submodel Valve
submodel SE
submodel SiF
submodel (SFc) SFCi (s0=600. , m=12.5 , T0=319. , rho=999.1)
submodel (SFc) SFCo (s0=600. , m=12.5 , T0=319. , rho=999.1)
submodel (mC) exC (C=104.E3) (ic e=319.)

connect SE at Valve:InHyd
connect SiF at Valve:OutHyd
SE.E0 = 2.48E6
SiF.F0 = 0.46

connect SFCi at Valve:InTherm
connect SFCo at Valve:OutTherm
SFCi.Vd = Valve.fh1
SFCo.Vd = -Valve.fh2

connect Valve:OutTherm at exC

Valve.y = 0.639
```

```
end
```

C.4 DEAR test program

```
{ ********************************************************************* }
@lib.bnd
{ Bond Graph bond }

model type bond
  cut A (x / y) B (y / -x)
  main cut C [A B]
  main path P <A - B>
end


{ ********************************************************************* }
@lib.sf
{ bond graph model of a flow source }

model type SF
  main cut A(./ -f)
  terminal F0
  F0=f
end


{ ********************************************************************* }
@lib.mgs
{bond graph conductance source for one dimensional cell}

model type mGS
  cut A(e1/f1),B(e2/-f2)
  main cut C[A B]
  main path P<A - B>
  parameter a=0.0,b=0.17
  terminal vwater
  local G1,G
  G1=a*vwater+b
  G=G1/e2
  G*e1 = f1
  f1*e1=f2*e2
end


{ ********************************************************************* }
@lib.cfl

{ bond graph model for capacitive field }

model type CFl

cut chem (mu/nu)
cut hyd (pvap/q)
cut therm(T/Sd)

parameter Cth = 75.906    { J/mol K }
parameter Chy = 1.2E-11   { m**5/N }
constant  rho = 1.8E-5    { m**3/mol }

terminal V,pd

constant Tc=373.15,pc=101330.,rr=4883.
local pd,Td,S,mud,n

{thermal capacitance : get Td}
Td = T*Sd/(n*Cth)
der(T) = Td
```

```
{terminal from CFl : get pd}
pd = pd

{Gibbs Duhem equation : get mud}
0 = -pd*V + Td*S + mud*n
der(mu) = mud

{vapor pressure curve : get pvap as theoretical pressure to
 determine equilibrium}
pvap = pc * EXP( rr*(T-Tc)/(T*Tc) )

{accumulation of through variables}
der(S) = Sd
der(n) = nu
V=rho*n        {might be der(V) = q as well}

end


{ ********************************************************************* }
@lib.cfg
{ bond graph model for capacitive field }

model type CFg

cut chem (mu/nu)
cut hyd (p/q)
cut therm(T/Sd)

parameter   Cth   = 36.306  { J/mol K }
constant    R     = 8.314 { J/mol K }
constant    kappa = 1.4    { dimensionless }

terminal V,pd

local pd,Td,S,mud,n,Chy

{thermal capacitance : get Td}
Td = T*Sd/(n*Cth)
der(T) = Td

{hydraulic capacitance : get pd}
Chy = V/(kappa*p)
pd = q/Chy

{Gibbs Duhem equation : get mud}
0 = -pd*V + Td*S + mud*n
der(mu) = mud

{equation of state : get p}
p*V=n*R*T

{related integrators}
der(S) = Sd
der(n) = nu

end


{ ********************************************************************* }
@lib.crt
{ bond graph model for chemical reactor for mass
  transfer reaction}

model type ChRT

cut chem (mu/-nu)
cut hyd (p/-q)
cut therm(T/-Sd)
terminal n
parameter R=8.314
```

```
local k
constant eps=1.E-4 , rho=1.8E-5
constant epst=10. , one=1.0

{reaction rate : get nu}
k = 1. / ( 1. + EXP(BOUND(-20.,20.,mu/1000.+p/1000.))) -0.5
nu = k * n / 50000.

{derivative of equation of state : get q}
q = rho * nu

{Gibbs equation : get Sd}
0 = -p*q + (T+epst*SIGN(one,T))*Sd + mu*nu

end


{ ********************************************************************* }
@hmt.dym
{ bond graph model of the hydraulic mass transfer system
  for systems with phase changes }

model type HMT

submodel(bond) B1,B2,B3,B4,B5,B6

cut flstore(pf/-qf)
cut gastore(pg/-qg)
cut rvap(pv/qv)
cut rcon(pc/qc)
node nc,nv

connect B1 from flstore to nv
connect B2 from nv to gastore
connect B3 from gastore to nc
connect B4 from nc to flstore

connect B5 from rvap to nv
connect B6 from rcon to nc

end


{ ********************************************************************* }
@cmt.dym
{ bond graph model of the chemical mass transfer system
  for systems with phase changes }

model type CMT

submodel(bond) B1,B2,B3,B4,B5,B6

cut flstore(muf/-nuf)
cut gastore(mug/-nug)
cut rvap(muv/nuv)
cut rcon(muc/nuc)
node nc,nv

connect B1 from flstore to nv
connect B2 from nv to gastore
connect B3 from gastore to nc
connect B4 from nc to flstore

connect B5 from rvap to nv
connect B6 from rcon to nc

end


{ ********************************************************************* }
@tmt.dym
```

```
{ bond graph model of the thermic mass transfer system
  for systems with phase changes }

model type TMT

submodel(bond) B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12
submodel(mGS) Gvap ,Gcon

terminal vvap,vcon
cut flstore(Tf/-Sdf)
cut gastore(Tg/-Sdg)
cut rvap(Tv/Sdv)
cut rcon(Tc/Sdc)
node nc,nv,nvc,ncc
node n1,n2,n3,n4

connect B1 from flstore to nv
connect B2 from nv to gastore
connect B3 from gastore to nc
connect B4 from nc to flstore

connect B5 from nv to n1
connect Gvap from n1 to gastore

connect B6 from nc to n2
connect Gcon from n2 to flstore

connect B7 from flstore to nvc
connect B8 from nvc to gastore
connect B9 from gastore to ncc
connect B10 from ncc to flstore

connect B11 from rvap to nvc
connect B12 from rcon to ncc

Gvap.vwater = vvap
Gcon.vwater = vcon

end


{ ********************************************************************* }
@deaer.dym
{bond graph model of an opened feedwater heater (deaerator)}

model type deaer

submodel(HMT) HMT
submodel(TMT) TMT
submodel(CMT) CMT
submodel(CFl) Cl ( ic T=375.,mu=-1583.8 ,S=1271.7 ,n=53.2 )
submodel(CFg) Cg ( ic T=370.,mu=-1019.  ,S=4.6995 ,n=3.5346E-2 )
submodel(ChRT) Cv
submodel(ChRT) Cc
submodel(SF) Sdwater ,Sdgas

parameter Area = 0.001 { m**2 }

{entropy sources}
connect Sdwater at TMT:flstore
connect Sdgas at TMT:gastore

{hydraulic subsystem}
connect HMT:flstore at Cl:hyd
connect HMT:gastore at Cg:hyd
connect HMT:rvap at Cv:hyd
connect HMT:rcon at Cc:hyd

{thermal subsystem}
connect TMT:flstore at Cl:therm
connect TMT:gastore at Cg:therm
```

```
connect TMT:rvap at Cv:therm
connect TMT:rcon at Cc:therm

{chemical subsystem}
connect CMT:flstore at Cl:chem
connect CMT:gastore at Cg:chem
connect CMT:rvap at Cv:chem
connect CMT:rcon at Cc:chem

{modulate transport reactors with molar mass n of attatched storage field}
Cv.n = Cl.n
Cc.n = Cg.n

{ modulate convective resistors }
TMT.vvap = HMT.qv / Area
TMT.vcon = HMT.qf / Area

{ constraint for isochoric model V=const }
Cl.V + Cg.V = 0.002

end


{ ************************************************************** }
@pc.dym

model pressurecooker

submodel deaer

deaer::Sdwater.F0 = 0.
deaer::Sdgas.F0  = 0.

end
```

References

[1] Edward L. Mitchell and Joseph S. Gauthier (1986), *ACSL: Advanced Continuous Simulation Language – User Guide and Reference Manual*, Mitchell & Gauthier Assoc., Concord, MA.

[2] Hilding Elmqvist (1978), *A Structured Model Language for Large Continuous Systems*, Ph.D. Dissertation, Report CODEN:LUTFD2/(TRFT-1015), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[3] Electric Power Research Institute (1984), *Modular Modeling System (MMS): A Code for the Dynamic Simulation of Fossil and Nuclear Power Plants. Volume 1: Theory Manual*, EPRI CS/NP–3016–CCM Volume 1, Electric Power Research Institiute, Palo Alto, CA.

[4] Electric Power Research Institute (1984), *Modular Modeling System (MMS): A Code for the Dynamic Simulation of Fossil and Nuclear Power Plants. Volume 2: Programmer's Manual*, EPRI CS/NP–3016–CCM Volume 3, Electric Power Research Institiute, Palo Alto, CA.

[5] Electric Power Research Institute (1984), *Modular Modeling System (MMS): A Code for the Dynamic Simulation of Fossil and Nuclear Power Plants. Volume 3: ACSL–Based MMS User's Manual*, EPRI CS/NP–3016–CCM Volume 3, Electric Power Research Institiute, Palo Alto, CA.

[6] Henry M. Paynter (1961), *Analysis and Design of Engineering Systems*, M.I.T. Press, Cambridge, MA.

[7] François E. Cellier (1991), *Continuous System Modeling*, Springer-Verlag, N.Y.

[8] Jean U. Thoma (1990), *Simulation by Bond Graphs; Introduction to a graphical method*, Springer-Verlag, N.Y.

[9] P. C. Breedveld et al.,*Bibliography of Bond Graph Theory and Application*, J. Franklin Institute, Vol. 328, No 5/6 1991, pp. 1067 - 1109

[10] Frank P. Incropera and David P. De Witt (1985), *Introduction to Heat Transfer*, John Wiley & Sons, N.Y.

[11] D. C. Karnopp (1972), *Bond graph models for fluid dynamic systems*, Trans. ASME J. Dynamic Syst. Measure. Control, Vol. 94, No. 3, pp. 222 – 229

[12] Raymond C. Binder (1973), *Fluid Mechanics*, Prentice-Hall, N.J.

[13] W. Beitz and K. H. Küttner (1986), *Dubbel. Taschenbuch für den Maschinenbau*, Springer-Verlag, N.Y.

[14] William C. Reynolds (1979), *Thermodynamic properties in SI. Graphs, tables and computational Equations for forty substances*, Department of Mechanical Engineering, Stanford University, Stanford, CA.

[15] Gordon J. Van Wylen and R. E. Sonntag (1986), *Fundamentals of Classic Thermodynamics*, John Wiley & Sons, N.Y.