

WinDali - An Open-Structured Component Modeling and Simulation Program Based on Standard Programming Languages

Morten Juel Skovrup
Dept. of Energy Engineering
Technical University of Denmark

1 Abstract

This paper presents a modeling and simulation program originally aimed at creating models of refrigeration plants by modeling the individual components in the plant and thereafter connecting the models to form a system-model.

The paper presents a method for connecting component models to form complete system-models. The method includes handling of discontinuities and state shifts both locally in the components and system-wide. To handle the states of the components the concept of local, input and output states is introduced.

Connecting components to system-models generate extra equations and a method for reducing the connection variables is described. This method also handles variables that are carried unchanged through a component.

The paper describes the structure and features of the modeling and simulation program.

2 Terms used in this paper

The following table summarizes the terms used in this paper.

Term	Explanation
Dynamic variables	Variables that appears differentiated (with respect to time) in the equations. The symbol y will be used for dynamic variables.
Static variables	Variables that do not appear differentiated in the equations. The symbol x will be used for static variables.
Independent variable	y is differentiated with respect to the independent variable. Normally this equals time, and the symbol t is used for the independent variable.
Parameters	Quantities that are set to a constant before simulation. The symbol p will be used for parameters.
States	For example a valve may be in on of two states: Open or Closed. These logical states will normally change the equations describing the physical system. The number of logical states for a model is the number of different sets of equations used to define the model. The symbol s will be used for states.
Discontinuities	Discontinuities are a way to describe abrupt changes in the physical system that is modeled. For example, by describing the process of a valve suddenly closing as a discontinuity, one avoids describing in detail the valve position while it closes. A discontinuity indicates that the physical system shifts to another logic state. So describing abrupt changes as discontinuities involves a description of the possible states of the physical system. The conditions causing the change of logical state must also be formulated.

Initial value problems	A problem where the present state of the system is known, and the future state is to be determined. Problems that can be formulated as initial value problems can be solved by WinDali (note that initial value problems does not in general require that the independent variable is time).
DAE	Differential Algebraic Equation. An equation system that consists of coupled ordinary differential equations and algebraic equations, for example: $\frac{d y}{d t} = x$ $x = \sqrt{x + y + \ln(x)}$
Solver	The numeric code that solves the system of DAE's. The solver integrates the differential equations, solves the algebraic equations and handles discontinuities.

3 Introduction

WinDali is a modeling and simulation system for Microsoft Windows™ 95, 98, NT 4.0 or later. “Win” states that it is made for Microsoft Windows®, and “Dali” that it solves Differential Algebraic Equations (in Danish equations is “Ligninger”). Dali was originally a simulation system made for the DOS platform by Kim Askjær [1]. In WinDali only the original solver from Dali is unchanged, and a graphical user interface and a component modeling system has been added.

4 Program architecture

The figure below shows the structure of WinDali.

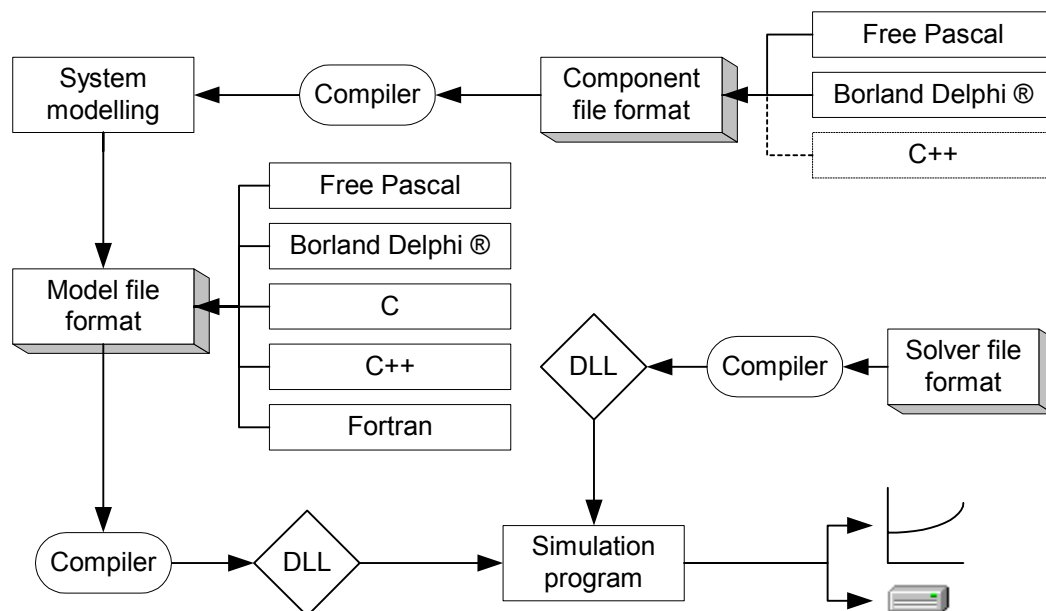


Figure 4.1. Structure of WinDali.

In Figure 4.1 there are four different types of boxes:

1. Square boxes indicate a computer program – that is an application with some sort of user interface.
2. Square boxes with shadows indicate a file format that has to be compiled with.

3. Rounded boxes indicate compilers, i.e. programs that convert source code to binary code.
4. Diamond shaped boxes indicating a Windows DLL (see later).

There are two points of attack when a model is created. Either the Component File Format or the Model File Format.

The Component File Format is for users who want to build system models using component modeling. Currently the components can be created in Free Pascal¹ or Inprise Borland Delphi™, but support for other languages is planned.

When the components have been compiled, they can be connected into systems in the System modeling program, and the result will be a model file, which complies with the Model File Format. Currently the System modeling application has not been developed, so the components have to be connected manually in the model file. Another limitation is that the language used to build the components (Free Pascal or Borland Delphi) also has to be used when the components are connected. This is due to the differences in the way programming languages implement objects.

The Model File Format can be used if component modeling is not being used. Because of this, non-objected oriented programming languages – like C and Fortran – can be used to create the model file. Examples of models created using Inprise Borland Delphi™, Free Pascal, Compaq Visual FORTRAN™ and Microsoft Visual C++™ are included with WinDali.

When the model has been created it is compiled to a Windows DLL (Dynamic Link Library). The concept of libraries is common for all programming languages. In Fortran they are called Modules, in C and C++ Libraries, and in Pascal Units. The difference between normal (static) libraries and dynamic link libraries is that a static library is linked to the program when it is compiled (and will be an integrated part of the executable), while dynamic link libraries are linked to the program at run-time. Figure 4.2 shows the difference between static and dynamic link libraries.

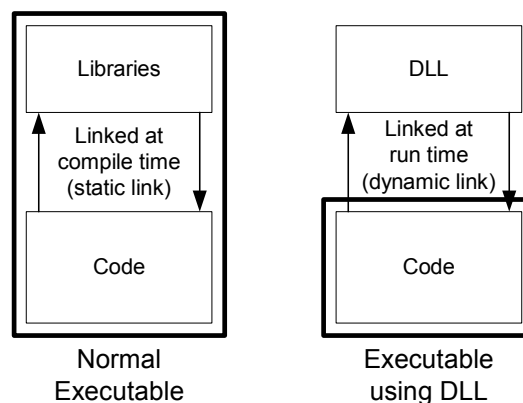


Figure 4.2. Difference between static and dynamic linking.

¹ Free Pascal is a freeware pascal compiler with support for object oriented programming [2]. Free Pascal is included with WinDali together with an integrated development environment (IDE) called Free Pascal Editor.

The advantage of using DLL's is that the Simulation Program is completely independent of the model, and that the model can be formulated in any programming language capable of creating Windows DLL's.

The model DLL can be loaded into the Simulation program, where simulation results can be plotted or saved to disk. The simulation program also enables automatic parameter studies.

As for the model, the equation solver is also loaded into the Simulation program at run-time. The solver should comply with the Solver File Format, but because the solver is also a DLL it can be programmed in any language capable of creating DLL's.

WinDali has the following features:

- Solves a system of semi-explicit differential algebraic equations.
- Solves initial value problems.
- Handles discontinuities and state shifts.
- Modeling is equation based. This implies that models are formulated by writing equations instead of graphically building the equations (as is the case in Simulink and VisSim).
- The user interface in the Simulation program is automatically generated from the information in the model DLL. This is possible because the model install itself into the Simulation program by calling a series of predefined functions. The same applies to the solver DLL, which also has the possibility of exposing parameters and settings to the user of the Simulation program.
- The structure of WinDali is open. As long as compliance with the Model File Format and the Solver File Format is ensured, any of the programs (square boxes in Figure 4.1) can be replaced.
- Any solver, capable of solving differential algebraic equations with discontinuities, can be used.
- It is easy to create distributable copies – it is simply a task of distributing the Simulation program, the model DLL and the solver DLL (this process is automated in the Simulation program).
- It is freeware.

The Model File Format, Component File Format and Solver File Format are explained in detail in [3], which serves as a user and programmers guide.

Compared to other modeling and simulation programs the advantages of WinDali are that the model is formulated in standard programming languages, which means that

- The modeler does not have to learn a new programming language.
- Debugging tools are already available.
- Using libraries with property data, mathematical routines etc. is simple.
- There is a complete freedom in creating the model – it can for example contain a specialized graphical user-interface.
- Component models can be in compiled form – i.e. you do not have to distribute the source code of component models.

5 Modeling environment

As stated previously WinDali is capable of solving systems of semi-explicit differential-algebraic equations (DAE's). This means you should be able to write the problem to be solved the following way:

$$\begin{aligned}\frac{d y}{d t} &= f(t, x, y, p, s) \\ 0 &= g(t, x, y, p, s)\end{aligned}\tag{5.1}$$

Problems, which can only be formulated as a system of implicit DAE's, cannot be solved by the current version of WinDali (but this is not due to the structure of WinDali but rather the capabilities of the included solver).

In equation (5.1) f denotes a vector of the right hand sides of the differential equations and g the static (algebraic) equations. Note that both the right hand sides and the static functions are functions of the current state s of the system. Furthermore WinDali allows the number of static equations (and thereby the number of static variables) to change between different states.

The equations constituting the model are specified in WinDali as a series of residual equations, implying that the equations are solved simultaneously. It also implies that WinDali does not require any causality in the model formulation, i.e. WinDali does not require the model to have an input-output structure.

The modeling environment that comes with WinDali consists of an IDE and a freeware Pascal compiler. Pascal was chosen as the primary language because the students at our department are familiar with Pascal.

The model is formulated by filling out 11 predefined Pascal-procedures in the model file. This is the case whether the model is formulated using component modeling or not (the only difference is that when using component modeling these 11 procedures are an integrated part of the component).

3 of the 11 procedures are of primary interest (i.e. the rest are optional): one where the problem is specified by specifying the number of states, the number of dynamic variables and the number of static variables in each state, one where the equations are formulated and finally one where the conditions for state-shifts are formulated.

The following figure shows the sequence in which the 11 procedures are called from the simulation program:

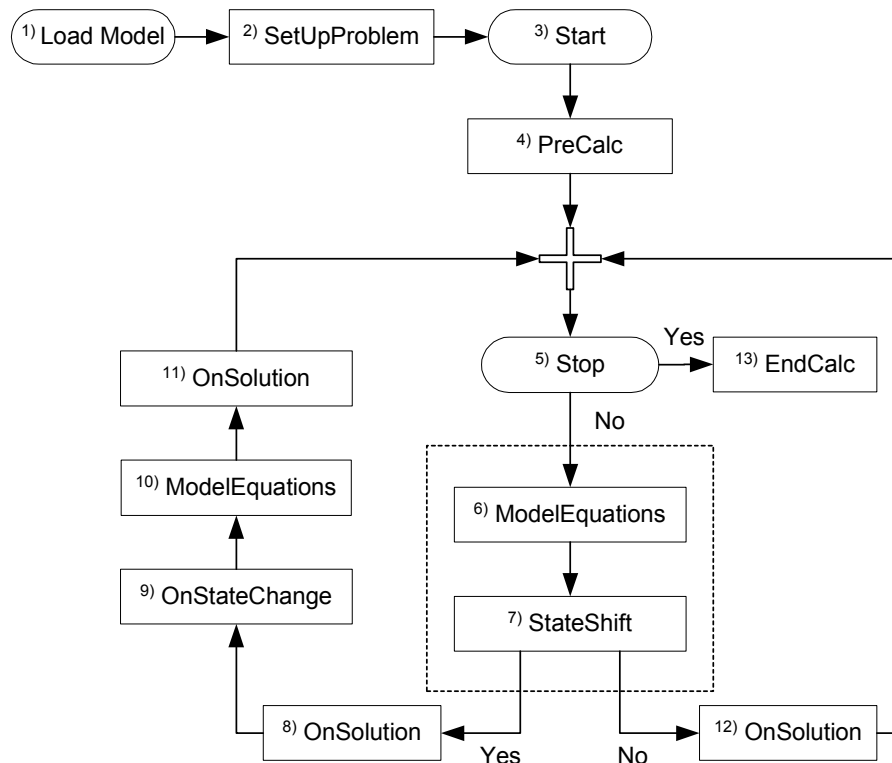


Figure 5.1 Calling sequence for the procedures in the model file

The rounded rectangles in Figure 5.1 represent user-actions in the Simulation program.

Note that the boxes 6 and 7 in general are called several times at each time step. This is because the solver iterates to find a solution. Four of the predefined procedures are not represented in Figure 5.1. The procedure **OnQuit** is called when the user closes the simulation program or selects another model. The procedures **OnUIValueChange**, **OnSaveSettings** and **OnLoadSettings** all have to do with controlling the user-interface. They enable the modeler to program actions, which are carried out in response to actions performed by the user of the simulation program (for example if the user presses a button, the model could display a specialized dialog, which enables the user to set some parameters).

The calling sequence with no state shift (branch 5-6-7-12) is straightforward. But the calling sequence with a state shift requires some explanation.

Lets say that we want to model a solenoid valve that is On when a temperature T is above 10°C , else it is Off. We want to record the temperature and the On-Off signal to the valve, and the initial temperature is 20°C . A plot of the temperature and the On-Off signal could look like this:

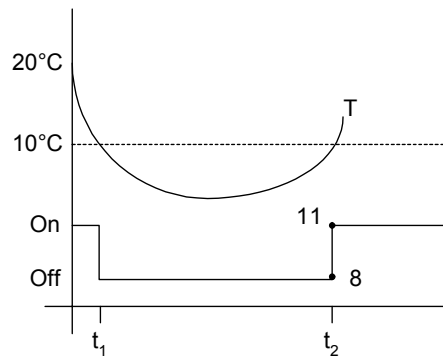


Figure 5.2 Temperature and On-Off signal.

When the solver reaches t_2 where the valve goes On, it first calls `OnSolution` (block 8) to inform the model file that a solution has been found. But at t_2 the equation system also shifts state, which means that the solver has to start all over, and possibly with a new set of static equations and corresponding variables.

The static equations have to be solved before the solver continues, and for this the solver needs guesses on the static variables.

In `SetUpProblem` default guesses on the static variables are given for each state, but if these guesses for some reason need to change, `OnStateChange` (block 9) is called. After `OnStateChange`, `ModelEquations` is called to solve the static equations in the new state, and before the solver continues, `OnSolution` is called to enable plotting of point 11 in Figure 5.2.

6 Component modeling interface

From the system point of view, a component is a black box, which can connect to other components and has some functionality.

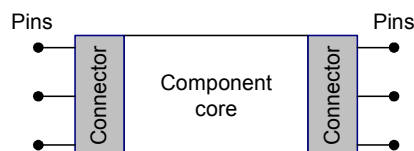


Figure 6.1. Conceptual component.

A component has a core, which defines the model of the component. It also has one or more connectors, which have one or more pins. A pin is just a connection variable – for example mass flow. Figure 6.1 shows a component that has two connectors with each three pins.

When a model of a component is created, one of the difficulties is to determine the number of equations that should be used to describe the component.

The simplest rule that can be used is to look at the number of pins on the connectors. If the connectors represent physical connections – as for example the suction and discharge port of a compressor – then the value of the variable that a pin represent could be measured on both sides of the component, and any change in a pin variable would

require an equation inside the component to describe that change. If on the other hand a pin value is unchanged over the component, then no equation is necessary for that pin.

The conceptual component in Figure 6.1 can be extended to indicate whether a connection variable is left unchanged while passing the component:

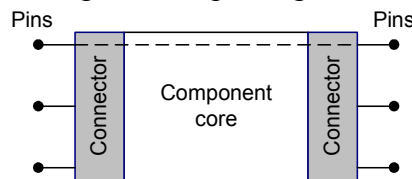


Figure 6.2. Component with variable carried unchanged through the component.

In Figure 6.2 the variable on the top pin is carried unchanged through the component. This is indicated with a dotted line.

Normally the pins act as pairs, that is the physical quantity on for example pin 2 on the left connector is the same as the physical quantity on pin 2 on the right connector. This is not always the case as can be seen from the following example:

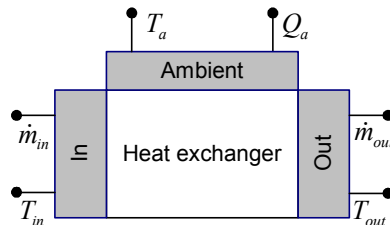


Figure 6.3. Heat exchanger example.

In the heat exchanger in Figure 6.3 the pins $\dot{m}_{in}$, $\dot{m}_{out}$ and T_{in} , T_{out} act as pairs. The connection to the ambient has no obvious pairs – the pins represent different physical quantities. A better way to draw the heat exchanger would be:

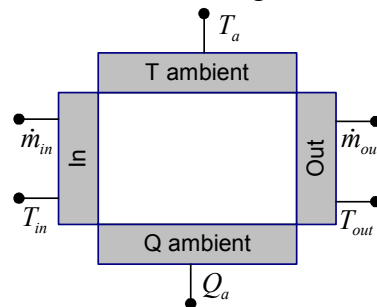


Figure 6.4. Heat exchanger with T-Q pair.

On this figure it is more clearly seen that Q and T to the ambient are related, and actually can be said to be a pair. An input/output description of the heat exchanger would be that it receives information about the ambient temperature and as output gives heat flow to the ambient. Defining the pins as pairs (in the following called connection pairs) is helpful when the necessary number of equations the component have is determined. Connection pairs do not necessarily have to be of only two pins. An example of this is a junction of three pipes, where the pressure of the three ports forms one pair and the mass flow of the three ports form another pair.

A rule of thumb that can be used when component models are created would then be:

Count the number of connection pairs – not counting connection variables that are carried unchanged through the component – this number equals the number of necessary equations.

6.1 Connecting components

When components are connected, the pins of one component are connected to the pins of another component and thereby creating connection equations. Connection equations are simply identity equations stating that:

$$Pin_{i,component1} = Pin_{i,component2} \quad (6.1)$$

Figure 6.5 shows an example where two components are connected. The components – C1 and C2 – each have two connectors with the names In and Out, and each connector has two pins named P and m. The connector names do not indicate that the components are connected according to some input/output scheme; they merely suggest that the component define a sign for the connection variables.

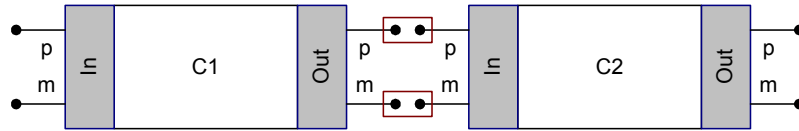


Figure 6.5. Two connected components.

The connection equations will be:

$$\begin{aligned} P_{C1,out} &= P_{C2,in} \\ m_{C1,out} &= m_{C2,in} \end{aligned} \quad (6.2)$$

Connection variables will almost always be static variables, and the connection equations will be static equations. The connection in Figure 6.5 will generate 2 static equations, but there are 4 static variables: $P_{C1,out}$, $P_{C2,in}$, $m_{C1,out}$, $m_{C2,in}$.

Counting the pins of components C1 and C2 gives 8 static variables, and assuming that $C1_{in}$ is connected to $C2_{out}$ then the connections will give 4 static equations. So there is still 4 equations missing.

These 4 equations have to come from the components and if the system were modeled as one equation system (that is no components) then these 4 equations would probably be the only static equations. This is because the connection equations are sort of dummy equations that could be replaced by a common variable:

$$\begin{aligned} P_1 &= P_{C1,out} = P_{C2,in} \\ m_1 &= m_{C1,out} = m_{C2,in} \\ P_2 &= P_{C2,out} = P_{C1,in} \\ m_2 &= m_{C2,out} = m_{C1,in} \end{aligned} \quad (6.3)$$

So now there is 4 static variables and 4 equations (that come from the components).

This reduction – assigning common variables to connections – is automated in WinDali. Every time two components are connected, the system assigns a common variable to each pin in the connection.

The second type of reduction that WinDali automates is when a variable is carried unchanged through a component, as illustrated in the following figure:

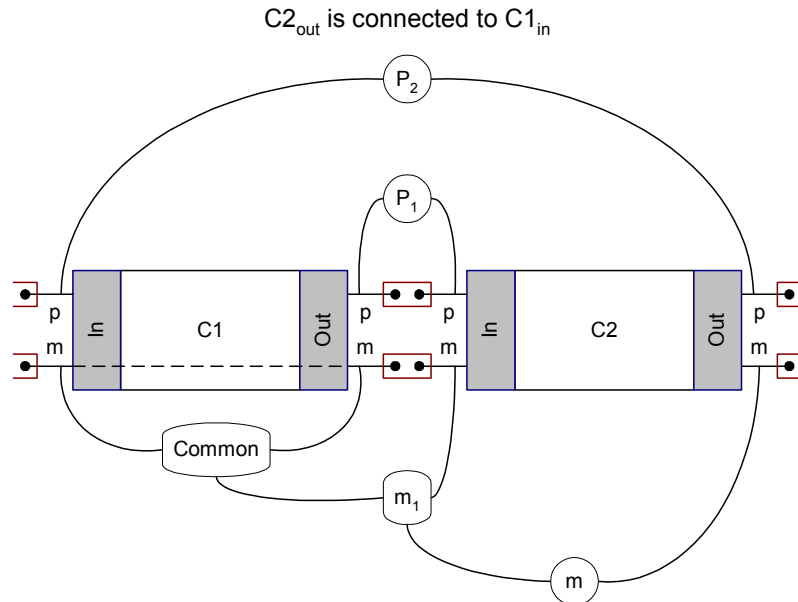


Figure 6.6. Reduction of variables. Note that $C2_{out}$ is connected to $C1_{in}$.

Figure 6.6 shows that when $P_{C1, out}$ is connected to $P_{C2, in}$ then the common variable P_1 is created, and that when $P_{C2, out}$ is connected to $P_{C1, in}$ then the common variable P_2 is created. When component C1 is created then it is immediately recognized that the m -pins of In and Out is equal, and a temporary common variable is created. When $m_{C1, out}$ is connected to $m_{C2, in}$ the common variable is replaced with m_1 . When $m_{C2, out}$ is connected to $m_{C1, in}$, the system recognize that $m_{C1, in}$ already has a variable assigned and m_1 is replaced by the variable m , which is also equal to $m_{C2, out}$.

The original system of 8 equations with 8 static variables is now reduced to 3 equations with 3 static variables: P_1 , P_2 and m .

As stated before the components are responsible for providing the three static equations that make it possible to solve the equations. But as the system handles the connection variables, the components should not regard the connection variables as variables owned by themselves – they should only provide equations for them.

When the components defines a sign for the connection variables, it is necessary to distinguish between two types of connection variables²:

1. Flow variables, which are variables that can have a direction – for example mass flow, heat flux and current.
2. Potential variables, which are variables that cannot have a direction – for example pressure, temperature and voltage.

² Flow variables are sometimes called “through” variables, while potential variables sometimes are called “across” or “effort” variables.

The variable reduction scheme requires that the system has complete control over the connection variables. This implies that each component has to define, which pins represent flow variables, and which direction a connector defines as positive for flow variables (either positive into the component or positive out of the component). If for example the Out connector of the C1 component in Figure 6.5 is connected to the Out connector of C2, and the mass flows from C1 to C2, then the mass flow into C2 would have the wrong sign (be positive), if the system did not realize that the common variable for the connection should change sign when the equations in C2 is called.

In the current implementation of the component system used in WinDali, it is not possible to specify that a connection variable is carried unchanged through a component in one state, but is not in another. This implies that if for example a pipe component defines a residual equation for the pressure drop in a situation where there is flow through the pipe, then it also has to define a residual equation for the pressure drop in a situation where there is no flow (i.e. a residual equation stating that pressure into the pipe is equal to pressure out of the pipe).

6.2 Handling states

Handling states is a bit more difficult than handling the connection of components and the connection variables.

In energy systems, shifting between states is normally initiated by the control equipment. If for example a thermostat is controlling a refrigeration plant, then the thermostat sets the compressor On or Off according to the temperature limits set in the thermostat. This kind of state shifting is recognized intuitively. But there might occur other types of state shifts in the system. If for example an evaporator is modeled, then the model might be designed in such a way that the equation set shifts when the fluid in the evaporator shifts from being two-phase to gas. This is an internal or local state shift in the evaporator; but as the solver only operates with system states it also changes the state of the system.

In WinDali there are three kinds of states:

1. Local states
2. Output states
3. Input states

Local states are, as in the evaporator example above, states that are local to a component. These states will change the state of the system, but other components will not be affected by the state change.

Output states are local states that can affect other components. If for example the thermostat from before sets the compressor Off, there might be other components in the system that requires to know that the compressor is Off – i.e. the Off state signal from the thermostat is an output state. It is important to note that the nature of state-shifting signals is input/output. The shifting of a state is controlled by one component and when it shifts this information can be used by other components.

WinDali supports output states to be combined by several local states. If for example a pump and its controller is modeled as one component, and the pump has two capacity levels, then the pump might define 3 local states:

1. Capacity level 1
2. Capacity level 2
3. Off

Other components might only be interested in whether the pump is On or Off, so state 1 and 2 are combined into one output state called On.

Input states are states of a component, but they are defined by other components. Taking the evaporator example from before then the evaporator might require knowledge of whether the compressor is On or Off (besides having the two local states TwoPhase and Gas). The evaporator component can specify that it has two input states, On and Off, and when the system model is created then the output states from the thermostat can be connected to the input states of the evaporator.

When a component has input states the total number of states it can be in changes. If for example the evaporator defines the two local states and the two input states, it can (logically) be in four states³:

1. TwoPhase, On
2. Gas, On
3. TwoPhase, Off
4. Gas, Off

But as the evaporator only defines 2 of the 4 states, it is not allowed to specify state shifts from state 1 or 2 to state 3 or 4 (or reversed)

Input states may be set as a result of several components shifting state. Consider the following example:

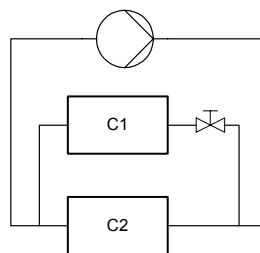


Figure 6.7. Type of input state.

Component C1 in Figure 6.7 has two input states, On and Off, which should be set if there is flow or no flow to the component. C2 is just a restriction – for example a pipe. The pump specifies two output states, On or Off, to indicate whether it is running or not, and the valve also specifies two output states, On or Off, to indicate whether is open or closed. The output states in the valve and the pump are both connected to the input states of component C1. The system can be in the following states:

³ If the evaporator defined three local states it could be in 6 states – the number of local states times the number of input states.

System state	Pump	Valve	C1
1	On	On	On
2	On	Off	???
3	Off	On	???
4	Off	Off	Off

Table 6.1. Unresolved input states.

The states 2 and 3 are indefinite, as C1 does not know which state it should be in when both its input states are set. This issue is resolved by requiring that an input state has a type – it can either be an OR state or an AND state. If the input state is OR then if just one component connected to the state tries to set it, then it will be set. If the input state is AND then all components connected to the state have to set it, before it will be set. In the example above C1 should specify that the input state On is an AND type, and that the input state Off is an OR type. In that case the system can be in the following states:

System state	Pump	Valve	C1
1	On	On	On
2	On	Off	Off
3	Off	On	Off
4	Off	Off	Off

Table 6.2. Resolved input states.

When the system model is created by connecting components, WinDali checks each component to generate the total number of states of the system.

7 Conclusions

WinDali is designed as an open-structured modeling and simulation package. This means that the modeling interface, the simulation interface and the numerical solver has been split into separate modules, which allows any of the modules to be replaced as long as new modules comply with the specified file formats.

The modeling language used in WinDali is based on existing standard programming languages. This has the advantage that the modeler has complete freedom in choosing the features the model should possess. On the other hand, the model file format is structured in such a way that people with a minimum of programming experience are able to use it. The component file format can be used without having knowledge of object oriented programming, but it is more complex than the modeling file format.

The component file format is used to build models of the components in the system. When this has been done, it is a simple task to connect the components and create the system model using the model file format. This does not mean that there is not a need for a graphical user interface where the components can be connected by simply drawing the connections, but this is left as suggestion for future work.

One of the main features of WinDali is that the user interface of the simulation program is created from the specifications in the model. Creating a graphical user interface is normally a very time consuming task, but when the interface is generated from the model, more time can be spent developing the model. The graphical user interface also makes parameter studies and keeping track of simulation cases easy.

The following figures show screendumps from the modeling and the simulation programs.

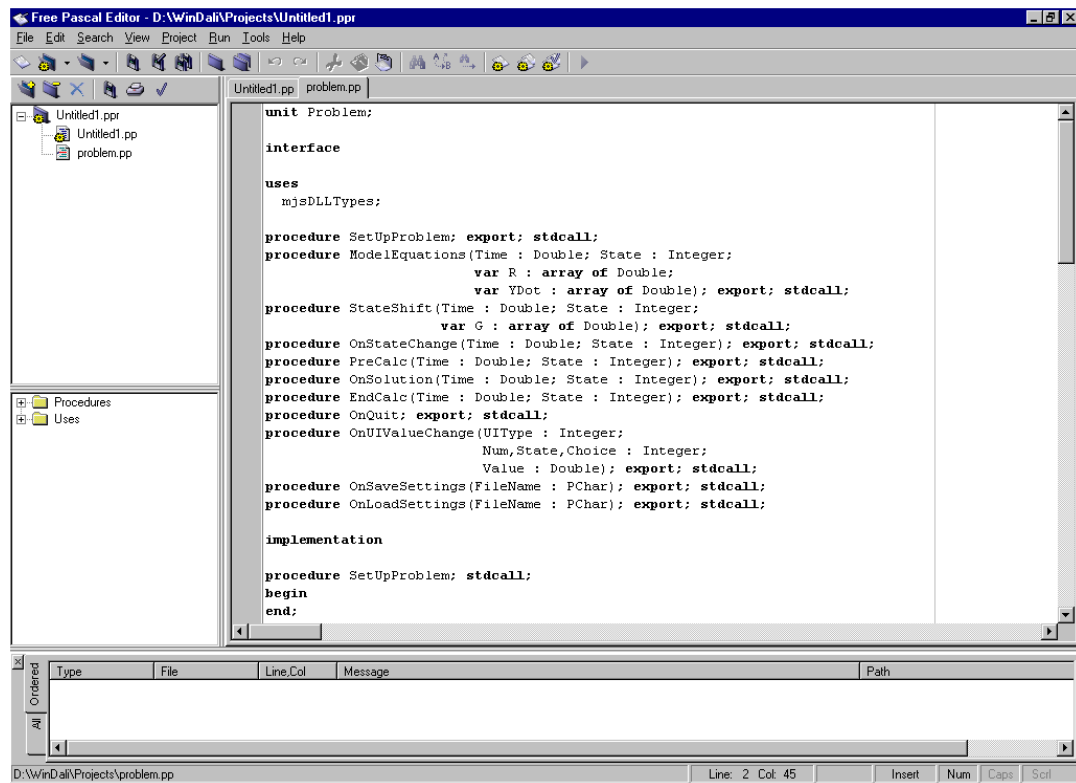


Figure 7.1 Screenshot from Free Pascal Editor.

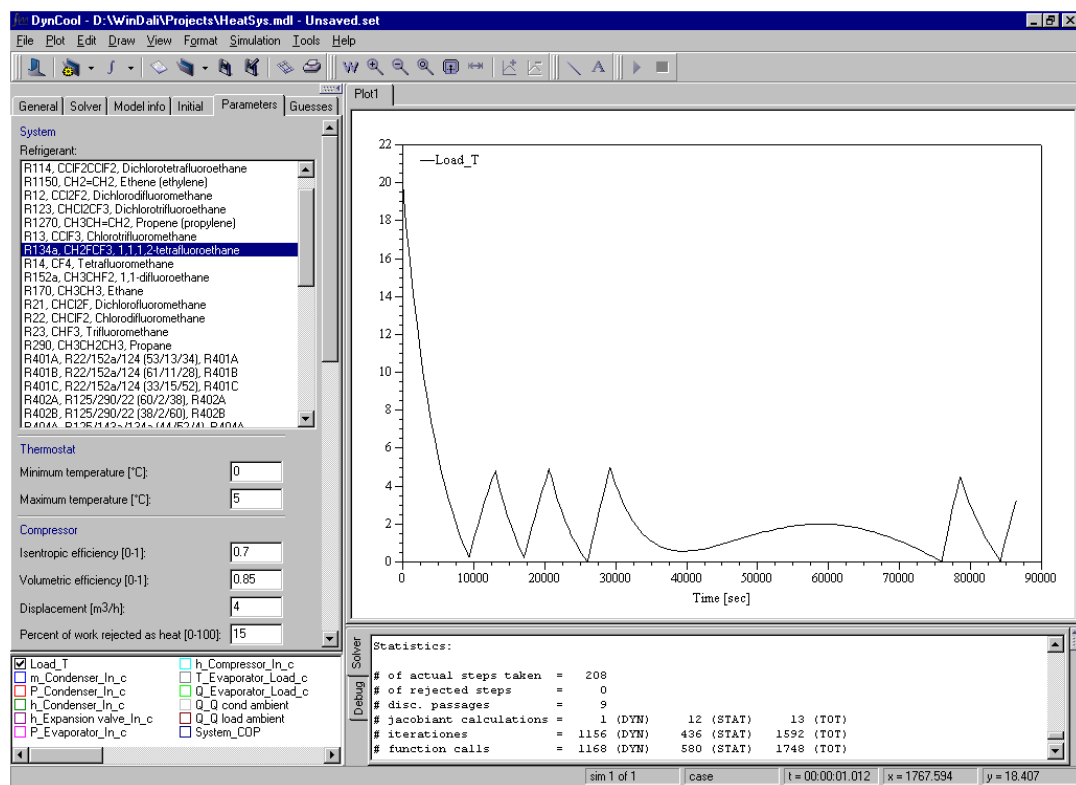


Figure 7.2 Screenshot from Simulation program.

8 References

- [1] Askjær, Kim Friis – *Brugervejledning til DALI. En differential-algebraisk ligningsløser*. Danmarks Tekniske Højskole, Laboratoriet for Køleteknik, F94-02. Januar 1986.
- [2] Free Pascal Compiler, <http://www.freepascal.org/>
- [3] Skovrup, Morten Juel – *WinDali – A Modeling and Simulation System for Microsoft Windows*. Technical University of Denmark, Department of Energy Engineering, ET-Ph.D. 2000-04. April 2000.
- [4] Skovrup, Morten Juel – *Systematic modeling and simulation of refrigeration plants*. Technical University of Denmark, Department of Energy Engineering, ET-Ph.D. 2000-04. April 2000.