CHEMICAL ENGINEERING TRANSACTIONS Volume 18, 2009
Editor Jiří Klemeš
Copyright © 2009, AIDIC Servizi S.r.l., ISBN 978-88-95608-04-4 ISSN 1974-9791
DOI: 10.3303/CET0918127

Web-Based Object-Oriented Modeling Environment for the Simulation of Chemical Processes

S. Kuntsche, H. Arellano-Garcia, G. Wozny
Chair of Process Dynamics and Operation, Berlin Institute of Technology
Sekr. KWT-9, Str. des 17. Juni 135, 10623 Berlin, Germany

In this work, a new equation system oriented modeling environment for process engineering is presented. The project is aimed to safe time and energy of engineers during the formulation of the equation systems and the desired problem statements. While code generation in several languages based on symbolic equations is one of the key tasks of the software, further aims are improved cooperation via internet and better accessibility of research results of co-workers. A motivation is given on how the modeling aspects focused in this project can facilitate research and development processes.

1. Introduction

1.1 Research and global cooperation

Research and development processes consist in general of the enrichment of existing knowledge, i. e. research done previously by other researchers is used as fundament for further investigations that deliver new results. These new results are made available (either on a public or on an enterprise wide level) for other researchers who in turn use the new results as fundament for their research. Thus, one can consider the ongoing research process as a cyclic procedure of retrieving published results and publishing new ones. Accordingly, research work is based on a cooperation that is independent from the location of the co-workers. The means of communication in that cooperation is publications or technical documentation. In process engineering new findings are often achieved by computer simulation, which in many cases means to solve large equation systems numerically. This goes along with analytic determination of derivatives and with programming of functions and derivatives. New results are achieved by alternating equations or by adding new ones. All these steps are time consuming and error-prone (Grossmann & Westerberg, 2000). The same applies to keeping documentation and publications up to date and consistent with the program code. The problem is that the mathematical information and its equivalent programming code are kept in different documents and that these documents are updated independently and without an enforced systematic (Brandt et al., 2008). On the other hand, the simulation problems are fully described in the literature or documentation in the form of symbolic equations and programming is thus only an auxiliary procedure.

1.2 Description of established modeling tools

To facilitate mathematical modeling, many software tools and standards have been developed. A thorough overview over modeling aspects and approaches addressing the respective difficulties is given in Zerry (2008). In this contribution, only a few tools and aspects shall be considered. The most basic approach of computer modeling is using a plain programming language s. a. FORTRAN and writing equations and jacobian matrix in a problem statement, which is subsequently solved by numeric methods taken from a library or by iteration algorithms developed especially for the problem at hand. Apart from that commercial software tools like AspenPlus or ChemCAD provide readily tested models for most of the equipment used in process engineering. While the approach of programming directly has the advantage of giving freedom in the problem description, readily programmed models are easy to use and available for more and more equipment and modeling approaches. However, it is often desired to adapt or extend existing models such that standard models in flow-sheeting tools like AspenPlus or ChamCAD cannot be used. Tools like gProms and MapleSim provide advanced facilities to enter and organize large equation systems while they perform symbolic determination of derivatives and provide powerful solving routines. Thus, these tools allow the engineer to concentrate on the technical application described, and to divide large problems into manageable parts. In particular MapleSim (among others) allows symbolic description of mathematic equations, which further narrows the gap between computer readable code and the information displayed in literature. In general, software tools are used to reduce complexity for the user and to perform standard tasks, such as numeric solving and determination of derivatives.

Established modeling tools work on local computers and do not provide generation of documentation. Therefore errors due to redundancy of data may arise in cooperation (Yang et al., 2004).

2. Addressed Aspects in Modeling

Symbolic modeling

As described in the introduction, the transfer of mathematical equations from symbolic formulation in literature into a code that can be evaluated by a computer is very important in modeling. The process of documentation can be considered as another transfer process of information, which is as important as the former one. The optimization of these transfer process are not yet addressed by current software. The project presented in this work aims to facilitate this part of modeling. The first step to do this, is to work with symbolic description of equation systems in order to avoid differences between the display in literature and in the computer code. Symbolic presentation of equations in numerical software is already offered by several tools on the market (see above). The aim in this project is to establish an engine for the evaluation of symbolic equations that is as close as possible to literature. At the same time, several ways of equation editing will be offered so that a choice for the most comfortable and efficient interface can be left to the user. One step towards literature and documentation is the mandatory description in a notation object for all symbols used in an equation or equation system.

Problem reformulation

One obstacle encountered when programming equation systems in languages like C or FORTRAN is that the decision for the iteration variables and design values is a fixed part of the code. In general, a reclassification, i. e. making one or more iteration variables design values, and replacing them by former design values, takes much programming effort and testing time. In the tool proposed, this step takes place before automated programming and can be undertaken without much work of the engineer.

Reusability and modularity

Well tested and long established equations and models should not be rewritten over and over again. Researchers need to use model parts from co-workers. It is obvious that different authors use different notations (Soetjahjo J., et al. 1998). However, the naming of variables in the documentation should be independent from their original model, since authors implementing model parts from others might want to present some of the imported equations in their own publication, consequently using a different notation. Thus, appropriate translation and name tracking methods are provided. Furthermore, a model library will be established to provide the standard equations and full equations systems for standard equipment in process engineering.

Internet cooperation

To strictly avoid redundancy of data, the tool will work on a server and its user interface will be accessible via internet browser. The modeling server will provide work spaces for groups of users, where models can be created and used independently from the location of the respective group member. A model library containing standard equations and model parts will be provided on that server.

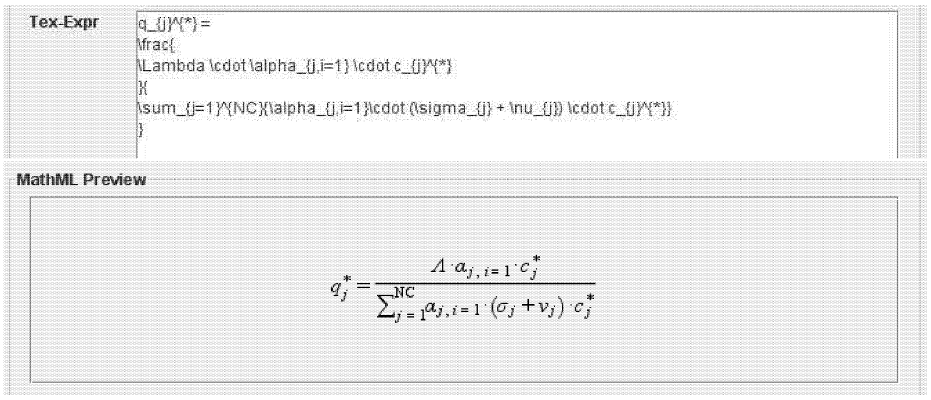
Output

As a first step, the software will generate numerical program code representing the problem described symbolically. This code can be executed on the server or an executable can be downloaded for local execution depending on the expected numerical burden. The program code itself can be downloaded by researchers, who want to use their own solvers. Furthermore, documentation is generated on the basis of the mandatory descriptions in the modeling structures, e. g. the nomenclature.

3. Results

In this section, the handling of the software developed in this work is shown using an example related to chromatography. The aim is to calculate the stationary-phase concentration q and the separation factor α according to equations stated by Brooks and Cramer 1995. The problem consists of two classes of equations:

$$\alpha_{j,1} = k_{1,j} \cdot \left(\frac{q_1^*}{c_1^*} \right)^{v_j - 1} \tag{1}$$



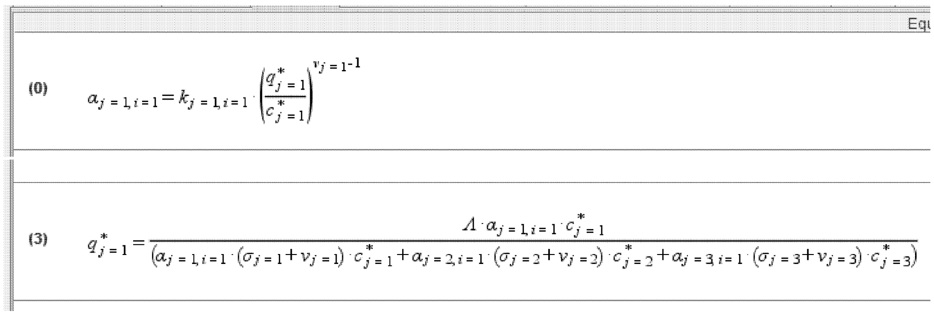
```
q_{j}^{*} =
\frac{
\Lambda \cdot \alpha_{j,i=1} \cdot c_{j}^{*}
}{
\sum_{j=1}^{NC}{\alpha_{j,i=1}\cdot (\sigma_{j} + \nu_{j}) \cdot c_{j}^{*}}
}
```

$$q_j^* = \frac{\Lambda \cdot \alpha_{j,\,i=1} \cdot c_j^*}{\sum_{j=1}^{NC} \alpha_{j,\,i=1} \cdot (\sigma_j + \nu_j) \cdot c_j^*}$$

Figure 1. Equation editing: In the basic approach tex expressions are translated to Presentation MathML code, which can be analysed by the modelling tool.

$$q_j^* = \frac{\Lambda \cdot \alpha_{j,1} \cdot c_j^*}{\sum\limits_{j=1}^{NC} \alpha_{j,1} \cdot (\sigma_j + \nu_j) \cdot c_j^*} \tag{2}$$

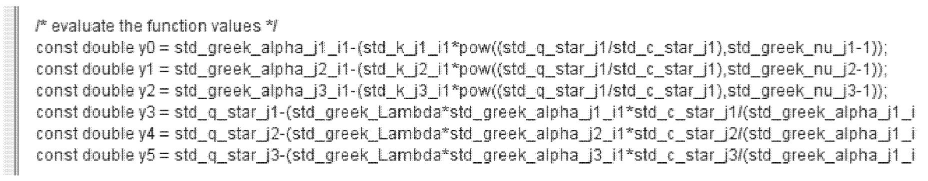
As a first step, we create a notation xml-file for this example, where all identifying elements are declared and provided with a description. After that the equations are entered by the help of an equation editor. The tool works with the mathematical web standard MathML. As entering the equations in MathML directly is an arduous task and as engineers time should not be wasted in that step, the equations are entered in the tex symbolic language. In further steps, the two equation classes are combined to an equation system. The index j is given the maximum value 3 in the next step and the equation system is instantiated accordingly. The result is shown in figure 2. Note that the sum expression in the bottom of equation (2) is reformulated to inline summands.



(0) $\alpha_{j=1,i=1} = k_{j=1,i=1} \cdot \left(\frac{q_{j=1}^*}{c_{j=1}^*}\right)^{\nu_{j=1}-1}$

(3) $q_{j=1}^* = \frac{\Lambda \cdot \alpha_{j=1,i=1} \cdot c_{j=1}^*}{\left(\alpha_{j=1,i=1} \cdot (\sigma_{j=1} + \nu_{j=1}) \cdot c_{j=1}^* + \alpha_{j=2,i=1} \cdot (\sigma_{j=2} + \nu_{j=2}) \cdot c_{j=2}^* + \alpha_{j=3,i=1} \cdot (\sigma_{j=3} + \nu_{j=3}) \cdot c_{j=3}^*\right)}$

Figure 2. Instantiation of the equations: Snippet showing equations (0) and (3), for index j=1. The summation in the denominator in eq (3) has been resolved to an explicit expression.



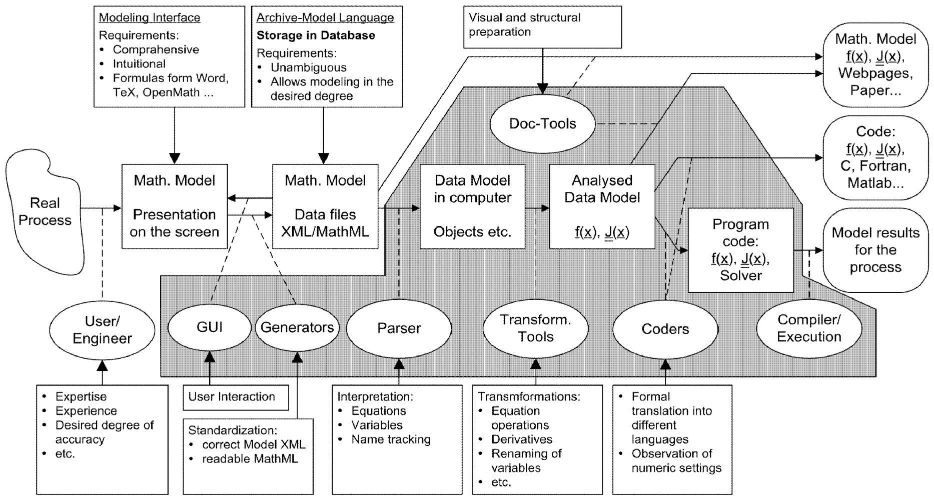
```
/* evaluate the function values */
const double y0 = std_greek_alpha_j1_i1-(std_k_j1_i1*pow((std_q_star_j1/std_c_star_j1),std_greek_nu_j1-1));
const double y1 = std_greek_alpha_j2_i1-(std_k_j2_i1*pow((std_q_star_j1/std_c_star_j1),std_greek_nu_j2-1));
const double y2 = std_greek_alpha_j3_i1-(std_k_j3_i1*pow((std_q_star_j1/std_c_star_j1),std_greek_nu_j3-1));
const double y3 = std_q_star_j1-(std_greek_Lambda*std_greek_alpha_j1_i1*std_c_star_j1/(std_greek_alpha_j1_i
const double y4 = std_q_star_j2-(std_greek_Lambda*std_greek_alpha_j2_i1*std_c_star_j2/(std_greek_alpha_j1_i
const double y5 = std_q_star_j3-(std_greek_Lambda*std_greek_alpha_j3_i1*std_c_star_j3/(std_greek_alpha_j1_i
```

Figure 3. Generated code: Snippet of the code for the calculation of the function values in C.

After successful instantiation, the variables have to be classified by the user into design values and iteration variables. This is done by appropriate intuitive user interface elements. The code generated subsequently is shown in figure 3. In this case the translation of the problem is done to the language C. For the solution, a root finding algorithm of the open source numeric library GSL (Gsl, 2008) is used. The simulation results can be stored and subsequently be loaded as initialization values. The problem can be reformulated easily by reclassifying the variables between design and iteration values respectively and by reassigning their values. Such problem statements can be stored separately.

Basic structure

An overview over the structure of the software is given in Figure 4. In this diagram, oval elements represent program parts, while the rectangle boxes in the middle represent the data structures carrying the model information in different levels of abstraction. The rectangle boxes at the bottom and the top of the diagram give additional information on program parts or data structures.



Modeling Interface
Requirements:
• Comprehensive
• Intuitional
• Formulas form Word, TeX, OpenMath ...
Archive-Model Language
Storage in Database
Requirements:
• Unambiguous
• Allows modeling in the desired degree
Visual and structural preparation
Math. Model f(x), J(x), Webpages, Paper...
Doc-Tools
Code: f(x), J(x), C, Fortran, Matlab...
Real Process
Math. Model Presentation on the screen
Math. Model Data files XML/MathML
Data Model in computer Objects etc.
Analysed Data Model f(x), J(x)
Program code: f(x), J(x), Solver
Model results for the process
User/ Engineer
GUI
Generators
Parser
Transform. Tools
Coders
Compiler/ Execution
• Expertise
• Experience
• Desired degree of accuracy
• etc.
User Interaction
Standardization:
• correct Model XML
• readable MathML
Interpretation:
• Equations
• Variables
• Name tracking
Transmformations:
• Equation operations
• Derivatives
• Renaming of variables
• etc.
• Formal translation into different languages
• Observation of numeric settings

Figure 4. Structure of the software. Model information is processed from left to right in the diagram.

4. Conclusions

A simulation environment using symbolic mathematical formulations has been established. The description of equation systems is kept close to the appearance of symbolic formulations as it is found in literature. Thus, the gap between the problem statement in literature and its representation in the simulation software is narrowed considerably. The process of code generation into different languages has been automatized so that time and energy is saved. Reformulating the simulation problem, i. e. reclassification of design and iteration values, is made very easy in comparison to reformulating program code. Sample cases could be executed successfully.

Outlook

Further steps will be the thorough testing and consolidation of the code. In a related project the necessary model database will be set up and filled with standard equations and model parts. The interface between literature and software will be improved so that the process of entering equations comes closer and closer to a copy-paste procedure.

5. Acknowledgements

The authors acknowledge support from the Cluster of Excellence "Unifying Concepts in Catalysis" coordinated by the Berlin Institute of Technology and funded by the German Research Foundation.

References

Brandt, S. C., Morbach, J., Miatidis, M., Theißen, M., Jarke, M., & Marquardt, W. An ontology-based approach to knowledge management in design processes. Comp. Chem. Engg., 32 (1-2), 320-342.

Brooks and Cramer, 1995, Solute Affinity in Ion-Exchange Displacement Chromatography, Chemical Engineering Science, Vol 51, No 15, 3847-3860

Grossmann, I.E., & Westerberg, A. W. (2000). Research challenges in process systems engineering. AIChE J., 46, 1700-1703.

Gsl, 2008, GSL – GNU Scientific Library, <www.gnu.org/software/gsl/>

MapleSim, <www.maplesoft.com/products/maplesim/index.aspx>

Soetjahjo, J., Go, Y. G., Bosgra, O. H., (1998). Diag – a structural diagnose tool for interconnection assignment in model building and re-use. Comp. Chem. Engg.Vol 22, Suppl., pp 933-936.

Zerry, R., 2008, MOSAIC, Eine webbasierte Modellierungs- und Simulationsumgebung für die Verfahrenstechnik. Shaker Verlag, Aachen, ISBN 978-3-8322-7148-0

Yang, A., Morbach, J.,, & Marquardt, W. (2004). From conceptualization to model generation: The roles of ontologies in process modeling. In C. A. Floudas & R. Agarwal (Eds.), Proceedings of the 6th international conference on foundations of computer-aided process design (pp. 591-594).