Development and Nonlinear Analysis of Dynamic Plant Models in ProMoT /Diana

Michael Mangold[*,1], Dmytro Khlopov[1], Gerrit Danker[1], Stefan Palis[2], Volodymyr Svjatnyj[3], Achim Kienle[1,2]

[1]Max-Planck-Institut für Dynamik komplexer technischer Systeme, Sandtorstraße 1, 39106 Magdeburg, Germany

[2]Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany

[3] Donetsk National Technical University, Faculty of Computer Science and Technology, Donetsk, Ukraine

Abstract:

The paper gives a short introduction to the process modeling tool ProMoT and the simulation tool Diana. Both are open-source programs intended for the dynamic analysis of chemical engineering and biological systems. They support the implementation and analysis of large nonlinear differential algebraic systems. An overview is given on the functionality of ProMoT and Diana. The use of the tools is illustrated by their application to an innovative fluidized bed crystallization process.

Keywords:

process model; computer aided modeling; differential algebraic systems; dynamic simulation; nonlinear analysis; crystallization; population balances.

1 Introduction

Dynamic simulation of chemical processes and chemical production plants has been widely accepted as a useful tool for process design. Dynamic simulation is applicable to the design of control strategies, to the development of start-up and shut-down procedures, to the stability analysis of desired operation points, or to the analysis and design of dynamic operation modes like batch operation, to name but a few examples. Consequently, there are a large number of commercial and non-commercial software tools available that support the numerical solution of systems of differential or differential algebraic equations, as they typically arise from the modeling of chemical plants. On the one hand, there are general purpose numerical programming environments like Matlab [1], GNU Octave [2], or Scilab [3]. These tools offer a wide range of numerical methods and visualization techniques. However, their numerical algorithms are not tailored specifically to the needs of chemical plant models, which typically are characterized by strong nonlinearities, a comparatively high system order (several thousand differential equations) and Jacobians without nice exploitable structural properties apart from sparsity. When implementing a complex plant model in one of these tools, the user may either have to accept quite slow numerical solution, or he may have to invest a lot of effort to increase efficiency by providing symbolic Jacobians, providing sparse matrix patterns etc. On the other hand, there are dedicated dynamic flowsheet simulators like

[*] Corresponding author, e-mail mangold@mpi-magdeburg.mpg.de

gPROMS [4], Aspen Plus Dynamics [5], or Dymola/Modelica [6] with strong modeling capabilities and specialized numerical solution techniques. As a drawback, the details of the algorithms inside these closed-source tools are usually hidden from the user. This makes it hard to debug the code in cases when a user defined model does not behave as expected or when its numerical solution fails. Further, it is often quite difficult or even impossible to add new numerical algorithms to a closed-source simulator. However, this may be desirable for experienced users who would like e.g. to test other equation solvers or to apply non-standard numerical approaches like continuation methods, novel experimental design methods etc.

Motivated by the shortcomings of existent simulation tools, we have developed an open modeling and simulation environment whose roots go back to earlier activities at the University of Stuttgart. The environment consists of a pair of two tools, ProMoT and Diana. The process modeling tool ProMoT [7-9] supports the implementation of structured and hierarchical modeling libraries. It translates symbolic model information into simulation code for a number of numerical simulation programs, one of which is Diana. Diana [10-12] is a platform for the dynamic simulation and nonlinear analysis of differential algebraic systems. A schematic overview on the use of the tools is given in Figure 1 and will be discussed in more detail in the subsequent sections.

<Figure 1>

In the last years, the ProMoT/Diana system has proven useful for various challenging modeling tasks, e.g. in the area of chemical engineering systems [13,14], energy systems [15] and biological systems [16,17].

ProMoT and Diana are free open-source software. They can be downloaded from the web page of the Max Planck Institute for Complex Dynamical Systems under http://www.mpi-magdeburg.mpg.de/projects/promot/ and http://www.mpi-magdeburg.mpg.de/projects/diana/ . ProMoT is available in form of binaries for Linux and Windows and as a source distribution. Diana can be built from sources on any recent Linux system.

The following two sections are introducing ProMoT and Diana in more detail. The use of the tools is then illustrated by the modeling and simulation of a fluidized bed crystallization plant, which is part of an ongoing research project.

2 Process Modeling Tool ProMoT

ProMoT is a modeling tool written in Common Lisp with a graphical user interface written in Java [7,8]. ProMoT supports the structured implementation of dynamic models described by systems of nonlinear implicit differential algebraic equations of the form

$$\begin{aligned} \boldsymbol{B}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t)\frac{dx}{dt} &= \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t) \\ \boldsymbol{y} &= \boldsymbol{h}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t)\,, \end{aligned} \tag{1}$$

where $\boldsymbol{u}$ denotes the input vector, $\boldsymbol{x}$ denotes the state vector, $\boldsymbol{y}$ is the output vector, $\boldsymbol{p}$ is a vector of constant model parameters, and t is the simulation time; $\boldsymbol{f}$ and $\boldsymbol{h}$ are arbitrary nonlinear function vectors; $\boldsymbol{B}$ is a – possibly singular – mass matrix, zero rows indicating implicit algebraic equations. $\boldsymbol{B}$, $\boldsymbol{f}$ and $\boldsymbol{h}$ may change during a simulation due to implicit switching events. In this way, it is possible to define models with discontinuities in ProMoT, e.g. caused by phase changes, reversal of fluid flow direction, opening and closing of valves etc. ProMoT itself is a purely symbolic modeling tool and hence has no restrictions with respect to numerical properties of the models like stiffness, differential index. Of course, such limitations occur when a numerical solution of the model is attempted and depend on the numerical algorithms employed. Limitations resulting from the use of Diana will be discussed in the next Section. However, on the ProMoT level the idea is to keep the model formulation separate from numerical requirements. ProMoT provides a rather general modeling language and outputs to different numerical solution tools. From these tools the user may choose the one that is most appropriate for his needs.

The modeler defines equation systems like (1) either by selecting predefined model building blocks from a graphical user interface or by using the text based model definition language MDL (see Figure 1). A simple example of MDL code is given in Figure 2. Switching events are described by Petri nets, with another simple example shown in Figure 3.

<Figure 2>

<Figure 3>

ProMoT offers two mechanisms for the structuring of complex models. First, inputs and outputs of several models of the form (1) may be connected via the graphical user interface of ProMoT, similar to classical flowsheet simulators. For an example see Figure 7, which is discussed in detail in Section 5. Second, ProMoT allows aggregation of modules to a composite modeling entity and multiple inheritance of one modeling entity from other modules. Each modeling entity in ProMoT inherits properties from a super-class. The class “simple_ode” in Figure 2 e.g. inherits from a general super-class “module”. Using the object oriented mechanisms of aggregation and inheritance, a modeler can build up hierarchically structured model libraries. This will be illustrated by the application example in Section 4.

After implementation of a model, ProMoT performs a structural analysis of the model. ProMoT tries to detect explicit algebraic equations. If there are no cyclic dependencies in the corresponding algebraic variables, then these variables are eliminated in order to reduce the number of unknowns for the numerical solution. Further ProMoT performs some simple structural solvability analysis of the model and issues an error message, if the model system is obviously not solvable, because e.g. the number of equations does not match the number of unknowns, or if a declared state variable does not appear in any equation. A visual editor [9] gives further insight in the internal dependencies of the model. ProMoT generates analytical Jacobians via an interface to the computer algebra system Maxima [18]. Analytical derivatives of arbitrary order with respect to the states and the model parameters can be generated, as they are required for sensitivity analysis, optimization, and singularity analysis.

As a final step, ProMoT converts the model information into simulation code for various numerical simulation tools. Currently, it is possible to export models to Diva [19], Matlab and Diana. Furthermore, model exchange (import and export) is possible via the Systems Biology Markup Language SBML [20].

3 Numerical Analysis Tool Diana

Diana [10-12] is a simulation tool for the solution and nonlinear analysis of differential algebraic systems, as they typically result from first principle modeling of chemical engineering systems and biochemical systems. The numerical core of Diana is written in C++ in order to ensure fast and efficient numerical solutions. Model equations also have to be implemented in C++ as an equation set object (ESO) using CAPE-OPEN [21] standard interfaces. However, a user of Diana never gets into immediate contact with C++ and is not required to write his own C++ code. Instead, the user may generate an ESO of his model with the help of ProMoT as explained above and illustrated in Figure 1. For the numerical analysis, the modeler accesses Diana via scripts written in the scripting language Python [22]. The advantage is that Python is very easy to learn and to program. Further, in Python there are powerful numerical and graphical libraries available the Diana user has full access to. Using these libraries, the Diana user can easily extend the functionality of Diana or post-process and visualize the numerical results obtained from Diana. For example, the NumPy library lets the user apply linear algebra methods from BLAS [23] and LAPACK [24] to the simulation results. Finally, Python scripts may be used to define and execute complex simulation scenarios like simulation of control recipes for chemical plants. Figure 4 shows a most simple example of a Python script with a call to Diana.

<Figure 4>

The numerical methods implemented in Diana are tailored to the needs of typical chemical plant models. The initial value problem solvers are able to handle stiff differential algebraic systems with a differential index of zero or one. The Jacobians should preferably be sparse, but apart from that no special structural properties are required. For best efficiency, the system order should lie approximately between 10^1 and 10^4. The lower limit results from the use of sparse numeric routines, which cause an additional computational burden for very small systems; the reason for the upper limit is that direct methods are used for solving the underlying linear algebra problems, whereas iterative methods may be more efficient for very large systems.

The numerical core libraries used in Diana are BLAS [23], LAPACK [24], UMFPACK [25], and ARPACK [26] for basic linear algebra algorithms, SUNDIALS/Ida [27] and DASPK [28] for the solution of differential algebraic initial value problems, ODESSA [29] for the solution of ODE initial value problems, SUNDIALS/Kinsol and NLEQ1S [30] for the solution of nonlinear algebraic equations; for further details see [12].

For the solution of optimization problems, Diana offers the interior point optimizer IPOPT [31], an implementation of the Nelder-Mead downhill simplex method [32], and genetic algorithms from the packages DIRECT [33] and GMFL [34].

Diana contains powerful methods for parameter continuation and bifurcation analysis of equilibrium points states, steady state singularities and periodic solutions [11,12]. A predictor corrector algorithm with step size control is used for continuation; the user can choose between a chord predictor and a tangent predictor for the prediction step, and between local parameterization and pseudo-arc length parameterization for the corrector step [12]. Diana permits the detection and parameter continuation of singularities up to codimension 3 (limit points, isolas , hysteresises, pitchforks, winged cusps) and of Hopf bifurcations. In combination with the analytical Jacobians provided by ProMoT, Diana generates the required augmented equation systems automatically. For the direct computation of stable and unstable periodic orbits, a single shooting algorithm has been implemented, which uses the Recursive Projection Method [35,36] for an efficient treatment of high-dimensional systems.

4 Application Example Fluidized Bed Crystallizer

The intention of the following Section is to illustrate the workflow of ProMoT/Diana. While toy examples can be found in the user documentation of the tools, a more serious application from recent research work is chosen here. The example is a plant model of a fluidized bed crystallizer system whose complexity is quite typical for the systems analyzed in ProMoT/Diana.

Fluidized bed crystallizers are an innovative method to continuously produce crystals with high purity [37]. Recently, fluidized bed crystallizers have been proposed for the continuous separation of enantiomers [38-40], which is a challenging problem e.g. in pharmaceutical industry.

In the following, the fluidized bed crystallization process sketched in Figure 5 will be considered. A supersaturated liquid solution comes from the feed tank and enters the bottom of the crystallizer. In the crystallizer, the fluid flow goes from bottom to top. The conditions in the crystallizer are chosen such that crystal growth occurs, but hardly nucleation. Small crystals are dragged upwards with the fluid. Larger crystals sink to the bottom due to gravity. They are fed into an ultrasonic (US) attenuator, crushed and recycled as seeding crystals into the crystallizer. A product flow is withdrawn at the side of the crystallizer. It goes through a sieve, where the product crystals are collected. The filtrate and the fluid outflow at the top of the crystallizer are fed back to the feed tank. The described process is a simplified version of the set-up suggested in [40], as it contains only one crystallizer instead of two. Therefore, it permits only the production of one type of crystals, e.g. of one enantiomer, while the counter-enantiomer remains in the liquid phase.

<Figure 5>

The objective of the process is to generate product crystals of a desired purity and size. The design variables are the temperatures and flow rates in the devices, as well as the shape of the crystallizer and the position of the product outlet. As a first step of the analysis in ProMoT/Diana a mathematical process model has to be formulated. A suitable model is presented in the next section.

4.1 Mathematical Process Model

4.1.1 Model of the fluidized bed crystallizer

The model of the fluidized bed crystallizer is adapted from [41]. It describes the particle size distribution as a function of time, of a space coordinate x in the direction of the fluid flow and of a property coordinate L for the characteristic crystal size. It is assumed that particles move inside the fluid flow with a velocity v_P in x direction that depends on the fluid flow velocity and the particle size. Particles grow with a growth rate $G(c)$, which is a function of the concentration c of the solute in the liquid phase. Nucleation, aggregation and attrition of particles are considered as negligible. A mass balance for the particles leads to the following population balance equation:

$$A(x)\left.\frac{\partial n}{\partial t}\right|_{x,L,t} = -A(x)G(c)\left.\frac{\partial n}{\partial L}\right|_{x,L,t} - \frac{\partial}{\partial x}\left\{A(x)v_p(x,L,t)n(x,L,t) - D\,A(x)\left.\frac{\partial n}{\partial x}\right|_{x,L,t}\right\} \qquad (2)$$

$$-\dot{N}_{US,out}(L,t)\,\delta\big(x - x_{US,out}\big) + \dot{N}_{US,in}(L,t)\,\delta\big(x - x_{US,in}\big)$$

$$(L > 0, 0 < x < h)$$

In (2), $n(x, L, t)$ is the number size density, i.e. the number of particles of a certain size per volume; $A(x)$ is the cross section of the crystallizer; D is a dispersion coefficient; $\dot{N}_{US,out}(L,t)$ and $\dot{N}_{US,in}(L,t)$ denote the particle flow to the ultrasonic attenuator at point $x_{US,out}$ and from the ultrasonic attenuator at point $x_{US,in}$ respectively; δ is the Dirac delta function; h is the total height of the crystallizer. The kinetic expression for the growth rate $G(c)$ is taken from [41].

A Dirichlet boundary condition is used for the L coordinate, and Danckwerts boundary conditions are used for the x coordinate:

$$n(x, 0, t) = 0 \tag{3}$$

$$v_P(0, L, t)\big(n(0, L, t) - n_{in}(L, t)\big) - D\,\frac{\partial n}{\partial x}\bigg|_{0,L,t} = 0, \tag{4}$$

$$v_P(h, L, t)\big(n(h, L, t) - n_{out}(L, t)\big) - D\,\frac{\partial n}{\partial x}\bigg|_{h,L,t} = 0, \tag{5}$$

where $n_{in}(L,t)$ is the size distribution of the particles entering at the bottom of the crystallizer, and $n_{out}(L,t)$ is the size distribution above the top of crystallizer, which is given as

$$n_{out}(L,t) = \begin{cases} n(h, L, t) & if\ v_P(h, L, t) > 0 \\ 0 & if\ v_P(h, L, t) < 0 \end{cases} \tag{6}$$

A formula for the particle velocity can be derived from a quasi-stationary momentum balance for the particles, assuming equilibrium between gravity force, buoyancy force and drag force [41]. The resulting relation reads for spherical particles with radius L:

$$v_P(x, L, t) = v_F(x, t) - \sqrt{\frac{8}{3}\,\frac{L\,g}{c_W(v_F)}\,\frac{\rho_P - \rho_F}{\rho_F}}, \tag{7}$$

where v_F is the flow velocity of the liquid, g is the gravity constant, ρ_F and ρ_P are fluid and particle density, respectively. The drag coefficient $c_W$is computed from the relation given in [41].

The flow velocity v_F of the incompressible liquid follows immediately from the volume flow $\dot{V}$and the free cross sectional area A_{eff}:

$$v_F(x,t) = \frac{\dot{V}(t)}{A_{eff}(x,t)}, \tag{8}$$

where

$$A_{eff}(x,t) = A(x)\left(1 - \int_0^\infty \frac{\pi}{6}\,L^3\,n(x, L, t)\,dL\right) \tag{9}$$

The mass balance of the liquid phase gives

$$\frac{\partial}{\partial t}\left(A_{eff}(x,t)\,c(x,t)\right) = -\frac{\partial}{\partial x}\left(A_{eff}(x,t)\,v_F(x,t)c(x,t)\right) + \frac{\partial}{\partial x}\left(A_{eff}(x,t)\,D_{eff}\,\frac{\partial c}{\partial x}\bigg|_{x,t}\right) + A(x)\,\frac{4}{3}\pi\,\frac{\rho_P}{\rho_F}\int_0^\infty L^3 G(c)\frac{\partial n}{\partial L}\bigg|_{x,L,t}\,dL \qquad (0 < x < h) \tag{10}$$

The three terms on the right hand side account for convective transport, dispersive transport with dispersion coefficient D_{eff} and consumption of solute due to crystal growth. The corresponding Danckwerts boundary conditions read

$$v_F(0,t)(c(0,t) - c_{in}(t)) - D\,\frac{\partial c}{\partial x}\bigg|_{0,t} = 0, \qquad \frac{\partial c}{\partial x}\bigg|_{h,t} = 0, \tag{11}$$

c_{in} being the inlet concentration of the solute at the bottom of the crystallizer.

4.1.2 Model of the ultrasonic attenuator

The US attenuator is described by the simple model suggested in [41]. It is assumed that the attenuator has negligible hold-up, i.e. the volume flow $\dot{V}_{US}$ from the crystallizer to the US attenuator is identical to that from the US attenuator back to the crystallizer, and the total particle mass flows from and to the attenuator are identical, as well. Further, it is assumed that the particles leaving the US attenuator always have the same size distribution $n_{US}(L)$. Under these assumptions, the particle flows $\dot{N}_{US,out}$ and $\dot{N}_{US,in}$ in (2) may be written as

$$\dot{N}_{US,out}(L,t) = \dot{V}_{US}\,n\left(x_{US,out}, L, t\right) \tag{12}$$

$$\dot{N}_{US,in}(L,t) = \dot{V}_{US}\,K_{US} n_{US}(L) \tag{13}$$

K_{US} in (13) is a scaling constant that follows from the mass conservation condition

$$\int_0^\infty \dot{N}_{US,out}(L,t)\,L^3\,dL = \int_0^\infty \dot{N}_{US,in}(L,t)\,L^3\,dL \tag{14}$$

4.1.3 Model of the product sieve

It is assumed that the product sieve works perfectly in the sense that the liquid flow rate and liquid composition at its inlet and its outlet are identical, and that the outlet flow contains no particles.

4.1.4 Model of the feed tank

For simplicity, the feed tank is assumed to be sufficiently large that it may be considered as a reservoir with constant composition of the liquid phase. Due to this idealization, the system is able to reach a steady state or stable periodic oscillation. A more realistic feed tank model with a finite hold-up would cause the system to be permanently in a transient state, because the amount of solute in the system would deplete constantly. This situation has also been studied in ProMoT/Diana, but is not discussed here, as it would distract from the main subject of this paper, which is the presentation of the tools.

4.2 Implementation in ProMoT

The process model from Section 4.1 is not yet ready for implementation in ProMoT, because currently ProMoT expects differential algebraic equations systems of type (1). Therefore, in a first step presented in 4.2.1 the partial differential equations (2) and (10) have to be approximated by a set of ordinary differential equations. The implementation of this set of equations using the modeling language MDL and the ProMoT GUI is described in 4.2.2.

4.2.1 Spatial discretization

The method of lines is used to convert the partial differential equations into sets of differential algebraic equations that can be solved by Diana. A finite volume scheme is applied with volume elements as shown in Figure 6.

<Figure 6>

An equidistant grid is used in x direction, and a logarithmic distribution of grid points in L direction. The discretization is largely straight-forward. Only the treatment of the convective transport term in the population balance equation (2) requires some care, because the particle velocity v_P may change its sign along the x and the L coordinate and a discretization scheme is needed that provides numerical stability and preserves mass conservation under these circumstances.

An appropriate discretization scheme is derived in the following. Integration of (2) over a volume element gives

$$\int_{L_{j-\frac{1}{2}}}^{L_{j+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} A(x) \frac{\partial n}{\partial t}\bigg|_{x,L,t} dx\, dL = \int_{L_{j-\frac{1}{2}}}^{L_{j+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} -\frac{\partial}{\partial x}\{A(x) v_p(x,L,t) n(x,L,t)\}\, dx\, dL + \cdots \tag{18}$$

(Only the critical convective term is shown on the right-hand side of (18), the other terms are omitted for brevity).

The integral on the left-hand side of (18) is approximated by

$$\int_{L_{j-\frac{1}{2}}}^{L_{j+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} A(x) \frac{\partial n}{\partial t}\bigg|_{x,L,t} dx\, dL \approx A(x_i)\, \frac{dn_{ij}}{dt}\, \Delta x_i\, \Delta L_j \tag{19}$$

The integral on the right-hand side of (18) is first solved in x direction and averaged in L direction:

$$\int_{L_{j-\frac{1}{2}}}^{L_{j+\frac{1}{2}}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} -\frac{\partial}{\partial x}\{A(x) v_p(x,L,t) n(x,L,t)\}\, dx\, dL \approx -\Delta L_j \left[A(x) v_p(x,L_j,t) n(x,L_j,t)\right]_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \tag{20}$$

Then, the remaining terms are approximated by

$$A\left(x_{i-\frac{1}{2}}\right) v_p\left(x_{i-\frac{1}{2}}, L_j, t\right) n\left(x_{i-\frac{1}{2}}, L_j, t\right) \approx \begin{cases} A_{i-\frac{1}{2}} v_{p,i-1,j}\, n_{i-1,j} & \text{if } v_{p,i-1,j} > 0 \\ A_{i-\frac{1}{2}} v_{p,i-1,j}\, n_{ij} & \text{if } v_{p,i-1,j} < 0 \end{cases} \tag{21}$$

$$A\left(x_{i+\frac{1}{2}}\right) v_p\left(x_{i+\frac{1}{2}}, L_j, t\right) n\left(x_{i+\frac{1}{2}}, L_j, t\right) \approx \begin{cases} A_{i+\frac{1}{2}} v_{p,ij}\, n_{ij} & \text{if } v_{p,ij} > 0 \\ A_{i+\frac{1}{2}} v_{p,ij}\, n_{i+1,j} & \text{if } v_{p,ij} < 0 \end{cases} \qquad (22)$$

($A_{i+\frac{1}{2}}$ is the cross section area at the right border of volume element i.)

By this approximation, it can be guaranteed that exactly the same amount of mass, as leaves one finite volume, enters another finite volume, independent of sign changes of v_P.

4.2.2 Model representation in ProMoT

As mentioned in Section 2, ProMoT offers two possibilities to structure a process model, in a "horizontal" way on one level of hierarchy, and in a "vertical way" by introducing different hierarchical levels. The horizontal structuring is achieved by defining model building blocks with inputs and outputs and by connecting these inputs and outputs graphically by the ProMoT GUI. For the example of the crystallizer process a very obvious approach is to represent each of the devices in the plant by one building block and to represent the piping between the devices by input-output connections. As a result one obtains an implementation of the crystallizer process model as is shown in the ProMoT screenshot in Figure 7.

<Figure 7>

The model structured displayed by the visual model editor on the right-hand side of Figure 7 is very close to the plant structure shown in Figure 4. It should also be note that the connections between input and output terminals in ProMoT are vectorial, i.e. each link may transport a vector of arbitrary length, containing e.g. the number size densities along the L coordinate at the crystallizer outlet and the corresponding liquid phase composition.

The "vertical" structuring method is also employed in the crystallizer example. This can be seen from the class browser shown on the left-hand side of Figure 7. One may recognize a module "grid", which is a super-class of the modules "crystallizer", "feed_tank", "filter", and "us" representing models of the devices in the plant. The module "grid" just contains a discretization of the property coordinate, i.e. the number and position of the grid points in L. By defining all modules which make use of the property coordinate as sub-classes of "grid" automatically ensures a consistent and redundancy free model implementation in the sense that all modules use the same property grid and modifications of the grid only have to be made at one point in the model. Of course, this is a very simple example of inheritance. More extensive use of aggregation and inheritance mechanisms in ProMoT has been made in [42,43]. In the present example, one could further use inheritance to fork different versions of the crystallizer model by putting the common model parts into one super-class and implementing model variants, e.g. different spatial discretization techniques, in sub-classes.

ProMoT automatically translates the generated plant model into C++ code for Diana. In this example, 124 grid points are chosen in x direction and 50 grid points are chosen in L direction, resulting in a equation system of 6324 differential equations and 353 algebraic equations. After a compilation step, this equation system is ready for numerical analysis in Diana.

4.3 Simulation Results in Diana

In a first step, dynamic simulation in Diana is used to study the start-up behavior of the crystallizer process. It is assumed that at the beginning of the simulation a seeding population of rather large crystals is placed in the upper half of the crystallizer. The feed flow rate, the product flow rate, and the flow rate to and from the US attenuator are kept constant. The solver DASPK is chosen for the integration of the differential algebraic model system. The simulation is carried out over 100000 s of simulation time, requiring 1020 s of CPU time on a laptop computer with 2.50 GHz processor and occupying about 670 MB of computer memory.

Figure 8 shows the resulting dynamic evolution of the crystal population. The seeding crystals grow due to the super-saturation of the liquid phase and gradually sink to the bottom of the crystallizer. When reaching the crystallizer's outlet to the ultrasonic attenuator, they are sucked out of the crystallizer, crushed and recycled as small crystals into the crystallizer. Hence, a population of smaller crystals forms at the bottom (time 500 s in Figure 8). This population grows in particle size and rises with the liquid flow (time 10000 s and 20000 s in Figure 8). The supply of fresh small particles at the crystallizer bottom slows down, when most of the large particles have been consumed in the US attenuator (time 30000 s in Figure 8). It takes a little time, until new large particles have formed and sunk to the ground (time 40000 s in Figure 8). Then the process repeats. By looking at the total mass in the crystallizer in Figure 9 one can see the periodic nature of the dynamic behavior more clearly (a simulation over a longer time horizon confirms that the system actually settles on a strictly periodic orbit). The physical explanation for the found behavior may be the selective removal of large particles from the crystallizer in combination with the crystal growth rate, which has been identified as a source of instabilities in other particle systems [44,45].

<Figure 8>

<Figure 9>

From an application point of view, the periodic oscillations are usually undesired, and the question arises what operational or constructive modifications could be made to stabilize the system. Tools for bifurcation and singularity analysis may help to solve this problem. As an example, the dependence of the system behavior on the feed flow rate is shown in the bifurcation diagram in Figure 10. One can see that the periodic oscillations occur over a quite large range of values for the feed flow rate. Actually, it is possible to suppress the oscillations by choosing a feed flow rate above the value of 9.35 l/h given by the Hopf bifurcation in Figure 10.

<Figure 10>

5 Conclusions and Outlook

ProMoT and Diana are a pair of free open-source tools for the dynamical modeling and numerical analysis of differential algebraic systems. ProMoT offers a powerful object oriented modeling language and a graphical user interface. It converts symbolic model information into simulation code for various simulation tools. One of the supported tools is Diana. Diana contains a collection of state-of-the-art numerical algorithms for steady state solution, integration, optimization, and singularity analysis of differential algebraic systems, as they typically result from the modeling of chemical and biochemical processes. Diana has a text based user interface via Python scripts and grants access to the complete functionality of the Python scripting language. This allows an effortless formulation of complex simulation tasks.

The application example of a fluidized bed crystallizer in Section 4 documents that ProMoT and Diana are able to support the implementation and numerical analysis of challenging models of chemical engineering systems and that their use provides insight in the dynamic behavior of state-of –the-art chemical processes.

Future work will simplify the implementation of models with partial differential equations. Further, ProMoT will be extended to support additional simulation tools.

6 Acknowledgement

The analysis of the fluidized bed crystallizer system is financially supported by DFG in the framework of SPP 1679, contract nr. MA 2051/2-1.

The development of ProMoT and Diana has been supported over many years by scholarships for master and Ph.D. students provided by Kompetenznetz Verfahrenstechnik Pro3.

Special thanks go to all the Pro3 interns who contributed with their programming work to the ProMoT / Diana system: Mykhaylo Krasnyk, Andriy Chut, Sergiy Gogolenko, Constantin Teplinskiy, Igor Babenko, Kateryna Bondareva, Michael Chayka, Maksym Dosta, Anna Guseva, Leonid Iakushyk, Antonia Isayenko, Yuriy Ivanov, Valdimir Kushnarenko, Oleksiy Milokhov, Oleksandr Miroshkin, Oleksandr Sherbakov, Kyrylo Sodatov, Vyachesla Trubarov.

7 References

[1] R. Schreiber, *Scholarpedia* **2007**, *2(7),* 2929. DOI: 10.4249/scholarpedia.2929

[2] Official website http://www.gnu.org/software/octave

[3] Official website http://www.scilab.org

[4] Official website http://www.psenterprise.com/gproms.html

[5] Official website http://www.aspentech.com/products/aspen-plus.aspx

[6] M. Tiller: *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, Dordrecht **2001**

[7] F. Tränkle, M. Zeitz, M. Ginkel, E. D. Gilles, *Mathematical and Computer Modelling of Dynamical Systems* **2000**, 6(3):283 - 307.

[8] M. Ginkel, A. Kremling, T. Nutsch, R. Rehner, E. D. Gilles, *Bioinformatics* **2003**, 19(9):1169-1176.

[9] S. Mirschel, K. Steinmetz, M. Rempel, M. Ginkel, E. D. Gilles, *Bioinformatics* **2009**, 25(5):687-689.

[10] M. Krasnyk, K. Bondareva, O. Milokhov, K. Teplinskiy, M. Ginkel, A. Kienle, in *Proc. of the 16th European Symposium on Computer Aided Process Engineering,* Elsevier, Amsterdam **2006.**

[11] M. Krasnyk, M. Ginkel, M. Mangold, A. Kienle, *Computers & Chemical Engineering* **2007,** 31:1100-1110.

[12] M. Krasnyk, *Ph.D. Thesis,* Otto-von-Guericke-Universität Magdeburg, **2008.**

[13] R. Kelling, G. Kolios, C. Tellaeche, U. Wegerle, V. Zahn, A. Seidel-Morgenstern, *Chemical Engineering Science* **2012**, 83:138-148.

[14] M. Mangold, A. Bück, R. Schenkendorf, C. Steyer, A. Voigt, K. Sundmacher, *Chemical Engineering Science* **2009**, 64:646-660.

[15] M. Mangold, M. Krasnyk, K. Sundmacher, *Chemical Engineering Science* **2004**, 59:4869-4877.

[16] K. Kolczyk, R. Samaga, H. Conzelmann, S. Mirschel, C. Conradi, *BMC Bioinformatics* **2012**, 13:251.

[17] S. Mirschel, K. Steinmetz, A. Kremling, in *Proceedings of 6th Vienna Conference on Mathematical* Modelling, MATHMOD, Vienna, **2009.**

[18] R. J. Fateman, P.N. de Souza, J. Moses, C. Yapp: *The Maxima Book.* URL http://maxima.sourceforge.net, **2004**.

[19] M. Mangold, A. Kienle, K.D. Mohl, E.D. Gilles, *Chemical Engineering Science* **2000,** 55:441-454.

[20] M. Hucka et al., *Bioinformatics* **2003,** 19(4):524-531.

[21] M. Jarke, J. Köller, B. Braunschweig, W. Marquardt, L. von Wedel, in *Proc. of 1st IEEE Conference on Standardisation and Innovation in Information Technology,* Aachen, **1999.**

[22] Official website http://www.python.org

[23] J. Dongarra, *International Journal of High Performance Applications and Supercomputing* **2002,** 16(1):1-111.

[24] G.H. Golub, C.F. Van Loan: *Matrix Computations,* John Hopkins University Press, Baltimore, **1996.**

[25] T.A. Davis, *ACM Transactions on Mathematical Software* **2004,** 30:196-199.

[26] R.B. Lehoucq, D.C. Sorensen, C. Yang: *ARPACK Users' Guide,* SIAM, Englewood Cliffs, **1998.**

[27] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, C.S. Woodward, *ACM Transactions on Mathematical Software* **2005,** 31(3):363-396.

[28] S. Li, L.R. Petzold, *Journal of Computational and Applied Mathematics* **2000**, 125(1-2):131-145.

[29] J.R. Leis, M.A. Kramer, *ACM Transactions on Mathematical Software* **1988**, 14(1):61-67.

[30] U. Nowak, L. Weimann, *A family of Newton codes for systems of highly nonlinear equations,* Technical Report, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, **1991.**

[31] A. Wächter, L. Biegler, *Mathematical Programming* **2006**, 106(1):25-57.

[32] J.A. Nelder, R. Mead, *The Computer Journal* **1965**, 7(4):308-313.

[33] C.D. Perttunen, D.R. Jones, B.E. Stuckman, *Journal of Optimization Theory and Application* **1993**, 79(1):157-181.

[34] J. Mockus, W. Eddy, A. Mockus: *Bayesian Heuristic Approach To Discrete and Global Optimization*, Kluwer Academic Publishers, Dordrecht, **2000.**

[35] G.M. Shroff, H.B. Keller, *SIAM Journal on Numerical Analysis* **1993**, 30(4):1099-1120.

[36] K. Lust, *Ph.D. thesis,* Katholieke Universiteit Leuven, **1997.**

[37] H. Tung, *Crystallization of organic compounds: an industrial perspective* .Wiley, Hoboken, **2009.**

[38] D. Binev, H. Lorenz, A. Seidel-Morgenstern, in *Produktgestaltung in der Partikeltechnologie.* Fraunhofer Verlag, Stuttgart, **2011.**

[39] D. Binev, H. Lorenz, A. Seidel-Morgenstern, in *Proc. of 18th Int. Workshop on Industrial Crystallization (BIWIC 2011),* Delft, **2011.**

[40] D. Binev, H. Lorenz, A. Seidel-Morgenstern, in *Proc. of 19th Int. Workshop on Industrial Crystallization (BIWIC 2012),* Tianjin, **2012.**

[41] S. Palis, D. Binev, H. Lorenz, A. Seidel-Morgenstern, A. Kienle, in *Proc. of 20th Int. Workshop on Industrial Crystallization (BIWIC 2013),* Odense, **2013.**

[42] M. Mangold, M. Ginkel, E.D. Gilles, *Computers & Chemical Engineering* **2004,** 28:319-332.

[43] R. Hanke, M. Mangold, K. Sundmacher, *Fuel Cells* **2005,** 5:133-147.

[44] R. Radichkov, T. Müller, A. Kienle, S. Heinrich, M. Peglow, L. Möhrl, *Chemical Engineering and Processing* **2006,** 45(10):826-837.

[45] S. Palis, A. Kienle, *Chemical Engineering Science* **2012**, 70:200-209.

8 Figure Legends

Figure 1: Schematic structure and usage of ProMoT and Diana (adapted from [12])

Figure 2: Implementation of the differential equation $\frac{dx}{dt} = -x,\ x(0) = 1$ in the model definition language MDL provided by ProMoT.

Figure 3: A very simple Petri net and its implementation in MDL: A system is initially in a state ‘not_full’ and goes to a state ‘full’, when a switching function $h_{max} - h$ becomes negative.

Figure 4: A simple example of the Python scripting language used to control the simulator Diana. A model named “Crystallizer” is loaded; the integrator DASPK is chosen for numerical solution; then dynamic simulation is executed until end time 1000.

Figure 5: Scheme of a fluidized bed crystallizer setup, consisting of a feed tank, the crystallizer, an ultrasonic attenuator and a sieve for product removal.

Figure 6: volume element for the spatial discretization of the crystallizer model.

Figure 7: Screen-shot showing the ProMoT implementation of the fluidized bed crystallizer system.

Figure 8: Dynamic simulation of the fluidized bed crystallizer system – contour plots of the number density function $n(x, L, t)$.

Figure 9: Dynamic simulation of the fluidized bed crystallizer system – total particle mass in the crystallizer.

Figure 10: Bifurcation diagram with feed flow rate as bifurcation parameter; solid line = stable steady state; dashed line = unstable steady state; filled square = Hopf bifurcation; filled circles = minima and maxima of the total particle mass on the periodic orbits; dotted line = time average of total particle mass on the periodic orbits.

Figure 1:

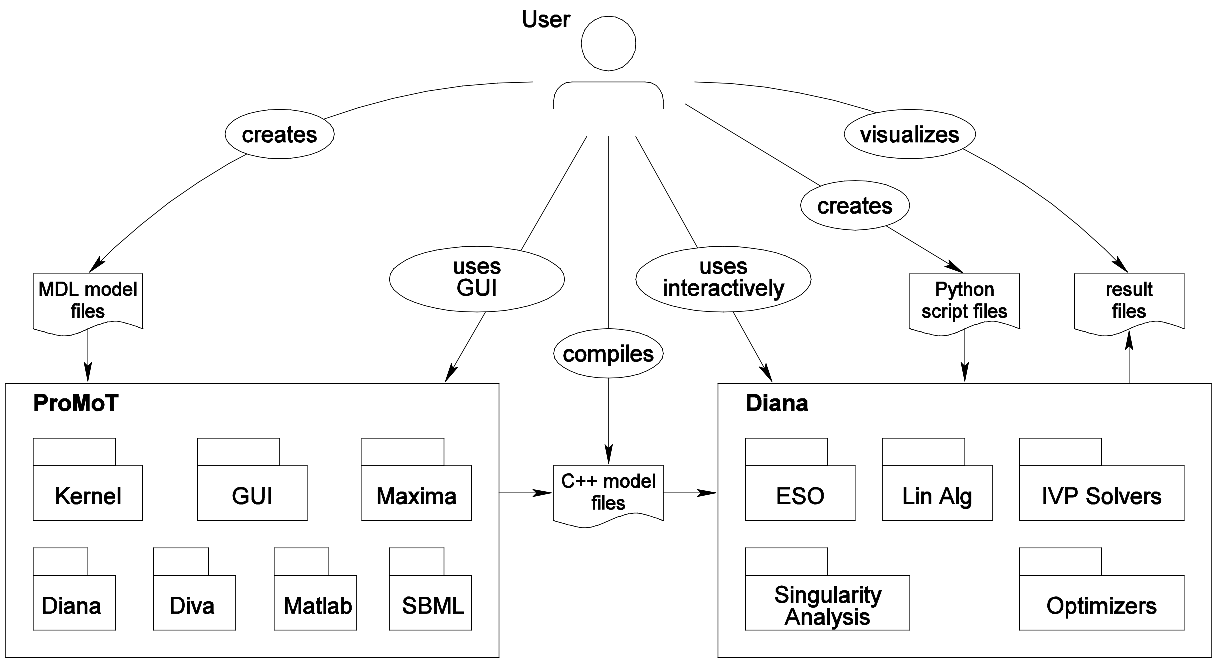
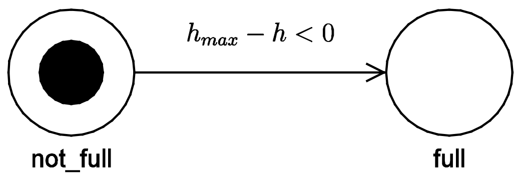


Figure 2:

```
(define-module :class "simple_ode"
  :super-classes ("module")
  :variables
  (
   ("x"
    :system-theoretic "state"
    :value "1.0"
    )
   )
  :equations
  (
   ("1st_order_ode"
    :relation ":diff<t> x == -x"
    )
   )
)
```

Figure 3:



```
(define-module :class "petri_net"
  :super-classes ("module")
  :places
  (
   ("not_full"
    :is-a "place"
    :initial-mark "yes"
    )
   ("full"
    :is-a "place"
    )
   )
  :transitions
  (
   ("transition_1"
    :phi-function "hmax-h"
    :from ("not_full")
    :to   ("full")
    )
   )
)
```

Figure 4:

```
dmain=diana.GetDianaMain()

# invoke diana model manager
mmanager=dmain.GetModelManager();

# initialize model "crystallizer"
model=mmanager.CreateModel(diana.CAPE_CONTINUOUS, "Crystallizer.so");
model.Initialize();

# initialize solver daspk
sfactory=dmain.GetSolverFactory();
solver=sfactory.CreateSolver(diana.CAPE_DAE, model, "daspk.so");
solver.Initialize();
solpar=solver.GetParameters();

# simulate crystallizer model until end time 1000.0
solpar["Tend"].SetValue(1000.0);
solver.Solve();
```

Figure 5:

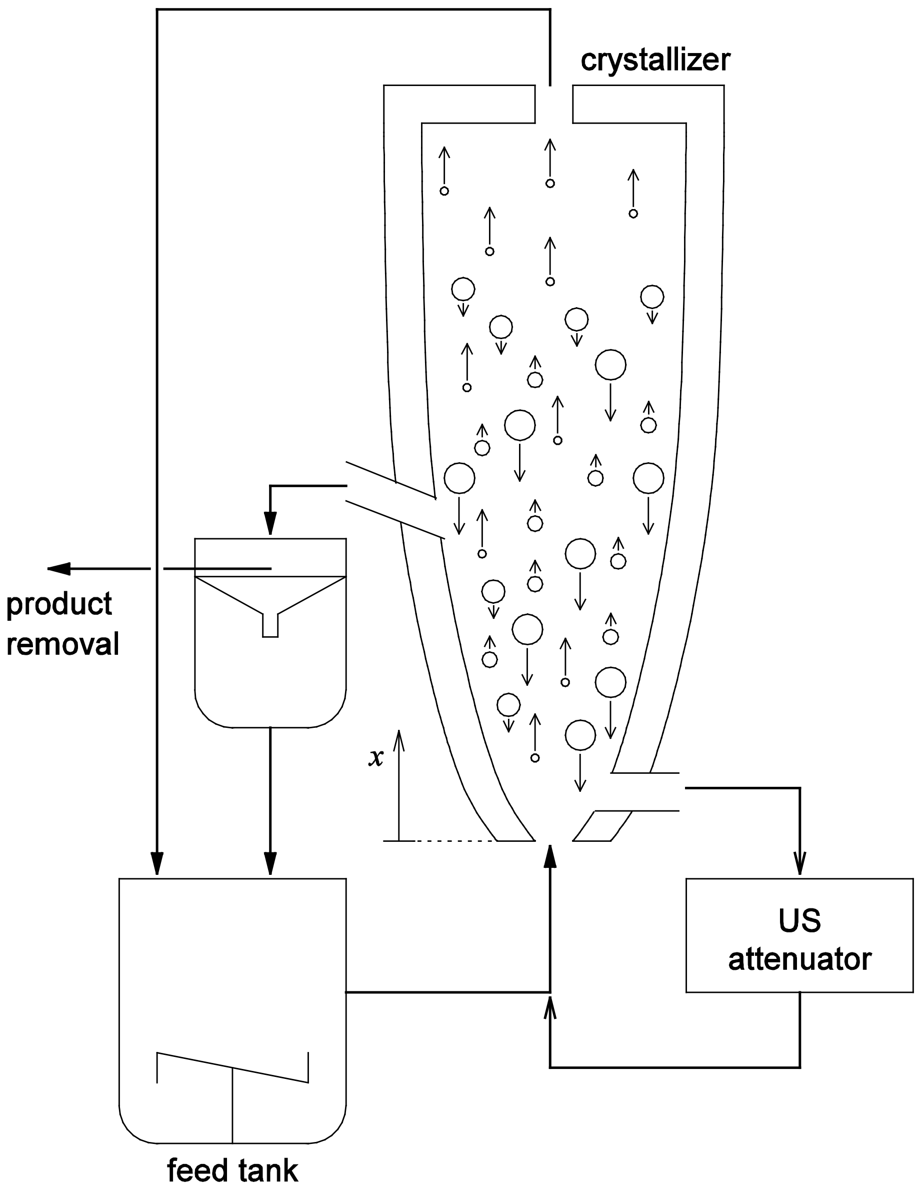


Figure 6:

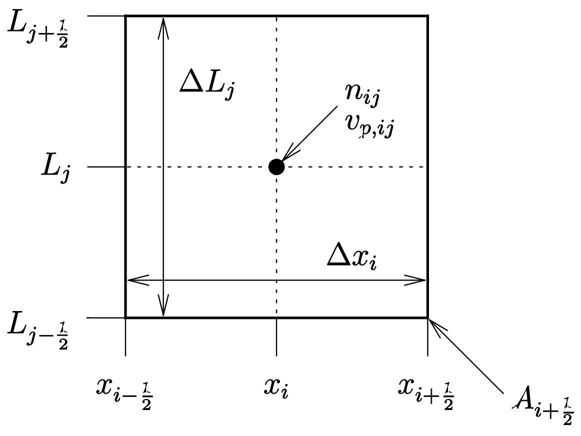


Figure 7:

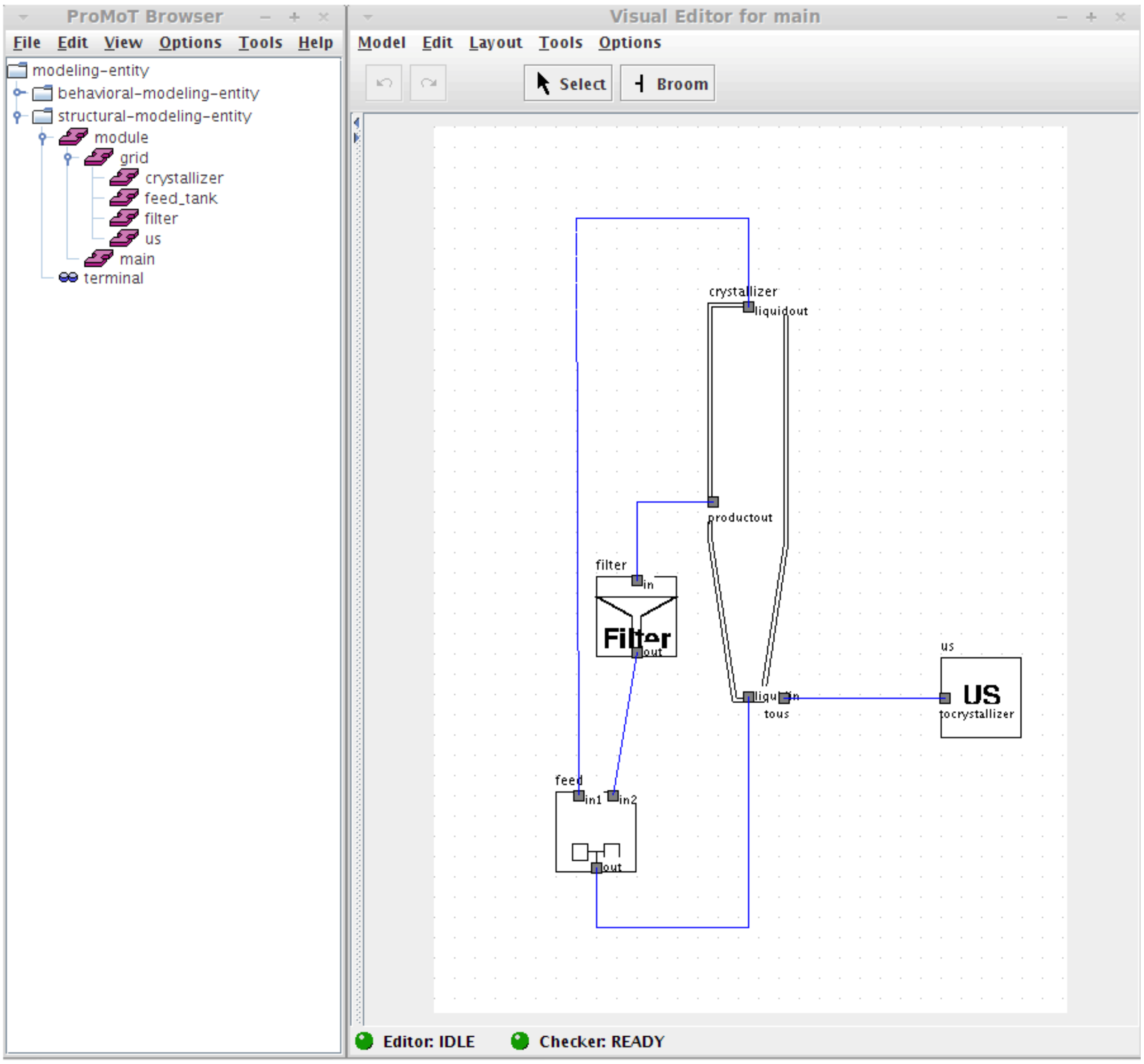

ProMoT Browser
File Edit View Options Tools Help
modeling-entity
behavioral-modeling-entity
structural-modeling-entity
module
grid
crystallizer
feed_tank
filter
us
main
terminal
Visual Editor for main
Model Edit Layout Tools Options
Select
Broom
crystallizer
liquidout
productout
filter
in
Filter
out
liquidin
tous
us
US
tocrystallizer
feed
in1
in2
out
Editor: IDLE
Checker: READY

Figure 8:

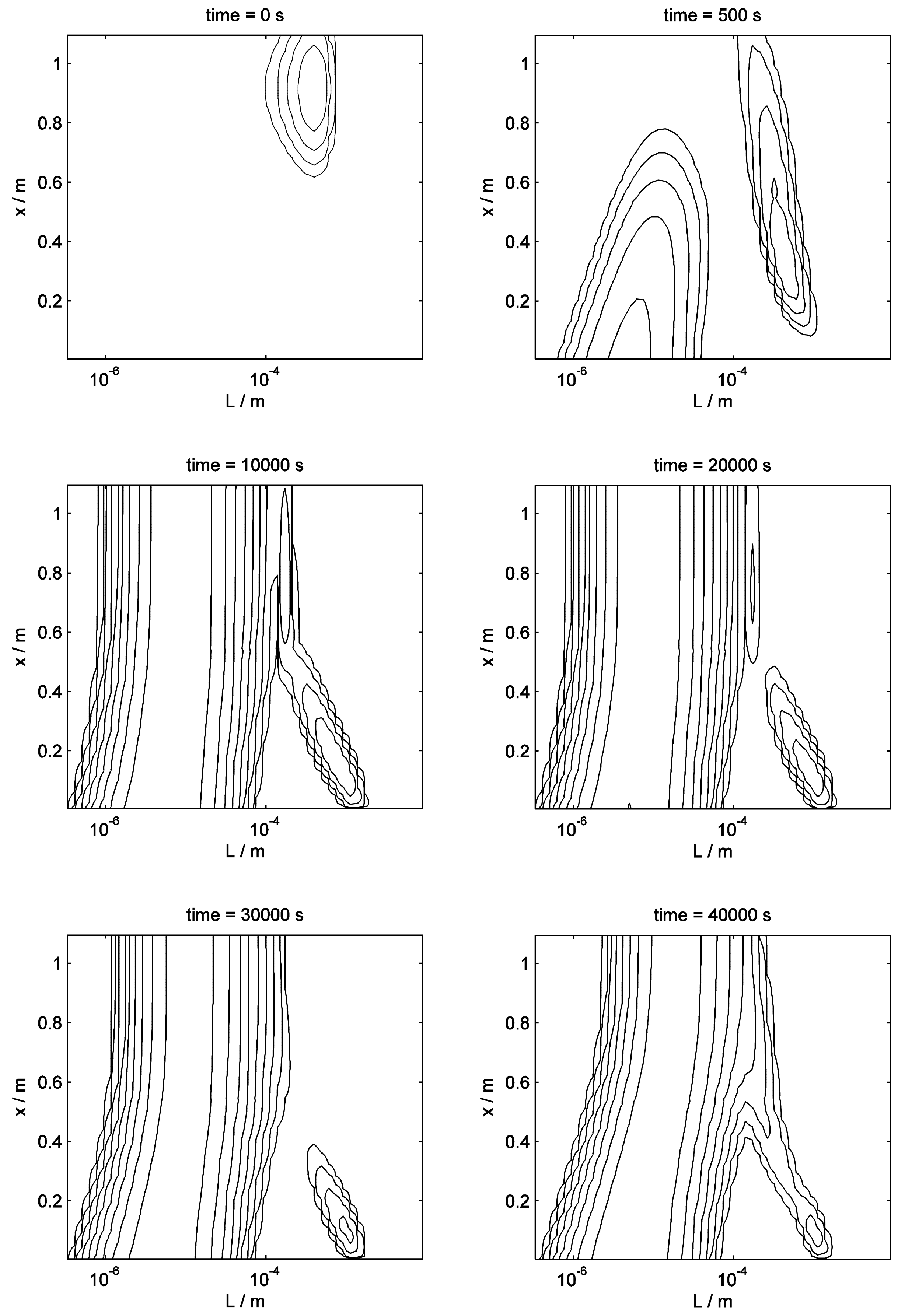
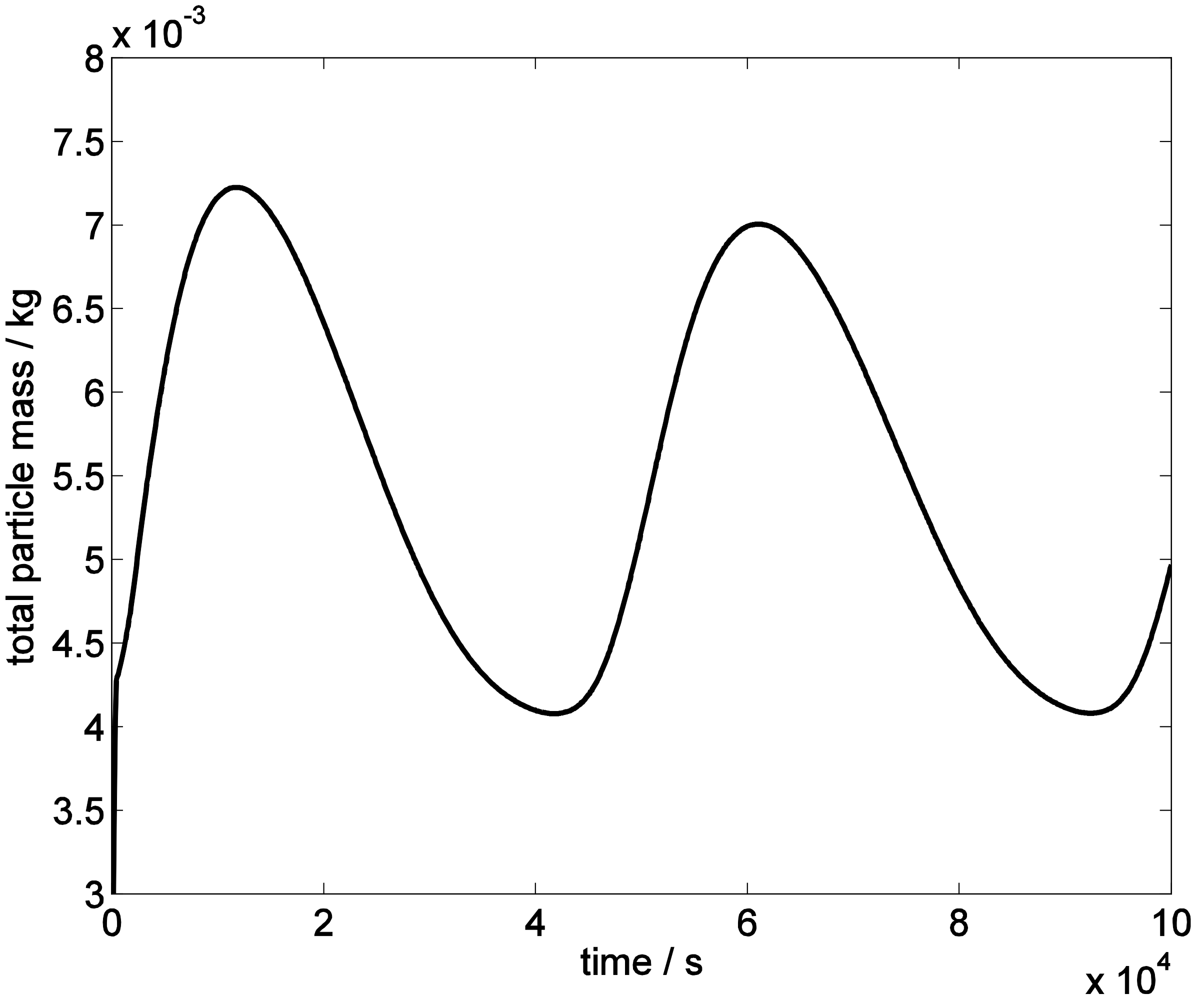
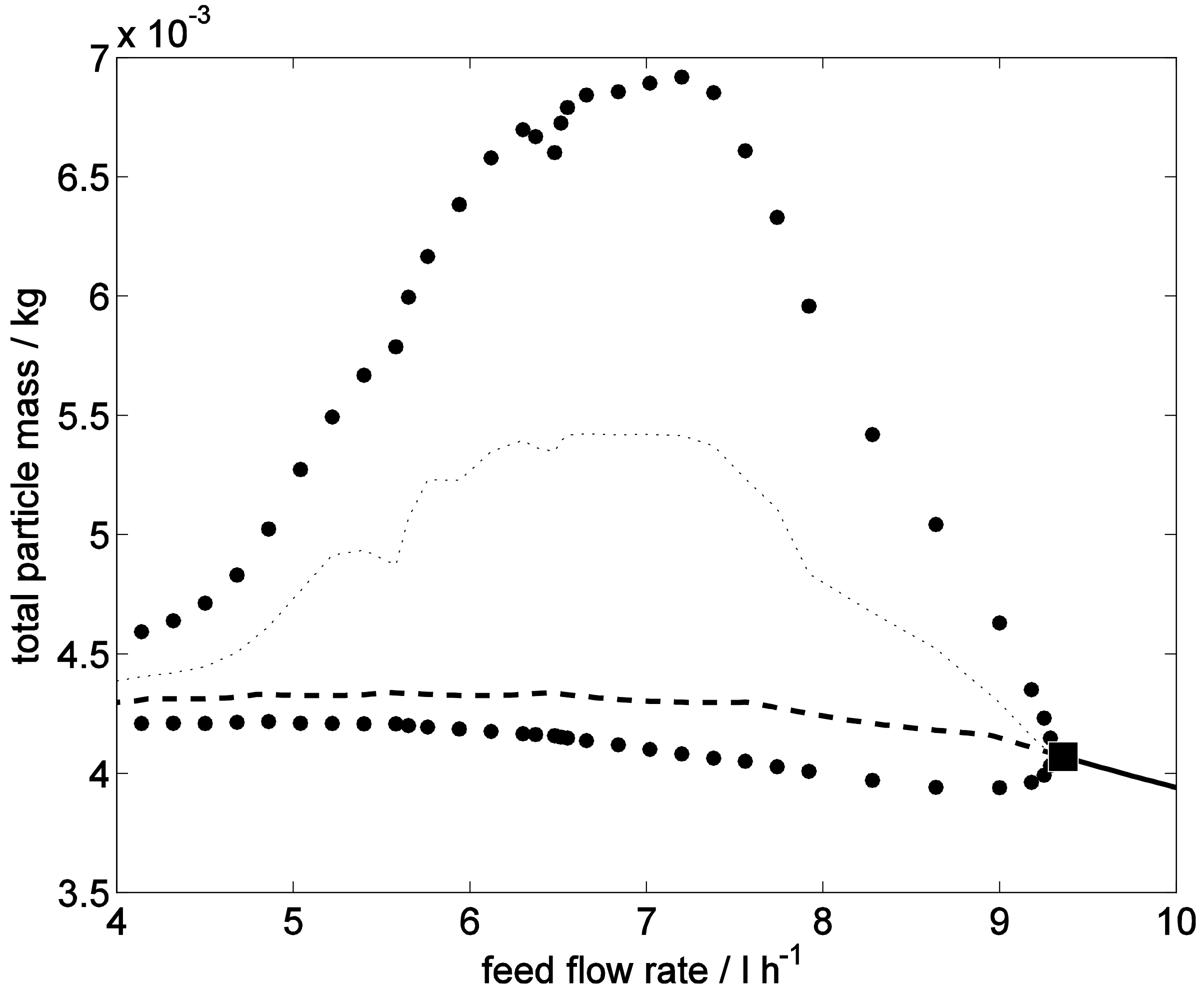

time = 0 s
time = 500 s
time = 10000 s
time = 20000 s
time = 30000 s
time = 40000 s
x / m
L / m
1
0.8
0.6
0.4
0.2
10^{-6}
10^{-4}

Figure 9:



x 10^-3
8
7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
total particle mass / kg
0
2
4
6
8
10
time / s
x 10^4

Figure 10:


x 10^-3
7
6.5
6
5.5
5
4.5
4
3.5
total particle mass / kg
4
5
6
7
8
9
10
feed flow rate / l h^-1