Mean Value Engine Modeling with Modelica

Daniel Silverlind

Reg nr: LiTH-ISY-EX-3242

6th December 2001

Mean Value Engine Modeling with Modelica

Master thesis

Performed in **Vehicular Systems**,
Dept. of Electrical Engineering
at **Linköpings Universitet**

Performed for **DaimlerChrylser**

by **Daniel Silverlind**

Reg nr: LiTH-ISY-EX-3242

Supervisor: **Thomas Stutte**
DaimlerChrysler
Ylva Nilsson
Linköpings Universitet

Examiner: **Dr. Mattias Nyberg**
Linköpings Universitet

Linköping, 6th December 2001.

Avdelning, Institution
Division, Department

Vehicular Systems,
Dept. of Electrical Engineering

Datum
Date

6th December 2001

Språk
Language
☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category
☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
—

ISRN
—

Serietitel och serienummer **ISSN**
Title of series, numbering —

LITH-ISY-3242

URL för elektronisk version
http://www.fs.isy.liu.se

Titel Medelvärdesmodeller av motorer med Modelica

Title Mean Value Engine Modeling with Modelica

Författare Daniel Silverlind
Author

Sammanfattning
Abstract

This thesis is a study of engine modeling in Modelica. It covers the range of models known as mean value engine models that often are used in development of control algorithms and evaluation of new hardware on the systems-level. The purpose is to create a package, a Modelica library, of basic components suitable for such models. The package is designed for a high flexibility with few rigid rules on their use and should be reusable for a range of applications.

The components are connected to a functioning engine, a four cylinder turbo diesel, which is verified against an already existing model of the same engine implemented in Simulink.

The work has been performed at DaimlerChrysler Research and Technology with assistance from MathCore.

Nyckelord
Keywords Modelica, MathModelica, object-oriented, non-causal, mean value engine model, thermodynamic

Abstract

This thesis is a study of engine modeling in Modelica. It covers the range of models known as mean value engine models that often are used in development of control algorithms and evaluation of new hardware on the systems-level. The purpose is to create a package, a Modelica library, of basic components suitable for such models. The package is designed for a high flexibility with few rigid rules on their use and should be reusable for a range of applications.

The components are connected to a functioning engine, a four cylinder turbo diesel, which is verified against an already existing model of the same engine implemented in Simulink.

The work has been performed at DaimlerChrysler Research and Technology with assistance from MathCore.

Keywords Modelica, MathModelica, object-oriented, non-causal, mean value engine model, thermodynamic

Preface

Thesis outline

The reader with a basic knowledge on engines can skip chapter 2, the reader who has come across engine models might also skip the third chapter and anyone familiar with Modelica should skip chapter 4.

Chapter 1, introduction.
Chapter 2 gives a brief description of automobile engines.
Chapter 3, some principals on engine modeling.
Chapter 4, the simulation tool.
Chapter 5 discuss design of a thermodynamic Modelica library.
Chapter 6 describes implementation of thermodynamic components.
Chapter 7 discuss modeling with the thermodynamic library.
Chapter 8 presents an engine model.
Chapter 9 makes conclusions of engine modeling with Modelica.
Chapter 10 presents ideas of future work.

The appendixes contains most of the Modelica-code from the presented models.

Acknowledgment

First a thank you to Thomas Stutte, DaimlerChrysler Research and Technology, for all the know-how in the function and modeling of turbodiesels, for all the material provided, and for "The Dilbert principle".

Thanks to all the people at MathCore (Jan Brugård, Mats Jirstrand, Andreas Idebrant, Johan Gunnarsson) for their help with Modelica and MathModelica.

Thanks to the people at Vehicular systems (Ylva Nilsson, Mattias Nyberg, Lars Eriksson) for their input along the way.

Finally a thank to all the Swedes and Germans in Esslingen for a great half year!

Contents

Chapter 1

Introduction

Efficient simulations are important in development of control algorithms and evaluation of new hardware in automobile and truck engines. Simulations save the time and expense of building hardware or implementing and testing software in real engines. The current trend is toward simulations of larger systems whereas it was earlier considered sufficient with separate simulations of components and subsystems representing a more narrow area of the physical process.

With the general simulation tools in use today, these complex systems are difficult to survey and maintain. Writing the models is error-prone, and little code can be reused in additional application as the models are valid for only a small range of simulations. Various specialized tools are efficient and powerful in reuse but cover only a single or a few physical domains like hydraulics or electric circuits.

Modelica is a object-oriented, multi-domain language for large physical systems. It overcomes the complexity of the general tools and extends the power of the domain-specific tools to multiple domains. This thesis attempts to produce a Modelica package for a range of engine models, the so called mean value model. The Modelica engine model will have several advantages over other common implementations, e.g. in Matlab. The functionality and appearance of the model and its subsystems will be very similar to those in the real engine. The submodels will be reusable in additional applications of engine modeling. Model maintenance will be effective as structural changes will be simple to perform. The very same model can be used for a wider range of simulations.

The purpose of the thesis is not to achieve the best possible model of any specific engine, although a Mercedes-Benz turbo-diesel is exemplified.

The work has been performed at DaimlerChrylsers Research and Technology facility in Esslingen am Neckar, Germany. The Simulation-

tool MathModelica 2.1 and support have been provided by MathCore AB.

Chapter 2

Internal combustion engines

An engine consumes fresh air and fuel and produces exhausts, usable work and heat. In an automobile this process takes place in usually four to eight cylinders. A number of systems that support and control the combustion are located around the cylinders, e.g. fuel-pumps, throttles, compressors and coolers. This chapter gives a brief description of these systems and of the basic functionality of an engine.

2.1 Cylinders

Air and fuel flows into the cylinders, the mixture reacts and usable energy is extracted from the heated gas which is then expelled. In a spark ignited (SI) engine the air is mixed with fuel before entering the cylinders. In a combustion ignited engine, or diesel, fuel is injected directly into the cylinders and ignites spontaneously due to the high pressure.

There are a hundred (!) aspects of the combustion process that anyone interested can study (see [1]), this chapter explains only a few aspects with bearing on this work.

2.1.1 Physical appearance

The cylinder is continuously swept by a piston which is connected to the crankshaft via a rod. The top of the cylinder houses intake and exhaust ports, a spark-plug in spark ignited engines and an injector in diesel engines.

2.1.2 Engine cycle

This section describes the diesel cycle, an SI-engine works in a similar manner.

Simplified, the engine works in four discrete strokes. In the first stroke fresh air is inhaled through the intake ports as the piston moves down, in the second stroke all valves are closed while the piston moves up and compresses the air. At the peak of compression fuel is injected and the expansion stroke starts, now the usable energy is extracted. In the fourth stroke the exhausts are ejected through the exhaust ports and the cycle is completed. Thus the crankshaft makes two revolutions to complete a single cycle.

2.1.3 Combustion efficiency

Abbreviated η, the combustion efficiency is a number between zero and one describing the portion of the fuels chemical energy that is transformed to usable mechanical work in the crankshaft. The efficiency varies with load and engine speed, common values range from 0.3 to 0.4 for diesel engines.

$$\dot{W}_{crank} = \eta \dot{m}_{fuel} Q_{HV}$$

2.1.4 Volumetric efficiency

A cylinder works as a pump. An ideal cylinder would inhale a mass of gas equal to its displacement volume multiplied with the inlet manifold (see section 2.2) density in each cycle. Due to resistance in the intake ports, back-flow through the ports and other effects the inhaled mass will typically be reduced which is expressed in the volumetric efficiency η_{vol}. The following equation express the flow through the cylinders in each cycle.

$$m_{cyl} = \eta_{vol} \rho_{inlet} V_{disp}$$

η_{vol} is usually higher than 0.8 and can in fact exceed 1.0.

2.1.5 Pumping losses

In an internal combustion engine the pressure on the intake side will normally be lower than on the exhaust side. Pumping gas from low to high pressure costs energy and this energy is taken from the crankshaft. The effect is largest in SI-engines on low load.

2.2 Manifolds

The cylinders are flanged on both sides by tubes called inlet and exhaust manifolds. There can be one or many of each manifold in an engine, maximum one of each per cylinder. In the following, multiple manifolds will be lumped together and referred to as a single inlet and a single exhaust manifold.

In SI-engines the inlet manifold pressure is reduced by a throttle in order to control the output torque. In diesel engines output torque is controlled via the fuel-injection and the inlet pressure is usually kept near or above ambient pressure. This explains the higher pumping losses in SI-engines.

2.3 Turbo-charger

Super-charging means raising the density in the inlet manifold by pumping air into it with a compressor. With a high density, more air can flow through the cylinders giving the engine a higher potential output of power. To further increase the density some of the heat that arises from the compression can be removed in an inter-cooler. This describes super-charging in general.

In a turbo-charger the power consumed by the compressor is provided by a turbine located directly after the exhaust manifold. Turbine and compressor are connected via a short shaft and works with a high rotational speed, ranging typically from 10 000 to 180 000 rev/min.

To control the turbo-charger speed, a valve is often placed so that it can by-pass the flow through the turbine and thus reduce its power output, a system called a waste-gate. Another design is controllable vanes prior to the turbine rotor called variable nozzle turbine or VNT. With heavily choked vanes the speed of the gases entering the turbine rotor will increase and the turbine will extract more energy, but the exhaust manifold pressure will build up making the engine pumping more power-consuming. While a waste-gate only can protect against harmful over-revving and over-charging, the VNT can be used in more intelligent control strategies, for example to accelerate the turbo to match an increased demand of torque.

2.4 Exhaust Gas Recirculation, EGR

To reduce harmful emissions some of the exhaust can be diverted back into the combustion process. There are two ways in which the recirculation can occur.

One is called internal EGR and refers to exhausts that remains in the cylinders after the exhaust-stroke or blows into the inlet manifold

when the intake-ports are opened. By controlling the timing of the ports the internal EGR can be controlled, but some recirculation is always present.

The second method is to connect the inlet and exhaust manifolds with a pipe and control the flow with a valve. In order to reduce the formation of nitrogen-oxides the EGR-system often include a cooler.

The rate of recirculation is typically less than 25 percent of the total amount of exhausts in SI-engines and up to 50 percent in diesels.

2.5 Gas path

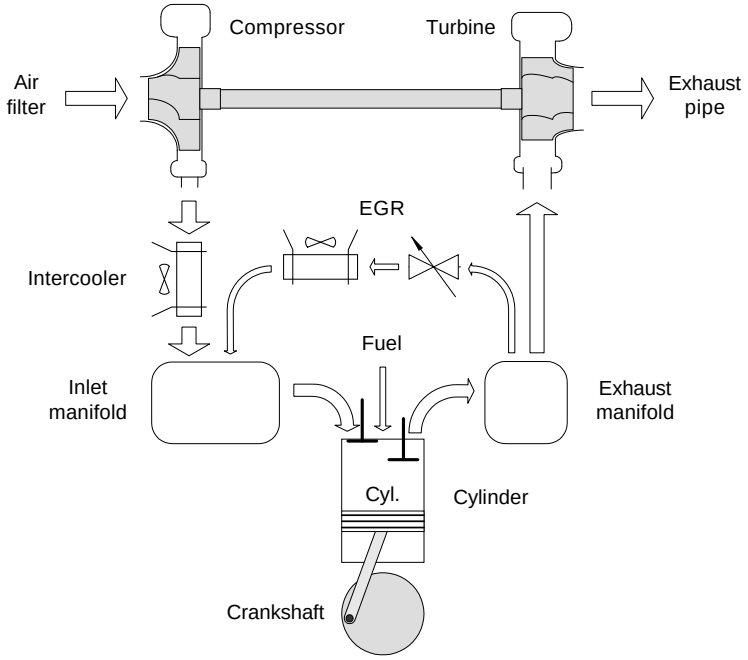

Figure 2.1: Gas path through a typical modern diesel engine.

With the major components and subsystems described, the paths of the gas can be constructed for a common type of engines. Via the arrows in figure 2.1 its typical direction can be followed.

Air first passes an air-filter, enters the compressor, the inter-cooler and the inlet manifold. It flows through the intake ports into the cylinder where it reacts with fuel. From the cylinders the gas flows via the exhaust manifold to the turbine and then via the exhaust system with

mufflers, catalysts etc out into the atmosphere. There can be paths parallel to the main path such as the EGR-system and ventilation pipes from crank-house, top-lock, filters etc.

2.6 Exhaust gases

In the SI-cycle all the air passing the cylinder takes part in the combustion leaving, ideally, neither oxygen nor fuel. To achieve this the fuel and air must be mixed in exact proportions according to the stoichiometry of the reaction.

The diesel cycle on the other hand works with an excess of air, so called lean burn. Depending on the conditions in the combustion process, a diesel engine can produce gases of quite various chemical compositions.

In the following the gas leaving the cylinder will be divided into fresh air, i.e. gas with ambient mixture, and exhausts, i.e. gases considered to have taken an active part in the combustion. The purpose is to be able to trace oxygen and other gases through the system, especially the mixing of fresh air and exhausts in the inlet manifold.

Chapter 3

Mean value engine models

Mean value engine models (MVEMs) attempts to capture dynamics in a time-scale spanning over several combustion cycles, or a few tenths of a second and longer. Faster events are not of interest other than their effects on a larger scale. The pulsating flow through the inlet port is modeled by its average value over a cycle, for example. The speed and torque output of the engine, dynamics of the turbo-charger and the pressure build-up in the inlet and exhaust manifolds are the aspects of most interest in MVEMs.

As a contrast, the in-cycle engine model is designed to capture details in the combustion-cycle like the pressure inside the cylinders over the strokes and effects of different valve-timings.

Modeling principles and approximations will vary between models depending on their intended use. Some of the common approximations and some of the choices made in this thesis are discussed in this chapter. More on MVEMs can be found in [2].

3.1 Static flow

Momentum of gas is of importance when modeling very rapid dynamics like sound waves propagating through a system. These dynamics are to fast and detailed for mean value models. Thus momentum can be left out, saving some variables and states.

3.2 One dimensional flow

Flow in two or three dimensions is of interest when modeling effects like mixing of gases and turbulence. This is too detailed for a MVEM in which distributions of mixture, temperature, velocity, etc are assumed uniform in a flow. Any aspect of the flow can therefore be described by a single value, e.g. a single transfer rate, as if it took place in a one-dimensional space.

3.3 Discretizised models

Flow through tubes, valves, etc are typically discretizised to points of stagnation and points of flow in order to simplify the mathematical representation. A tube with a narrow cross-section is not modeled with a continuous pressure drop but as a row of restrictions and volumes. Models of compressors, turbines and coolers are modeled in the same manner with pointwise changes of pressure and temperature. If the flow can be considered one-dimensional, the model components can be regarded zero-dimensional.

3.4 Bidirectional flow

Gas flows in an engine are essentially not only one-dimensional but also one-way. The engine inhales fresh air and exhales rest products at the other end of the line. Flow in the wrong direction is rare but can happen, especially in an EGR-system during highly dynamic events when the inlet pressure for a brief moment is higher than the exhaust pressure and the EGR-valve is open (the ambition is to close the valve at transients, but the valve is often not very fast). In order to catch these rare but not unimportant events the model could support flow in two directions, at least in some subsystems. This requirement is not standard, in many applications unidirectional flow is sufficient. The models in this work will however support bidirectional flow.

3.5 Single gas or mixture

Modeling a one-component gas is often sufficient in MVEMs, perhaps with different gases in different subsystems. There are however situations which require simulation of a mixture, for example when tracing the exhausts flowing into the inlet manifold through the EGR-system. Support for mixtures is not self-evident in MVEMs, but is included in this thesis.

3.6 Ideal gas law

Physical properties of the gases such as specific heats are often considered constant with respect to temperature. This can seem quite a rough simplification as the range of temperatures in an engine can be about 300-1200 K, but since the range is much more narrow in each subsystem the properties can be set to locally typical values.

The properties are further often modeled as constant with respect to the mixture of gases, which depending on the application also can have a narrow range locally.

In this work the temperature dependency is left out but each gas component is assigned its own physical properties giving average values dependent of the composition of gases.

3.7 Spontaneous diffusion

Two entities of gas in diffusive contact will mix until their temperatures and mixtures reaches equilibrium in $lim_{t\to\infty}$. This holds even if no net flow of gas between the entities occur.

Such spontaneous diffusion between entities is typically not modeled in MVEMs since this process is slow compared with the rate of gas exchange through the engine. Thus, the only way two volumes of gas can mix in the model is by a net flow from one volume to the other. Inside a volume mixing is considered instant and total though.

Chapter 4

Modeling and simulation tool

This chapter gives a brief introduction to the modeling-language Modelica and a presentation of the modeling and simulation tool MathModelica.

4.1 Modelica

Modelica is a object-oriented, non-causal language for modeling of large, multi-domain systems. It is an international effort of accomplishing a De-facto standard language for physical modeling. Modelica is declarative, it lets the user write equations in their general, textbook form. The Modelica-tool solves for individual variables and decides the computational order, i.e. the causality.

Modelica models are typically built from classes representing limited subsystems. Connected classes form more complex ones in a model hierarchy with each level representing a larger area of the physical process. In the automobile example, the car is a class containing the engine which in turn include the classes cooler, valve, cylinder, etc. Classes may inherit and extend each other in a perpendicular class hierarchy.

Modelica lets the user construct libraries of classes and includes a number of standard libraries like the analog electrical and mechanical libraries. With a set of such libraries defined, modeling will appear much like systems-construction in reality, choosing classes from the library like choosing components from ELFA[1].

A key issue is re-usability, two similar subsystems should be modeled only once as a generic class. Re-usability is further facilitated by the

[1] A well-known Swedish catalogue of electrical components.

variable causality of the classes; since a model can be simulated with various directions of signal flow it can be used in a more flexible manner, e.g. a DC-motor model can be used either as an engine or as a generator much like the real motor.

A powerful feature of Modelica is the restricted class `connector` which may declare a set of variables but no equations or algorithms. There are other restricted classes like `model` and `type`. The connector specifies the communication between classes and will determine much of the design and modeling capabilities. A suitable connector is typically the first thing that is defined when building a model or a library. Transitions between domains are usually accomplished by including different connectors in a class, e.g. a DC-motor would include connectors from the electrical and rotational-mechanical domains. The following Modelica-code defines an electrical connector with voltage potential and current.

```
connector Pin          ''Electric interface''
   Real v;                 ''Voltage''
   flow Real i;            ''Current''
end Pin;
```

A connect-statement with two connectors as arguments automatically creates equations which can be seen as a communication channel between components. Other means of communication exist, via global variables and between model levels, but the connections are the standard interfaces. If a and b are electrical pins with variables current i and voltage potential v, the statement `connect(a,b)` will create equations corresponding to Kirchhoff's laws at a junction.

$$a.v = b.v;$$
$$a.i + b.i = 0;$$

The key-word `flow` in Pin states that i obeys a sum-to-zero relationship.

With Pin available a simple component, a resistor, can be defined. $Resistor$ have the two interfaces n and p, a parameter and two equations.

```
model Resistor         ''Resistor with two interfaces''
   Pin p;
   Pin n;
   parameter Real R;       ''Resistance''
equation
   p.i+n.i=0;
   p.v-n.v=R*p.i;          ''Ohm's law''
end Resistor;
```

With two additional components, a voltage source and a ground, the simple electrical circuit in figure 4.1 can be assembled. The definitions of the ground and source are as straight forward as the resistor and are not shown here. Some model parameter are set at instantiation in *SimpleCircuit*; the resistors are of 1 and 10 kΩ, the sinus frequency 50 Hz, etc.

```
model SimpleCircuit   ''Circuit with four componets''
   SinSource S(f=50, A=320);
   Resistor R1(R=10^3);
   Ground G(V=0);
   Resistor R2(R=10*10^3);
equation
   connect(S.p, R1.p);     ''S connected to R1''
   connect(R1.n, G.p);
   connect(R1.n, R2.p);
   connect(R2.n, S.n);
end SimpleCircuit;
```

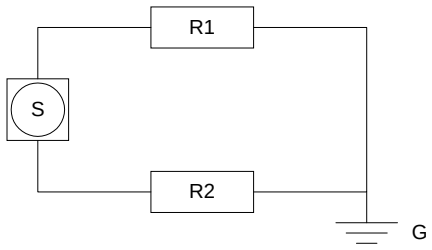

Figure 4.1: Simple circuit.

Further information on Modelica is found in [3] and [4].

4.2 Mathematica

Mathematica is a tool for numerical and analytical computations. It uses a notebook environment for all activities such as programming, graphics, input, etc. The strength of Mathematica is the structured environment and its ability of symbolical manipulations.

4.3 MathModelica

MathModelica from MathCore is an environment for textual and graphical modeling with Modelica. It uses Mathematica notebooks for modeling, simulations, plots, ordinary mathematical computations and documentation. The standard Modelica syntax as well as a special syntax utilizing some of the notebooks typesetting capabilities can be used, the circuit example above is written in the Modelica syntax. MathModelica also includes a graphical modeling environment based on MS Visio which can be imported into the notebooks. The Dymola [5] kernel is used for parsing the Modelica-code to C, for compilation and simulation.

More on MathModelica can be found in [6].

Chapter 5

Implementation of a thermodynamic library

This chapter discuss principles, ideas and previously identified problems associated with implementing a Modelica library suitable for MVEMs. The thermodynamic connector and the super-class sets the basic design of the library and are described here, the complete components are described in the next chapter.

5.1 Strategy

The object-oriented Modelica supports packaging of knowledge in basic components with maintained appearances and semantics as in the physical reality. In thermodynamics it is common to use idealized models such as reservoirs and volumes, and it seems reasonable to choose these as the basic components of a MVEM. A set of such components are defined in the library in chapter 6.

A thermodynamic library already exists; the ThermoFlow library [7]. ThermoFlow is preliminary and very generic, it should be seen as a base on which libraries can be built rather than a complete, usable library itself. Due to its generality it is quite complex, e.g. its connector contains eight variables whereas this work discuss the need of three or four variables. The complexity of ThermoFlow and the fact that it is preliminary makes it seem more efficient to define a new library for the MVEM application.

The classes in the self-designed library are designed with a generic number of gas-components and connectors, though most components will be restricted to two connectors. The design is such that it should be possible to connect any two classes directly to each other with maintained physical semantics. There are some limitations to this set by

the numerical and modeling characteristics described in section 7.1. No general laws that limits the use of the classes are however written into the design, like consecutive volumes and flow components as in ThermoFlow.

Physical constants like specific heats could be placed at a high level in the model hierarchy giving all components access to them. Instead the constants are parameterized in each component which opens the possibility to use different physical properties in different subsystems. For example the intake and exhaust systems could each be modeled with a single component gas while maintaining the different physical behavior of exhausts and fresh air. This would be especially applicable in SI-engines without EGR and would save a few states. For the same reason it is not recommended to include the constants in the connectors.

Even though bidirectional flow is of interest only in some subsystems (EGR) in an engine, all classes will support this in order to keep the components as general as possible. The intention has been to construct components for general internal combustion engines although some adaptations to the OM611-model (see chapter 8) could not be avoided, in particular in the turbo-charger, combustion chamber and coolers.

5.2 Previously identified problems

5.2.1 Square roots

Thermodynamic models are often very non-linear with square-roots describing flow through restrictions, compressors, etc. Consider equation (5.1) describing flow of an incompressible fluid through a restriction, $\dot{m}$ is mass-flow, p_a and p_b are pressures at interface a and b. Previous work [8] include discussions on how to implement this relationship.

$$\dot{m}^2 r T_a = p_a(p_a - p_b) \tag{5.1}$$

The squared representation can not be used since, given the causality pressures to mass-flow, two solutions to equation (5.1) exist and forcing the solver to choose the physical correct one is not possible.

A square-root expression with $\dot{m}$ on the right-hand side should not be used either. Square (and other) roots are generally a problem for numerical solvers when the solution is close to zero since their derivatives becomes very large. But even when the solution is not close to zero the solver might start the iteration there, causing it to abort. Instead modified roots can be used, functions which are actual roots at large argument values and take the value of some other function close to zero. The modified roots used in this work utilizes second or third degree polynomials when evaluated near zero, see *LinearRoot* (n:th

roots) and *LinearRootS* (square roots only) in appendix A for implementation. Alternative fixes includes an interpolating table of the square-root at discrete points, or functions resembling roots in some interval.

Further discussions involve potential benefits of using $\dot{m}|\dot{m}|$ on the left-hand side of equation (5.1) to enforce a certain direction of mass-flow. There seems to be no logical reason why this representation should not give the same problems as the square-root, and without the simple fix of modified roots. It might be that the Dymola solver happens to treat this representation in a better way.

5.2.2 If without else

An implication of bidirectional flow in combination with certain connectors.

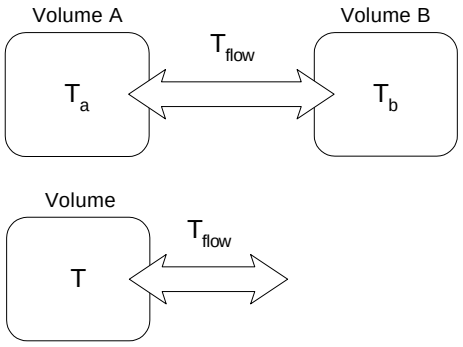

Figure 5.1: Gas exchange between volumes.

Imagine two volumes exchanging gas as in the top of figure 5.1. Unless identical properties are assumed in both volumes some information about the exchanged gas is needed, here this is the temperature T_{flow}. The temperature of the exchanged gas switches with the direction of the flow and this must be expressed via an event.

```
if flow A ⇒ B then
    T_flow = T_a;
else
    T_flow = T_b;
end if;
```

In the object-oriented approach such basic equations should be hided inside the classes but neither volumes has access to the tem-

perature in the other volume. The name-spaces are protected and the only mean of communication is via the connectors. (As said in 4.1 other methods exist, but these are not recommendable.) Since only the information inside a class should be used, the equations should be written for the case in the bottom scheme of figure 5.1 and only the first clause of the statement above could be expressed.

```
if flow out then
    T_flow = T;
end if;
```

Such a half if-statement is not allowed since it represents an equation only under some conditions and could therefore, as far as the solver knows, produce a system with a varying number of equations but a constant number of variables. But since outflow from one volume always means inflow to the other, there is always exactly one active equation. The solution by MathCore is to identify those half if-statements with equal but opposite condition-clauses and combine them before simulation. This function is preliminary and does not yet work with arrays.

Note that non-dynamic components with strictly two interfaces do not need half if-statements. This includes restrictions, valves, compressors, turbines, coolers and other components with equations like $\sum \dot{m} = 0$. The combination routine is able to identify opposite condition clauses through these components, i.e. it solves and reduces the mass-flow equations until the condition-clauses of the statements can be directly compared.

5.3 Connectors

In the thermodynamic domain, one can think of a connector as a cut through a flow (hence named *FlowCut*). The connector variables represent the conditions of the gas it cuts through. Its use in a model is as a communication channel between classes, the connector variables are therefore worth choosing with care since they determine much of the modeling capabilities. An alternative view of the connector is discussed in [9].

Three different sets of the variables pressure p, temperature T, mass-flow $\dot{m}$ and enthalpy-flow $\dot{H}$ have been discussed in previous work, see [8]. These are the $\{p, \dot{m}, \dot{H}\}$, $\{p, T, \dot{m}\}$ and $\{p, T, \dot{m}, \dot{H}\}$ connectors. Using the first two connectors require half if-statements for support of bidirectional flow whereas the third supports this due to the redundancy $\dot{H} = c_p T \dot{m}$. The statement below could control the flow in the bottom scheme of figure 5.1 if implemented with $\{p, T, \dot{m}, \dot{H}\}$.

```
if flow out then
    T_flow = T;
else
    Ḣ_flow = c_p ṁ T_flow;
end if;
```

Assuming access to half if-statements, the choice between $\{p, \dot{m}, \dot{H}\}$ and $\{p, T, \dot{m}\}$ will be shown to be a matter mainly of personal preferences. Many sets of three variables will function just as well, but at least one sum-to-zero (prefix flow) and one equality variable is needed.

One argument for $\{p, \dot{m}, \dot{H}\}$ have been the ability of using a connector in more than one connection in order to construct three-way joints. Figure 5.2 shows four combinations of volumes, restriction and valve. The black squares represent those thermodynamic connectors associated with half if-statements, i.e. only the connectors in the volumes are shown.

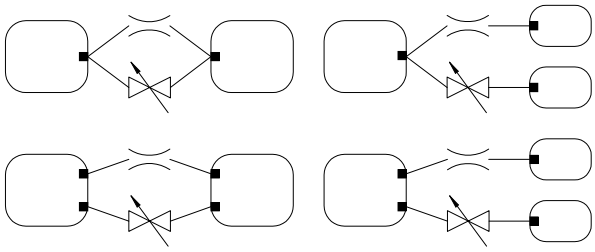

Figure 5.2: Combinations of volumes, restriction and valve.

The first combination would not be solvable with $\{p, \dot{m}, T\}$ since the restriction declares equal temperature in the two connectors while the valve might declare them different. With $\{p, \dot{m}, \dot{H}\}$ this would not be a problem, but the upper right combination would be impossible also with this connector because it would declare three half if-statements. Three half if-statements can not be combined to an integer number of complete statements. This is not simply a problem for the combination routine, it would be logically troublesome as either one or two condition-clauses would be evaluated true at any time giving a variable number of equations. Even with $\{p, \dot{m}, \dot{H}\}$ the general rule would be to use a connector in maximum one connection and to include additional connectors instead. This rule involves neither extra work nor additional code and is a less error-prone usage of the connectors. Therefore the argument for $\{p, \dot{m}, \dot{H}\}$ falls at least partly. The two combinations in the bottom of figure 5.2 are better alternatives to the two upper regardless of the connector in use. Observe that the two combinations to the

right are not entirely equivalent, if the left-hand volume were replaced by a junction-class they would however be mathematically identical. A junction can be pictured as a volume-less volume, i.e. instant mixing but no buffering of mass or energy.

Left to consider in the connector choice is only the convenience of using the connector when modeling components. The $\{p, \dot{m}, \dot{H}\}$-connector makes dynamic components such as volumes neat and comprehensible while the $\{p, T, \dot{m}\}$ looks better with non-dynamic ones such as valves, compressors and restrictions. These components dominate in the library and the choice is therefore the variables $\{p, T, \dot{m}\}$ where $\dot{m}$ is a vector in order to support flow of multi-component gas.

An MVEM would also include components of other domains like rotational mechanics and signals, requiring additional connector classes. These are quite standard, like the rotational-mechanical $Flange$ and the electrical Pin, and there is no need for a description here. Only the $EnvCut$-connector is non-trivial, designed to communicate properties of the atmosphere and surrounding environment to components in thermal or diffusive contact with it.

The definitions of the connectors can be found in appendix B.

5.4 Super-classes

$Base$ is a partial model, a base on which to construct components. It includes thermodynamic connectors and physical properties of the gas and it specifies help-variables with simplified notations. All thermodynamic components will extend $Base$.

The number of gases modeled and the number of thermodynamic connectors are parameterized via the two integers $Ngas$ and $Ncon$ and can thereby be chosen at instantiation. The rest of the variables and parameters, except π, are non-scalar. The mass-flow variable $\dot{m}$ is a $Ngas * Ncon$ matrix representing flow of a $Ngas$-component gas through $Ncon$ interfaces. The variables p and T are $Ncon$-dimension vectors of pressures and temperatures at the connectors. The physical properties c_p, c_v, R and γ are $Ngas$-dimension parameters.

Due to the number of components with strictly two connectors a subclass with two connectors, $TwoPin$, is defined with further simplified notations. E.g. the temperature in connector 1 is available in the variable T_a and the mass-flow in connector 2 in $\dot{m}_b$.

Additional partial models could be imagined, e.g. a superclass with zero volume, or no pressure drop, or no enthalpy-change, etc. A component could extend several of these classes (multiple inheritance is supported), but there is a limit to the number of partial classes that makes life more easy. Having just two partial models seem to be the right compromise here. The implementations are found in appendix B.

Chapter 6

Thermodynamic components

The intention of this chapter is to give a view of the basic functionalities of the thermodynamic components in the MVEM library. The full implementations are found in appendix B.

Most components are described with a few sample equations. Flow is always positive in the direction into a component in these equations which differs somewhat from common thermodynamic conventions where work is often positive in the outward direction. In classes with strictly two thermodynamic connectors flow is assumed from connector a to b (subscripts). The variable $\dot{m}_a$ denotes the mass-flow in connector a, T_a the temperature in a, etc. The if-statements expressing bidirectional flow, vector expressions and other detailed notations are left out. Due to the problems discussed in section 5.2.1, square-roots are always implemented with modified root-functions.

Component	Connectors	Characteristics
Reservoir	n *FlowCut*s, (*EnvCut*)	Static, source
CtrlVol	n *FlowCut*s	Dynamic
Restriction	2 *FlowCut*s	Static
Valve	2 *FlowCut*s, *Pin*	Static
Cooler	2 *FlowCut*s, (*EnvCut*)	Static, source
Compressor	2 *FlowCut*s, *Flange*	Static
Turbine	2 *FlowCut*s, *Flange*	Static
Cylinder	2 *FlowCut*s, *Flange*, *Pin*	Static, source

Figure 6.1: The thermodynamic components with interfaces. Referenced connectors within parenthesis.

Figure 6.1 summarizes the thermodynamic components with some features. A few groupings based on fundamental differences in functionality can be made. All components except *Volume* and *Reservoir* have strictly two thermodynamic connectors and are considered volume-less, i.e. no buffering of gas. All components except the cooler and combustion chamber are modeled as perfectly insulated.

6.1 Infinite reservoir

In thermodynamics it is common to consider very large reservoirs. A small system can interact with such a reservoir without changing its temperature, pressure or other properties.

The purpose of *Reservoir* is to terminate one or more gas-paths and thereby setting boundary conditions to the model. Since the reservoir is a source of mass the temperature and mixture of the flow should be governed by half if-statements according to section 5.2.2.

Pressure, temperature and gas composition in the reservoir are either parameters or taken from a reference to an *EnvCut*-connector. If the *EnvCut* is connected to a model of the outside world, introducing time or distance dependent ambient conditions in a drive-cycle simulation is simplified.

6.2 Control volume

Another standard model in thermodynamics is the control volume. It is idealized to model mixing of in-flowing gases as instant giving uniform pressure and temperature throughout the volume. The control volume is a buffer of mass and energy.

Control volumes can be modeled with different approximations. If temperature and mixture can be considered constant a one-state volume is sufficient. If an energy balance is required another state must be added and modeling a dynamic mixture adds one state per extra gas-component.

The volume is named *CtrlVol* in appendix B and is implemented with mass and energy dynamics. In a common simulation the states will be U, total stored energy, and the elements of M, the vector of stored masses. To be precise, the solver chooses the states and it might reduce one of them, for example if the pressure is parameter bound. Pressure, temperature and gas composition are non-state variables.

Simplified, the equations could look like this, imagine a one-component

gas and a single connector a:

$$\begin{aligned} \dot{M} &= \dot{m}_a \\ \dot{U} &= \dot{m}_a c_p T_a \\ U &= M c_v T \\ pV &= MRT \\ p &= p_a \end{aligned}$$

Half if-statements should govern the temperature and mixture of the gases entering and leaving the volume.

```
if ṁ_a < 0 then
    T_a = T;
end if;
```

6.3 Restriction

A flow restriction can be pictured as a slightly choked tube. The pressure difference across the restriction is small enough to consider the density equal on both sides, thus modeling an incompressible fluid. The mass flow to pressure ratio relationship is described as

$$\dot{m}_a \sqrt{rT_a} = \sqrt{p_a(p_a - p_b)}$$

where r is a discharge coefficient describing the physical appearance of the restriction such as effective area and friction. See [10] for deduction of the relation. The restriction for incompressible flow is called *RestrictionI* in appendix B.

If large pressure differences can be expected the restriction should be modeled assuming compressible fluids as described in section 6.4. The restriction for compressible fluids, *RestrictionC*, is the same as the valve but with constant opening. *RestrictionC* should be used whenever high pressure ratios is expected to occur during a simulation, it is thus the more general of the restriction models. It also seems that the Dymola solver handles the equations for compressible flow better than those in *RestrictionI*.

6.4 Valve

Flow through a valve should not be idealized to an incompressible medium, its use in systems implies that the pressure drop across the valve can be large.

The flow is modeled as equation (6.1) for pressure ratios larger than the critical ratio where the velocity of sound is reached, relation (6.2). The critical ratio is used in the place of the actual ratio in equation (6.1) when the pressure difference is too large, under so called choked flow. The temperature is unchanged across the valve since only stagnation pressures and temperatures are considered. Deductions of the following relations can be found in [10].

$$\frac{\dot{m}_a\sqrt{T_a}}{p_a} = A\sqrt{\frac{1}{R}\frac{2\gamma}{\gamma-1}\left(\left(\frac{p_b}{p_a}\right)^{\frac{2}{\gamma}} - \left(\frac{p_b}{p_a}\right)^{\frac{\gamma+1}{\gamma}}\right)} \quad (6.1)$$

$$\left(\frac{p_b}{p_a}\right)_{critical} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma}} \quad (6.2)$$

A signal connector controls the valve opening via A, which represent effective area lumped with a discharge coefficient. The signal/effective area relation is parameterized via a one-dimensional interpolating table.

6.5 Cooler

The cooler extracts heat from the gas by setting it in thermal contact with a cooling medium. The efficiency η is a measure of how effective the thermal contact is and could be compared with a heat transfer coefficient. The equation below governs the temperature difference between the gas flowing in and the gas flowing out via η and the temperature T_{medium} of the cooling medium.

$$T_b = T_a - \eta(T_a - T_{medium})$$

In *Cooler* the efficiency is determined by the flow through the cooler and the speed of the cooling medium via a two-dimensional map. The temperature and speed of the cooling medium is either parameterized in *Cooler* or taken from a reference to a *EnvCut*-connector. The latter can be used in order to support varying air-speed and ambient temperature in a drive-cycle simulation and is the only cooler defined in appendix B.

The cooler model has no volume and no states, neither is there any pressure drop across the cooler. *Cooler* is a energy-source.

6.6 Compressor

Two very common equations in compressor models, derivation in [1]:

$$M\omega = \dot{m}_a c_p (T_b - T_a) \tag{6.3}$$

$$\frac{T_b}{T_a} = 1 + \frac{1}{\eta}\left(\left(\frac{p_b}{p_a}\right)^{\frac{\gamma-1}{\gamma}} - 1\right) \tag{6.4}$$

Equation (6.3) states that the enthalpy increase in the gas equals the power transfer through the turbo-axle (torque multiplied with speed). In (6.4) the isentropic efficiency η tells us that the energy input is not an ideal compression, there is friction and other effects in the compressor heating the gas more than an ideal compression would.

The compressor needs two relations that can not be derived from basic physics and must instead be chosen to fit measurements. These two black-boxes can be chosen with some freedom, three approaches have been tested in this work. The first is fitted polynomials for real and isentropic enthalpy change (and thereby isentropic efficiency). In the second attempt a polynomial of turbo speed and pressure ratio governs mass-flow while a polynomial of mass-flow and pressure ratio determines the compressor efficiency. The third approach uses an interpolating table with speed and pressure ratio inputs for mass-flow while a table of mass-flow and pressure ratio governs the efficiency.

The two compressor models in appendix B, $CompressorS$ and $CompressorR$, includes the two last black-box approaches. $CompressorR$ is a compressor lumped with two restrictions and therefore includes a lookup table of pressure ratio across the compressor, the remaining pressure difference is considered to occur across the restrictions.

6.7 Turbine

The equations (6.5) and (6.6) are identical to those in section 6.6 except for the inverse efficiency. See [1] for deduction.

$$M\omega = \dot{m}_a c_p (T_b - T_a) \tag{6.5}$$

$$\frac{T_b}{T_a} = 1 + \eta\left(\left(\frac{p_b}{p_a}\right)^{\frac{\gamma-1}{\gamma}} - 1\right) \tag{6.6}$$

The power $M\omega$ should normally be negative since energy is extracted from the gas, T_b will be lower than T_a. Neither the turbines nor the compressors includes any inertia and should therefore be connected to some rotational mechanical component like a stiff inertial axle.

Since the turbine model cannot be derived entirely from basic physics it needs two black boxes or fitted parameters. Three attempts have

been made, all modeling turbines equipped with variable vanes (see section 2.3).

The first attempt models the flow as being through a restriction (compressible fluid) corrected with the two terms A and K in equation (6.7), idea from [11]. It has been found that the two terms are well described by the two low-degree polynomials in equation (6.8)- (6.9), where X is the vane position. Efficiency is modeled with a polynomial of turbo speed and blade speed ratio UC_s, i.e. the ratio between the speeds of the rotor blade tips and the speed of the exhausts entering the rotor.

$$\frac{\dot{m}_a\sqrt{T_a}}{p_a} = A\sqrt{\frac{2\gamma}{R(\gamma-1)}\left(\left(\frac{p_b}{p_a}\right)^{\frac{2}{\gamma}}-\left(\frac{p_b}{p_a}\right)^{\frac{\gamma+1}{\gamma}}\right)}\left(k_0\frac{p_b}{p_a}\right)^K \quad (6.7)$$

$$A = k_1X + k_2 \quad (6.8)$$

$$K = k_3X^2 + k_4X + k_5 \quad (6.9)$$

In the second attempt efficiency is obtained by interpolating between five polynomials of turbo speed and blade speed ratio, each polynomial representing a discrete turbine vane setting. Mass-flow is modeled with a polynomial of pressure ratio and vane setting.

The third attempt uses an interpolating table of pressure ratio and vane-position for mass-flow. A table of UC_s, turbo speed and vane-setting governs the efficiency. The three-dimensional interpolating table is developed by MathCore and is not part of the Modelica standard library.

Only the two last approaches named $TurbineS$ and $TurbineR$ are shown in appendix B. The turbine and compressor components are not very general; since so little physical knowledge about them is used they may well need modification to fit different turbo-chargers.

6.8 Combustion chamber

The cylinders are modeled as a volume-less, steady-flow pump that heats the gas and extracts energy. In reality the flow of gas and fuel are far from steady as the ports are opened and fuel is injected only once every cycle. Since the cylinders buffers gas over almost a complete cycle there is a time-delay between changes on the intake side and the effects on the outlet side. This delay is not modeled; *Cylinder* is considered volume-less with $\sum \dot{m} = 0$. In the same manner the effect of fuel injection on output torque is modeled as being instant whereas in reality there is an average delay of one crankshaft revolution between changes in injection signal and output torque.

A rotational-mechanical *Flange*-connector enables the cylinder to be connected directly to the crankshaft, a model of the connecting rod is not necessary. A signal connector, *Pin*, controls the fuel-injection. The cylinder uses four interpolating tables; volumetric efficiency (η_{vol}), combustion efficiency (η), heating of exhaust gases (η_{exh}) and a one-dimensional map used to normalize the fuel injection.

Equation (6.10) to (6.13) are samples from *Cylinder*. They govern the flow through the cylinders, the power ($M\omega$, torque and angular speed) delivered from the crankshaft, the heating of the gas and the power consumed when pumping gas from the inlet to the exhaust manifolds. V is the displacement volume, Q_{HV} the energy density of the fuel and $\dot{W}_{pump}$ the pumping power.

$$\dot{m}_a = \eta_{vol}\frac{\omega}{4\pi}V\rho_a \tag{6.10}$$

$$M\omega = \dot{W}_{pump} - \eta\dot{m}_{fuel}Q_{HV} \tag{6.11}$$

$$0 = \dot{H}_a + \dot{H}_b + \eta_{exh}\dot{m}_{fuel}Q_{HV} \tag{6.12}$$

$$\dot{W}_{pump} = \frac{\omega}{4\pi}(p_b - p_a)V \tag{6.13}$$

Cylinder models a two-component gas, fresh air and exhausts, both on the inlet and the outlet side. The chemical reactions changes the relative amounts of the gases, leaving more exhausts and less fresh air. This transformation is difficult to express in a manner which can cope with an arbitrary number of gas-components. Therefore *Cylinder* must be modified if one wish to simulate a different set of gases.

The combustion process makes the cylinder an energy-source. Since fuel evaporates and adds to the mass-flow the cylinder model is also a source of mass. It should therefore include half if-statements governing mixtures and temperatures of the flow. A cylinder model with half if:s have not been tested due to the problems described in section 5.2.2.

6.9 Additional components

In all but the most trivial models there is a need to construct special purpose components. This section describes how additional components can be written.

To function with the rest of the library, any new thermodynamic component need to include the *FlowCut* connector (can of course also include other connectors) and it needs the correct number of equations. In a component with only thermodynamic connectors the correct number is determined by the numbers of declared variables, connectors and gas-species, see equation (6.14). Here n-dimensional vector expressions

equals n equations and half if expressions counts as half an equation.

$$N_{equations} = N_{variables} + N_{connectors}(1 + \frac{1}{2}N_{gases}) \tag{6.14}$$

Components obeying these rules can be connected to a solvable system without any extra equations outside the components, assuming access to half if-statements.

The easiest way to create new components is to extend the superclass *Base* which provide thermodynamic connectors, a few other variables and parameters like c_p and π.

Chapter 7

Systems of thermodynamic components

This chapter describes general model-construction with the thermodynamic library and shows two example systems.

7.1 Combining components

The components of the library can not be combined quite arbitrarily, there are disallowed and troublesome combinations. E.g. a reservoir connected to another reservoir typically gives an inconsistent system. The connection specifies that pressure is the same throughout the system while the pressures in the reservoirs are parameter-bound, possible to different values. For the same reason any system without pressure-drop that terminates with two reservoirs will be inconsistent.

Two volumes can be directly connected, thus declaring their pressures equal although temperatures and gas mixtures can differ. A row of volumes can be connected to simulate a long tube with uniform pressure but non-uniform temperature and gas-composition.

Having two consecutive non-dynamic components, i.e. classes without thermodynamic states like compressors and valves, is not impossible but might give numerical problems. It is a good idea to separate the components with a small volume or to lump the parts together in a new class. In *CompressorR* (section 6.6) the compressor and nearby restrictions have been merged in this way.

As discussed in section 5.3 a connector can take part in at most one connection.

7.2 A simple system

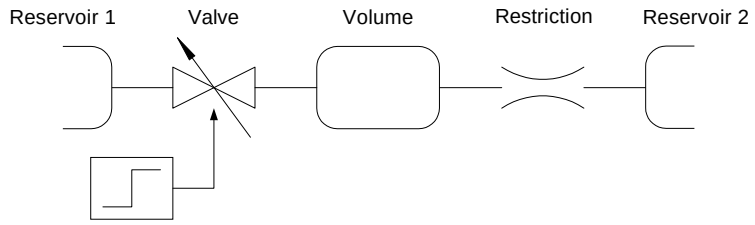

Figure 7.1: Simple system

An example system is pictured in figure 7.1. It includes three of the most basic classes; reservoir, control volume, restriction and valve. The example demonstrates fundamental functionalities like pressure equalization, adiabatic compression and bidirectional flow.

The system is simulated with high pressure in *Reservoir*1 and lower in *Reservoir*2, temperatures follow the same distribution. Each reservoir, and initially also the volume, contains a unique species of gas giving a three-component gas model. Simulation starts with $Valve$ closed and equal temperatures in *Reservoir*2 and $Volume$. The pressure in the volume is initially lower than in *Reservoir*2, the restriction can be considered a valve that opens at $t = 0$.

The gas-component that origins from *Reservoir*1, variables in *Reservoir*1 and in $Valve$ are represented with dashed lines in 7.2 and 7.3. Dot-dashed lines represent *Reservoir*2 and *Restriction* while solid lines represent the volume.

During the first second, with $Valve1$ closed, the pressures in $Volume$ and *Reservoir*2 nearly reaches equilibrium and the temperature in the volume increases due to the compression. When $Valve$ is opened at $t = 1.0$ the flow through *Restriction* is reversed, observe the instant switch of temperature of the flowing gas. The temperature in $Volume$ increases further after $t = 1.0$, first transient due to the compression and after about $t = 1.5$ s in a slower rate as hot gas from *Reservoir*1 flows into the volume.

7.3 A mechanical and thermodynamical system

This example shows a system of components from the mechanic and thermodynamic domains and illustrates energy conservation and trans-

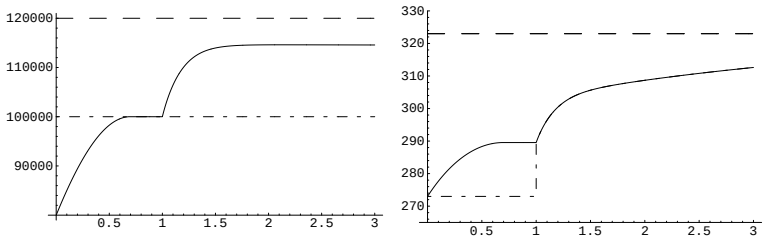

Figure 7.2: Simulation of Simple system. Left: pressures [Pa] in $Volume$ (solid) and reservoirs (dashed and dot-dashed). Right: temperatures [K] in $Volume$ (solid), $Valve$ (dashed) and *Restriction* (dot-dashed). Time [s] on x-axes.

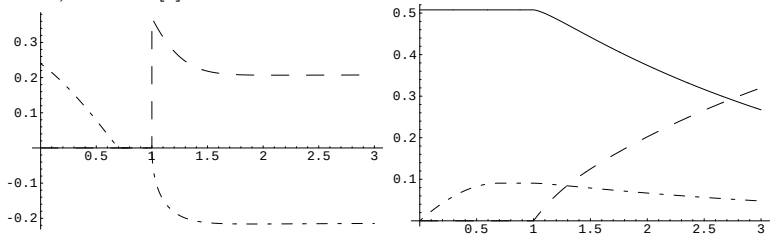

Figure 7.3: Simulation of Simple system. Left: mass-flows [kg/s] through $Valve$ (dashed) and *Restriction* (dot-dashed). Right: masses [kg] in $Volume$.

lation of energy between different domains.

Two volumes, a compressor and a turbine form the closed gas-path in figure 7.4, an inertia connects the compressor and the turbine. The turbo-charger is assembled with $CompressorS$ and $TurbineS$ (section 6.6 and 6.7), i.e. the models with efficiency and mass-flow polynomials.

Simulation starts at a state of high turbo-charger/inertia speed and near ambient conditions in the volumes. When the system is released at $t = 0$ the high turbo-charger speed makes the compressor pump gas from the low pressure to the high pressure volume. The gas flows back via the turbine and thereby returns some of the energy to the turbo-charger. Figure 7.5 shows the resulting pressures and temperatures in the two volumes, figure 7.6 turbo-charger speed and energies. The irreversible compression and expansion transforms the kinetic energy in the inertia to heat in the gas, solid and dashed lines in the right plot in figure 7.6. The dotted line represent the sum of kinetic and thermal energy which is of course constant. A thermodynamic interpretation of the process is that energy is transformed from a zero-entropy mechanical state to a high-entropy heat state.

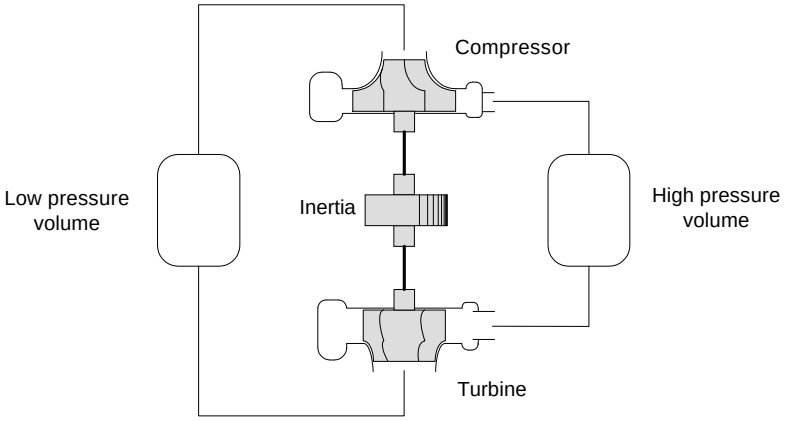

Figure 7.4: Closed system with turbo.

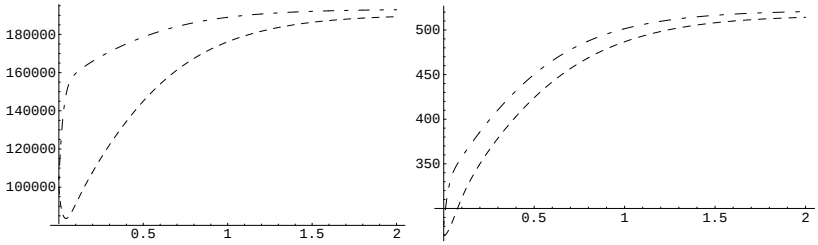

Figure 7.5: Simulation of Closed system. Left: pressures [Pa] in the low (dashed) and high (dot-dashed) volumes. Right: temperatures [K] in the volumes.

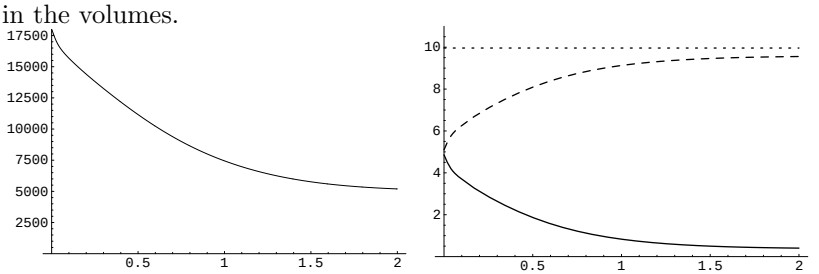

Figure 7.6: Simulation of Closed system. Left: turbo-speed [rad/s]. Right: kinetic (solid), thermal (dashed) and total (dotted) energies [kJ].

Chapter 8

Engine model

This chapter describes modeling of a specific engine using the thermodynamic library. The purpose is to verify the functionality of the components and to acquire a flexible model structure.

8.1 OM611 turbo diesel

The OM611 is a 2.2 liter, four-cylinder diesel engine from Mercedes-Benz. It is equipped with a cooled EGR-system, a turbo-charger with variable turbine vanes and an inter-cooler. Details can be found in [12].

8.2 Mean value model of OM611

The Modelica mean value model of OM611 is formed by 14 thermodynamic components, two mechanical parts and one signal block (ECU) as shown in figure 8.1. A two component gas mixture of air and combustion products is modeled in all parts of the gas path, although anything else than fresh air is unlikely to appear before the inlet manifold.

The ECU controls the two valves, the fuel-injection and the turbine vanes, the signal-connections are dashed in figure 8.1 and only partly drawn. The ECU-signals, engine speed or load and ambient conditions can be considered the model input.

The engine includes two interfaces for communication with the outside world; a rotational-mechanical *Flange* and a *EnvCut*-connector, both found in the bottom right of figure 8.1. No actual connections to the *EnvCut* can be seen on this model level but the coolers and the reservoirs refers to it.

In figure 8.2 the complete automobile in the simulation arrangement is pictured. The various transmission models used here are very

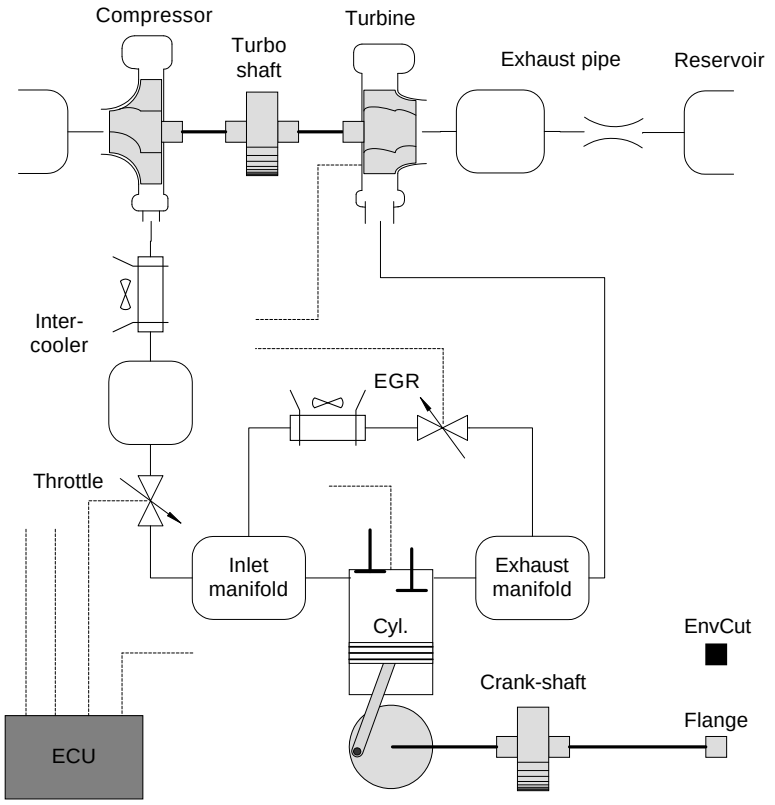

Figure 8.1: Topology of Modelica OM611 MVEM. Thermodynamic connections in solid, mechanical in thick solid and signal in dashed lines.

simple, declaring either a speed or a speed-dependent load. More realistic transmission models would include variable gear ratio, connections to the environment model and possibly connections to an automobile body.

There is no model of the driver present, a such could be placed in the same model level as the engine and transmission, i.e. in figure 8.2, and would communicate with the engine via additional signal connectors carrying torque demand and/or other reference signals.

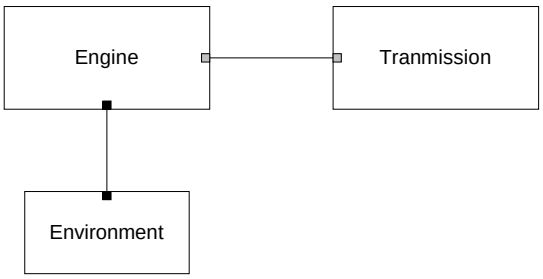

Figure 8.2: Automobile model used in simulation

8.3 Simulation of Modelica OM611 MVEM

8.3.1 Driving a load

The figures 8.3- 8.5 all show the same simulation where the engine model is driving an inertia with a speed-dependent load while all actuators are tested back and forth. The engine uses the turbo model with polynomial fits of efficiency and mass-flow, i.e. $CompressorS$ and $TurbineS$. The inputs are in the forms of steps at the times t according to the scheme in figure 8.6. Typical values of ambient conditions are used. The load is a second degree polynomial of engine speed resembling the aerodynamic and frictional forces affecting a car.

The simulation starts on low load and choked VNT. When the vanes are opened at t=1 s both the turbo-charger speed and the manifold pressures, figure 8.3-8.4, decreases rapidly which is expected. The EGR-valve and throttle settings have small effects on turbo speed and pressures but none of the inputs during the first six seconds affects engine speed or output torque much. At t=6 s the injection reference signal is doubled and the vanes are choked simultaneously, the same injection input is made at t=10 s but with open vanes. The difference can be seen in the turbo speed which accelerates much more after the first step, the apparent overshot is because the vanes opens again after 0.5 s. The two periods of high fuel-injection have nearly identical effects on engine speed and torque, which depend almost entirely on the fuel input as long as the air to fuel ratio does not fall under the stoichiometric ratio.

The turbine efficiency plot in figure 8.5 is somewhat extreme reaching too high values during the periods immediately after the actuator steps. $TurbineS$ is based on static test-bed measurements which do not cover the entire operating range of a turbine application on an engine, extrapolation can therefore be expected during highly dynamic events.

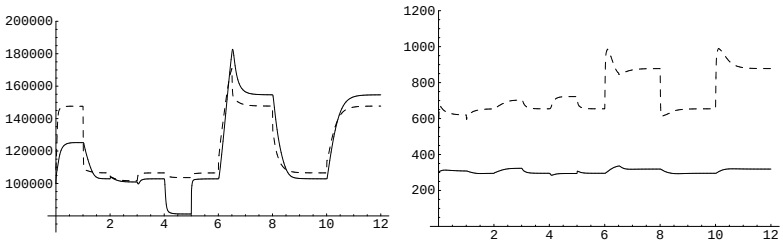

Figure 8.3: Modelica MVEM driving a load. Left: pressures [Pa] in inlet (solid) and exhaust (dashed) manifolds. Right: temperatures [K] in manifolds.

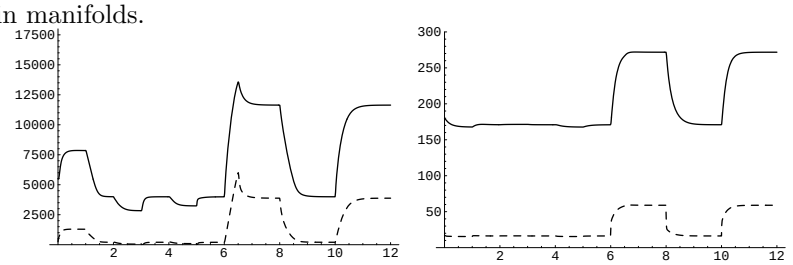

Figure 8.4: Modelica MVEM driving a load, speeds (solid) and powers (dashed). Left: turbo speed [rad/s] and power [W]. Right: engine speed [rad/s] and power [kW].

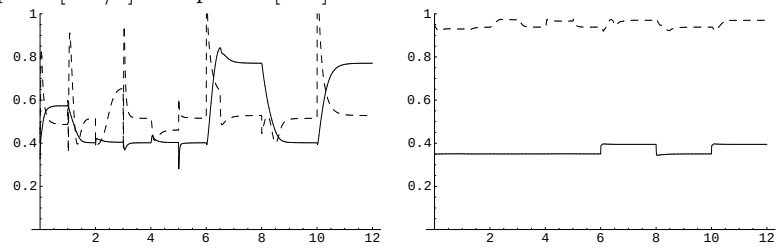

Figure 8.5: Modelica MVEM driving a load. Left: compressor (solid) and turbine (dashed) efficiencies. Right: combustion (solid) and volumetric (dashed) efficiencies.

	Value	t	Value	t	Value	t	Value	
Fuel	20	6.0	40	8.0	20	10.0	40	mg/shot
VNT	10	1.0	100	6.0	60	6.5	100	% open
EGR	0	2.0	100	3.0	0	-	0	% open
Throt.	100	4.0	20	5.0	100	-	100	% open

Figure 8.6: Modelica MVEM driving a load. Actuator reference signals during the time-intervals between the events t.

Also within the range of the measurements the static test-bed is somewhat different from the engine-application giving too high efficiency in general. As a consequence the inlet pressure is higher than the exhaust pressure (figure 8.3) at high engine loads which is rare in automobiles. The lack of an air-filter model also contributes to the high inlet pressure.

8.3.2 EGR back-flow

Section 3.4 claims that back-flow can occur through the EGR-system during brief moments. Figure 8.7 shows a simulation where the actuators are manipulated to achieve reversed flow, this time using the compressor and turbine with interpolating tables (see section 6.6- 6.7). The tabular values of the turbo-charger, as well as all other tables, are taken from the Simulink reference model described in section 8.4.

From a point of high load and choked turbine vanes the fuel is nearly cut off and the vanes are opened at $t = 2$. The result is a rapid pressure drop in the exhaust manifold and a slower decrease in the inlet manifold, the left plot in figure 8.7. During a short period the inlet pressure is higher than the exhaust pressure resulting in a reversed flow through the EGR-valve, the left plot of figure 8.8.

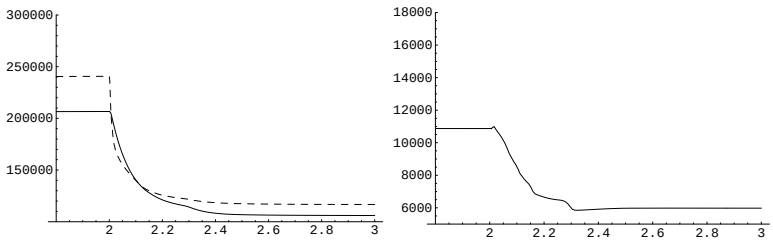

Figure 8.7: Modelica MVEM back-flow. Left: pressures [Pa] in inlet (solid) and exhaust (dashed) manifolds. Right: turbo speed [rad/s].

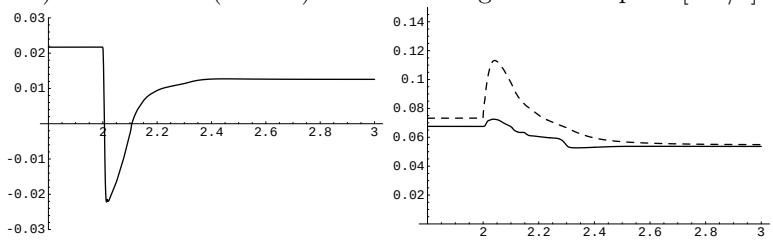

Figure 8.8: Modelica MVEM back-flow. Left: mass-flow [kg/s] through EGR-valve. Right: mass-flows [kg/s] through compressor (solid) and turbine (dashed).

8.4 Comparison with reference model

The Modelica OM611 model is verified against a reference model implemented in Simulink which is thoroughly tested and fits measurements well. Using only another model for verification instead of comparing with reality is of course not a recommendable approach in general, but it will do here since the OM611 MVEM is merely an example application.

The Modelica model is the same as in the "back-flow"-simulation, sharing most of the maps and equations with the reference model. But some is also different; many of the saturations, rate-limiters etc in the Simulink model have not been transfered to the Modelica model and therefore a very good numerical agreement should not be expected. If all equations and maps were copied directly from the Simulink model a near perfect agreement could probably be accomplished and only the differences in Matlabs and Dymolas solvers would remain. But there would certainly be more effective ways of comparing two solvers than writing two identical MVEMs.

It will instead be sufficient if the qualitative behavior match; the two models should respond in approximately the same way to input. In order to compare the dynamics of the models a series of simulations with nearly identical inputs have been made. Two of these have been chosen to represent the main similarities and differences between the models. Both simulations span over three seconds but the first part is used simply to reach the desired operating point and only the time between 1.8 and 3.0 s is displayed in the plots.

The first simulation compares the responses to a fuel-injection step input. At a constant engine speed of 3500 rev/min the fuel reference signal is increased from 15 to 50 mg/shot at $t = 2.0$ s, figure 8.9-8.12 shows some results. The Modelica turbo acceleration is somewhat shaky in the left plot in figure 8.10. The right plot represents the same simulation with the reference model which have smoother turbo dynamics and higher steady-state speeds. The rest of the differences can be explained by the poor Modelica turbo-charger model.

In the second simulation the vanes are choked from 100% to 40% at $t = 2$, again under constant engine-speed of 3500 rev/min. Figure 8.14 shows that the problem with the turbo is the turbine efficiency, it oscillates heavily and its peaks and valleys can be traced in the acceleration of the turbo. It also stays at a lower value than in the Simulink simulation. The rest of the plots, figure 8.13, 8.15 and 8.16, shows reasonable agreement if the effects of the turbine behavior are subtracted. Temperatures in the manifolds are approximately the same as well as the pressure difference, even though the absolute pressures differ due to the different turbo speeds.

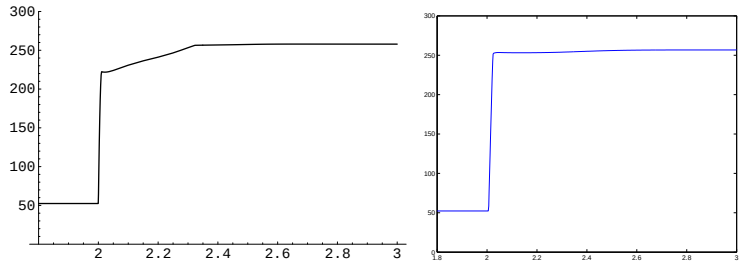

Figure 8.9: Step in fuel-injection, engine torque [Nm]. Left: Modelica, right: Simulink.

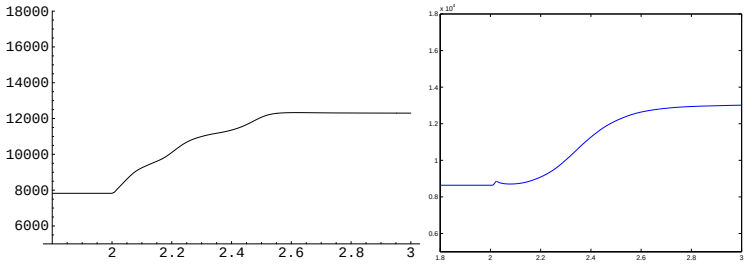

Figure 8.10: Step in fuel-injection, turbo speed [rad/s]. Left: Modelica, right: Simulink.

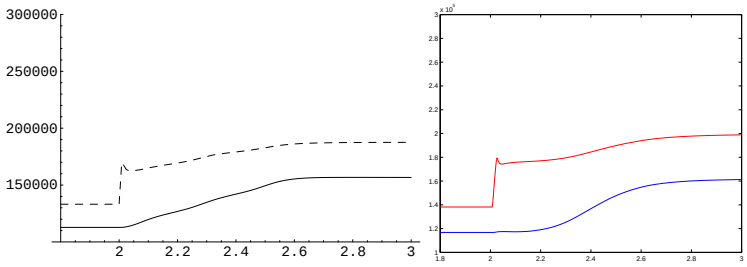

Figure 8.11: Step in fuel-injection, pressure [Pa] in inlet (solid) and exhaust (dashed) manifolds. Left: Modelica, right: Simulink.

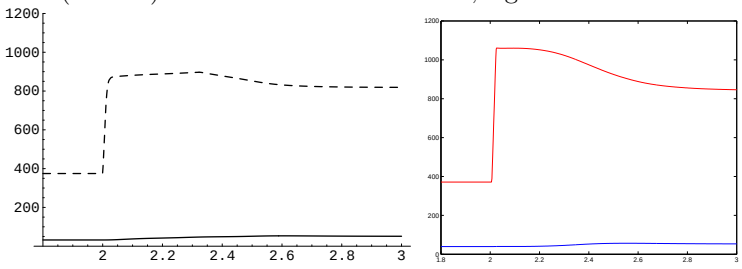

Figure 8.12: Step in fuel-injection, temperatures [°C] in inlet (solid) and exhaust (dashed) manifolds. Left: Modelica, right: Simulink.

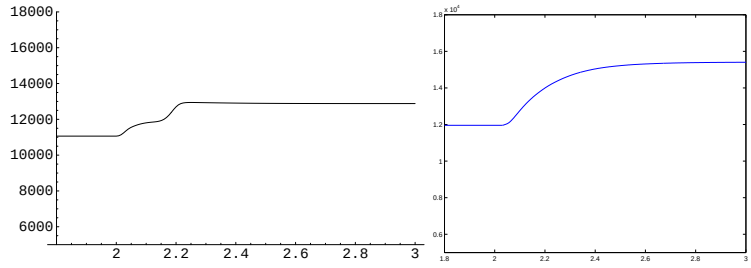

Figure 8.13: Step in turbine vanes, turbo speed [rad/s]. Left: Modelica, right: Simulink.

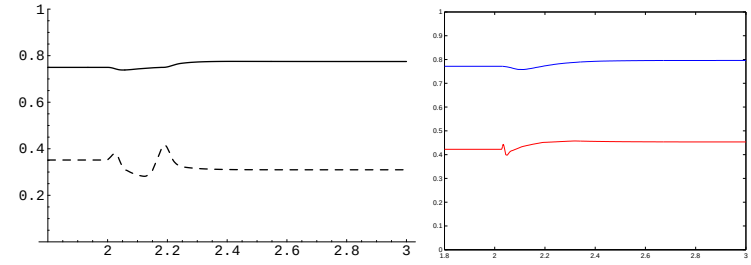

Figure 8.14: Step in turbine vanes, compressor (solid) and turbine (dashed) efficiencies. Left: Modelica, right: Simulink.

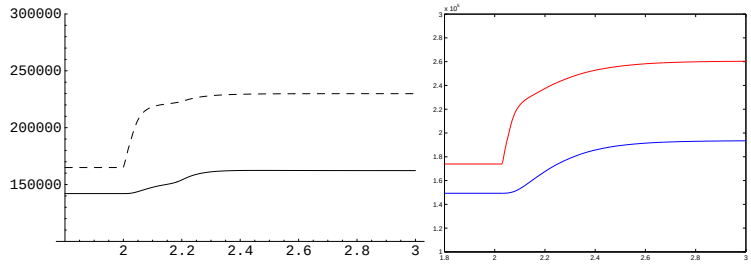

Figure 8.15: Step in turbine vanes, pressure [Pa] in inlet (solid) and exhaust (dashed) manifolds. Left: Modelica, right: Simulink.

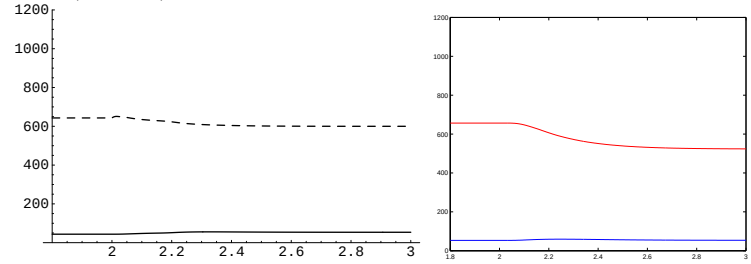

Figure 8.16: Step in turbine vanes, temperatures [°C] in inlet (solid) and exhaust (dashed) manifolds. Left: Modelica, right: Simulink.

8.5 About the simulations

The success of a modeling effort rely heavily on the the simulation tool and its numerical solver. This section deals with some aspects and experiences with MathModelica and Dymola.

MathModelica allows the user to specify initial values of states in the simulation call. Finding an initial state that enables the solver to find a solution at time zero is not always easy. The intuitive approach of choosing a physically realistic state will often not do, in the thermodynamic domain it is usually better to choose somewhat higher values. The simulate call also lets the user suggest initial values of non-state variables. This can help the solver which otherwise starts the iteration with all non-state variables at zero.

The initial-value problem is a well-known problem in numerical solvers and Dymola is not unique. Another, and far more severe, problem during the simulations have been the instability of MathModelica. Simulation calls locks up the Dymola kernel on a regular basis, sometimes locks up Mathematica and occasionally the entire operating system. No explanation or solution of this problem have come forth during the work; different models have been tested and Windows NT have even been reinstalled in one occasion without improvements. Hardware faults is a possibility but seems an unlikely cause. Instead the problem seems to be correlated with the MathModelica version 1.4 and NT. On computers running Windows 2000 Professional lock ups do not occur, at least not frequently.

Another problem have been Mathematicas dubious handling of files, sometimes destroying days of stored work. This too seems correlated with Windows NT.

Chapter 9

Conclusions

A functional package for mean value engine modeling have been presented. It can be used in a flexible manner for general internal combustion engines and is easy to expand with new classes. It supports energy and pressure dynamics, mixture of any number of gases but not momentum or non-ideal gases. The implementation is made under the assumption that a function for managing half if-statements would be present.

The engine models presented verifies that fairly large systems can be built with the library, the problems were extrapolation of one efficiency map and a poor implementation of the other. The models also provide a structure for further MVEMs with connections to atmosphere and transmission models.

The Modelica language is well suited for MVEMs The physical, object-oriented approach gives a vivid implementation even though much of the model is "black-box". Modeling is faster (given improved stability of the tool), less error-prone and produces easier maintained models than in Matlab/Simulink, C or other general tools due to the declaration of equations instead of algorithms. The notebook environment is convenient with code, graphs, documentation and mathematical computations in one place.

MathModelica makes modeling, simulation and visualization more structured if the work is performed more or less exclusively in the notebooks. On the other hand it can be argued that MathModelica is yet another environment to handle on top of those already in use. It lacks a coupling to Matlab, which whether one like it or not is the industrial standard today. An ability to communicate with Matlab would improve MathModelicas practical use in the automotive industry. The version of MathModelica used in this work suffer from severe problems with stability, at least when used on Windows NT computers. The initial-value problems are common to many numerical simulation tools. Yet

another, quite severe, problem have been Mathematicas somewhat unsafe routines for handling files when used on NT-machines.

Chapter 10

Outlook

This chapter presents some ideas for future extensions.

When a function for half if-statements is available an engine model could be tested with the proper definitions of the components. The preliminary version of the function seems promising but can only be used with quite simple models. Alternatively the modeling principle suggested in [9] could be tested.

It could be advantageous to implement the MVEM-components with the ThermoFlow library [7], which is yet under development. The chance is that [7] will become the standard base in the thermodynamic domain, perhaps also included in the Modelica standard library. If so the MVEM-library could add to a more general collection of libraries which would make exchange between MVEM:s and other thermodynamic models possible as more people would be familiar with the basic design.

References

[1] John B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill, New York, USA, 1988.

[2] L. Nielsen and L. Eriksson. Course material vehicular systems. Technical report, Vehicular Systems, Department of Electrical Engineering, Linköpings Universitet, Linköpings Universitet, Linköping, Sweden, 1999.

[3] Modelica Association. Modelica tutorial version 1.4. December 2000. www.modelica.org/documents.sthml.

[4] Modelica Association. Modelica language specification version 1.4. December 2000. www.modelica.org/documents.sthml.

[5] Dymola. Dynasim ab. www.dynasim.se.

[6] Mats Jirstrand. Mathmodelica - a full simulation tool. MathCore AB. www.mathcore.com/documents/index.html.

[7] J. Eborn, H. Tummescheit, and W. Wagner. Thermoflow, a thermo-hydraulic library in modelica. Technical report, Department of Automatic Control, Lund University, Sweden and Department of Energy Engineering, Technical University of Denmark, 1999. www.control.lth.se/ hubertus/ThermoFluid/.

[8] A. Stankovic. Modelling of air flows in automotive engines using modelica. Master's thesis LiTH-ISY-EX-3065, Department of Electrical Engineering, Linköpings Universitet, Linköping, Sweden, August 2000. www.fs.isy.liu.se/Publications/Masters_Theses2000.

[9] J. Brugård, L. Eriksson, and L. Nielsen. Mean value engine modeling of a turbo charged spark ignited engine - a principle study. Report LiTH-ISY-R-2370, Vehicular Systems, Department of Electrical Engineering, Linköpings Universitet, August 2001.

[10] Y. Nakayama and F. Boucher, R. *Introduction to Fluid Mechanics*. Arnold, London, Great Britain, 1999.

[11] R. Isermann, O. Lost, and A. Schwarte. Modellgestützte reglerentwicklung für einen abgasturbolader mit variabler turbinengeometrie an einem di-dieselmotor. *Motortechnische Zeitung*, (3):61, March 2000.

[12] R. Klingmanm, W. Fick, H. Brueggemann, D. Naber, K.H. Hoffmann, and A. Peters. Die neuen common-rail dieselmotoren mit direkteinspritzung in der modellgepflegten e-klasse. *Motortechnische Zeitung*, 7,8,9, 1999.

Notation

Symbols, abbreviations and other notations used in the thesis.

General

Modelica-constructs like `Connect`-statements are typeset in `plain style` in the report while self-defined types and variables like the connector-type *FlowCut* and its instance *a* are typeset in *mathematical style*.

Variables and parameters

c_v	Specific heat at constant volume.
c_p	Specific heat at constant pressure.
R	Gas constant, c_p-c_v.
γ	Ratio of specific heats, c_p/c_v.
p_a	Pressure at point (connector) a.
T_a	Temperature at point (connector) a.
$\dot{m}_a$	Mass-flow through cut (connector) a or of substance a.
$\dot{H}_a$	Enthalpy-flow through cut (connector) a.
ρ_a	Density at point (connector) a.
M	Context volume: mass. Context mechanics: torque.
U	Total energy.
W	Mechanical work.
ω	Rotational speed.
η	Efficiency, different contexts.
η_{vol}	Volumetric efficiency.
η_{exh}	"Efficiency" of heating exhausts.
V	Volume.
Q_{HV}	Energy density of fuel.
UC_s	Blade speed ratio.

Abbreviations

MVEM	Mean Value Engine Model
VNT	Variable Nozzle Turbine.
EGR	Exhaust Gas Recirculation.
SI	Spark Ignited.
OS	Operating System.
NT	MS Windows NT.

Appendix A

Modelica functions

```
function LinearRoot
   input Real x "input x";
   input Real p "power";
   input Real dx "linearization limit";
   output Real y "output";

protected
   Real Gamma;
   Real C3;
   Real C1;
   Real adx;
   Real padx;

algorithm
   adx := abs(dx);
   if x > adx then
      y := x^(p);
   else
      if x < -adx then
         y := -(-x)^(p);
      else
         padx := adx^((p-1));
         C1 := (1.5-0.5*p)*padx;
         C3 := 0.5*(p-1.)*padx/(adx*adx);
         y := (C1+C3*x*x)*x;
      end if;
   end if;
end LinearRoot;

function LinearRootS
```

```
        input Real x;
        input Real dx;
        output Real y;

algorithm
        if x > dx then
                y := x^(0.5);
        else
                y := 3/(2*dx^(1/2))*x - 1/(2*dx^(3/2))*x^(2);
        end if;
end LinearRoot;

function AvgConst
        input Real[2] C;
        input Real[2] m;
        output Real Cavg;

algorithm
        if abs(sum(m)) > 10^(-3) then
                Cavg := (C*m)/sum(m);
        else
                Cavg := sum(C)/size(C,1);
        end if;
end AvgConst;
```

Appendix B

Basic components

B.1 Connectors

```
connector FlowCut
   parameter Integer n;
   Real p;
   Real T;
   flow Real[n] mdot;
end FlowCut;


connector Flange
   Real omega;
   flow Real M;
end Flange;


connector Pin
   Real signal;
end Pin;


connector EnvCut
   parameter Integer n;
   Real p;
   Real T;
   Real[n] MassFraction;
   Real AirSpeed;
   Real Incline;
end EnvCut;
```

B.2 Super-classes

General with parameterized number of connectors and specialization with two connectors.

```
model Base
   parameter Integer Ngas;
   parameter Integer Ncon;
   FlowCut[Ncon] Cut(n=Ngas);
   Real[Ncon,Ngas] mdot;
   Real[Ncon] p(start=10^(5)*ones(Ncon));
   Real[Ncon] T(start=293*ones(Ncon));
   parameter Real[Ngas] c_p=1005*ones(Ncon);
   parameter Real[Ngas] gamma=1.4*ones(Ncon);
   Real[Ngas] R;
   Real[Ngas] c_v;
   constant Real Pi=Modelica.Constants.pi;

equation
   p = Cut.p;
   T = Cut.T;
   mdot = Cut.mdot;
   R = c_p-c_v;
   for i in 1:Ngas loop
      c_v[i] = c_p[i]/gamma[i];
   end for;
end Base;


model TwoPin
   extends Base(Ncon=2);
   Real[Ncon] mdot_a, mdot_b;
   Real T_a, T_b, p_a, p_b;

equation
   mdot_a = mdot[1,:];
   mdot_b = mdot[2,:];
   T_a = T[1];
   T_b = T[2];
   p_a = p[1];
   p_b = p[2];
end TwoPin;
```

B.3 Infinite reservoirs

```
model ReservoirP
   extends Base;
   parameter Real p_Infinity;
   parameter Real T_Infinity;
   parameter Real[Ngas] MassFraction;

equation
   p = p_Infinity*ones(Ncon);
end Reservoir;


model Reservoir
   extends Base;
   outer EnvCut Env(n=Ngas);
   EnvCut env;
   Real p_Infinity;
   Real T_Infinity;
   Real[Ngas] MassFraction;

equation
   connect(env, Env);
   p_Infinity = env.p;
   T_Infinity = env.T;
   MassFraction = env.MassFraction;
   p = p_Infinity*ones(Ncon);
end Reservoir;
```

B.4 Control volume

```
model CtrlVol
   extends Base;
   Real[Ngas] M;
   Real U_Infinity, M_Infinity;
   Real p_Infinity, T_Infinity;
   Real[Ngas] MassFraction;
   parameter Real V=1;

equation
   der(M) = transpose(mdot)*ones(Ncon);
   M_Infinity = sum(M);
   der(U) = sum(diagonal(mdot*c_p)*T);
   U = (c_v*M)*T_Infinity;
```

```
   p_Infinity*V = (M*R)*T_Infinity;
   p = p_Infinity*ones(Ncon);
   MassFraction = M/M_Infinity;
end CtrlVol;
```

B.5 Restrictions

Restrictions for incompressible $(\ldots I)$ and compressible $(\ldots C)$ flow.

```
model RestrictionI
   extends TwoPin;
   parameter Real r=10^(7);
   parameter Real sigma=10^((-2));

equation
   mdot_a+mdot_b = zeros(2);
   T_a = T_b;
   if p__a > p_b then
      sum(mdot_a) = LinearRootS(p_a*(p_a-p_b)/(r*T_a),sigma);
   else
      sum(mdot_b) = LinearRootS(p_b*(p_b-p_a)/(r*T_b),sigma);
   end if;
end RestrictionI;


model RestrictionC
   extends TwoPin;
   Real p_r;
   Real Psi;
   Real gamma_avg, R_avg;
   Real p_rC;
   parameter Real A=2*10^((-3));

equation
   mdot_a+mdot_b = zeros(2);
   gamma_avg = AvgConst(gamma,mdot_a);
   R_avg = AvgConst(R,mdot_a);
   p_rC = (2/(gamma_avg+1))^((gamma_avg/(gamma_avg-1)));
   T_b = T_a;
   p_r = if p_a < p_b then
      p_a/p_b
   else
      p_b/p_a;

   Psi = if p_r > p_rC then
```

```
      LinearRoot((2*gamma_avg/(gamma_avg-1))*
                   (p_r^((2/gamma_avg))-
                     p_r^(((gamma_avg+1)/gamma_avg))),
                 0.5,10^((-6)))
   else
      LinearRoot((2*gamma_avg/(gamma_avg-1))*
                   (p_rC^((2/gamma_avg))-
                     p_rC^(((gamma_avg+1)/gamma_avg))),
                 0.5,10^((-6)));

   if p_a > p_b then
      sum(mdot_a) = A*p_a/LinearRoot(R_avg*T_a,0.5,1000)*Psi;
   else
      sum(mdot_b) = A*p_b/LinearRoot(R_avg*T_b,0.5,1000)*Psi;
   end if;
end RestrictionC;
```

B.6 Valve

```
model Valve
   extends TwoPin;
   input Pin pin;
   Real X, A;
   Real p_r;
   Real Psi;
   Real gamma_avg, R_avg;
   Real p_rC;
   parameter Real[:,:] EffArea;
   ModelicaAdditions.Tables.CombiTable1D
                            AreaTable(table=EffArea);

equation
   X = pin.signal;
   mdot_a+mdot_b = zeros(2);
   gamma_avg = AvgConst(gamma,mdot_a);
   R_avg = AvgConst(R,mdot_a);
   p_rC = (2/(gamma_avg+1))^((gamma_avg/(gamma_avg-1)));
   T_b = T_a;
   p_r = if p_a < p_b then
      p_a/p_b
   else
      p_b/p_a;
   Psi = if p_r > p_rC then
      LinearRoot((2*gamma_avg/(gamma_avg-1))*
```

```
                (p_r^((2/gamma_avg))-
                  p_r^(((gamma_avg+1)/gamma_avg))),
                0.5,10^((-6)))
   else
      LinearRoot((2*gamma_avg/(gamma_avg-1))*
                  (p_rC^((2/gamma_avg))-
                     p_rC^(((gamma_avg+1)/gamma_avg))),
               0.5,10^((-6)));
   {X} = AreaTable.u;
   A = AreaTable.y[1];
   if p_a > p_b then
      sum(mdot_a) = A*p_a/LinearRoot(R_avg*T_a,0.5,1000)*Psi;
   else
      sum(mdot_b) = A*p_b/LinearRoot(R_avg*T_b,0.5,1000)*Psi;
   end if;
end Valve;
```

B.7 Cooler

```
model Cooler
   outer EnvCut Env(n=Ngas);
   EnvCut env;
   extends TwoPin;
   parameter Real[:,:] efficiency;
   ModelicaAdditions.Tables.CombiTable2D
                        EfficiencyTable(table=efficiency);
   Real eta;
   Real nu;
   Real T_amb;

equation
   connect(env, Env);
   nu = env.AirSpeed;
   T_amb = env.T;
   mdot_a+mdot_b = zeros(2);
   p_a = p_b;
   nu = EfficiencyTable.u1;
   abs(sum(mdot_a)) = EfficiencyTable.u2;
   eta = EfficiencyTable.y;
   if sum(mdot_a) > 0 then
      T_b = T_a-eta*(T_a-T_amb);
   else
      T_a = T_b-eta*(T_b-T_amb);
   end if;
```

```
end Cooler;
```

B.8 Compressors

Polynomial (...S) and tabular interpolation (...R) of mass-flow and efficiency.

```
model CompressorS
   extends TwoPin;
   Flange f_a;
   Real M, omega, N, phi, eta, N_corr, p_r;
   Real limit, LowFlow, gamma_avg, c_pavg;
   parameter Real D_2;
   parameter Real c_1, c_2, c_3, c_4;
   parameter Real c_5, c_6, c_7, c_8;
   parameter Real a_1, a_2, a_3;
   parameter Real k_1, k_2, k_3, k_4, k_5;
   parameter Real k_6, k_7, k_8, k_9;

equation
   omega = f_a.omega;
   omega = (2*Pi/60)*N;
   M = f_a.M;
   c_pavg = AvgConst(c_p,mdot_a);
   gamma_avg = AvgConst(gamma,mdot_a);
   phi = sum(mdot_a)*LinearRoot(T_a,0.5,100)/p_a;
   N_corr = N/LinearRoot(T_a,0.5,100);
   p_r = p_b/p_a;
   mdot_a+mdot_b = zeros(2);
   M*omega = sum(mdot_a)*c_pavg*(T_b-T_a);
   T_b/T_a = 1+1/eta*(p_r^(((gamma_avg-1)/gamma_avg))-1);
   limit = a_1+a_2*N_corr+a_3*N_corr^(2);
   LowFlow = c_1+c_2*N_corr+c_3*N_corr^(2)+c_4*p_r+
             c_5*N_corr^(2)*p_r+c_6*p_r^(2)+
             c_7*N_corr*p_r^(2)+c_8*N_corr^(2)*p_r^(2);
   phi = if p_r < limit then
      LowFlow
   else
      10^((-6));
   eta = k_1+k_2*phi+k_3*phi^(2)+k_4*p_r+k_5*phi*p_r+
         k_6*phi^(2)*p_r+k_7*p_r^(2)+k_8*phi*p_r^(2)+
         k_9*phi^(2)*p_r^(2);
end CompressorS;
```

```
model CompressorR
   extends TwoPin;
   Flange f_a;
   parameter Real[:,:] efficiency;
   parameter Real[:,:] phitab;
   parameter Real[:,:] pressureratio;
   ModelicaAdditions.Tables.CombiTable2D
                           EfficiencyTable(table=efficiency);
   ModelicaAdditions.Tables.CombiTable2D
                           CorrMassTable(table=phitab);
   ModelicaAdditions.Tables.CombiTable2D
                           CorrPRTable(table=pressureratio);
   Real phi, M, omega, N, omega_corr;
   Real eta, p_r, gamma_avg, c_pavg;

equation
   M = f_a.M;
   omega = f_a.omega;
   omega = (2*Pi/60)*N;
   c_pavg = AvgConst(c_p,mdot_a);
   gamma_avg = AvgConst(gamma,mdot_a);
   omega_corr = omega/LinearRoot(T_a,0.5,100);
   sum(mdot_a) = p_a/LinearRoot(T_a,0.5,100)*phi;
   mdot_a+mdot_b = zeros(2);
   M*omega = sum(mdot_a)*c_pavg*(T_b-T_a);
   T_b/T_a = 1+1/eta*(p_r^(((gamma_avg-1)/gamma_avg))-1);
   omega_corr = CorrMassTable.u1;
   p_b/p_a = CorrMassTable.u2;
   phi = CorrMassTable.y;
   omega_corr = CorrPRTable.u1;
   p_b/p_a = CorrPRTable.u2;
   p_r = CorrPRTable.y;
   phi = EfficiencyTable.u1;
   p_r = EfficiencyTable.u2;
   eta = EfficiencyTable.y;
end CompressorR;{inj_
```

B.9 Turbines

Polynomial ($\ldots S$) and tabular interpolation ($\ldots R$) of mass-flow and efficiency.

```
model TurbineS
   extends TwoPin;
   Flange f_a;
```

```
   input Pin pin;
   Real eta, M, omega, N;
   Real nu, N_corr, p_r(start=0.59), phi, gamma_avg, c_pavg;
   Real eta_1, eta_2, eta_3, eta_4, eta_5;
   Real X;
   parameter Real D;
   parameter Real X_1, X_2, X_3, X_4, X_5;
   parameter Real k_1, k_2, k_3, k_4, k_5, k_6;
   parameter Real a_1, a_2, a_3, a_4, a_5, a_6;
   parameter Real b_1, b_2, b_3, b_4, b_5, b_6;
   parameter Real c_1, c_2, c_3, c_4, c_5, c_6;
   parameter Real d_1, d_2, d_3, d_4, d_5, d_6;
   parameter Real e_1, e_2, e_3, e_4, e_5, e_6;

equation
   omega = f_a.omega;
   omega = (2*Pi/60)*N;
   M = f_a.M;
   c_pavg = AvgConst(c_p,mdot_a);
   gamma_avg = AvgConst(gamma,mdot_a);
   X = 10.4*(1-pin.signal);
   p_r = p_b/p_a;
   phi = sum(mdot_a)*LinearRoot(T_a,0.5,100)/p_a;
   N_corr = N/LinearRoot(T_a,0.5,100);
   nu = Pi*D*N/(60*LinearRoot(2*c_pavg*T_a*
             (1-p_r^(((gamma_avg-1)/gamma_avg))),0.5,0.1));
   mdot_a+mdot_b = zeros(2);
   M*omega = sum(mdot_a)*c_pavg*(T_b-T_a);
   T_b/T_a = 1-eta*(1-p_r^(((gamma_avg-1)/gamma_avg)));
   phi = k_1+k_2*p_r+k_3*p_r^(2)+k_4*p_r*X+
             k_5*p_r^(2)*X+k_6*X^(2);
   eta_1 = a_1+a_2*N_corr+a_3*nu+a_4*nu*N_corr+
             a_5*nu^(2)+a_6*nu^(2)*N_corr;
   eta_2 = b_1+b_2*N_corr+b_3*nu+b_4*nu*N_corr+
             b_5*nu^(2)+b_6*nu^(2)*N_corr;
   eta_3 = c_1+c_2*N_corr+c_3*nu+c_4*nu*N_corr+
             c_5*nu^(2)+c_6*nu^(2)*N_corr;
   eta_4 = d_1+d_2*N_corr+d_3*nu+d_4*nu*N_corr+
             d_5*nu^(2)+d_6*nu^(2)*N_corr;
   eta_5 = e_1+e_2*N_corr+e_3*nu+e_4*nu*N_corr+
             e_5*nu^(2)+e_6*nu^(2)*N_corr;
   eta = (1+sign(X_2-X))/2*
           ((X_2-X)/(X_2-X_1)*eta_1+(X-X_1)/(X_2-X_1)*eta_2)+
           (1+sign(X-X_2))/2*(1-sign(X-X_3))/2*
```

```
           ((X_3-X)/(X_3-X_2)*eta_2+(X-X_2)/(X_3-X_2)*eta_3)+
           (1+sign(X-X_3))/2*(1-sign(X-X_4))/2*
           ((X_4-X)/(X_4-X_3)*eta_3+(X-X_3)/(X_4-X_3)*eta_4)+
           (1+sign(X-X_4))/2*
           ((X_5-X)/(X_5-X_4)*eta_4+(X-X_4)/(X_5-X_4)*eta_5);
end TurbineS;


model TurbineR
   extends TwoPin;
   Flange f_a;
   input Pin pin;
   Pin PulsePin;
   parameter Real[:,:,:] efficiency;
   parameter Real[:,:] pulse;
   ModelicaAdditions.Tables.CombiTable2D
                          PulseTable(table=pulse);
   parameter Real[:,:] phitab;
   ModelicaAdditions.Tables.CombiTable2D
                          PhiTable(table=phitab);
   Real M, omega, eta, N, N_eng, eta_comp,  omega_corr;
   Real nu, capitalPhi, tableID, X, gamma_avg, c_pavg;
   constant Real D;

equation
   M = f_a.M;
   omega = f_a.omega;
   omega = (2*Pi/60)*N;
   X = 1-pin.signal;
   PulsePin.signal = N_eng;
   c_pavg = AvgConst(c_p,mdot_a);
   gamma_avg = AvgConst(gamma,mdot_a);
   omega_corr = omega/LinearRoot(T_a,0.5,100);
   nu = D*omega/(2*LinearRoot(2*c_pavg*T_a*(1-(p_b/p_a)^
                 (((gamma_avg-1)/gamma_avg))),0.5,0.001));
   sum(mdot_a) = capitalPhi*p_a/LinearRoot(T_a,0.5,100);
   mdot_a+mdot_b = zeros(2);
   M*omega = sum(mdot_a)*c_pavg*(T_b-T_a);
   T_b/T_a = 1-eta*eta_comp*(1-(p_b/p_a)^
                         (((gamma_avg-1)/gamma_avg)));
   N_eng = PulseTable.u1;
   p_a/p_b = PulseTable.u2;
   eta_comp = PulseTable.y;
   tableID = dymTable3Init(efficiency,0);
```

```
   eta = dymTableIpo3(tableID,omega_corr,nu,X);
   p_a/p_b = PhiTable.u1;
   X = PhiTable.u2;
   capitalPhi = PhiTable.y;
end TurbineR;
```

B.10 Combustion chambers

CylinderPin extends with an extra interface for engine-speed.

```
model Cylinder
   extends TwoPin;
   Flange f;
   input Pin pin;
   parameter Real[:,:] maxfuel;
   parameter Real[:,:] combeff;
   parameter Real[:,:] exhaust;
   parameter Real[:,:] voleff;
   ModelicaAdditions.Tables.CombiTable1D
                          MaxFuelingTable(table=maxfuel);
   ModelicaAdditions.Tables.CombiTable2D
                          EfficiencyTable(table=combeff);
   ModelicaAdditions.Tables.CombiTable2D
                          ExhaustEnergyTable(table=exhaust);
   ModelicaAdditions.Tables.CombiTable2D
                          VolEffTable(table=voleff);
   Real omega, N, M, mdot__fuel, eta__vol, q__comb;
   Real q__pump, rho__in, Hdot__a, Hdot__b;
   Real inj__fuel, inj__norm, inj__max, eta, exhWpart;
   parameter Real V=0.00215;
   parameter Real num_cyl=4;
   parameter Real T_fuel=273+30;
   parameter Real Q_HV=42.9*10^(6);
   parameter Real Q_vapour=400*10^(3);
   parameter Real AF_s=14.7;
   parameter Real cp_Fuel=800;

equation
   omega = f.omega;
   omega = (2*Pi/60)*N;
   M = f.M;
   pin.signal = inj_fuel;
   rho_in = p_a/(AvgConst(R,mdot_a)*T_a);
   N = VolEffTable.u1;
   rho_in = VolEffTable.u2;
```

```
   eta_vol = VolEffTable.y;
   {N} = MaxFuelingTable.u;
   {inj_max} = MaxFuelingTable.y;
   inj_norm = inj_fuel/inj_max;
   N = EfficiencyTable.u1;
   inj_norm = EfficiencyTable.u2;
   eta = EfficiencyTable.y;
   N = ExhaustEnergyTable.u1;
   inj_norm = ExhaustEnergyTable.u2;
   exhWpart = ExhaustEnergyTable.y;
   sum(mdot_a) = eta_vol*N/(2*60)*V*rho_in;
   mdot_fuel = num_cyl*N/(2*60)*inj_fuel;
   mdot_b[2]+mdot_a[2]+mdot_fuel+
        (if mdot_fuel < mdot_a[1]/AF_s then
             AF_s*mdot_fuel
         else
             mdot_a[1]) = 0;
   Hdot_a = (c_p*mdot_a)*T_a;
   Hdot_b = (c_p*mdot_b)*T_b;
   q_comb = if mdot_fuel < mdot_a[1]/AF_s then
      mdot_fuel*Q_HV
   else
      mdot_a[1]/AF_s*Q_HV;
   q_pump = omega*(p_b-p_a)*V/(4*Pi);
   M*omega = q_pump-eta*q_comb;
end Cylinder;


model CylinderPin
   extends Cylinder;
   Pin PulsePin;

equation
   PulsePin.signal = N;
end CylinderPin;
```

B.11 Stiff axle/inertia

```
model AxelTwo
   Flange f_a, f_b;
   Real omega, N;
   parameter Real I=1;
   constant Real Pi=Modelica.Constants.pi;
```

```
equation
   f_a.omega = f_b.omega;
   omega = f_a.omega;
   omega = (2*Pi/60)*N;
   I*der(omega) = f_a.M+f_b.M;
end AxelTwo;_a*
```

B.12 Control block

Interpolating time-tables for reference-signals filtered to actual actuator-settings.

```
model Controller
   output Pin xvgt;
   output Pin inj;
   output Pin throttle;
   output Pin egr;
   parameter Real tau_t=0.01;
   parameter Real tau_i=0.006;
   parameter Real tau_x=0.02;
   parameter Real tau_e=0.01;
   Real t;
   Real e;
   Real i;
   Real x;
   parameter Real[6,2]
         SpjallTabell={{0.,1.},{1.,1.},{2.,1.},
                       {4.,1.},{8.,1.},{10.,1.}};
   parameter Real[6,2]
         SoppaTabell={{0.,35*10^((-6))},{1.,50*10^((-6))},
                      {2.,50*10^((-6))},{2.,10*10^((-6))},
                      {4.,10*10^((-6))},{7.,10*10^((-6))}};
   parameter Real[6,2]
         VgtTabell={{0.,1.},{1.,0.2},{1.1,0.2},
                    {2.,0.2},{2.,1.},{8.,1.}};
   parameter Real[6,2]
         EgrTabell={{0.,0.45},{2.,0.45},{3.,0.45},
                    {4.,0.45},{8.,0.45},{10.,0.45}};
   Modelica.Blocks.Sources.TimeTable
                           ThrottleTable(table=SpjallTabell);
   ModelicaAdditions.Tables.CombiTableTime
                           InjectionTable(table=SoppaTabell);
   Modelica.Blocks.Sources.TimeTable
                           XVGTTable(table=VgtTabell);
   Modelica.Blocks.Sources.TimeTable
```

```
                        EGRTable(table=EgrTabell);

equation
   tau_t*der(t)+t = ThrottleTable.y[1];
   throttle.signal = t;
   tau_e*der(e)+e = EGRTable.y[1];
   egr.signal = e;
   tau_i*der(i)+i = InjectionTable.y[1];
   inj.signal = i;
   tau_x*der(x)+x = XVGTTable.y[1];
   xvgt.signal = x;
end Controller;
```

Appendix C

OM611 engine

C.1 Input data

Example of how the 3D turbine efficiency table could be constructed in Mathematica. Data and grid-values must be merged into a single tensor. The values of η are here chosen at random (well, via an obscure function), the real data used in the model is a 5*40*60 table.

The input variables to the table are corrected turbo speed, blade speed ratio and VNT-position.

```
TurbCorrSpeed = {150.0, 400.0, 600.0};
TurbUCs = {0.0, 0.67, 1.33, 2.0};
TurbXVGT = {0.0, 1.0};
TurbEfficiency =
  Table[Max[0.1, (0.5+0.05j)(1+0.3k)Sin[i]],
        {k, 0, 1}, {j, 1, 3}, {i, 0, 3}];
```

```
For[i=1; temp = {}, i<=Length[TurbXVGT], i++,
    temp = Append[temp,
        Prepend[Transpose[Prepend[TurbEfficiency[[i]],
                                  TurbUCs]],
            Prepend[TurbCorrSpeed, TurbXVGT[[i]]]]]];
TurbEffMatrix = Transpose[temp, {1, 3, 2}];
```

$$\begin{pmatrix} \begin{pmatrix} 0. \\ 0. \\ 0.67 \\ 1.33 \\ 2. \end{pmatrix} & \begin{pmatrix} 150. \\ 0.1 \\ 0.4628 \\ 0.5001 \\ 0.1 \end{pmatrix} & \begin{pmatrix} 400. \\ 0.1 \\ 0.5049 \\ 0.5456 \\ 0.1 \end{pmatrix} & \begin{pmatrix} 600. \\ 0.1 \\ 0.5470 \\ 0.5910 \\ 0.1 \end{pmatrix} \\ \begin{pmatrix} 1. \\ 0 \\ 0.67 \\ 1.33 \\ 2. \end{pmatrix} & \begin{pmatrix} 150. \\ 0.1 \\ 0.6017 \\ 0.6501 \\ 0.1009 \end{pmatrix} & \begin{pmatrix} 400. \\ 0.1 \\ 0.6563 \\ 0.7093 \\ 0.1101 \end{pmatrix} & \begin{pmatrix} 600. \\ 0.1 \\ 0.7110 \\ 0.7684 \\ 0.1192 \end{pmatrix} \end{pmatrix}$$

The remaining tables are one- or two-dimensional. Below the construction of a 2D combustion efficiency table. Grid-values and data are merged here too. The η-values are of course gibberish also here, typical actual values are smaller and the size of the real table much larger.

```
SpeedList = Table[1000.0+i*1500, {i, 0, 2}];
FuelProportion = Prepend[Table[0.2+i*0.4, {i, 0, 2}], 0.0];
EtaList = {{0.5, 0.5, 0.5}, {0.5, 0.5, 0.5}, {0.5, 0.5, 0.5}};
EtaMatrix =
    Prepend[Transpose[Prepend[Transpose[EtaList],
                              SpeedList]],
            FuelProportion];
```

$$\begin{pmatrix} 0.0 & 0.2 & 0.6 & 1.0 \\ 1000.0 & 0.5 & 0.5 & 0.5 \\ 2500.0 & 0.5 & 0.5 & 0.5 \\ 4000.0 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

C.2 Engine model

This is the OM611 model with interpolating tables in turbine and compressor, the model with polynomial fits is not presented. The if-statements should be replaced by half-if:s in the basic components when the function that handles them is available.

To be able to read data from the Mathematica work-space, the model is written as a Mathematica-function which reads tabular values for various maps. The function-call at the end of the section actually defines the engine-model.

```
ReadTables[Maxfuel_, Combeff_, Exhaust_, Voleff_,
```

```
    InterCoolerEff_, EgrCoolerEff_, EgrArea_, ThrotArea_,
    CompEff_, CompPR_, CompPhi_, TurbEff_, TurbPhi_,
    TurbPulse_]:=

model EngineR
   parameter Integer ngas=2;
   parameter Real[ngas] cp={1005,1150};
   parameter Real[ngas] gamma1={1.4,1.39};
   inner EnvCut Env(n=ngas);
   Flange f;
   Reservoir AmbIn(Ncon=1,Ngas=ngas,c_p=cp,gamma=gamma1);
   Reservoir AmbOut(Ncon=1,Ngas=ngas,c_p=cp,gamma=gamma1);
   CtrlVol Inter(V=6.3*10^((-3)),Ncon=2,
                 Ngas=ngas,c_p=cp,gamma=gamma1);
   CtrlVol Inlet(V=3.2*10^((-3)),Ncon=3,
                 Ngas=ngas,c_p=cp,gamma=gamma1);
   CtrlVol Exmani(V=0.844*10^((-3)),Ncon=3,
                  Ngas=ngas,c_p=cp,gamma=gamma1);
   CtrlVol Expipe(V=0.01,Ncon=2,Ngas=ngas,
                  c_p=cp,gamma=gamma1);
   CylinderPin Cyl(Ngas=ngas, maxfuel=Maxfuel,
                   combeff=Combeff, exhaust=Exhaust,
                   voleff=Voleff, c_p=cp,
                   gamma=gamma1);
   Axel CrankShaft(I=0.02);
   CompressorR Komp(Ngas=ngas, efficiency=CompEff,
                    phitab=CompPhi, pressureratio=CompPR,
                    c_p=cp, gamma=gamma1);
   TurbineR Turb(Ngas=ngas, efficiency=TurbEff,
                 phitab=TurbPhi, pulse=TurbPulse,
                 c_p=cp, gamma=gamma1);
   Axel TurboShaft(I=7.71*10^(-6));
   Cooler InterCooler(Ngas=ngas,
                      efficiency=InterCoolerEff,
                      c_p=cp, gamma=gamma1);
   Cooler EgrCooler(Ngas=ngas, efficiency=EgrCoolerEff,
          c_p=cp, gamma=gamma1);\[IndentingNewLine]
   Valve MainThrottle(Ngas=ngas, EffArea=ThrotArea,
          c_p=cp, gamma=gamma1);
   Valve EgrThrottle(Ngas=ngas,EffArea=EgrArea,c_p=cp,
                     gamma=gamma1);
   RestrictionC R(Ngas=ngas,A=1.5*10^((-3)),c_p=cp,
                  gamma=gamma1);
   Controller ECU;
```

```
equation
   connect(AmbIn.Cut[1], Komp.Cut[1]);
   connect(Komp.Cut[2], InterCooler.Cut[1]);
   connect(InterCooler.Cut[2], Inter.Cut[1]);
   connect(Inter.Cut[2], MainThrottle.Cut[1]);
   connect(MainThrottle.Cut[2], Inlet.Cut[1]);
   connect(Inlet.Cut[2], Cyl.Cut[1]);
   connect(Cyl.Cut[2], Exmani.Cut[1]);
   connect(Exmani.Cut[2], Turb.Cut[1]);
   connect(Turb.Cut[2], Expipe.Cut[1]);
   connect(Expipe.Cut[2], R.Cut[1]);
   connect(R.Cut[2], AmbOut.Cut[1]);

   connect(Exmani.Cut[3], EgrCooler.Cut[1]);
   connect(EgrCooler.Cut[2], EgrThrottle.Cut[1]);
   connect(EgrThrottle.Cut[2], Inlet.Cut[3]);

   connect(Cyl.f, CrankShaft.f_a);
   connect(CrankShaft.f_b, f);
   connect(Komp.f_a, TurboShaft.f_a);
   connect(TurboShaft.f_b, Turb.f_a);

   connect(Turb.pin, ECU.xvgt);
   connect(Cyl.pin, ECU.inj);
   connect(MainThrottle.pin, ECU.throttle);
   connect(EgrThrottle.pin, ECU.egr);
   connect(Cyl.PulsePin, Turb.PulsePin);

   if sum(AmbIn.mdot[1,:]) < 0 then
      AmbIn.T[1] = AmbIn.T_Infinity;
   else
      Inter.T[1] = Inter.T_Infinity;
   end if;
   if sum(AmbIn.mdot[1,:]) < 0 then
      AmbIn.mdot[1,2] =
        AmbIn.MassFraction[2]*sum(AmbIn.mdot[1,:]);
   else
      Inter.mdot[1,2] =
        Inter.MassFraction[2]*sum(Inter.mdot[1,:]);
   end if;
   if sum(Inter.mdot[2,:]) < 0 then
      Inter.T[2] = Inter.T_Infinity;
   else
```

```
   Inlet.T[1] = Inlet.T_Infinity;
end if;
if sum(Inter.mdot[2,:]) < 0 then
   Inter.mdot[2,2] =
     Inter.MassFraction[2]*sum(Inter.mdot[2,:]);
else
   Inlet.mdot[1,2] =
     Inlet.MassFraction[2]*sum(Inlet.mdot[1,:]);
end if;
if sum(Inlet.mdot[2,:]) < 0 then
   Inlet.T[2] = Inlet.T_Infinity;
else
   Cyl.Hdot_a+Cyl.Hdot_b+Cyl.exhWpart*Cyl.q_comb = 0;
end if;
if sum(Inlet.mdot[2,:]) < 0 then
   Inlet.mdot[2,2] =
     Inlet.MassFraction[2]*sum(Inlet.mdot[2,:]);
else
   sum(Cyl.mdot_a)+Cyl.mdot_fuel+sum(Cyl.mdot_b) = 0;
end if;
if sum(Cyl.mdot_b) < 0 then
   Cyl.Hdot_a+Cyl.Hdot_b+Cyl.exhWpart*Cyl.q_comb = 0;
else
   Exmani.T[1] = Exmani.T_Infinity;
end if;
if sum(Cyl.mdot_b) < 0 then
   sum(Cyl.mdot_a)+Cyl.mdot_fuel+sum(Cyl.mdot_b) = 0;
else
   Exmani.mdot[1,2] =
     Exmani.MassFraction[2]*sum(Exmani.mdot[1,:]);
end if;
if sum(Exmani.mdot[2,:]) < 0 then
   Exmani.T[2] = Exmani.T_Infinity;
else
   Expipe.T[1] = Expipe.T_Infinity;
end if;
if sum(Exmani.mdot[2,:]) < 0 then
   Exmani.mdot[2,2] =
     Exmani.MassFraction[2]*sum(Exmani.mdot[2,:]);
else
   Expipe.mdot[1,2] =
     Expipe.MassFraction[2]*sum(Expipe.mdot[1,:]);
end if;
if sum(AmbOut.mdot[1,:]) < 0 then
```

```
      AmbOut.T[1] = AmbOut.T_Infinity;
   else
      Expipe.T[2] = Expipe.T_Infinity;
   end if;
   if sum(AmbOut.mdot[1,:]) < 0 then
      AmbOut.mdot[1,2] =
        AmbOut.MassFraction[2]*sum(AmbOut.mdot[1,:]);
   else
      Expipe.mdot[2,2] =
        Expipe.MassFraction[2]*sum(Expipe.mdot[2,:]);
   end if;
   if sum(Exmani.mdot[3,:]) < 0 then
      Exmani.T[3] = Exmani.T_Infinity;
   else
      Inlet.T[3] = Inlet.T_Infinity;
   end if;
   if sum(Exmani.mdot[3,:]) < 0 then
      Exmani.mdot[3,2] =
        Exmani.MassFraction[2]*sum(Exmani.mdot[3,:]);
   else
      Inlet.mdot[3,2] =
        Inlet.MassFraction[2]*sum(Inlet.mdot[3,:]);
   end if;
end EngineR;


ReadTables[MaxFuelMatrix, EtaMatrix, ExhMatrix,
    VolEffMatrix, EffMatrixInter, EffMatrixEGR,
    AreaMatrixEGRThrot, AreaMatrixMainThrot,
    CompEffMatrix, CompCorrPRMatrix,
    CompPhiMatrix, TurbEffMatrix,
    TurbPhiMatrix, TurbPulseMatrix];
```

C.3 Automobile model

C.3.1 Atmosphere/environment

```
model Environment
   parameter Integer ngas=2;
   EnvCut Env(n=ngas);
   parameter Real p_amb=10^(5);
   parameter Real T_amp=293;
   parameter Real[:] Mixture={1,0};
   parameter Real Incline=0;
```

```
equation
   Env.p = p_amb;
   Env.T = T_amp;
   Env.MassFraction = Mixture;
   Env.AirSpeed = 30;
   Env.Incline = Incline;
end Environment;
```

C.3.2 Dummy transmission models

```
model AxelLoad
   Flange f;
   Real omega, N, M;
   parameter Real I=1;
   parameter Real B_dry=1;
   parameter Real B_wet=1;
   parameter Real B_aero=1;
   constant Real Pi=Modelica.Constants.pi;

equation
   omega = f.omega;
   omega = (2*Pi/60)*N;
   M = f.M;
   I*der(omega) = M-B_dry-B_wet*omega-B_aero*omega^(2);
end AxelLoad;


model AxelSpeed
   Flange f;
   Real omega, N, M;
   constant Real Pi=Modelica.Constants.pi;
   parameter Real N_const=2500;

equation
   omega = f.omega;
   omega = (2*Pi/60)*N;
   M = f.M;
   N = N_const;
end AxelSpeed;omega;
```

C.3.3 Complete automobiles

This is the model used in the "Back-flow through EGR"-simulation and in the comparison with the Simulink model.

```
model Speed
   EngineR engine;
   AxelSpeed clutch;
   Environment environment;

equation
   connect(engine.f, clutch.f);
   connect(engine.Env, environment.Env);
end Speed;
```

This model is used in the "Driving a load"-simulation. The engine model *EngineS* is not accounted for here, but it includes *CompressorS* and *TurbineS*.

```
model AutoLoad
   EngineS engine;
   AxelLoad clutch;
   Environment environment;

equation
   connect(engine.f, clutch.f);
   connect(engine.Env, environment.Env);
end AutoLoad;
```

Appendix D

Simulations

The simulation-calls with initial values of states and inputs via the ECU and transmission modules.

D.1 Driving a load

```
load=Simulate[AutoLoad, {t, 0, 12},

  ParameterValues -> {
    clutch.B_dry == 5,
    clutch.B_wet == 0.1,
    clutch.B_aero == 2.5*10^(-3),
    clutch.I == 0.2,
    engine.ECU.SoppaTabell ==
                {{0.0, 20*10^(-6)}, {6.0, 20*10^(-6)},
                 {6.0, 40*10^(-6)}, {8.0, 40*10^(-6)},
                 {8.0, 20*10^(-6)}, {10.0, 20*10^(-6)},
                 {10.0, 40*10^(-6)}, {12.0, 40*10^(-6)}},
    engine.ECU.VgtTabell ==
                {{0.0, 0.1}, {1.0, 0.1},
                 {1.0, 1.0}, {6.0, 1.0},
                 {6.0, 0.6}, {6.5, 0.6}
                 {6.5, 1.0}, {12.0, 1.0}},
    engine.ECU.EgrTabell ==
                {{0.0, 0.0}, {2.0, 0.0},
                 {2.0, 1.0}, {3.0, 1.0},
                 {3.0, 0.0}, {12.0, 0.0}},
    engine.ECU.SpjallTabell ==
                {{0.0, 1.0}, {4.0, 1.0},
                 {4.0, 0.2}, {5.0, 0.2},
```

```
                {5.0, 1.0}, {12.0, 1.0}}},

  InitialValues -> {
    engine.TurboShaft.omega == 5.5*10^3,
    clutch.omega == 180,
    engine.Inter.M == {7.3*10^(-3), 0.1*10^(-3)},
    engine.Inter.U == 1600,
    engine.Inlet.M == {3.6*10^(-3), 0.1*10^(-3)},
    engine.Inlet.U == 800,
    engine.Exmani.M == {0.5*10^(-3), 0.1*10^(-3)},
    engine.Exmani.U == 230,
    engine.Expipe.M == {4.5*10^(-3), 0.1*10^(-3)},
    engine.Expipe.U == 2500,
    engine.ECU.i == 20*10^(-6),
    engine.ECU.x == 1,
    engine.ECU.t == 1,
    engine.ECU.e == 0,
    engine.Cyl.mdot_a == {0.08, 0.01},
    engine.Cyl.mdot_b[[1]] == -0.09,
    engine.Exmani.T[[1]] == 1000,
    engine.Exmani.T[[2]] == 900,
    engine.Inlet.T[[2]] == 390},

  NumberOfIntervals->12000
];
```

D.2 Back-flow through EGR

```
backflow=Simulate[AutoSpeed, {t, 0, 4},

  ParameterValues -> {
    clutch.N_const == 3500,
    engine.ECU.SoppaTabell ==
                {{0.0, 35 10^(-6)}, {1.0, 50*10^(-6)},
                 {2.0, 50*10^(-6)}, {2.0, 10*10^(-6)},
                 {4.0, 10*10^(-6)}, {7.0, 10*10^(-6)}},
    engine.ECU.VgtTabell ==
                {{0.0, 1.0}, {1.0, 0.2}, {1.1, 0.2},
                 {2.0, 0.2}, {2.0, 1.0}, {8.0, 1.0}},
    engine.ECU.EgrTabell ==
                {{0.0, 0.45}, {2.0, 0.45}, {3.0, 0.45},
                 {4.0, 0.45}, {8.0, 0.45}, {10.0, 0.45}},
    engine.ECU.tau_i == 0.006, engine.ECU.tau_x == 0.02},
```

```
  InitialValues -> {
    engine.TurboShaft.omega == 7.5*10^3,
    engine.Inter.M == {10*10^(-3), 0.0001*10^(-3)},
    engine.Inter.U == 3000,
    engine.Inlet.M == {5*10^(-3), 0.0001*10^(-3)},
    engine.Inlet.U == 1500,
    engine.Exmani.M == {0.5*10^(-3), 0.0001*10^(-3)},
    engine.Exmani.U == 800,
    engine.Expipe.M == {0.0064, 0.001*10^(-3)},
    engine.Expipe.U == 4200,
    engine.ECU.i == 35*10^(-6),
    engine.ECU.x == 1}
];
```

D.3 Fuel-step

```
fuelstep=Simulate[AutoSpeed, {t, 0, 4},

  ParameterValues -> {
    clutch.N_const == 3500,
    engine.ECU.SoppaTabell ==
                {{0.0, 35*10^(-6)}, {1.0, 15*10^(-6)},
                 {2.0, 15*10^(-6)}, {2.0, 50*10^(-6)},
                 {4.0, 50*10^(-6)}, {10.0, 15*10^(-6)}},
    engine.ECU.VgtTabell ==
                {{0.0, 1.0}, {2.0, 1.0}, {2.0, 1.0},
                 {3.5, 1.0}, {3.6, 1.0}, {4.0, 1.0}},
    engine.ECU.EgrTabell ==
                {{0.0, 0.0}, {2.0, 0.0}, {3.0, 0.0},
                 {4.0, 0.0}, {8.0, 0.0}, {10.0, 0.0}},
    engine.ECU.tau_i == 0.006,
    engine.ECU.tau_x == 0.02},

  InitialValues -> {
    engine.TurboShaft.omega == 7.5*10^3,
    engine.Inter.M == {10*10^(-3), 0.0001*10^(-3)},
    engine.Inter.U == 3000,
    engine.Inlet.M == {5*10^(-3), 0.0001*10^(-3)},
    engine.Inlet.U == 1500,
    engine.Exmani.M == {0.5*10^(-3), 0.0001*10^(-3)},
    engine.Exmani.U == 800,
    engine.Expipe.M == {0.0064, 0.001*10^(-3)},
    engine.Expipe.U == 4200,
    engine.ECU.i == 35*10^(-6),
```

```
    engine.ECU.x == 1}
];
```

D.4 VNT-step

```
vgtstep=Simulate[AutoSpeed, {t, 0, 4},

  ParameterValues -> {
    clutch.N_const ==  3500,
    engine.ECU.SoppaTabell ==
                {{0.0, 35*10^(-6)}, {2.0, 35*10^(-6)},
                 {2.0, 35*10^(-6)}, {4.0, 35*10^(-6)},
                 {7.0, 15*10^(-6)}, {10.0, 15*10^(-6)}},
    engine.ECU.VgtTabell ==
                {{0.0, 1.0}, {2.0, 1.0}, {2.0, 0.4},
                 {4.0, 0.4}, {5.0, 0.4}, {10.0, 0.4}},
    engine.ECU.EgrTabell ==
                {{0.0, 0.0}, {2.0, 0.0}, {3.0, 0.0},
                 {4.0, 0.0}, {8.0, 0.0}, {10.0, 0.0}}},

  InitialValues -> {
    engine.TurboShaft.omega == 5.5*10^3,
    engine.Inter.M == {10*10^(-3), 0.0001*10^(-3)},
    engine.Inter.U == 3000,
    engine.Inlet.M == {5*10^(-3), 0.0001*10^(-3)},
    engine.Inlet.U == 1500,
    engine.Exmani.M == {0.5*10^(-3), 0.0001*10^(-3)},
    engine.Exmani.U == 800,
    engine.Expipe.M == {0.0064, 0.001*10^(-3)},
    engine.Expipe.U == 4200,
    engine.ECU.i == 35*10^(-6),
    engine.ECU.x == 1}
];
```