17th IEEE International Conference on Control Applications
Part of 2008 IEEE Multi-conference on Systems and Control
San Antonio, Texas, USA, September 3-5, 2008

Modular Modelling and Simulation Approach - Applied to Refrigeration Systems

Kresten K. Sørensen[a,b] and Jakob Stoustrup[a]
[a]*Section for Automation and Control, Department of Electronic Systems*
Aalborg University, Fredrik Bajers Vej 7C, 9220 Aalborg, Denmark
[b]*Lodam electronics a/s, Kærvej 77, 6400 Sønderborg, Denmark*
E-mail: kresten@es.aau.dk jakob@es.aau.dk

***Abstract*— This paper presents an approach to modelling and simulation of the thermal dynamics of a refrigeration system, specifically a reefer container. A modular approach is used and the objective is to increase the speed and flexibility of the developed simulation environment. The refrigeration system is divided into components where the inputs and outputs are described by a set of XML files that can be combined into a composite system model that may be loaded into MATLAB™. A set of tools that allows the user to easily load the model and run a simulation are provided. The results show a simulation speed-up of more than a factor of three by partitioning the model into smaller parts, and thereby isolating fast and slow dynamics. As a cost there is a reduction in accuracy which in the example considered is less than one percent.**

I. INTRODUCTION

Numerical simulation is extensively used for experiments within the field of control engineering, and with the increasing power of computers it has become possible to simulate very large dynamical systems on a normal desktop PC at reasonable speed. But larger system models results in larger and more complex equation sets that are difficult to handle and therefore a range of simulation tools capable of handling large system models is available. When choosing a simulation tool it is worth considering the ease of modelling, simulation speed and accuracy because these parameters vary from tool to tool.

A common tool utilized within control engineering is SIMULINK™ [5], which allows the user to create and simulate large models from built-in or user developed component libraries through a Graphical User Interface (GUI). The simulation model composed of component models may be solved by a range of block oriented input/output solvers that automatically adjusts the size of the numerical integration step. SIMULINK™ variable-step solvers change the step size during simulation [10], reducing the step size to increase accuracy when the states of a simulation model are changing rapidly and increasing the step size to avoid taking unnecessary steps when the models states are changing slowly. Computing the step size adds to the computational overhead at each step but can reduce the total number of steps, and hence simulation time, required to maintain a specified level of accuracy for models with rapidly changing or piecewise continuous states. The selected integrations step is however inherited by blocks further down the signal path and if the components further down the path have slow dynamics compared to the preceding blocks they will be simulated with unnecessarily small integration step sizes, leading to a computational overhead. Such a system is referred to as stiff and while MATLAB™ and SIMULINK™ have specific solvers for these types of problems they do not address the problem with computational overhead in stiff systems.

Another approach is used by DYMOLA™ [7] that implements the MODELICA [6] language which is an equation-based object-oriented modelling language. In MODELICA symbolic equations that define the dynamical behavior of a component may be entered in a non-causal way, leaving the task of ordering and reducing the final set of equations to the simulation engine before a simulation can be run. Stiff problems may be solved using implicit methods that allow larger step-sizes at the cost of solving a set of non-linear equations at each time step. DYMOLA™ uses mixed-mode integration [9] that takes a middle course where the system is split up into fast and slow states. Only the fast states are discretized implicitly leaving a smaller set of nonlinear equations to solve at each time step and thus a faster simulation. To speed up the simulation even further inline integration [8] is supported. The discretization formulas are inserted (in-lined) into the problem and DYMOLA™'s symbolic engine is applied to the resulting equations [9]. The automatic model reduction and partitioning approach used by DYMOLA™ does not, however, take advantage of a-priori system knowledge that already exists.

In this paper an attempt was made to build a small and fast simulation environment that provides easy modular modelling and rapid prototyping of refrigeration systems from a library of component models. A modelling and simulation environment for MATLAB™ has been developed to enable simulation experiments on nonlinear models consisting of a mix of models based on Ordinary Differential Equations (ODE), Differential Algebraic Equations (DAE) and purely algebraic equations. The objective of the tool is to provide rapid and flexible refrigeration system model development from a predefined set of refrigeration component models. A refrigeration container model has been chosen as an example for this study.

This work was supported by Lodam electronics a/s and the Ministry of Science, Technology and Innovation

Refrigeration containers are used to move many different types of cargo between all areas in the world and this puts some unusual requirements on the refrigeration system with respect to the temperature range on both the cold and the hot side. The goods transported may require a stable temperature between -30C° and +20C° and the temperature of the air around the container can be between -30C° and +50C°. Because of the nonlinear nature of the refrigeration system and the large temperature range it is infeasible to use a linear model for simulation experiments and therefore a nonlinear model must be used. The requirements for such a model is that it should match the real system close enough to be used for closed loop control experiments and simulations should run at least one order of magnitude faster than experiments on the real system. Another important requirement is that it must be possible to change the model configuration without having to rewrite or reorganize the entire set of equations for the model by hand. Therefore a modular approach is selected where the model of the refrigeration system is composed of a set of interchangeable component models that are based on first principles and assumptions where appropriate.

A model of a refrigeration container is developed and used as a test case for the simulation environment. The system has both fast and slow states but the fast states are isolated in a single component leading to a potential speed-up of the simulation with a modular approach. Therefore it was attempted to decouple component models with slower dynamics from models with fast dynamics by simulating each of the component models separately and only exchange input/output values between the component models at fixed discrete times. The number of steps and average step time for the solvers for each of the components is used to calculate the difference in simulation time of the example system, as either a modular model or a monolithic model.

II. Modelling

The model of the refrigeration system is divided into components that each represent a physical component of the system, i.e. a condenser or an evaporator. Each **component model** is described by two files; an `m` file that holds the input/output equations of the model and an `XML` file that describes the properties of the inputs and outputs of the `m` file. The **simulation model** is the overall model for the refrigeration plant and its properties are described by an `XML` file that holds a list of included component models and the connections between them. Therefore the structure of the simulation model is defined by the simulation model definition file, and from this the model loader can create a simulation object that is used by the simulator.

A. Component Model Syntax

Modelling of component models is basically the same as for normal ODE model functions that may be solved by MATLAB™'s built-in `ode` solvers, but additional info is needed by the model loader in order to do type-checking when connecting the inputs and outputs of the component models. Each component model is described by an `m` file containing the input/output equations, an `XML` file describing the input/output properties of the model, its execution mode, and the name of the corresponding `m` file. The syntax for writing the component model `XML` file is shown in the box below:

Component Model Syntax

```
 <?xml version="1.0" encoding="UTF-8"?>
<component name=["Component Name]">
<inputs>
  <input name=["Input1 Name"] type=["Input1 Type"]
    description=["Input1 Description"]/>
  .
  .
  <input name=["InputN Name"] type=["InputN Type"]
    description=["InputN Description"]/>
</inputs>

<states>
  <state name=["State1 Name"] type=["State1 Type"]
    default=["State1 Start Value"]
    description=["State1 Description"]/>
  .
  .
  <state name=["StateN Name"] type=["StateN Type"]
    default=["StateN Start Value"]
    description=["StateN Description"]/>
</states>

<connectors>
  <connector type=["type"] conid=["Conn. Name"]>
    <{input, state} name=["Name"]
      type="Physical Entity"/>
    <{input, state} name=["Name"]
      type="Physical Entity"/>
   </connector>
  .
  .
</connectors>

<control_inputs>
  <input name=["Input Name"] type=["Input Type"]
    description=["Input Description"]/>
  .
  .
</control_inputs>

<simulation method={"ode15s", "call"}
  call=[".m File Function Name"]/>
<filename>[.m File Name]</filename>
</component>
```

The `<inputs>` section contains a list of the inputs to the component model, and it is important that the inputs are listed in the same order as they occur in the input vector of the model function. Each input has a name, a type used for type-checking when inputs are connected, and a description. There must be the same number of inputs in the input list as the length of the input vector of the corresponding model function. The `<states>` section is similar to the input section except that it describes the states or outputs of the function and that a default value must be declared. The default value is used as initial value in simulations when the simulation tool is not given an initial state vector to start from. Signals may be grouped together in connectors that allows the user to connect a set of signals from one component model to a similar set on another component model in one operation.

Because this environment is used for refrigeration systems it has a built-in connector class for refrigeration pipe interfaces but obviously, for other applications, other connections

will be relevant - see Simulation Model Syntax in Subsection II.B. On the refrigeration pipe interface three variables exist; a mass flow $\dot{m}$, a pressure p, and an enthalpy h. The model loader will return an error if each refrigeration pipe interface does not contain exactly one of each of the mentioned aforementioned types. The individual variables may be either an input or a state but the model loader will check that each of the inputs can be connected to a state of the correct type when two refrigeration pipe interfaces are connected. This saves the user the work of having to connect the variables manually, but when building the component models attention must be given to where the different states that are shared between models are located.

B. Simulation Model Syntax

The syntax for the simulation model `XML` file are listed below:

Simulation Model Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<simulation_model name=["Simulation Model Name"]>
<component_path>[Component Library Path]
</component_path>

<components>
  <component name=["Component Model Name"]
    file=["Component Model XML File Name"]/>
  .
  .
  <component name=["Component Model Name"]
    file=["Component Model XML File Name"]/>
</components>

<connections>
  <connector name=["Connector Name"]>
    <component name=["Component Name"]
      conid=["Connection Name"]/>
    <component name=["Component Name"]
      conid=["Connection Name"]/>
  </connector>

  <connection name=["Connection Name"]>
     <component name=["Component Name"]
       type="Input" input=["Input Name"]/>
     <component name=["Component Name"]
       type="Output" output=["Output Name"]/>
  </connection>

</connections>
</simulation_model>
```

The simulation model is composed of a set of component models and their connections described by the simulation model `XML` file, containing a list of the included component models and a description of how the components are connected. The `<components>` section lists the component models used in the simulation model and it is allowed to use a component model more than once if they are given unique names. In the `<connections>` section all the connections in the simulation model are listed. A connector connection is established as in the `<connector>` sections by giving the name of the two component models and the connector on each of the components. Inputs and outputs that are not associated with connector interfaces, such as control inputs to actuators, are connected in a `<connection>` section by listing the two components one by one. In each of the `<component>` sections it must be stated whether the signal is an input or an output, and what the name of the signal is. The model loader provides type checking on connections between components and gives a precise description, with the names of the implicated components and signals, in case of an eventual error in the set of connections between components.

The model may be loaded into MATLAB™ with a model loader function that loads each of the components and creates a struct containing all the information necessary for simulation. During simulation the states of the component models are kept in a single vector, denoted $\mathbf{X}$, and therefore two matrices that maps between $\mathbf{X}$ and component model I/O are generated for each component model. The matrix $\mathbf{Z}_n$ maps from $\mathbf{X}$ to the component model x-vector $\mathbf{x}_n$ such that

$$\mathbf{X} = \mathbf{Z}_n \cdot \mathbf{x}_n \quad (1)$$

$$\mathbf{x}_n = \mathbf{Z}_n^T \cdot \mathbf{X} \quad (2)$$

where the index n denotes the component model number. Component model inputs $\mathbf{u}_n$ are mapped from $\mathbf{X}$ by the matrix $\mathbf{CM}_n$ such that

$$\mathbf{u}_n = \mathbf{CM}_n \cdot \mathbf{X} \quad (3)$$

Because $\mathbf{CM}_n$ maps from state variables that reflect a physical value to inputs that take a physical value, it is a one-zero matrix and must have exactly one ”1” in each row. When the model loader has created the mapping matrices they are used to check the connection integrity of the system model such that all inputs are connected to exactly one state.

III. Simulation

Simulation of the system model is done in discrete time steps defined in the time vector given in the simulation function call. In each of the time steps the component models are simulated separately, according to the method defined in the component model `XML` file and the results are then combined into the X vector containing the states for all of the component models. Component models that are purely algebraic are evaluated in one operation like a normal MATLAB™ function, and dynamical models are solved by one of MATLAB™'s built-in `ode` functions. Fig. (1) shows a possible structure for a simulation model consisting of four component models.

The discretization of the signals between the component models is equivalent to inserting a zero order hold between all models as shown on Fig. (1).

The input to the individual component models are calculated by applying (2) and (3) and given as arguments to the appropriate simulation function such that

$$\mathbf{x}_n(k) = f_n([\mathbf{t}(k-1) \quad \mathbf{t}(k)], \quad \mathbf{Z}_n \cdot \mathbf{x}_n, \quad \mathbf{CM}_n \cdot \mathbf{X}) \quad (4)$$

where f_n is the simulation function for the referenced component model and $[\mathbf{t}(k-1) \quad \mathbf{t}(k)]$ is the time interval in which to simulate. The results from the each of the component simulations are then combined into the system state-vector X by

$$\mathbf{X}(k) = \sum_{n=1}^{N} \mathbf{Z}_n \cdot \mathbf{x}_n(k) \quad (5)$$

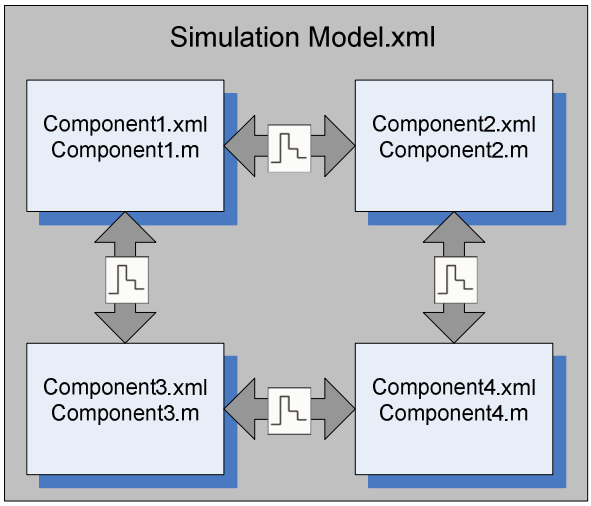

Fig. 1. Example of a Simulation Model Structure

where N is the total number of components in the model. The simulation environment may then proceed and simulate the next time step with the same procedure as above.

IV. Reefer Model

The refrigeration system used as an example here utilize an economizer to increase the efficiency of the system at high pressure differences between the cold and hot side. The compressor efficiency is lower at high pressure differences which is exist when there is a large difference between the evaporation and condensation temperatures.The economizer arrangement increases the refrigeration capacity and improves the coefficient of performance (COP) [11]. A schematic of the system can be seen in Fig. 2. The elements of the model are described in the sequel. The explicit 44 equations of the model are not derived here due to space limitations, but they are available in [12].

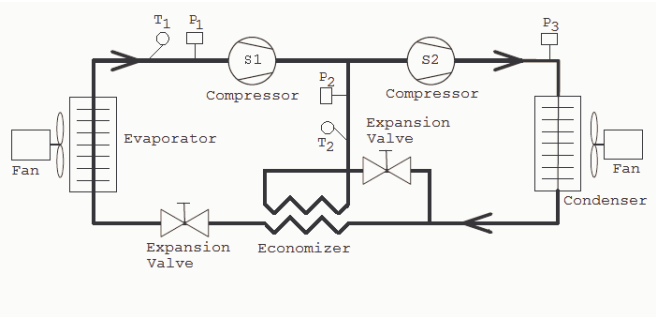

Fig. 2. Refrigeration System for a Reefer Container

The system consists of a two-stage piston compressor, a condenser, an evaporator, and an economizer [11] which is a counterflow plate heat exchanger. The compressor is equipped with a frequency converter which enables it to run at variable speed. The expansion valves are electromagnetically pulsed on/off valves and the fans may run at half speed, full speed, or be turned off entirely. There are three major pressure levels in this setup; p_1 is the evaporator pressure, p_2 is the intermediate pressure between the two compressor stages, and p_3 is the condenser pressure. Since p_2 is also the evaporation pressure on the cold side of the economizer it is coupled closely to the evaporation temperature of the economizer, and hence influential on the inlet temperature of the refrigerant to the evaporator.

According to [1] the dominant dynamics of a refrigeration system are the thermal time constants of the metal surfaces in the heat exchangers and refrigerant mass time constants, with respect to control applications. Also according to [1] some of the components have dynamics that are so fast compared to the dominant dynamics that they may be replaced with algebraic equations, thus reducing the model order while preserving the physical behavior of the model on the dominant dynamics. The components that may be modelled algebraically are the expansion valves and the compressors while the rest of the components are modelled using first principles or assumptions. Two pipe junction models are needed in order to model the joining and splitting of refrigerant flows that occur between the compressor stages and after the receiver, respectively.

A. Pipe Joining Junction

The pipe junction model has three states; Pressure p, internal mass M, and output enthalpy h_{out}. It also has five inputs; the mass flows on all three interfaces $\dot{m}_{in1}$, $\dot{m}_{in2}$, $\dot{m}_{out}$, and the input enthalpy for both refrigerant inputs h_{in1} and h_{in2}. The pressure of this component model has fast dynamics because it is a small volume containing vapor with a high mass flow and no boiling liquid to dampen pressure oscillations and therefore the exact dynamics are neglected and the absolute pressure simply calculated instead.

B. Compressor

The compressor has two almost identical stages where the only difference is that the displacement volume of the first stage is twice as large as that of the second stage. The compressor stages are modelled by two algebraic functions giving the mass flow and output enthalpy as a function of input pressure, output pressure, input enthalpy, the speed of the compressor, and the temperature of the compressor. The compression is assumed adiabatic and the physical behavior of the model includes harmful volume, and valve pressure loss. The mass of the refrigerant in the compressor is neglected and therefore the mass flows on the input and output are equal.

C. Expansion Valve

The expansion valve model is purely algebraic and modelled as a continuous valve giving the average mass flow of the electromagnetically pulsed on/off valves used on the reefer. A lookup table is used to find the mass flow at full opening as a function of the pressures on both sides and this is multiplied with the ON time, which is the fraction of time it is turned on. The expansion is assumed adiabatic end therefore there is no change to the enthalpy of the refrigerant.

D. Economizer

The hot side of the economizer is filled with liquid refrigerant running from the receiver to the evaporator expansion valve and is therefore modelled as a single region with uniform heat transfer from liquid to metal. The liquid volume is assumed to have uniform pressure and enthalpy and a constant pressure drop from input to output. The cold side, where refrigerant evaporates, is modelled as a single volume where the amount of energy transferred from the metal to the refrigerant is dependent on the difference in temperature between the refrigerant and the metal walls. The thermal capacitance of the metal walls acts as a damper on the dynamics and it is therefore included in the model.

E. Evaporator

The evaporator is modelled as in [2] which is a lumped model with a moving boundary between the two phase and the vapor volume.

F. Condenser

The condenser is modelled as in [3] and [4] which is a lumped model with a moving boundary between the two-phase volume and the vapor volume.

G. Receiver

The receiver is a buffer tank for excess refrigerant. The refrigerant is led from the condenser into the top of the receiver and liquid refrigerant to the expansion valves are taken from the bottom. The receiver is usually either neglected in dynamical models of refrigeration plants or not existing in the modelled plant. It is, however, not entirely without influence on the refrigeration system's dynamics, especially in startup situations and during fast pressure changes. The reason for this is that the liquid in the receiver acts as a buffer and has a dampening effect on pressure transients from the condenser, but this can also lead to problems with vapor bubbles in the feed line to the expansion valves which severely degrades the mass flow. The liquid in the receiver that goes to the expansion valves may be either sub-cooled or at the boiling point. If the liquid starts to boil it will turn into a two phase mixture of liquid and vapor with a quality that depends on how much it is boiling. When the condenser fan is switched on, the pressure in the receiver can drop rapidly, and if the temperature of the liquid in the receiver is close to the boiling point it will begin to boil until the temperature drops below the boiling point.

H. Box

The box is the largest thermodynamic capacitance in the reefer due to its mass, but in this example an empty container has been used and therefore the thermodynamic capacitance consists mainly of the aluminium T-floor and the air inside the container. Energy is exchanged by air circulating from the evaporator, over the floor, and back to the evaporator again along the sides and roof of the container. The air is heated by energy leaking through the insulated walls, floor and roof of the container.

V. Results

A. Simulation Speed

An experiment has been carried out in order to measure the increase in speed gained by simulating the system as separate component model functions instead of a large single-function model. The reefer model used in the experiment has 17 discrete and 35 continuous states and are divided into 10 component models, where four are purely algebraic and the remaining six are continuous. A simulation of a 4000 s period has been carried out using a laptop equipped with a 2.0 GHz Core 2 Duo processor and 2 GB of RAM. The discrete time step size of the simulation environment was set to one second and the simulation completed in 156.8 s, i.e., 25.5 times faster than real time experiments.

Fig. 3 shows the time used to simulate each of the components at the discrete time steps during the 4000 s simulation, and Fig. 4 shows the number of simulation steps used by each of the components for each discrete time step.

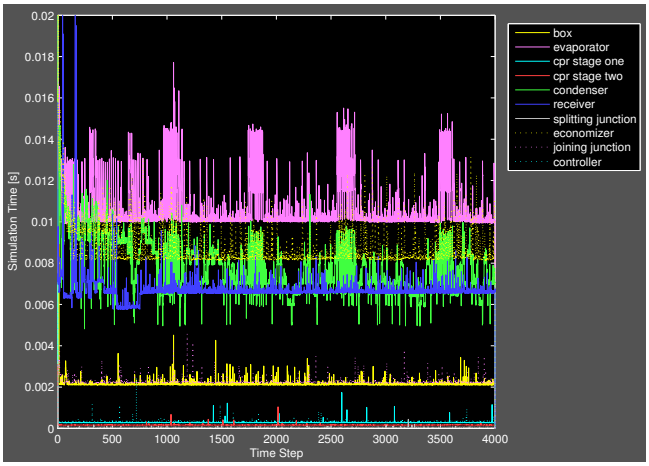

Fig. 3. Simulation Time for Each of the Components

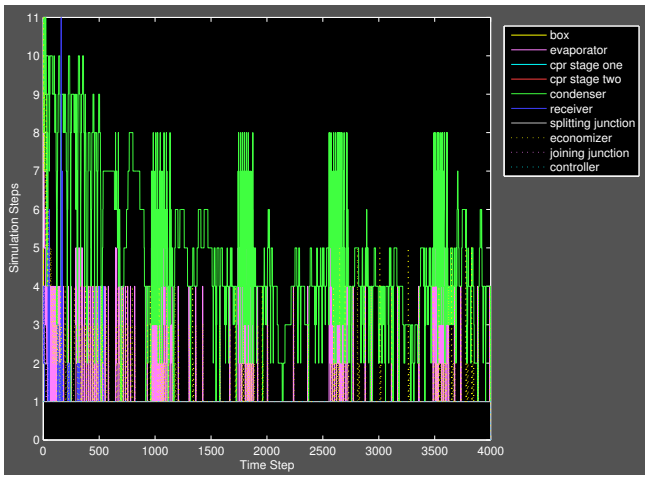

Fig. 4. Number of Simulation Steps for Each of the Components

The time, T_{uni}, that it would have taken a solver to simulate the model if it had been unified into a single, monolithic,

function is calculated using the total number of simulation steps and the average time for a single simulation step for each model. The total number of simulation steps S_n is the number of times that a model component function has been called during the entire simulation, either by the ODE solver if it it is continuous or directly by the simulation environment if it is algebraic. The average time for a single simulation step t_n is found by

$$t_n = \frac{T_n}{S_n} \quad (6)$$

where T_n is the total time that the CPU has spent solving a particular model during the entire simulation.

The time saved by simulating in smaller components has been calculated by averaging over the entire simulation period as in

$$T_{uni} = \sum_{n=1}^{N} S_{max} \cdot t_n x \quad (7)$$

where N is the total number of component models, S_{max} is the number of simulation steps used by the component model that used the most simulation steps during the entire simulation. Equation (7) yields a total simulation time of 528.7 s for a monolithic model which corresponds to a speed-up of

$$\frac{528.7s}{156.8s} = 337.2\% \quad (8)$$

for the modular model compared to a monolithic model. The average error of the modular simulation is 0.734%, relative to a simulation of the same model unified into a monolithic function, which is acceptable when considering the improvement in simulation speed.

VI. DISCUSSION AND FUTURE WORK

A. Discussion

There are both benefits and drawbacks to the modular approach; it is up to the user to determine a suitable size of the discrete time steps with respect to the fastest dynamics in the set of component model states that are used as inputs to other component models. It is however possible for a component model to have fast internal dynamics, that is, a state that is not used as input to other component models and thereby discretized by the simulation environment.

The separation of the component models, however, makes it possible to simulate systems with both fast and slow dynamics faster than it is possible with a unified system model, where all the equations of the composite model are rewritten as one function that can be used for simulation. The reason for this is that a solver for a unified model would need to evaluate all the equations for the entire model for each of the time steps, which would be sized with respect to the fastest dynamics of the unified model. By simulating the system as separate component models it is possible to have one or more components with fast internal dynamics that are simulated using smaller time steps than is necessary for components with slow dynamics. This results in an overall reduction in calculations needed to simulate the system for a given period of time and thereby a faster simulation.

The decentralized nature of the simulation has a potential cost on the achieved accuracy of the simulation result due to the sequential computation. On the other hand solving smaller algebraic equations might have a positive influence on the accuracy of the result. In fact, in some cases, a numerically infeasible simulation might be rendered feasible by decentralization for some systems.

B. Future Work

A mathematical formulation that describes the implications that the discretization has on the accuracy of the simulation results and provides a guideline, or even automatic identification of the largest possible discrete integration step that can be used for a given model without exceeding the target accuracy of the simulation.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the assistance from Kim Madsen, Maersk Container Industri and Lars Mou Jessen, Lodam electronics a/s, for counselling on modelling of the reefer container components.

REFERENCES

[1] Bryan Rasmussen, Andrew Musser, and Andrew Alleyne, Model-Driven System Identification of Transcritical Vapor Compression Systems, IEEE Transactions on Control Systems Technology, vol. 13, no. 3, pp. 444-451, May 2005, DOI: 10.1109/TCST.2004.839572

[2] Wei-Jiang Zhang and Chun-Lu Zhang, A Generalized Moving-Boundary Model for Transient Simulation of Dry-Expansion Evaporators Under Larger Disturbances, International Journal of Refrigeration Vol. 29, No.7, November 2006, pp. 1119-1127.

[3] N. B. O. L. Pettit, M. Willatzen and L. Ploug-Srensen, A General Simulation Model for Evaporators and Condensers in Refrigeration Part I: Moving Boundary Formulation of Two-Phase Flows with Heat Exchange, International Journal of Refrigeration, Vol. 21 No. 5, August 1998, pp. 398-403, DOI: 10.1016/S0140-7007(97)00091-1.

[4] N. B. O. L. Pettit, M. Willatzen and L. Ploug-Sørensen, A General Simulation Model for Evaporators and Condensers in Refrigeration Part II: Simulation and Control of an Evaporator, International Journal of Refrigeration, Vol. 21 No. 5, pp. 404-414, DOI: 10.1016/S0140-7007(97)00092-3.

[5] SIMULINK™,http://www.mathworks.com/products/simulink/

[6] MODELICA, http://www.modelica.org/documents/ModelicaSpec30.pdf

[7] DYMOLA™, http://www.dynasim.se/index.htm

[8] H. Elmqvist, M. Otter and F. Cellier, Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential– Algebraic Equation Systems, Keynote Address, Proc. ESM'95, European Simulation Multiconference, Prague, Czech Republic, June 5–8, 1995, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.572

[9] Soejima S, Matsuba T, Application of Mixed Mode Integration and New Implicit Inline Integration at Toyota, 2nd International Modelica Conference, Oberpfaffenhofen, Germany, 2002, Proceedings, pp. 65-1 - 65-5, http://www.modelica.org/events/Conference2002/papers/p09_Soejima.pdf

[10] SIMULINK™, Choosing a Solver, http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/ug/f11-69449.html

[11] Sven Jonsson, Vattenfall Energisystem AB, Box 528, 16215, Vallingby, Sweden, Performance Simulations of Twin-Screw Compressors with Economizer, International Journal of Refrigeration Volume 14, Issue 6, November 1991, pp. 345-350 DOI:10.1016/0140-7007(91)90031-B

[12] Kresten K. Sørensen and Jakob Stoustrup, Modular Modelling of a Refrigeration Container, Currently being written.