Advanced modeling and simulation techniques in MOSILAB: A system development case study

Christoph Nytsch-Geusen[1]
Thilo Ernst[1] André Nordwig[1]
Peter Schwarz[2] Peter Schneider[2]
Matthias Vetter[3] Christof Wittwer[3]
Andreas Holm[4] Thierry Nouidui[4]
Jürgen Leopold[5] Gerhard Schmidt[5]
Alexander Mattes[6]

[1]Fraunhofer Institute for Computer Architecture and Software Technology
Kekuléstr. 7, D-12489 Berlin, christoph.nytsch@first.fhg.de

Fraunhofer IIS/EAS[2], Fraunhofer ISE[3], Fraunhofer IBP[4], Fraunhofer IWU[5], Fraunhofer IPK[6]

1 Abstract

The design and the optimization of complex technical systems can be supported efficiently by using simulation methods and tools. For this reason, the generic simulation tool MOSILAB (Modeling and Simulation Laboratory) is being developed by a consortium of six Fraunhofer institutes in the GENSIM project. For the modeling process, MOSILAB uses the object- and equation-oriented model description language Modelica®, with a backwards-compatible extension to incorporate elements for describing model structure dynamics [1].

In this article we will use illustrate how MOSILAB's advanced modeling and simulation techniques support the user, with the help of two case studies: a complex energy system and a cutting tool system. Thus, the case studies illustrates very different uses MOSILAB.

2 Case studies

2.1 Complex energy system

The case study of a solar heating system will demonstrate MOSILAB's advanced modeling and simulation techniques, such as model-based development, model structure dynamics, external simulator coupling, or the distributed execution of simulation experiments. The considered system model includes a solar energy plant model, a building model, a model for the control strategy and an environment model for the climate parameters (see Figure 1).

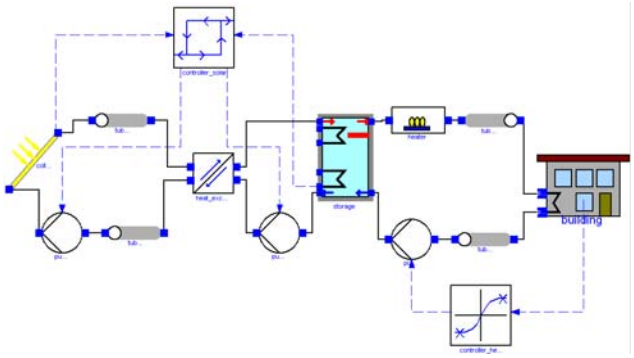

Figure 1: Energy system for solar heating system

The solar energy plant model consists of a primary solar cycle with the collector field, the solar pump and some tubes. The solar energy is transferred by a counterflow heat exchanger to the secondary storage cycle, where a storage pump loads the thermal storage. A discrete two-point controller switches on both mentioned pumps, if the temperature difference between the collector output is higher than the fluid temperature in the lowest point of the storage. The other side of the storage provides the building model with heating energy by a heating cycle. A continuous controller regulates the mass flow between zero and a maximum value, subject to the difference of the current room temperature and the set room temperature. An auxiliary heater delivers additional thermal energy, if the set flow temperature isn't achieved by the storage output temperature.

2.2 Cutting tool system

At present high performance and high precision cutting tools often are designed as modular systems with a complex mechanical behavior. Static and dynamic tool deformations or deflections affect the reliability of the technological process as well as the quality of the workpiece. Tool designers need convenient modeling techniques to predict the tool behavior under working conditions in order to optimize the tool design. Simulation can also help users to choice the most suitable tool and to optimize the cutting conditions.

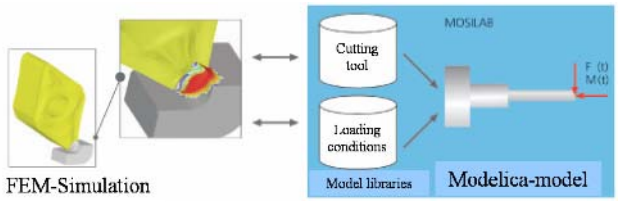

Figure 2: Coupling of FEM-simulators and MOSILAB for cutting tool systems

In the case study of a machining tool, MOSILAB is coupled with two external domain-specific FEM (Finite-Element-Methods) simulators. To simulate the behaviour of high performance cutting machine tools, the machining processes are considered to determine the loading conditions, the tool deformation, the cutting edge displacement and possible malfunctions caused by overloads. Non-linear effects at interfaces between components of a modular cutting tool are also included.

The mechanical and thermal tool loads undergo changes while complex workpiece geometries are machined e.g. dies and molds. Detailed knowledge of the occurring forces and temperatures caused by the chip building for any section of the tool path enables an adjustment of the process parameters to the specific cutting conditions. Thus, the feed rate is optimized by using FEM and analytically based simulation approaches.

This coupled consideration of tool loading and the corresponding tool behaviour enables the choice of the most suitable tool, an estimation of the workpiece quality and provides significant improvements in the efficiency of machining operations.

3 Model-based development with the MOSILAB-IDE

An integrated development environment offers users support at every step of the simulation – from model building to simulation to post-processing [6]. In order to the traditional component diagrams (compare Figure 2), which give overviews about the structures of the plant- or sub-models, further UML^{H} - diagram types are available. The class diagrams are used to organize classes and their relationships in libraries. Statechart diagrams can be used to model reactive behaviour of components, e.g. the drivers for model structural dynamics. An integrated meta-model ensures model consistency for all diagram types [8].

Thus, the behaviour of a solar thermal plant during „normal operation“ or in different unscheduled states can be represented in an integrated state-dependent structure variable model. Unscheduled states could be the plant behaviour whilst damaged pumps or self-activated pressure control valves, when the solar collector becomes overheated.

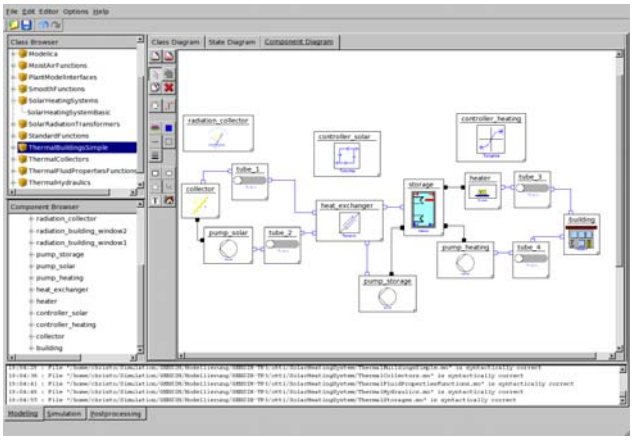

Figure 2: Solar heating system as component Diagram in the MOSILAB-IDE

To support a gap-free model-based development process, a code generator plug-in can be used to produce native embedded system code for controller relevant sub-models.

With this feature, a newly designed controller algorithm can be tested in combination with the virtual model of the controlled system. After successful testing, the same controller algorithm can work on the real controller hardware.

Technically, the description of such controller models uses Modelica’s block and algorithm concepts. Each block implementation can be automatically transformed into controller code for the target operating system. The approach was tested on the embedded Linux derivate BOSS [2].

4 Use of model structural dynamics

Using model structural dynamics [1], MOSILAB is able to adapt the model description, depending on the model state. One example of MOSILAB’s flexi-

bility is its capability to switch between simulation models varying in local resolution. We have chosen the application of a 1D-thermal storage model, embedded in the solar heating system model to illustrate this advantage.

During periods of low collector temperatures or when the storage pump is off, the thermal stratification in the storage can be calculated sufficiently with few numerical nodes (n_zones = 4). When hot water enters the storage, it is necessary to use a storage model with substantially more numerical nodes for the thermal gradient calculation (n_zones = 12, compare Figure 3).

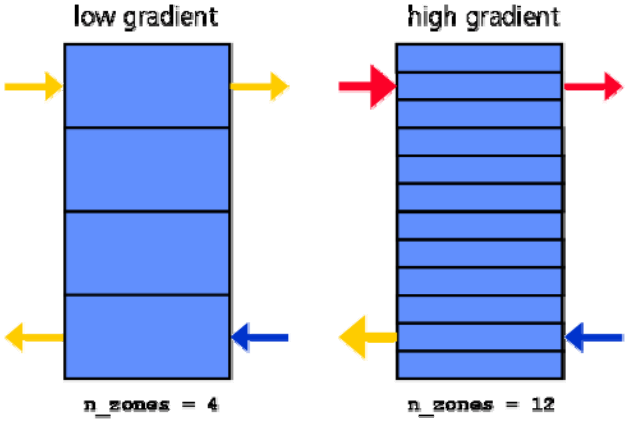

Figure 3: Structural variable storage model, which uses a different number of zones in dependency of the current thermal layering.

The following code fragments of the system model show the implemented strategy for switching between both models. The first part includes the declaration of the component models.

```
model SolarHeatingSystem
  ...
  ThermalCollectorDynamic collector(...);
  Pump pump_solar(...), pump_storage(...),
       pump_heating(...);
  Tube tube1(...), tube2(...), tube3(...),
       tube4(...);
  HeatExchangerCounterflow heat_exchanger(...);
  TwoPointController controller_solar(...);
  TanhController controller_heating(...);
  FlowHeater heater(...);
  ThermalBuildingHeatEx building(...);
  dynamic Storage storage, tempStorage;
  event Boolean finer(start=false);
```

The dynamic parts of the system model are marked with the prefix **dynamic**, in our use case the storage model. Further, the Boolean-variable *finer* has the prefix **event**, which is needed to trigger the replacement from the coarser to the finer storage model.

```
equation
  finer = pre(finer) or
          collector.out.T-pump_storage.in.T > 3.0
          and controller_solar.out > 0;
```

The first equation in the **equation**-section is true, if the difference of the collector temperature and the temperature in the lowest storage zone exceeds 3 K (and the solar pump is on). Then the storage model has to switch from 4 to 12 zones for a better reproduction of the thermal gradient.

The following code illustrates that only the static **connect**-equations are available in the **equation**-section. All dynamic connects between the storage model and its surrounding components are not closed:

```
  // controller solar and storage cycle
  collector.out.T = controller_solar.in1;
  pump_storage.in.T = controller_solar.in2;
  pump_solar.alpha = controller_solar.out;
  pump_storage.alpha = controller_solar.out;

  // controller heating cycle
  building.T_air = controller_heating.in2;
  273.15 + 20.0 = controller_heating.in1;
  pump_heating.alpha = controller_heating.out;
  ...
  // solar circle:
  connect(collector.out, tube1.in);
  connect(tube1.out, heat_exchanger.in1);
  connect(heat_exchanger.out1, tube2.in);
  connect(tube2.out, pump_solar.in);
  connect(pump_solar.out, collector.in);
  // storage solar circle:
  // no static  connect between
  // heat_exchanger.out2 and storage.in_supply1
  // no static connect between
  // storage.out_supply1 and pump_storage.in
  connect(pump_storage.out,heat_exchanger.in2);
  // heating circle:
  // no static connect between
  // storage.out_load_1 and  heater.in
  connect(heater.out, tube3.in);
  connect(tube3.out, building.in);
  connect(building.out, tube4.in);
  connect(tube4.out, pump_heating.in);
  // no static connect between
  // pump_heating.out and storage.in_load1
  ...
```

In the **statechart**-section, which is responsible for the model structure dynamics, the states of the system model (*startState, lowResolution, highResolution*) are declared and the transitions between the states (*startState -> lowResolution, lowResolution -> highResolution*) are modelled:

```
statechart
  state SolarHeatingSystemBasic
    extends State;
    State lowResolution, highResolution;
    State startState(isInitial = true);

    entry action
      storage := new Storage(n_zones = 4,
                             volume = 30.0,
                             ...);
    end entry;
```

At the beginning of the simulation experiment (*startState -> lowResolution*) the storage model is added to the system model and the connections of the storage model to its surrounding components are closed.

```
transition startState -> lowResolution
  add(storage);
  connect(heat_exchanger.out2,
          storage.in_supply1);
  connect(storage.out_supply1,
          pump_storage.in);
  connect(storage.out_load1, heater.in);
  connect(pump_heating.out,
          storage.in_load1);
end transition;
```

If the transition *lowResolution -> highResolution* is triggered by the variable *finer* during the simulation experiment, the connections from the storage model are cut by using **disconnect**(*a.p,b.p*) and the old storage model is removed.

```
transition lowResolution -> highResolution
  event finer action
  disconnect(heat_exchanger.out2,
             storage.in_supply1);
  disconnect(storage.out_supply1,
             pump_storage.in);
  disconnect(storage.out_load1, heater.in);
  disconnect(pump_heating.out,
             storage.in_load1);
  remove(storage);
```

Now a new storage model is instantiated with **new** in a higher resolution (*n_zones = 12*). The start values of the new storage model are determined from the current state of the old storage model:

```
tempStorage := new Storage(n_zones = 12,
                           volume = 30.0,
                           ...):
tempStorage.content.T_zone[1] :=
                storage.content.T_zone[1];
tempStorage.content.T_zone[2] :=
                storage.content.T_zone[1];
tempStorage.content.T_zone[3] :=
                storage.content.T_zone[1];
tempStorage.content.T_zone[4] :=
                storage.content.T_zone[2];
...
tempStorage.content.T_zone[10] :=
                storage.content.T_zone[4];
...
```

Then the new storage model substitutes the old model, must be added to the system model and the connection to its adequate components are closed again.

```
storage := tempStorage;
add(storage);
connect(heat_exchanger.out2,
        storage.in_supply1);
connect(storage.out_supply1,
        pump_storage.in);
connect(storage.out_load1, heater.in);
connect(pump_heating.out,
        storage.in_load1);
  end transition;
 end SolarHeatingSystem_SC;
end SolarHeatingSystem;
```

The simulation experiment with MOSILAB for this system model for a summer day is shown in Figure 4. The diagram shows the implemented behaviour. During the morning hours, the solar controller switches the pumps on first (third curve). Two hours later the temperature between the collector output and the temperature in the lowest layer of the storage is greater than 3 K. (This temperature is equal to the input temperature of the storage pump.) As a result, MOSILAB exchanges the coarse storage model with the higher-resolution model (*n_zones = 4 -> n_zones = 12,* fourth curve).

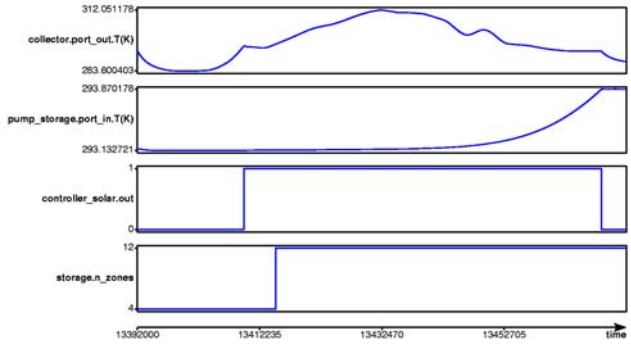

Figure 4: Simulation experiment for a summer day: MOSILAB switches to the detailed model, when hot water enters the storage model and its thermal gradient has to be recalculated in a finer resolution.

5 Numerical coupling with external simulators

Building on the MOSILAB platform, reusable components for simulator coupling have been developed within the GENSIM project. The components support integration with standard tools, such as MATLAB/Simulink or FEMLAB/COMSOL Multiphysics and also domain-specific FEM-Tools such as MARC and DEFORM. This represents a departure from and an improvement upon the typical separate handling of system simulation and FEM (Finite Element Method) simulation.

5.1 MATLAB/Simulink

MOSILAB offers an optional generic interface for MATLAB/Simulink [3]. Thus, it is possible to develop control strategies for embedded systems within MATLAB/Simulink and combine them with a Modelica model of the mixed-continuous discrete system environment. In this scenario each sub-

system is modelled in with the appropriate modelling paradigm within adequate simulation engineering tools.

For a smooth integration of both modelling views, a proxy object is introduced in each view. Within a view, the proxy object represents the wrapped simulator which is realized in the other view. This leads to symmetric model perspectives, which are close to the mental model of the engineer.

In MATLAB/Simulink a generic MOSILAB proxy model can be imported and parameterized via the block parameter dialog. (Compare with Figure 5.)

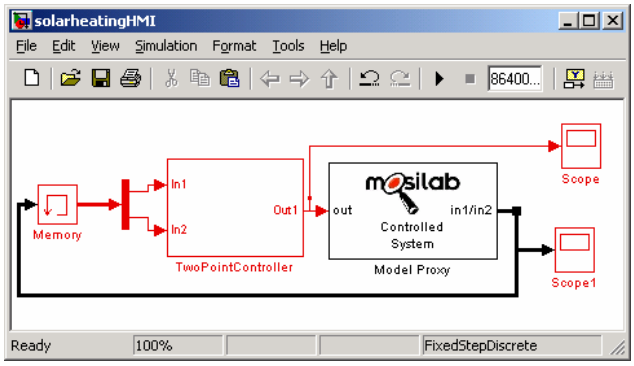

Figure 5: Simulink with an embedded MOSILAB-model

The controlled system model itself (in the case study, the solar energy plant and the building model) is developed using MOSILAB and will be associated with this proxy model, which is shown in the following code fragment:

```
block RemoteModel
 constant Boolean isRemoteModel=true;
 parameter Integer nInp, nOutp;
 input  Real inp[nInp];
 output Real outp[nOutp];
end RemoteModel;
```

The constant *isRemoteModel* indicates the presence of a further simulator/driver behind this model. Thus, the numeric algorithms can handle the input and output vectors correctly. The number of input and output variables can be given by *nInp* and *nOutp*. The vectors itself are given by *inp* and *outp*.

The following code illustrates the direct use of this generic remote interface within a Modelica model:

```
model SolarHeatingSystem
  ThermalCollectorDynamic collector
  Pump pump_solar(...);
  StorageSimple storage(...);
  ...
  // the Simulink interface model
  RemoteModel ctrl_solar(nInp=2, nOutp=1);
  ...
equation
  ...
  // controller solar cycle
  collector.port_out.T = ctrl_solar.inp[1];
  storage.content.T_zone[4] = ctrl_solar.inp[2];
  pump_solar.alpha = ctrl_solar.outp[1];

  pump_storage.alpha = ctrl_solar.outp[1];

  // solar cycle:
  connect(collector.out, tube1.in);
  connect(tube1.out, heatexchanger.in1);
  connect(heat_exchanger.out1, tube2.in);
  connect(tube2.out,pump_solar.in);
  connect(pump_solar.out,collector.in);

  // storage solar cycle:
  connect(heatexchanger.out2,storage.in_supply1);
  connect(storage.out_supply1,pump_storage.in);
  connect(pump_storage.out,heat_exchanger.in2);
  ...
end SolarHeatingSystems;
```

In this configuration the simulation is driven by MATLAB/Simulink as the master simulator. Figure 6 illustrates a coupled simulation experiment for the solar heating system during a simulation period of one week in spring. The top screen shows the output signal of the discrete controller, calculated in MATLAB/Simulink. This signal switches the solar pump depending on the temperature difference between the collector output temperature and the temperature in lowest level within the water storage.

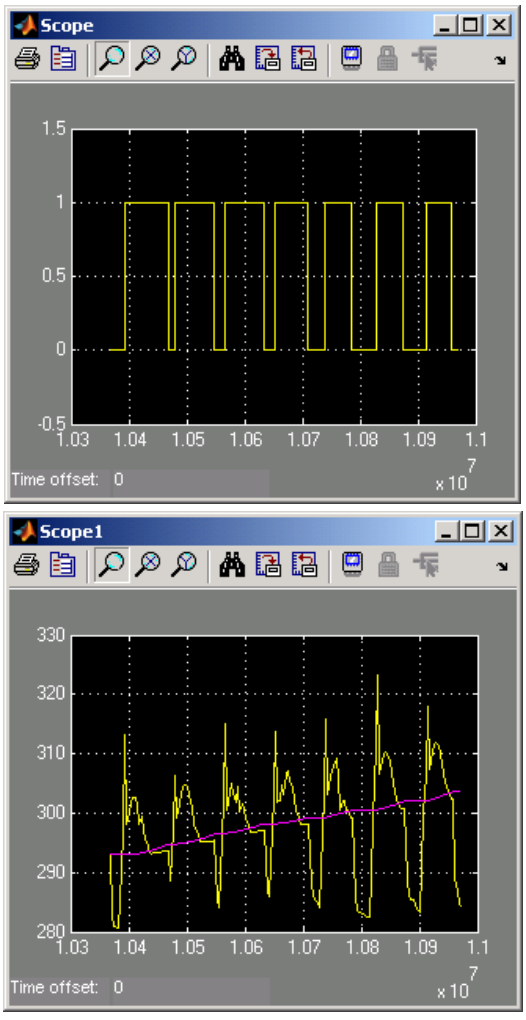

Figure 6: Coupled simulation of MOSILAB with MATLAB/Simulink.

The bottom screen illustrates the dynamic behaviour of the controlled system, calculated in MOSILAB, for the same time period. The curves represent both state variables, which are the input signals of the controller (collector output and storage temperature).

5.2 FEMLAB/COMSOL Multiphysics

One other aspect within the project was the development of a numeric coupling between the simulators MOSILAB and FEMLAB [4]. For simulator couplings which incorporate FEMLAB, two basic principles exist:

1. Coupling within the MATLAB Framework – here the MATLAB engine is used in a C-program or a dedicated coupling model is implemented based on the MEX-interface.
2. FEMLAB is used as a stand-alone simulator – within FEMLAB Java-API models can be loaded, the simulation can be controlled, and the data exchange can be organized.

The second principle is used for this implementation. Figure 7 illustrates the basic structure of the coupling.

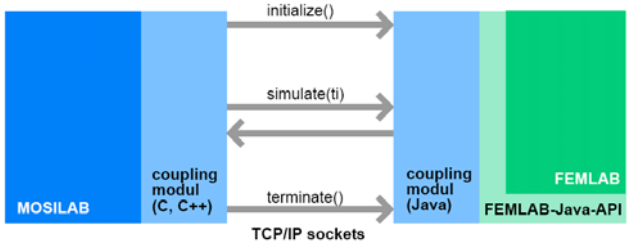

Figure 7: Numerical coupling between MOSILAB and FEMLB/ COMSOL Multiphysics

The communication between the two sides is handled by TCP/IP sockets. This extends the usage of the simulator coupling for a distributed computer environment. Due to a lean coupling implementation, the communication time for the data exchange is much shorter than the simulation time for simple FEM-models. This allows an effective simulation including realistic transient boundary conditions even in combination with models requiring small simulation time steps.

Hence, simulator coupling is suitable for a wide range of applications. This enables the analysis of control systems with a detailed consideration of the controlled process. Furthermore, components in a complex system can be analysed in detail using the MOSILAB-FEMLAB environment, e.g. the multi-dimensional flow within the heat storage as a part of a solar heating system.

5.3 MARC

The finite element code MARC can be used to model complex nonlinear mechanical and thermo-mechanical structures such as machining tools consisting of different components with contact and friction problems. Thus, it is possible to simulate complex system behavior which cannot be adequately described by analytical functions. For example, nonlinear load dependent contact behavior between tool components may results in non-linear tool deformations which require an expensive finite element analysis (FEA). Because of long computing times with FEA the coupling between MOSILAB and MARC will be off-line in most cases. For that purpose a special interface has been developed. The coupling of MOSILAB with MARC will enable to opt between analytical models for relatively simple cutting tools or the more complex FEM-models. That way it is possible to optimize the accuracy of the description of the tool behavior and the expense of the calculations.

Finite element analyses will run outside of MOSILAB and the input and output streams to respectively from the MOSILAB databases will be realized by *readData()* and *writeData()* commands. Using that interface it will also be possible to use predefined FEM-models from a tool model library without special knowledge of finite element modeling.

The loading conditions required by analytical or finite element analyses are provided by the simulation of the cutting process (see chapter 5.4).

5.4 DEFORM

Originally developed for metal forming processes, the FEM-tool DEFORM is also suited for simulation of the chip formation during the machining process. DEFORM is advantageous for an efficient handling of the mesh distortion, which is caused by the high strains within the chip formation zone. Through the remeshing function it is possible to generate a new mesh and to transfer the interpolated values for each node. Thereby the program is able to simulate the mechanical and thermo-mechanical behavior.

In addition, a simplified and fast model for the chip building process in Modelica was developed, which is based on analytical equations (e.g. cutting force calculation according to Kienzle [5]). First of all, the parameters of the simplified Modelica-model have to be calculated with a large number of detailed DEFORM simulations. As a result, the fast Modelica model can be used in the area of validity of these DEFORM calculations.

The cooperation between MOSILAB and DEFORM for the determination of these parameters has been fully automated. First, a routine for automated pre- and post processing for DEFORM was developed. The execution of DEFORM by an external program is possible using the text mode of the software. This enables set up and run of simulations without going through the graphic user interface. The routine needs initial input information about tool geometry and machining parameters provided in a text file. To transfer amongst others, the values for the angles of the cutting wedge or for the feed rate and width of cut from MOSILAB the commands *readData()* and *writeData()* are applied.

```
model DataExchange
  model Kienzle
  ...
  end Kienzle;

  parameter String fname = "inputData.txt" ;
  parameter String fnameOut = "outputData.txt";
  Kienzle k;
  algorithm
    when initial() then
      readData(fname, k);
    end when;
    when terminal() then
      writeData(fnameOut,k);
    end when;
end DataExchange;
```

These loose coupling of MOSILAB with DEFORM helps to combine the advantages of both methods: First, the short computation time, when solving analytical equations in Modelica and second, the manifold possibilities by analyzing the chip building process through FEM-Analysis.

6 Distributed execution of simulation experiments

Simulators developed using MOSILAB can be generated in various configurations – from a "barebone" variant suitable for constrained environments, such as embedded systems, to a regular desktop application, to a web service for distributed simulation.

6.1 Simulator Services and Interoperability

MOSILAB follows a service-based architectural style. For all configurations except the minimal one, the simulators generated by MOSILAB are created as services communicating through a standard interface. The standard interface is based on the W3C/OASIS web services protocol suite (most importantly, HTTP and SOAP), which allows MOSILAB-developed simulators to be controlled from a wide variety of software environments such as Java, C++, C#/.NET, MATLAB, Python, Perl, and Ruby. MOSILAB also supports a more bandwidth-efficient proprietary stream command interface, a direct C++ API, and a Python API. The Python layer abstracts from the underlying transport mechanism; i.e. the same Python experiment script can be used to control a simulator running as a local subprocess and communicating via OS standard I/O pipes, or to control a simulation web service running on a remote machine but having been generated from the same Modelica model (compare figure 8).

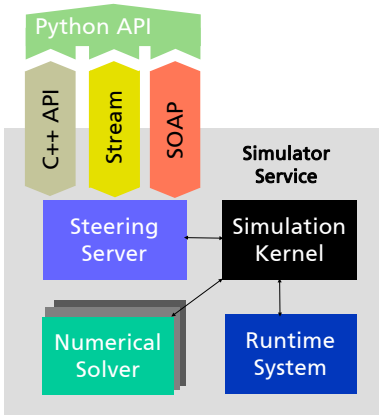

Figure 8: MOSILAB steering interface options

These interfaces are all manifestations of one and the same abstract protocol (called the *MOSILAB unified steering protocol*), which is only expressed in different programming languages. The generic interfaces to other simulators described in section 5 have been developed using these interfacing options specific to MOSILAB, in addition to Modelica's standard external function interface.

6.2 Speeding up parameter studies by distributed simulation

Often, the system design task at hand requires a large number of simulation runs with differing parameter values, e.g. to obtain knowledge about the system's behaviour under parameter variations ("robust design"), or to approximate a certain desired property of the system being designed ("optimization"). In the system model from the case study, it makes sense to consider variations of the model parameters "collector area", "heat store volume" or "building orientation", as well as parameters of the controller model. The following Figures 9 and 10 illustrate a variation of the collector area parameter.

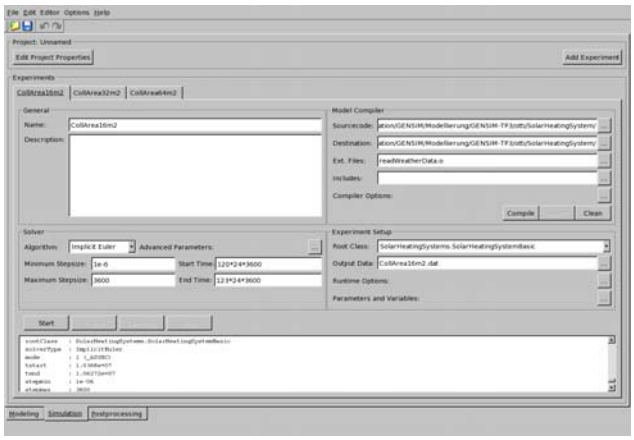

Figure 9: Multiple simulation experiments in the MOSILAB-IDE for varying the collector area

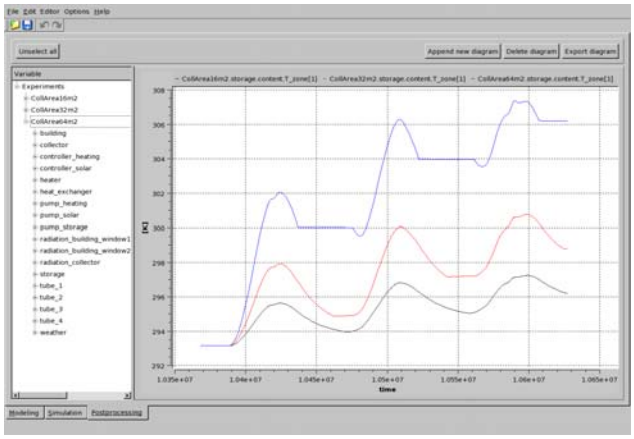

Figure 10: Impact of different collector areas on the storage temperature during a period of 3 days

Variations of multiple parameters lead to multidimensional variant spaces, the size of which (i.e. the total number of simulation runs needed) soon becomes impractical, due to the sheer computation time needed. Statistics-based methods exist to achieve a substantial reduction of the variant space with only a marginal loss of result quality, but even with such methods in place, a large number of necessary simulation experiments are likely to remain. MOSILAB's service-based architecture allows for distribution of simulation experiments as independent, parallel jobs in clusters and computational grids, thus empowering the user to make optimal use of the computational resources available. The individual distributed simulators can nevertheless be interactively controlled and supervised from the MOSILAB-IDE (see Figure 10).

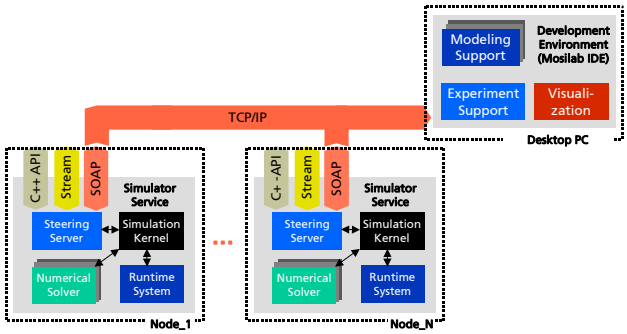

Figure 10: Executing simulator services in the Grid

For very large numbers of parallel experiments, central steering limits scalability, and interactive supervision becomes impractical. In this case, MOSILAB-generated simulators can be distributed in the Grid as independent batch jobs. For more information on MOSILAB and Grid computing, see [7].

References

[1] Nytsch-Geusen, C. et al. MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics. Proceedings of the 4th International Modelica Conference TU Hamburg-Harburg, 2005.

[2] Nordwig, A. et. al: Codegenerierung aus Simulationsmodellen von heterogenen technischen Systemen am Beispiel einer Pendelsteuerung, VSEK-Report. FKZ01ISC65, 2005.

[3] Nordwig, A.: Coupling of Modelica and Matlab/Simulink models. Technical Report, Fraunhofer FIRST 2006.

[4] Clauß, C. et. al: Simulatorkopplung mit FEMLAB. Proceedings of the 1th FEMLAB Conference, Frankfurt am Main, 2005.

[5] König, W.: Fertigungsverfahren – Band 1: Drehen, Fräsen, Bohren. VDI-Verlag, 1990.

[6] MOSILAB-Homepage: http://www.mosilab.de

[7] Ernst, T. et al.: MOSILAB: Modelica Simulation from Desktop to Grid. 2. Workshop "Grid-Technologie für den Entwurf technischer Systeme", Dresden, 2006.

[8] Nordwig, A.: Integration von Sichten für die objektorientierte Modellierung hybrider Systeme, Verlag dissertation.de, ISBN 3-89825-692-8, 2003.