NTNU – Trondheim
Norwegian University of
Science and Technology

Ontology-based Computer-aided modelling Tool

Sebastian Roll

Chemical Engineering and Biotechnology
Submission date: July 2012
Supervisor: Heinz A. Preisig, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Abstract

A long-lived software project for using the advantages of a computer in modelling of physical-chemical-biological processes has been under development by my supervisor Prof Heinz A. Preisig. The software is designed to assist the modeller in making consistent and structurally solvable models, reducing the time frame and the amount of random mistakes. The model is built as a basic graph in the bottom, with subsequent layering of information onto the graph, describing its physical nature. The finished models is meant to have automatic code generation for existing solvers, so that it can be used for simulation, system identification or for control optimization problems.

The scope of the project has been to assist Prof Presig with software design and overall strategy. The biggest concepts faced were on how to deal with phase change in models, what physical descriptions should be available for the graph and how these physical descriptions are best structured to prevent infeasible systems. The current version of the computer-aided modelling project is written in Python, using the graphical user interface environment PyQt4.

Acknowledgments

I would like to thank my parents and siblings for an unwavering support throughout my studies. My partner Ruth has also earned my deepest gratitude. She is my anchor.

Thank you, Prof Heinz A. Preisig, for every single discussion we have had. The academically themed talks were insightful, but the personal conversations are the ones that I will remember when I look back at my life as a student.

Contents

List of Figures

List of Tables

Nomenclature

A Helmholtz energy, (J)
G Gibbs energy, (J)
U Internal energy, (J)
H Enthalpy, (J)
h Molar enthalpy, (J/mole)
S Entropy, (J/K)
N Mole number, (mole)
k Heat transfer coefficient, $(W/(m^2K))$
p Pressure, (Pa)
C Number of components, (-)
P Number of phases, (-)
μ Chemical potential, (J/mole)
β Valve constant, (-)
D Diffusion coefficient, (m^2/s)
V Volume, (m^3)
c_p Heat capacity at constant pressure, (J/K)
T Temperature, (K)

Chapter 1

Introduction

1.1 Goal and scope

Process modelling is often a time-consuming and arduous task. It is prone to inconsistencies and accidental errors. There is much to gain from having a modelling tool to automate what can be automated, enforcing consistency in the model while maintaining richness in modelling options.

The tool described in this report is meant to ensure completeness and correctness in the process of modelling. It is designed around the concept of ontology, which can be understood as a formal, explicit specification of a shared conceptualization. The finished software is meant to create models for process simulation, system identification or optimal control solving.

This report will introduce the concept and functions of the modelling tool, as well as some theoretical basis.

1.2 Problem description

The objective of the project is to generate an ontology to be used in connection with the process modeller developed by Prof Preisig. The project involves structuring of the knowledge associated with generating models for chemical and related processes. Further, software objects are being defined to capture the knowledge, being first used to define the ontology, for which a wizard-class editor is to be adapted accordingly. The ontology is then used in configuring and utilization of the modelling tool.

1.3 Outline of thesis

Chapter 2 presents the program and the basis upon which its modules are built. The directed graph is explained and the way it is given context by use of attributes. The issue of phase transition in a software modelling context is also discussed. Chapter 3 presents the software interfaces and gives an example on how the software would be used in practice. Conclusion and future work is summarized in chapter 4.

Chapter 2

Software basis

2.1 Modelling

The approach to creating a complete description of a model is best shown in figure 2.1 [6].

The variables explained:

- x^0: initial conditions of primary states
- $v(x, \theta)$: secondary states are computed from primary state variables and parameters
- $\tilde{x}(v)$: transposition is dependent on the state and parameters
- $\hat{x}(v, \theta)$: transport equations from secondary state variables and parameters
- $\dot{x}(\hat{x}, \tilde{x})$: state time differential from flows and transpositions
- $x = \int_t^{t+dt} \dot{x}$: integrate state time differential for a new primary state value

The procedure of modelling starts with a directed graph that shows capacities and how they interact. The graph consists of nodes and connection, and these units are coloured with certain attributes. This graph gives rise to the balance equations. For a mass system:

$$\dot{\underline{n}}_s = \underline{\underline{F}}^n_s \hat{\underline{n}}_s + \underline{\underline{R}}^T_s \tilde{\underline{n}}_s \tag{2.1}$$

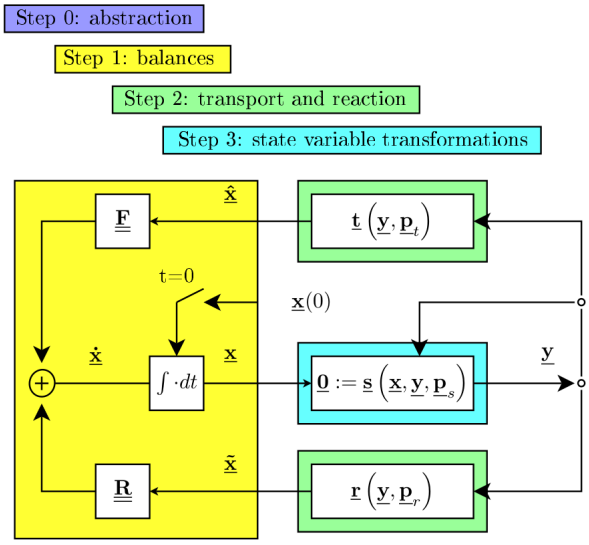

Figure 2.1: DiagramModelling

Here, the time differential of component mass in system s ($\underline{\dot{n}}_s$) is equal to the sum of flows to and from the system ($\underline{\underline{F}}^n_s \underline{\hat{n}}_s$) and the transposition of species inside the system ($\underline{\underline{N}}^T_s \underline{\tilde{n}}_s$). $\underline{\underline{F}}^n_s$ is the directionality matrix for the species in the graph, $\underline{\hat{n}}_s$ is the flow rate of the species, $\underline{\underline{N}}^T_s$ is the stoichiometric matrix and $\underline{\tilde{n}}_s$ is the kinetics of species transposition.

The next step is determining the flow rate $\underline{\hat{n}}_s$ and transposition rate $\underline{\tilde{n}}_s$. They usually depend on some conjugate variable of the energy function, such as temperature, pressure or chemical potential. These variables are called secondary state variables, because they in turn depend on the primary state variables. The primary state variables is most likely, but not necessarily, defined in the Kalman sense, meaning "the minimal information required to compute the future given the current input." [8]. For all part of this thesis, the primary state variables are component mass and internal energy.

The secondary state variables are expressed as functions of other secondary state variables, primary state variables, and parameters. The final link is the one between the primary state variables and the balance equations [7].

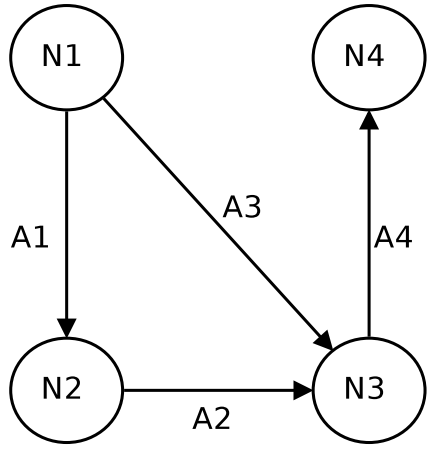

Figure 2.2: Graph of a mass transfer system

Let us consider a simple set of nodes and connections. The figure 2.1 shows the concept of systems (nodes) and how they interact. The nodes are containers of an extensive quantity, in this case mass, and the connections are mathematical boundaries in which the nodes transport their extensive quantities. The graph gives rise to the incidence matrix.

2.2 Incidence matrix

An incidence matrix is matrix that shows the connection between two different types of objects. If an element of row-indexed object A is connected to an element of column-indexed object B, the intersection of the column and row is 1. In a modelling context, the incidence matrix is row-indexed with nodes and column-indexed with arcs.

The model 2.1 has the incidence matrix:

$$\begin{bmatrix} -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Reading from the first column (node 1), it has non-zero elements in the first and third column, indicating that node 1 is connected to connection 1 and 3.

2.3 Software overview

The modelling software is made of the following parts:

- Graph editor and appearance+automaton editor
- Attributes and attribute editor
- Rules and rules editor
- Equations and equation editor

The graphic in figure 2.16 shows the parts and their information flow (graphic from *Ontology approach to model construction* by H.A. Preisig and Tore H. Warberg (2012))

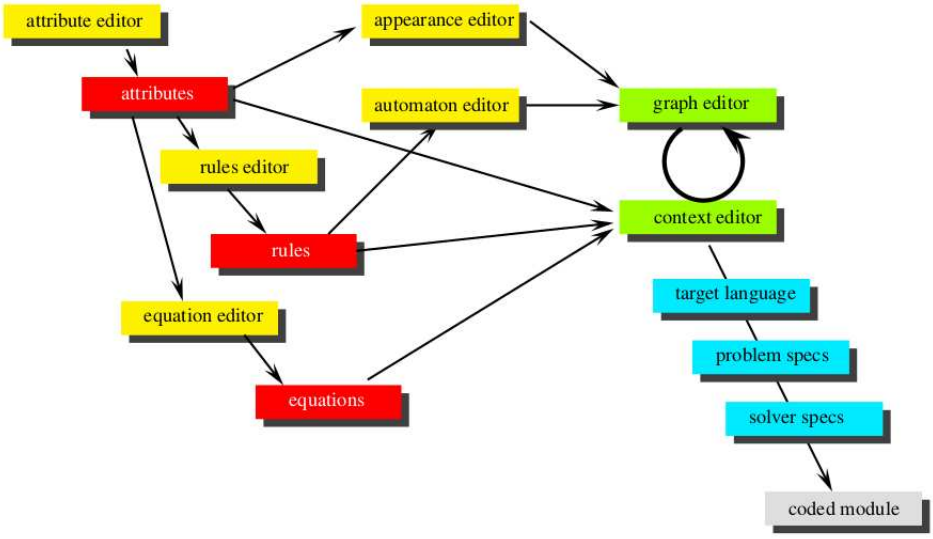

Figure 2.3: Architecture for the software *ProcessModeller*.

2.4 Ontology

The Greek word ontology comes from the word onto, "being, that which is" and logia, "study,theory". One definition popular in information science is: "An ontology is a formal, explicit specification of a shared conceptualisation. (Gruber, 1993). A more simplified explanation proposed by Fredrik Arvidsson and Annika Flycht-Eriksson is: "An ontology provide a shared vocabulary, which can be used to model a domain, that is, the type of objects and/or concepts that exist, and their properties and relations" [1]. In the specific case of the modelling tool, the ontology is explained as the information framework that is needed to give meaning to the basic graph.

The process modelling tool is based on an ontology approach in the fact that the graph editor is decoupled from what gives the graph meaning. Since the basic graph structures (nodes and connections) are so fundamental, they can be used for electric circuit modelling, information flow and much else. A super-user can add to or alter the existing attribute library (attributes is the data that gives context to the graph), or edit the rules that the attributes adhere by. In computer-science terms, this is like being able to edit the class itself (the ontology) to be able to create a different type of object (the instantiated model).

2.5 Nodes and connections

In this report, nodes and connections are collectively termed *units*.

Nodes

The model in its most fundamental form consists of nodes and arcs. Nodes are considered to be indivisible objects that may hold mass, energy or information. The entirety of any extensive property exists inside nodes.

Connections

A connection, or arc, represents interactions between nodes. It is defined as a transport of extensive quantities across a mathematical boundary that couples two systems together. This means that even though connections are transporting units, they don't actually hold any of the quantity that is transported.

The connection has a head node and a tail node, or is bi-directional (no head nor tail). If "physical", arcs carry transport equations, either in mass or energy. Arcs can also represent information flow, in which case there is no transport equation. A connections driving force is the potential created by a deviation between the head and tail node of an intensive property. [5] Examples of intensive quantities that create boundary potential are:

- Pressure: Drives mass flow through convective transport
- Chemical potential: Drives mass flow by diffusion
- Temperature: Driving force for heat transport
- Electrical potential: Electrical work

Mass flow through a connection also applies a transport of energy, in the form of internal, kinetic and potential energy, in addition to the pressure-volume work needed to displace the surroundings. The internal energy is the energy required to create the mass in its thermodynamic state from a vacuum, standing still and being outside any force fields. Kinetic energy constitutes the momentum gained from motion of the mass as a whole, and potential energy is the energy that comes from the mass being in non-equilibrium to a force field (e.g. gravity). Since mass transport implies energy transport, this means that both systems adjoined by the boundary

must be considered to be an energy capacity, albeit potentially a negligible capacity, unless energy as a whole is not in the scope of the modelling process. In other words, if a system has mass dynamics, it must also have energy dynamics. Energy dynamics, however, does not imply mass dynamics.

2.6 Attributes

Nodes and connections need a context to give it meaning. Nodes and arcs in itself only give information about the network structure of the model. In order to use it, one must give an explanation as to what the nodes and connections represent, and what is in them. This is where the concept of attributes come in. For physical-chemical-biological systems, reasonable attributes for describing a model can be found in table 2.1. The attributes with values are shown in table 2.6.

Table 2.1: Attributes for giving context to the basic graph.

Attribute	Description
Distinction	is the unit a system (node) or a link (connection).
Nature	is a node or connection of physical nature or is it a pure information unit?
Directionality	Does the flow in a connection go one way or both ways?
Dynamics	can we make any assumptions on the capacity of the node?
Distribution	lumped system or distributed in any of the three euclidean space dimensions?
State	is there energy, mass or information (or a combination of them) in the unit?
Phase	What phase is a node/connection in?
species	are there any species in the node/connection?

Table 2.2: Attributes with values.

Attribute	Values
distinction	node | dynamic
system	constant | dynamic | event | -
directionality	unidirectional | bidirectional | -
nature	physical | information
phase	liquid | solid | gas | fluid | - | ?
morphology	distrubuted | lumped
state	mass | energy
species	*

There are two goals that should be aspired for setting up the attributes and the structure that holds the attributes. Firstly, it needs to give sufficient information about the basic nature of the units in the model. Secondly, the structure of attribute combinations should only reflect reasonable combinations. An information node cannot have mass in it. It would give no meaning to use the dynamics attribute for connections. An important objective when designing the attribute structure is to minimize the amount of higher-level rules necessary to ensure feasible models.

2.7 Dynamics

The attribute *dynamics* is intimately related to the node. The value *dynamic* represents a system where the extensive properties (primary state variables) change when there is transport from the system, and both the intensive and extensive properties changes when transport comes in to it.

Event-dynamic nodes have no capacity, so there is no accumulation of mass or energy in the node. This reduces its differential algebraic equation (DAE) to an algebraic equation because the primary states does not change from in- and outflow (because they are always zero). Event-dynamic nodes can carry components of different phases. This enables them to be used as phase boundaries.

A reservoir is defined as a node that has a constant state, i.e. it has no dynamics. Their state differentials are not included in the equation set, but rather their intensive physical properties such as temperature, pressure and component concentration. These properties usually show up in the transport equations in and out of the reservoir.

2.8 Tokens

At this point it is helpful to introduce a concept termed tokens. When looking at the attributes that can be assigned to a units, one sees that some of them can influence neighbouring units. This is the case for the state, phase and species attribute. If a connection has the attribute *state* with value *mass*, then the same must apply for the head and tail node, and this attribute value should be added if it's not there already. A token is a collection of the attributes which describes the extensive quantity that is held and transported within a unit.

Tokens have some interdependencies. If the token *mass* exists in a system, then there should be a mass balance over that system. But there is now also energy in that system, since mass has enthalpy (the internal energy of the mass in addition to the pressure-volume work done by the mass). Therefore, when a system receives a mass token, it should also receive an energy token and a momentum token if energy is considered in the model scope. Each token gives a differential equation for one of the primary states. A mass token gives a differential equation in component mass and an energy

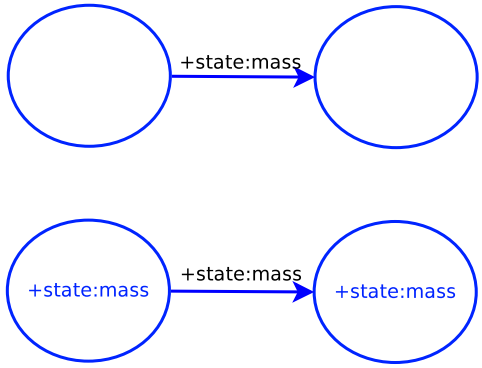

Figure 2.4: Token attributes influence neighbouring units.

token gives a differential equation for internal energy.

Figure 2.5-2.7 shows token copy from Node A to Connection 1, and again from Connection 1 to Node B. The transitions are fired by the algorithm if specific rules are met.

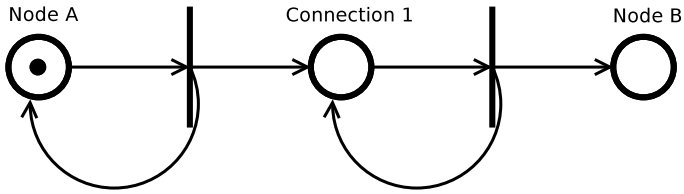

Figure 2.5: Token attributes influence neighbouring units.

Token propagation from node:
if nature in Connection 1 is *physical*
AND token is not in Connection 1
AND ((head of Connection 1 is not Node A) OR (Connection 1 is bidirectional)) **then**
Token → Connection 1
end if
Token propagation from connection:
if nature in Node B is *physical* **then**

if token is not in Node B **then**
Token → Node B

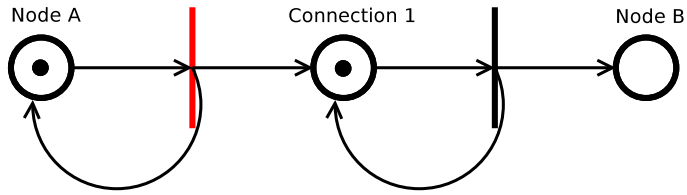

Figure 2.6: Token attributes influence neighbouring units.

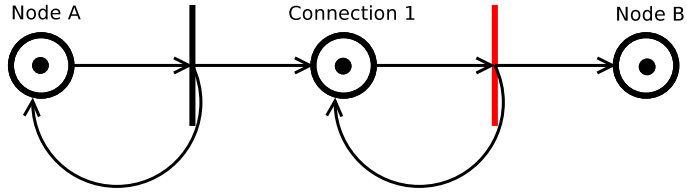

Figure 2.7: Token attributes influence neighbouring units.

end if

if token is not in Node A **then**
Token → Node A
end if
end if

Tokens are propagated as shown in table 2.3 [8]. Here t is mass and θ is energy. The first frame shows that a token injection in a connection will trigger the copy of the token to the connections nodes. The second and thirds frame shows mass and energy tokens spreading through unidirectional and bidirectional nodes respectively.

2.8.1 Token as energy

An energy token in a node represents the internal energy of the node. While in transport, the energy token can be in the form of conductive heat, radiation, work or similar. The head node receives the energy token and there it will be added to an internal energy balance. When mass-less energy is transported from the node to another node, the token in the connection cannot be considered internal energy. If the form of energy is to be represented in the node, the energy token would have to be transformed. First the token would be internal energy, then it would have another form in the connection (conductive heat, radiation, work), and back to internal energy again in the receiving node.

Table 2.3: Propagation rules for tokens.

state	node	arc	node	rule
Initialise				
before	-	t	-	
after	t	t	t	0
before	-	θ	-	
after	θ	θ	θ	0
unidirectional				
before	t,θ	t	t	
after	t,θ	t	t,θ	1
before	t	t	t,θ	
after	t	t	t,θ	-
bidirectional				
before	t,θ	t	t	
after	t,θ	t	t,θ	1
before	t	t	t,θ	
after	t,θ	t	t,θ	1

This method of keeping the form of energy in the attribute definition space requires rules to maintain. After deliberation with supervisor, the best place to handle the ambiguity in energy form is in the equation editor. By this logic, the attributes only presents the picture "this is energy transfer", not specifying the form off energy. This is done when the user chooses the energy transport equation. The equations should then have a tag, indicating if this is an equation typical for conduction, radiation, work or otherwise.

2.9 Attributes from a tree structure

The first thought out way of structuring attribute definitions is to set arrange them in a tree structure. The root is a dictionary with key="graph", and the value was a list. The list contains other lists. Each list is unique and always contains the same information, regardless of where it is being used. The end branches of the tree structure represent a full definition of the items, in this case nodes and arcs. All branches in the tree structure is unique and together they form the full spectre of attribute permutations.

The node and arc types defined by the attribute tree seems to work, up

until the phase boundary is determined to be an event-dynamic node carrying two phases. The list describing the phases in the tree structure always points to some other list. If an event-dynamic node is to be defined with two phases, the phase list will create an infinite loop, since all its values will point back to the phase list.

phase = [liquid — solid — gas — used-defined — *], choose liquid
liquid = [phase], choose phase
phase = [liquid — solid — gas — used-defined — *], choose solid. Now we have the two phases that is needed.
solid = [phase], must choose phase. This is an infinite loop.

One could define a second phase attribute, say *phase2*, which list elements point to the original phase list. Then the sequence for defining two phases for the event-dynamic node would be:

phase2 = [liquid2 — solid2 — gas2 — used-defined2 — *], choose liquid
liquid2 = [phase], choose phase
phase = [liquid — solid — gas — used-defined — *], choose solid. Now we have the two phases that is needed.
solid = [species], Next attribute, in this case species.

The elements in the phase2 attribute list must be different from the elements of the phase list. This solution is unintuitive and messy, as it adds at least four new lists (phase2, liquid2, solid2, gas2). Additionally, any new user-defined phases must be added two times, once to each phase list. This requires higher-level rules to implement.

2.10 Attribute as patterns

As described earlier, using the attribute tree is a challenge when it comes to event-dynamic systems having two phases, because of the following:

phase = [liquid]
liquid = [-]

The solution that is implemented in the software is as follows: You start with defining a list consisting of the list name, for example phase, and property values, such as [liquid | solid | gas | fluid | - | ?]. The list would then look like this: **phase** = [liquid | solid | gas | - | *]). Here ”-” is undefined

and * is user defined.

A unit keeps more information about itself than just the phase, so one would need a list of these lists to define the unit. This list of possible permutations is called a pattern generator in the ontology. If an instance of the pattern generator is made in which a value or a set of values is chosen among the alternatives, this instance is then called a pattern. A pattern contains the information connected to an object. It is fully specified, but it can also be edited, in which case the connected objects might also be affected.

If there were to be only one pattern generator containing all possible properties, it would imply the existence of many higher-level rules that asserts certain conditions. For example, if the following pattern generator was the only one:

<arc|node><directionality><system><token><phases><state><species*>, then there would have to be rules that stated:

- If <token> = information, <species*> must be set to "undefined".
- If <arc|node> = node, <directionality> is "undefined.
- etc..

Therefore, a set of possible pattern generators are necessary to minimize the number of higher-level rules.

2.10.1 Proposed attribute pattern generators

When defining a set of pattern generators, some syntax must be introduced. The "+" marker indicates that the value can be edited by the user. When a bracket is replaced with a value, such as when <nature> is replaced by physical, this means that <nature> is already specified to be physical and can have no other value.

Syntax:
<nature>: The choice between physical|information is exclusively calculated.
+<nature>: The choice is calculated, but can also be edited by the user.
+physical: The choice is user specified and it cannot be edited thereafter.

physical: The choice for <nature> is calculated and determined to be *physical*.

nodes:
+<constant|dynamic>.+physical.+<phase>.mass.+distribution>.+<species*>
+<constant|dynamic>.+physical.+<phase>.energy.+distribution>
+<constant|dynamic>.+physical.+<phase>.entropy.+distribution>
+<constant|dynamic>.+information

+event.+physical.+<phase a>.+<phase b>.mass.+<distribution>.¡species*>
+event.+physical.+<phase a>.+<phase b>.energy.+<distribution>
+event.+physical.+<phase a>.+<phase b>.entropy.+<distribution>
event.+information.+<token>

arcs:
+unidirectional.+physical.+<phase>.+<state.<distribution>.<species*>
+bidirectional.+physical.+<phase>.+<state.<distribution>.<species*>
+unidirectional.information

2.11 Phase and species

There is a way of handling phase transposition described in Westerweele, 2003 [9]. The flow matrix $\mathbf{F}^n$ usually does not have any information about the phase of the species. A species in the liquid phase must be distinguishable from the same species in the gaseous phase. The distinction can be made by treating them like two different species altogether in the flow matrix. The flow matrix would then use the product of species and phase instead of only species. For a hypothetical species C, a phase transition of C would increase the species space by 1, from [C*(old phase)] to [C*(old phase), C*(new phase)].

2.12 Rules

Rules facilitate the need for wanted behaviour that does not come from anywhere else. As an example, the attribute patterns makes sure that a user cannot define a node that has the unidirectional/bidirectional distinction attribute that is meant for connections. The attribute patterns do not, however, dictate what nodes can be connected to each other. Generally, if a certain behaviour can be accomplished without use of explicit rules, then that is preferred.

2.12.1 Connection rules

Not allowed:

- An arc cannot have mass node at tail and energy node at head.
- A node that is not event-dynamic cannot have more than one phase.
- A reservoir can not have both incoming and outgoing arcs.

The program will run a check for rule breach after every user action. If a rule breach is identified, the program reverts to request user action.

2.12.2 Rules for graph editor

The rules can also be regarding interface. These are just a few examples of what rules should exist for the graph editor.

- If a units *nature* attribute only has the element *information*, then represent the unit with dashed lines. Else (mass,energy,nothing), use a solid line.
- Different node representations between elements in the dynamics attribute.
- Colour the graph differently when it is deemed inconsistent or incomplete.

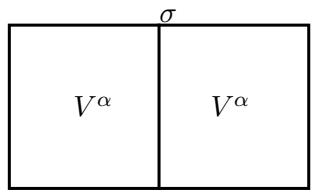

Figure 2.8: Gibbs interface

2.13 Treatment of phase boundary

There are several different ways of handling the concept of a phase boundary. The Gibbs model treats the phase boundary as ideally thin (figure 2.8), so the volume of the boundary is $V_B = 0$, and $V_\alpha + V_\beta = V_{tot}$. This could very well be a sufficient assumption for many situations and it should be possible to create with the software, but it is an over-simplification. In reality, the interface is a thin stratum in which the physical properties will vary continuously. [2]. This representation of a phase boundary should also be possible to create in the model.

2.13.1 Phase boundary as lumped capacity

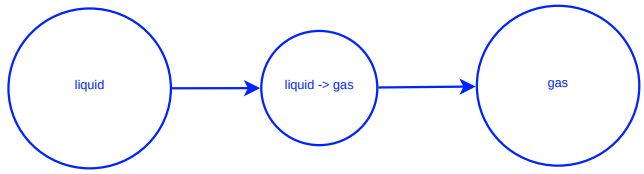

Figure 2.9: Phase boundary as lumped capacity.

Let's first consider the possibility of using a lumped capacity to describe a phase boundary, visualized in figure 2.9. A phase boundary contains more than one phase. The definition of a lumped system is a control volume with uniform intensive properties. A multi-phase system does not have uniform intensive properties, therefore a lumped system cannot be used as a phase boundary.

2.13.2 Event-dynamic system

The phase boundary can be defined as an event-dynamic system, i.e. a node with instantaneous dynamics. Two different phases can reside in this zero

or negligible capacity node.

The sequential routine for creating a liquid-gas system is shown in figure 2.10. The steps are:

1. Nodes are established
2. Connection between the nodes
3. The middle node is set to zero capacity by setting the attribute *dynamics* to *event-dynamic*
4. The *phase* attribute is set to *liquid*. The phase will spread to the head and tail node because of the rule set.
5. The boundary already has a phase *liquid*. The second phase is set to *gas*.
6. The second phase is spread to the right node because of the rule set.

2.13.3 Boundary with capacity

For the cases where a boundary must be treated like a three-dimensional space that can hold a capacity, the software needs to be able to facilitate this. One way to do it is to introduce two (or more) capacity nodes on each side of the event-dynamic node, as shown in figure 2.11. These capacity nodes can be used to include the continuous change of physical properties over the phase boundary.

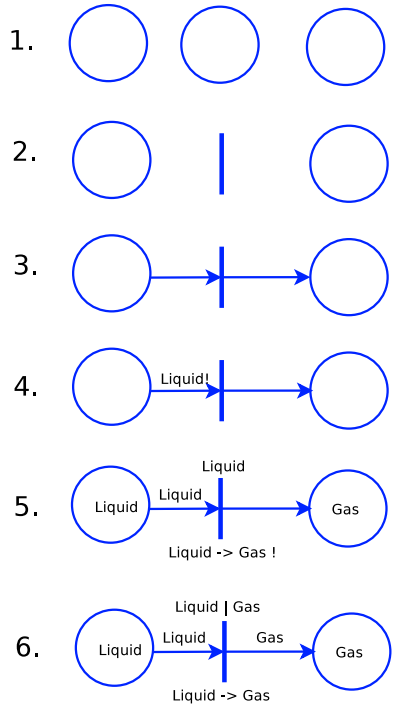

Figure 2.10: Method for establishing a phase boundary.

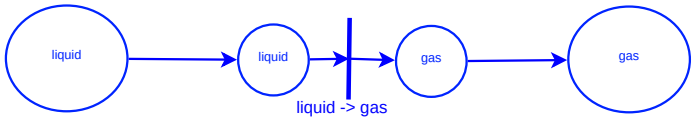

Figure 2.11: Boundary with capacity.

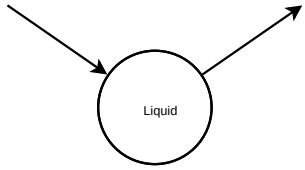

Figure 2.12: Liquid capacity with two mass connections.

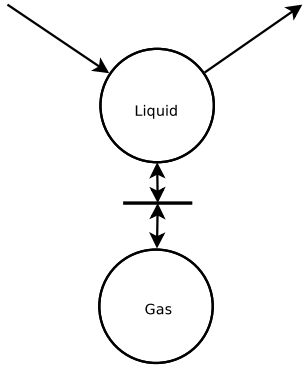

Figure 2.13: The liquid capacity becomes a two-phase system in computation time.

2.13.4 Handling undeclared phase transition

When a model is made and ready for computational work, each physical system has a phase assigned to it. In figure 2.12, a system has a liquid phase.

If the intensive properties in a system changes, so that a species would be in a different phase than the one specifies, the program can handle this automatically by creating a new node and a boundary (event-dynamic node). The new system is then connected to the existing one with bidirectional arcs. Figure 2.13 shows a new system with gas phase. Note that the gas is not transported anywhere but back into the liquid phase.

2.14 Software structure

Software architecture is something that should be carefully considered when working on big projects. A big software project is usually made of separate parts that are designed to be as independent of each other as possible. Sometimes some parts also have a stand-alone editor feature in the places where it is useful.

2.14.1 Levels of Design

When starting a new software project, it can be tempting for a beginner to skip the design process. Its a common mistake to start writing classes and routines before a firm overview of the project is established. This can lead to poor structure, redundant code and a lot of time spent on rewriting.

A developer should take design into consideration in all levels of the software system. According to *Code Complete* (McConnell, 2004) [4], the levels are:

1. Software system: What is the wanted functionality for the software system?
2. Division into subsystems/packages: What subsystems can I divide the software into? How should these subsystems communicate with each other?
3. Division into classes within packages: What classes seems natural to define? How do they relate to the rest of the system?
4. Division into data and routines within classes: Are the data and routines consistent within each class?
5. Internal routine design: What algorithm is best suited for doing what is supposed to be done by the routine?

The most challenging part of software design is the third point, dividing the program into subsystems.

2.14.2 Division into subsystems

Imagine a request for a piece of software that could scour statistical data from the web, and represent this data in a 3D environment. The natural thing to do is to identify the canonical subsystems. An example is shown

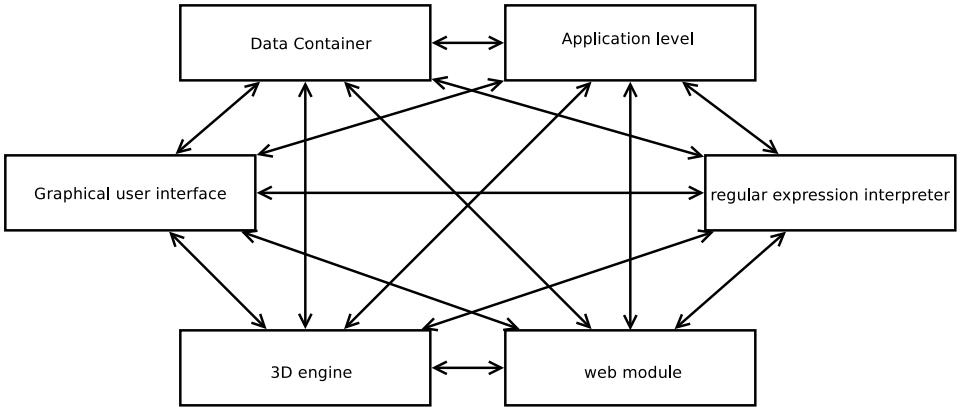

Figure 2.14: Subsystems with anarchic hierarchy

in figure 2.14.2.

The readers attention is probably on how the subsystems are linked in terms of communication. The figure shows an anarchic structure, which is not good for several reasons:

- A developer working on one subsystem probably needs to know something about the other subsystems it is connected to.
- The subsystems cannot easily be copied into different software projects because of all of its dependencies.
- If one wants to replace a subsystem, it could have an impact on many other subsystems.

By introducing structure in the way subsystems communicate, they become more portable, easier to maintain and more comprehensible for other developers. Allow communication between subsystems on a need to know basis-and it had better be a good reason. [4]

The computer-aided modelling software has a strict communication hierarchy between its subsystems. It is sequential in many parts, in that one subsystem inherits objects from previous ones. This structure is called context free, and is highly modular.

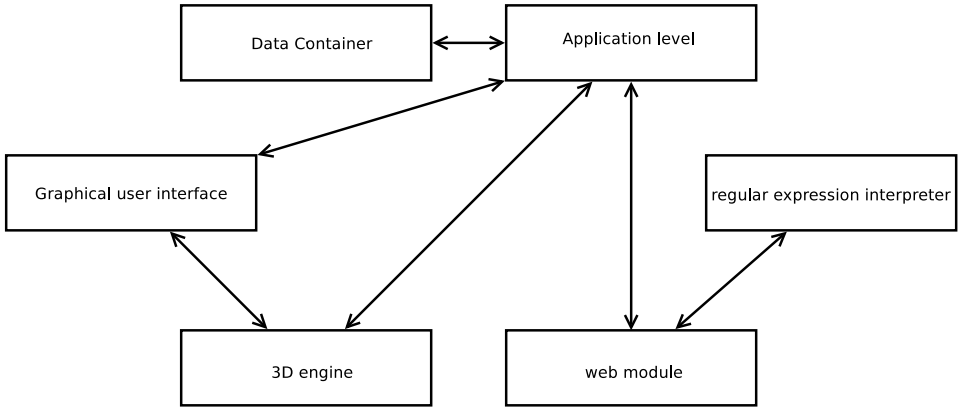

Figure 2.15: Subsystems with structured hierarchy

2.14.3 Modulization

Special focus has been made on separating information handling into compartments. The compartments are made so that if a change must be made in one area of the program, the editing process is confined to as few parts of the software as possible. As an example, if one needed to have commands that were specific for one operating system, for example folder management, it would be a good idea to make a separate module for these operating system dependent commands. So that instead of doing a Windows-specific procedure everywhere in the code, one could do a call to the module that would then do the procedure. In such a case, porting the program to a different operating system would not require searching the entire code for these procedures, only require editing in that particular module.

2.14.4 Attributes

Because of this modular way of designing the way attributes are handled, completely new phenomena can be incorporated into the ontology very easily. If a user wished to expand the scope of the model to include electrical charge build-up in capacity boundaries, he could add attributes to the event-dynamic system to facilitate for this in his models.

2.14.5 Equations

This module contains equations that the user can choose among when constructing a model. It contains secondary states equations, transport equations, reaction kinetic equations and parameters. Equations can be added,

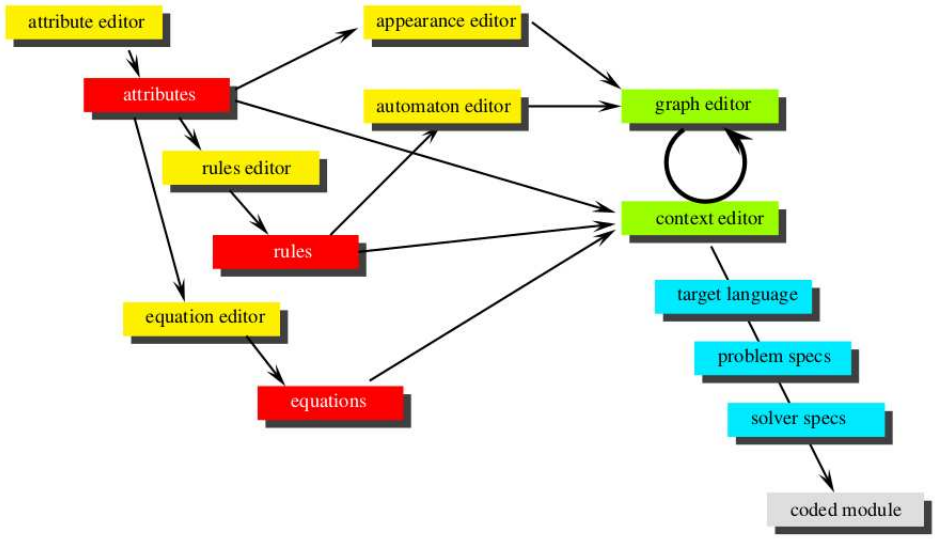

Figure 2.16: Architecture for the software *ProcessModeller*.

deleted and edited by an the editor. There may be one single or several different equations for a particular variable. As an example, mass transport from one node to another could have a range of equations to be chosen from.

Note that the primary state differential equations are not generated here, as they are automatically generated from the information provided by the graph. There may also be many distinct *sets* of equations, depending on what the model is meant to reflect. In cases where only mass should be considered as primary state, a new equation set could be generated, omitting all energy equations. in that case, the user is not presented with any choice regarding energy transfer. The equation editor accepts variables and equations that are specific for the ontology one wishes to construct. One can make a simplified ontology that is suited for basic modelling projects, or a more comprehensive ontology that might require more user input data from the model.

Deleting variables

When deleting variables, one will have to check if it is located in any of the equations. Since the variable cannot exist in equations made prior to it (equations can only use pre-existing variables), only the equations that were made after the variable have to be checked.

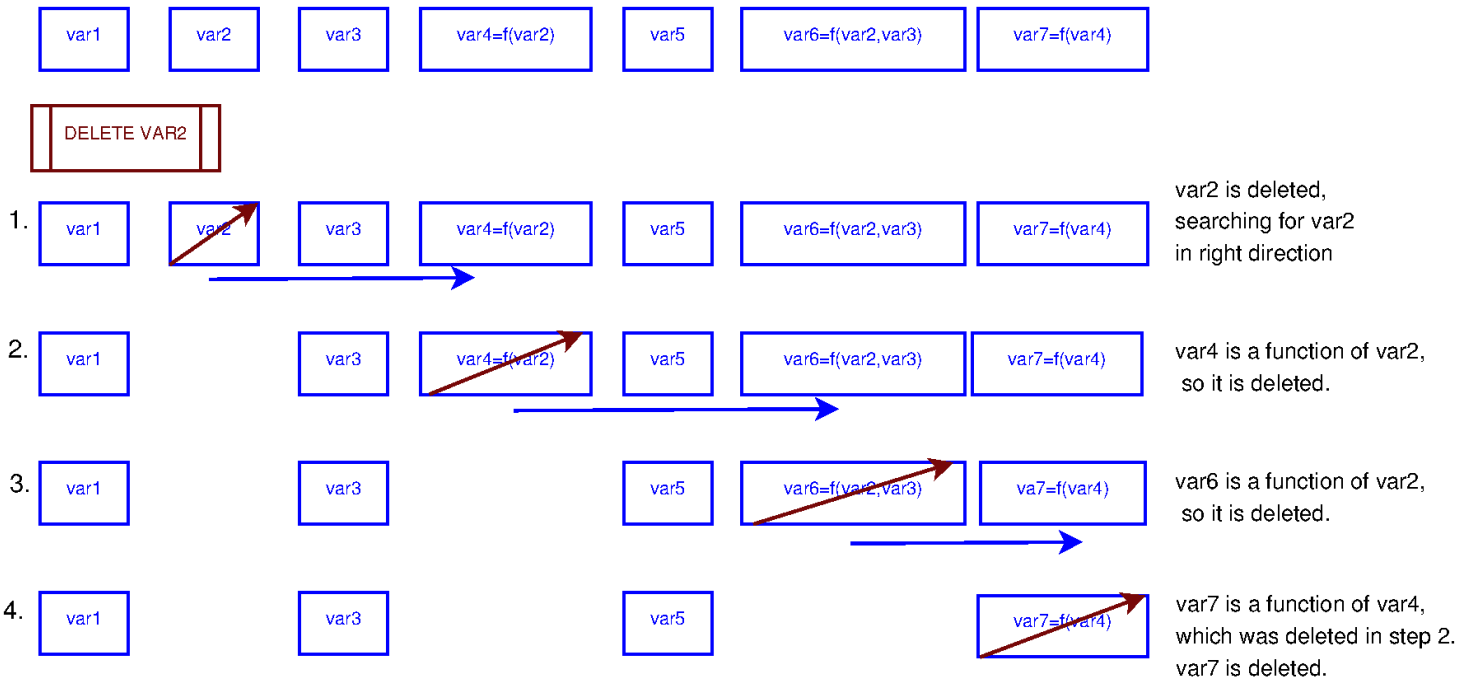

Figure 2.17: Deleting equations from editor.

Chapter 3

Implementation and discussion

3.1 Software

Figure 3.1 was an editor for the old attribute setup, the tree structure. The structure is itself represented as an expanded pane tree. Attributes could be added and deleted as branches through the *edit* option.

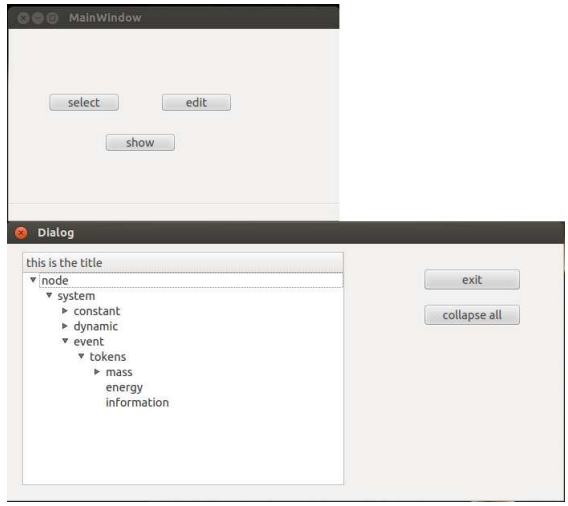

Figure 3.1: Old editor for attribute tree structure.

The tree structure editor in figure 3.1 was replaced by an editor for the new pattern generator attribute structure (figure 3.4). The radio box

define indicates possibility for a user to choose between the attribute values. The radio box *combi* allows multiple values to be chosen for that particular attribute. The attribute *token:[mass, energy]* is an example of where both values should be possible in a system.

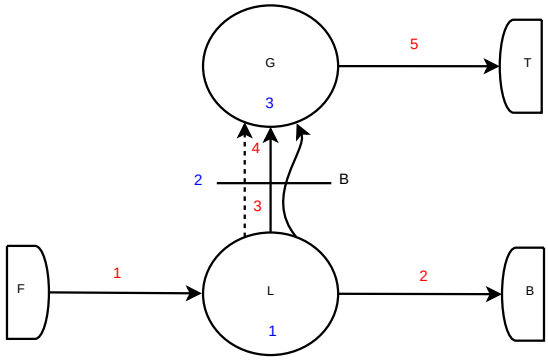

Figure 3.2: Model of a flash tank

3.2 Modelling example

As the modules are still not finalized at the moment of writing this thesis, the best way to describe the modelling procedure is to do it manually and explain where the different steps would take place in the software.

Figure 3.2 is a model of a flash tank, coloured with dynamics, state, phase and morphology. The model also shows a work energy transfer from liquid to gas phase. In terms of the software, this means that the connection is *state=energy*, and that the equation for that particular energy transfer is chosen to be a work term equation.

The program starts by defining primary states (figure 3.1).

Table 3.1: Primary states

symbol	function	description
n		Component mass
U		Internal energy

3.2.1 Directed graph

The graph modelling part is the users interface for making the basic graph and typing the graph with attributes.The interface is shown in figure 3.3. The multiple drop-down boxes in the lower left quartile of the window are used for typing the graph with attribute patterns.

The available attribute patterns generators would have already been created using the pattern generator editor shown in figure 3.4.

The sequence for building the graph with attribute patterns is shown in figure 3.5-3.9.

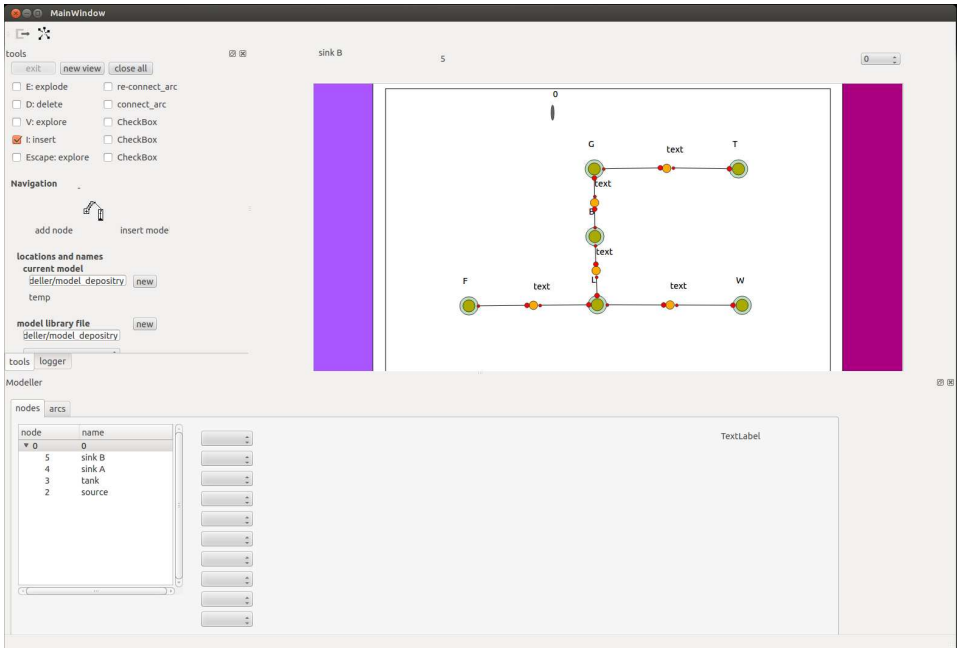

Figure 3.3: Graph modeller.

3.2.2 Species flow matrix

The Khatri-Rao product is the matrix formed by a block-by-block Kronecker product. The Khatri-Rao product has syntax $\star$, while the Kronecker product has syntax $\otimes$.

$$A \star B = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix} \star \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \\ B_{41} & B_{42} \end{bmatrix} = \begin{bmatrix} A_{11} \otimes B_{11} & A_{12} \otimes B_{12} \\ A_{21} \otimes B_{21} & A_{22} \otimes B_{22} \\ A_{31} \otimes B_{31} & A_{32} \otimes B_{32} \\ A_{41} \otimes B_{41} & A_{42} \otimes B_{42} \end{bmatrix} \tag{3.1}$$

In the mass balance vector equation, one needs the stoichiometric matrix, the flow matrix, the flow rate equations and the transposition rate equations. The following example shows how to get the species directionality matrix by using the Khatri-Rao product.

The mass flow matrix F is computed directly from the user-generated graph. There are five mass connections (columns) and three nodes with mass balance equations. The sources and sinks have attribute *dynamics=constant*,

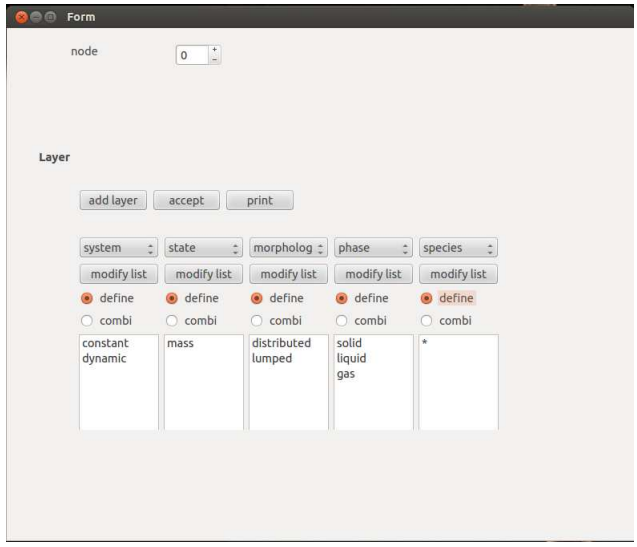

Figure 3.4: Editor for attribute pattern generators.

so they are not included because their mass balance is not consistent:

$$F = \begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

S_s: all species (A, B, C, c)

$$\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

S_n^1: species in node L,

$$\begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}$$

S_m^1: species in arc $L|F$,

$$\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$$

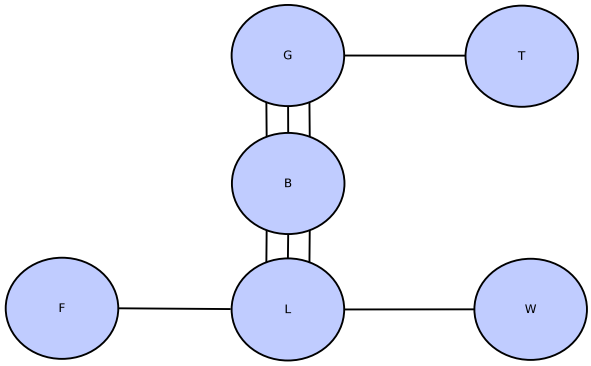

Figure 3.5: Model consists of nodes and connections. This is the basic graph, nodes representing feed, liquid bulk, phase boundary, gas bulk, top sink and waste sink.

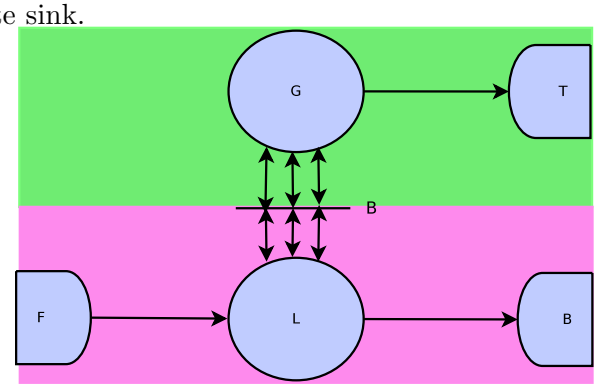

Figure 3.6: Attributes are defined for nodes and arcs. Dynamics, phases, morphology, directionality

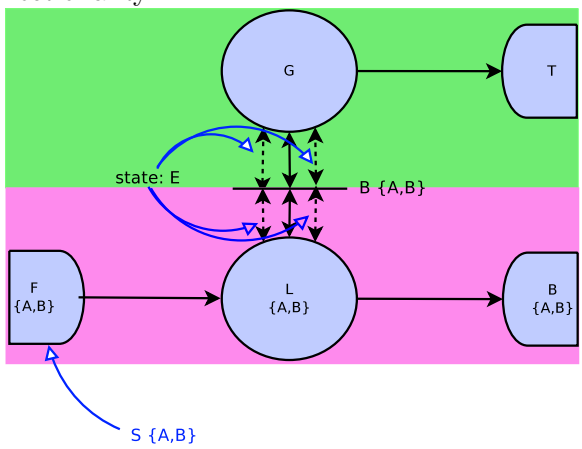

Figure 3.7: Energy token placed in some arcs. Species placed in Feed node, this then propagates the mass token and species to other nodes of the same phase.

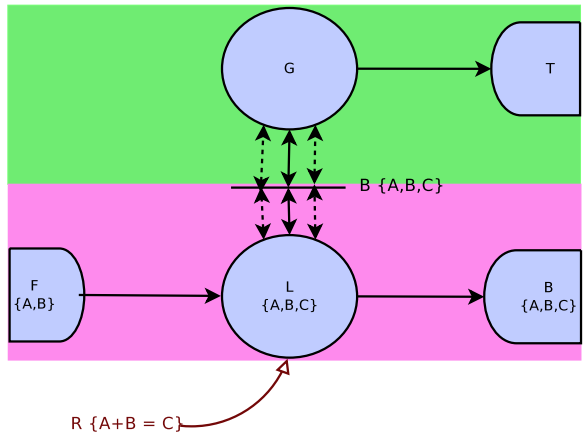

Figure 3.8: Reaction takes place in Liquid node.

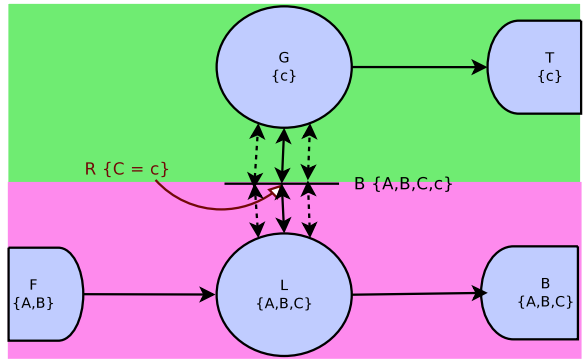

Figure 3.9: Phase transition "reaction" of component C at the phase boundary. C in gas phase, now named c, is propagate throughout the gas phase nodes.

$S_{S_n^1,S_S}$: Diagonalized species in node L,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

S_{S_S,S_m^1}: Diagonalized species in arc L—F,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$S_{1,1} = S_{S_n^1,S_S} \times S'_{S_S,S_m^1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$S_{1,1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, S_{1,2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, S_{1,3} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, S_{1,4} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, S_{1,5} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$S_{2,1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, S_{2,2} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, S_{2,3} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, S_{2,4} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, S_{2,5} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$S_{3,1} = \begin{bmatrix} 0 & 0 \end{bmatrix}, S_{3,2} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, S_{3,3} = \begin{bmatrix} 0 \end{bmatrix}, S_{3,4} = \begin{bmatrix} 1 \end{bmatrix}, S_{3,5} = \begin{bmatrix} 1 \end{bmatrix}$$

All the diagonalized species matrices are then inserted into a block matrix S.

$$S = \left[\begin{array}{cc|ccc|c|c|c} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right]$$

Taking the Khatri-Rao product of S with the flow matrix F:

$$\underline{\underline{S}} \star \underline{\underline{F}} = \underline{\underline{F}}^n \left[\begin{array}{cc|ccc|c|c|c} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \star \begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$\underline{\underline{F}}^n = \left[\begin{array}{cc|ccc|c|c|c} 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{array}\right]$$

The generated species flow matrix are used in the mass balance equations and also in the energy balance equations, since mass flow induces enthalpy change in the system.

3.2.3 Balance equations

The balance equations written out are:

$$\dot{\underline{n}}_L = \hat{\underline{n}}_{F|L} - \hat{\underline{n}}_{L|B} - \hat{\underline{n}}_{L|W} + V_R \underline{\underline{N}}^T \tilde{\underline{\eta}}_L \tag{3.2a}$$

$$\dot{\underline{n}}_B = \hat{\underline{n}}_{L|B} - \hat{\underline{n}}_{B|G} + V_R \underline{\underline{N}}^T \tilde{\underline{\eta}}_B \tag{3.2b}$$

$$\dot{\underline{n}}_G = \hat{\underline{n}}_{B|G} - \hat{\underline{n}}_{G|D} \tag{3.2c}$$

No potential or kinetic forces (system is not moving): E = U + P + K = U

$$\dot{U}_L = \hat{H}_{F|L} - \hat{H}_{L|W} - \hat{H}_{L|B} - \hat{q}_{L|B} - \hat{w}_{L|B} \tag{3.3a}$$

$$\dot{U}_B = \hat{H}_{L|B} + \hat{q}_{L|B} + \hat{w}_{L|B} - \hat{H}_{B|G} - \hat{q}_{B|G} - \hat{w}_{B|G} \tag{3.3b}$$

$$\dot{U}_G = \hat{H}_{B|G} - \hat{q}_{B|G} - \hat{w}_{B|G} - \hat{H}_{G|D} \tag{3.3c}$$

3.2.4 Transport

The transport equations are chosen from the ones available from the equation library. The same goes for secondary state equations and parameters.

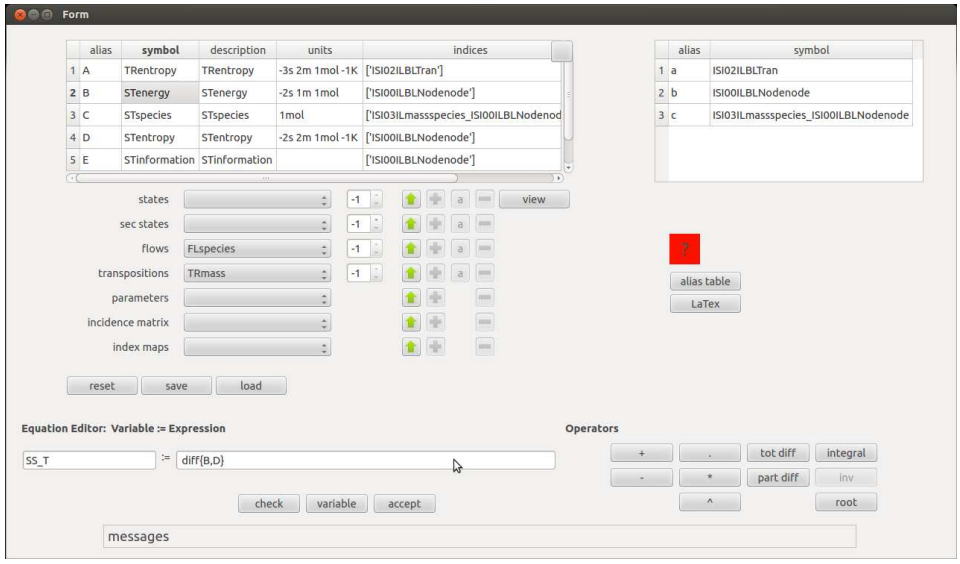

Figure 3.10: Equation editor.

The equation library for the specific ontology is created in the equation editor, shown in figure 3.10. The physical units are checked on both sides of the equation to ensure correctness. Algebraic and differential equation can be made.

Mass transport

For the flash tank example, there is diffusive transport between liquid phase and boundary, and between boundary and gas phase:

$$\hat{\underline{n}}_{L|B} = -D_{L|B}(\underline{\mu}_B - \underline{\mu}_L) \tag{3.4a}$$

$$\hat{\underline{n}}_{B|G} = -D_{B|G}(\underline{\mu}_G - \underline{\mu}_B) \tag{3.4b}$$

Convective transport, to and from the reservoirs (source (F) and sinks (W and D)).

$$\underline{\hat{n}}_{F|L} = \frac{\underline{c}_{F|L} * \hat{V}}{\underline{M}_F} = \frac{\underline{c}_{F|L}}{\underline{M}_F}(-\beta_{F|L} sign(p_L - p_F)\sqrt{|p_L - p_F|}) \tag{3.5a}$$

$$\underline{\hat{n}}_{L|W} = \frac{\underline{c}_{L|W} * \hat{V}}{\underline{M}_L} = \frac{\underline{c}_{L|W}}{\underline{M}_L}(-\beta_{L|W} sign(p_W - p_L)\sqrt{|p_W - p_L|}) \tag{3.5b}$$

$$\underline{\hat{n}}_{G|D} = \frac{\underline{c}_{G|D} * \hat{V}}{\underline{M}_G} = \frac{\underline{c}_{G|D}}{\underline{M}_G}(-\beta_{G|D} sign(p_D - p_G)\sqrt{|p_D - p_G|}) \tag{3.5c}$$

Energy transport

$$\hat{q}_{L|B} = -k_{L|B}(T_B - T_L) \tag{3.6a}$$

$$\hat{q}_{B|G} = -k_{B|G}(T_G - T_B) \tag{3.6b}$$

$$\hat{H}_{F|L} = \underline{h}_F^T \underline{\hat{n}}_{F|L} \tag{3.6c}$$

$$\hat{H}_{F|W} = \underline{h}_F^T \underline{\hat{n}}_{L|W} \tag{3.6d}$$

$$\hat{H}_{L|B} = \underline{h}_L^T \underline{\hat{n}}_{L|B} \tag{3.6e}$$

$$\hat{H}_{B|G} = \underline{h}_B^T \underline{\hat{n}}_{B|G} \tag{3.6f}$$

$$\hat{H}_{G|D} = \underline{h}_G^T \underline{\hat{n}}_{G|D} \tag{3.6g}$$

$$\underline{h} = \underline{h}^0 + \underline{c}_p(T - T_{ref}) \tag{3.6h}$$

Secondary states (table 3.2) will have been chosen with the assistance of the program. They would have been previously been generated in the equation editor.

3.2.5 Transposition

Phase transposition

The boundary has zero capacity in mass and energy:

$$\underline{\hat{n}}_B = 0 \tag{3.7a}$$

$$\hat{U}_B = 0 \tag{3.7b}$$

Flow to and from boundary is fast:

$$C_{C,L} = C_{C,B} \tag{3.8a}$$

$$C_{C,G} = C_{C,B} \tag{3.8b}$$

Table 3.2: Secondary states

symbol	function	description
A	$U - TS$	Helmholtz energy
T	$\frac{\delta U}{\delta S}$	temperature
μ	$\frac{\delta U}{\delta n}$	chemical potential
$\underline{p}_i$	$\frac{nRT}{V}$, bernoulli	partial pressure
p	$\underline{e}^T \underline{p}_i$	total pressure
V	$\frac{\delta U}{\delta p}$, $\frac{n_t}{\rho_i}$	volume
c	$\frac{n_s}{V_s}$	concentration
$\underline{h}$	$n^0 + cp(T - T_r ef)$	molar enthalpy
h	$\frac{V}{A}$	height

Phase transition is fast:

$$C_{C,B} = k * C_{c,B} \tag{3.9}$$

Steady-state reduction in boundary:

$$\dot{n}_{C,B} = 0 \tag{3.10a}$$
$$\dot{n}_{c,B} = 0 \tag{3.10b}$$

Here we need to treat C (liquid C) and c (gaseous C) as the same species.

$$n_B = \hat{n}_{L|B} - \hat{n}_{L|B} = 0 \tag{3.11a}$$
$$- D_{L|B}(\mu_B - \mu_L) - (-D_{B|G}(\mu_G - \mu_B)) = 0 \tag{3.11b}$$
$$\underline{\mu}_B = -\frac{D_{L|B}\underline{\mu}_L - D_{B|G}\underline{\mu}_B}{D_{L|B} + D_{B|G}} \tag{3.11c}$$

$$\underline{\underline{N}} = \begin{bmatrix} -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

A+B→C reaction in L is proportional to concentration A and B, inverse proportional to concentration of C.

$$\tilde{\xi}_L = k_{AB} C_A C_B - k_C C_C = \frac{k_{AB} n_a n_b M_a M_B - k_c n_C M_C}{V_L} \tag{3.12}$$

Phase transposition is equal to to transport of C to the boundary:

$$\tilde{\underline{n}}_{L|B,C} = -D_{L|B}(\underline{\mu}_{B,C} - \underline{\mu}_{L,C}) \tag{3.13}$$

$$\tilde{\underline{n}} = \underline{\underline{N}}^T \xi = \begin{bmatrix} -1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{k_{AB} n_a n_b M_a M_B - k_c n_C M_C}{V_L} \\ -D_{L|B}(\underline{\mu}_B - \underline{\mu}_L) \\ 0 \end{bmatrix}$$

$$\underline{\underline{F}}^n \underline{\hat{n}} = \left[\begin{array}{cc|ccc|c|c|c} 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{array}\right] \begin{bmatrix} -\frac{\varepsilon_{F|L}}{\underline{M}_F}(\beta_{F|L} sign(p_L - p_F)\sqrt{|p_L - p_F|} \\ -\frac{\varepsilon_{L|W}}{\underline{M}_L}(\beta_{L|W} sign(p_W - p_L)\sqrt{|p_W - p_L|} \\ -D_{L|B}(\underline{\mu}_B - \underline{\mu}_L) \\ -D_{B|G}(\underline{\mu}_G - \underline{\mu}_B) \\ -\frac{\varepsilon_{G|D}}{\underline{M}_G}(\beta_{G|D} sign(p_D - p_G)\sqrt{|p_D - p_G|} \end{bmatrix}$$

3.2.6 Non-modelled reactions

The phase transition is assumed to be a pseudo steady-state system. This reaction can be separated from the other.

$$\dot{\underline{n}} = \underline{\underline{\Gamma}}_{\sum} \underline{\underline{A}}_{k} \hat{\underline{n}} + \underline{\underline{\Gamma}}_{\sum} \underline{\underline{S}}_{eq} \underline{\xi}_{eq} + \underline{\underline{\Gamma}}_{\sum} \underline{\underline{S}}_{r} \underline{\xi}_{r} \tag{3.14a}$$

Ω is defined as the left null space of $\underline{\underline{\Gamma}}_{\sum} \underline{\underline{S}}_{eq}$, transposed. Finding the left null space of a matrix is equal to finding the null space of the transposed matrix.

$$\underline{\underline{S}}_{eq}^{T} = \begin{bmatrix} 0 & 0 & -1 & 1 \end{bmatrix} \tag{3.15}$$

$$\underline{\underline{\Gamma}}_{\sum} = \underline{\underline{I}} \tag{3.16}$$

$$\Omega = nullspace(\underline{\underline{\Gamma}}_{\sum} \underline{\underline{S}}_{eq})^{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.17}$$

Define a new primary state variable n^*:

$$n^* = \Omega n \tag{3.18a}$$

$$\dot{\underline{n}}^* = \underline{\underline{\Omega\Gamma}}_{\sum} \underline{\underline{A}}_{k} \underline{\underline{\Gamma}}_{m}^{T} \hat{\underline{n}} + \underline{\underline{\Omega\Gamma}}_{\sum} \underline{\underline{S}}_{r} \underline{\xi_r} \tag{3.18b}$$

$$K_1 = \frac{c_c}{c_C} \tag{3.18c}$$

This type of index reduction can also be done with unmodelled flows. This reduces the primary state space by 2.

3.2.7 Solvability

In computation, a new value for the internal energy is calculated for every time step by $U_{t+\delta t} = U_n + \int_t^{t+\Delta t} U \, \mathrm{d}t$. Using this value for U, one can use a thermodynamic package to determine the temperature and volume of the system. By saying that the calculated U should be equal to $U(T, V, \underline{n})$ from the package, the equation system has one less degree of freedom.

For a phase boundary system with phases L and G, the temperature and volume can be gained by solving the equation set:

$$U(T,V,\underline{n})_L = U_{calc,L} \tag{3.19a}$$
$$U(T,V,\underline{n})_G = U_{calc,G}V_L + V_G = V_{tot}p_L = p_G \tag{3.19b}$$

$$U(T,V,\underline{n})_L - U_{calc,L} = 0 \tag{3.20a}$$
$$U(T,V,\underline{n})_G - U_{calc,G} = 0 \tag{3.20b}$$
$$V_L + V_G - V_{tot} = 0 \tag{3.20c}$$
$$p_L - p_G = 0 \tag{3.20d}$$

The internal energy is a primary state for the equation system. It is however more convenient to get Helmholtz energy then internal energy from a thermodynamic package. This is solved by transforming internal energy to a Helmholtz surface using Legendre transformation:

$$U(S,V,\underline{n}) = A(T,V,\underline{n}) - T(\frac{\delta A}{\delta T})_{V,\underline{n}} \tag{3.21}$$

Pressure is expressed as a Helmholtz derivative:

$$p = (\frac{\delta A}{\delta V})_{T,\underline{n}} \tag{3.22}$$

$(\frac{\delta A}{\delta T})_{V,\underline{n}}$ and $(\frac{\delta A}{\delta V})_{T,\underline{n}}$ are also obtained from the thermodynamics package. Insert (3.22) into (3.20d), and (3.21) into (3.20a) and (3.20b):

$$A(T,V,\underline{n})_L + T(\frac{\delta A}{\delta T})_{V,\underline{n},L} - U_{calc,L} = 0 \tag{3.23a}$$
$$A(T,V,\underline{n})_G + T(\frac{\delta A}{\delta T})_{V,\underline{n},G} - U_{calc,G} = 0 \tag{3.23b}$$
$$V_L + V_G - V_{tot} = 0 \tag{3.23c}$$
$$(\frac{\delta A}{\delta V})_{T,\underline{n},L} - (\frac{\delta A}{\delta V})_{T,\underline{n},G} = 0 \tag{3.23d}$$

This set of four equations can be solved numerically, using for example Newtons method (appendix B), to get the four unknowns T_L, T_G, V_L, V_G. Equation (3.23c) is a constraint, and (3.23d) is a coupling condition.

3.2.8 Equations and variables

The following secondary state equations are defined:

$$A = U - TS \tag{3.24a}$$
$$H = U + pV \tag{3.24b}$$
$$T = root(A(T, V, \underline{n})) \tag{3.24c}$$
$$V = root(A(T, V, \underline{n})) \tag{3.24d}$$
$$S = -(\frac{\delta A}{\delta T})_{V,\underline{n}} \tag{3.24e}$$
$$\mu = (\frac{\delta U}{\delta n})_{T,V} \tag{3.24f}$$
$$p = -(\frac{\delta A}{\delta V})_{T,\underline{n}} \tag{3.24g}$$
$$\underline{p_i} = p\underline{x}$$
$$c = \frac{n}{V}S$$
$$\underline{h}_{B|G} = n^0_{B|G} + c_{p,B}(T_B - T_ref) \tag{3.24h}$$
$$c_p = (\frac{\delta H}{\delta T})_p \tag{3.24i}$$

The variables that needs to be defined for modelling is:

- Temperatures T_L, T_B, T_G. $T_a = \left.\frac{\partial U_a}{\partial S_a}\right|_{V,\underline{n}}$
- Pressures p_F, p_L, p_W, p_G, p_D. $p_a = \left.\frac{\partial U_a}{\partial V_a}\right|_{S,\underline{n}}$
- Chemical potentials $\underline{\mu}_L, \underline{\mu}_B, \underline{\mu}_G$. $\mu_a = \left.\frac{\partial U_a}{\partial \underline{n}_a}\right|_{S,V}$
- Molar masses of all components $M_i, i \in (A, B, C, c)$.
- Volumes V_L, V_G.
- Transport parameters $D_{A|B}, \beta_{A|B}, k_{A|B}$.
- Molar enthalpies $\underline{h}_F, \underline{h}_L, \underline{h}_B, \underline{h}_G$.

Necessary parameters listed in table 3.3.

Table 3.3: Parameters

symbol	description
ρ	density
A	area
g	gravitational field
h^0	enthalpy
β	valve constant
D	diffusion coefficient
k	heat transfer coefficient
M	molar mass
C_p	heat capacity, constant pressure
C_v	heat capacity, constant volume
T_{ref}	reference temperature
h^0	reference enthalpy

Chapter 4

Conclusions and future work

The ongoing development of a computer-aided modelling program was done together with supervisor Prof Heinz A. Preisig. Underlying principles that serve as the basis for the software were discussed at length, at the basic graph level and on the ontology level. Much of the program was redesigned, both in code and interface.
The biggest theme of interest was the attribute story. Attributes are the entities that gives physical meaning to the directed graph. The attributes defined in the current ontology are:

distinction: node | dynamic
system: constant | dynamic | event | -
directionality: unidirectional | bidirectional | -
nature: physical | information
phase: liquid | solid | gas | fluid | - | ?
morphology: distributed | lumped
state: mass | energy
species: *

A new attribute structure was created, called the attribute pattern generator. A pattern generator consists of a string containing a specific order of attribute values. The pattern generator structure is designed so that the systems and connections are ensured only feasible combinations of attributes. This way, the number of higher level rules are kept at a minimum.

The modelling program is still a project under development. The following work remains to be done:

- Finish the *rules* and context editor module.

- Connect modules together.
- Included and test a thermodynamics package together with the equation editor.
- Facilitate creation of distributed systems.

Bibliography

[1] A. Arvidsson, F.; Flycht-Eriksson. Ontologies i. Technical report, Retrieved 26 November 2008.

[2] E. A. Guggenheim. THERMODYNAMICS - An Advanced Treatment for Chemists and Physicists. Elsevier Science Publishers B. V., 1967.

[3] Bjrn Tore Lvfall. Computer Realization of Thermo-dynamic Models Using Algebraic Objects. PhD thesis, Norwegian University of Science and Technology, December 2008.

[4] Steve McConnell. Code Complete: A Practical Handbook of Software Construction, Second Edition. Microsoft Press, 2004.

[5] Heinz A Preisig. Dynamic systems and control technology. Technical report, 1998.

[6] Heinz A. Preisig. Constructing and maintaining proper process models. 2010.

[7] Heinz A Preisig. Lecture notes in chemical process systems engineering. Technical report, 2010.

[8] Heinz A Preisig. An ontology for phys-chem-bio systems 2. Technical report, 2012.

[9] Mathieu R. Westerweele. Five Steps for Building Consistent Dynamic Process Models and their Implementation in the Computer Tool Modeller. PhD thesis, Universiteitsdrukkerij TU Eindhoven, 2003.

Appendix A

Thermodynamic basics

A.1 Degrees of freedom

Gibb's phase rule states that the number of independent variables needed to define a system is:

$$F = C - P + 2, \tag{A.1}$$

where C is the number of components and P is the number of phases. The variable F is called the degrees of freedom (DOF) of the system. For a multi-component system, a system constraint arises, namely that the sum of all component fractions must be 1. This reduces the DOF by one.

A.2 Conjugate pairs

Variables that are needed describe the energy of a system come in pairs, one extensive and one intensive. These pairs are called conjugate variables. S and T, V and p, n and μ, are conjugate variable pairs. You only need one of each to describe the system. The chosen variables dictates which energy function is natural to use. The energy functions have what is called canonical variables, which are the natural variables for describing the system. For example,

$$dA = SdT - pdV - \sum_{i=1}^{n} \mu_i dN_i \tag{A.2}$$

Shows that the canonical variables for A is T,V and n. If they are known, then S, p and μ_i can be found as they are the partial derivatives of A(T,V,n).

A.3 Legendre transformation

A Legendre transform transforms a real-value function of a real value into another. The function variable can be transformed into the function derivative of that variable so that:

$$f_i(\epsilon_i, x_j, x_k, ..., x_n) = f(x_i, x_j, x_k, ..., x_n) - \epsilon_i x_i, \epsilon_i = (\frac{\delta f}{\delta x_i})_{x_j, x_k, ..., x_n} \quad (A.3)$$

The internal energy has canonical function variables S,V and N. We define another function $H(S, pi, \underline{n})$

$$H(S, \frac{\delta U}{\delta V}), N) = U - \frac{\delta U}{\delta V})_{S,N} dV = U + pV \quad (A.4)$$

Where $-\frac{\delta U}{\delta V}$ is defined as p. This energy function is called enthalpy. Enthalpy is the internal energy in addition to the pressure-volume work required to displace its environment and to replace it with its own volume and pressure. It is natural to use enthalpy when describing the energy of incoming or outgoing mass, since the mass must "make room" for itself when it moves in space.

Helmholtz energy, $A(T, V, \underline{n})$, is defined as the Legendre transform of $U(S, V, \underline{n})$ with respect to S:

$$A(T, V, N) = U - \frac{\delta U}{\delta S})_{V,N} dS = U - TS, \frac{\delta U}{\delta T} = T \quad (A.5)$$

Gibbs energy, $G(T, p, \underline{n})$, is defined as the Legendre transform of $U(S, V, \underline{n})$ with respect to S and V:

$$G(T, p, N) = U - (\frac{\delta U}{\delta S})_{p,N} dS - \frac{\delta U}{\delta V})_{T,N} dV = U - TS + pV \quad (A.6)$$

These relationships can all be used to fully describe the energy of a system, since they contains the same amount of information. They have, however, different uses. Helmholtz energy is a function of temperature, volume and component mass. It is useful for cases where the systems volume is constant. Enthalpy is best suited is one wants the energy of a system at constant pressure, as it is a function of entropy, pressure and component mass. [3]

A.4 Maxwell relations

The Maxwell relations arise from the commutative nature of the energy function derivatives. They are derived from second derivatives of the energy functions:

$$\frac{\delta^2 f}{\delta x \delta y} = \frac{\delta^2 f}{\delta y \delta x} \tag{A.7}$$

A.5 Phase equilibrium

Phase equilibrium in the thermodynamic sense means equal mass transfer in both directions of the phase boundary. The most common specification is equality of μ. Phase equilibrium conditions can be found by a minimization of all the energy functions. Minimum found by total differential of function equal to zero Minimization of Helmholtz energy at constant temperature:
$A_{eq} = minA(V, N)_T$
The sum of V and n of both phases is V_0 and n_0

A thermodynamic phase equilibrium is the state in which a thermodynamic system has minimum total energy. There are no driving forces acting on the system, as the transport between the phases are the same in each direction.

U(S,V,N), H(S,p,N), A(T,V,n) and G(T,p,n) is minimized at equilibrium, meaning that the total differential of the energy function is zero. The chemical potential μ is equal in both phases.

Appendix B

Newton's method

Newton's method is a numerical method used to solve real-valued functions of the form f(x)=0. It uses the functions derivative, so it must be continuous in neighbourhood of a zero. It starts with the definition of a simple derivative:

$$f'(x_n) = \frac{f(x_n) - 0}{x_n - x_{n+1}} \tag{B.1}$$

$$x_{n+1} = x_n - \frac{f(x_n}{f'(x_n)} \tag{B.2}$$

and converges if the function is well-behaved. If the the function uses an x_i value in which $f'(x_i)$ is close to zero, the method may fail, as the next point may fall far from the corresponding zero function value.

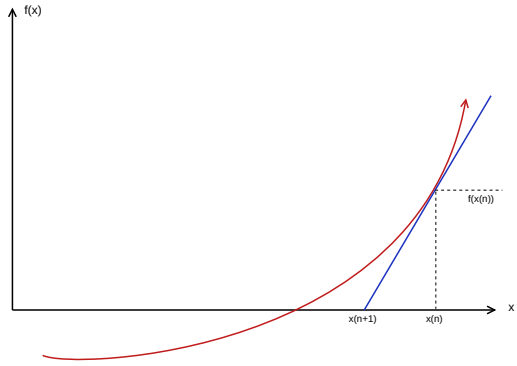

Figure B.1: Newton-Raphson method for one variable

Appendix C

Pydot

C.1 Attribute tree

Graphical representation of an attribute tree, using Pydot:

```
def makeDiagram(self, f):
    self.graph = pydot.Dot(graph_type='graph') #instantiates a Pydot object

    lists = self.tree.data
    root = self.tree.root
    tree = ListTree(lists, root)
    paths = tree.getPaths()
    counter = 0

    idTag = {}
    idTag[counter] = root
```

```
#First path is drawn
for i in range(1, len(paths[0])):
    counter += 1
    idTag[counter] = paths[0][i]
    root1 = pydot.Node(counter-1, label='"'+idTag[counter-1]+'"',style="filled",fillcolor="red")
    var = pydot.Node(counter, label='"'+idTag[counter]+'"',style="filled",fillcolor="green")
    self.graph.add_node(root1)
    self.graph.add_node(var)
    edge = pydot.Edge(root1, var)
    self.graph.add_edge(edge)

for i in range(2, len(paths)): #runs through the list of lists
    first = False #flag. Needed because the first node to be drawn in each list needs to be
    connected to the existing tree. The following nodes in the list just
    connects to each other
    openIfStatement = False
    #flag. Needed to activate the if-sentence after the first requirement is fulfilled
    for j in range (0, len(paths[i])): #runs through the elements in each list of lists
        if paths[i][j] not in paths[i-1] or openIfStatement == True:
        #the first node that is not in a previous list (and the following ones) are connected
            openIfStatement = True
            counter += 1
            idTag[counter] = paths[i][j] #puts a unique tag on the new node
            #print paths[i][j-1], "-->", paths[i][j]
            if first== False: #the first new node is connected to the existing tree structure
                lastUniqueNode = [item[0] for item in idTag.iteritems() if item[1] == paths[i][j-1]]
                root1 = pydot.Node(lastUniqueNode[-1], label='"'+idTag[lastUniqueNode[-1]]+'"',\
                style="filled", fillcolor="red") # changed variable name from root to root1
                first=True
            else:
                root1 = pydot.Node(counter-1, label='"'+idTag[counter-1]+'"', style="filled",\\
                fillcolor="red") # changed variable name from root to root1
```

```
                var=pydot.Node(counter,label='"'+idTag[counter]+'"',style="filled",fillcolor="green")
                self.graph.add_node(root1) #adds node to pydot object
                self.graph.add_node(var) #adds node to pydot object
                edge = pydot.Edge(root1, var)
                self.graph.add_edge(edge) #adds arc to pydot object
    f = f + '.png' #filename
    self.graph.write_png(f) #writes the graph to folder as a png
```

C.2 Bipartite graph

```
def draw(listA , listB , filename="temp.png"):
    digraph = pydot.Dot(graph_type='graph', rankdir='LR', splines="false")#, rotate='90')
    clusterEqs = pydot.Cluster("Equations", label="Equations")
    clusterVars = pydot.Cluster("variables", label="Variables")
    variables = {}
    equations = {}
    for item in listA:
        variables[item] = pydot.Node("var"+str(listA.index(item)), label=str(item), \
        shape = "box", style="filled", fillcolor="green", width="3", penwidth="1")
        clusterVars.add_node(variables[item])
    for key in listB:
        count = 0
        count2 = 0
        if type(listB[key][0]) is list and len(listB[key])>1:
            nice = pydot.Cluster(str(count), label=str(count))
            count += 1
            for i in range(len(listB[key])):
                count2 += 1
                named = "eq"+str(key)+str(count2)
                b = pydot.Node(named, label=str(key), shape = "hexagon", style="filled",\
```

```
            fillcolor="red", width="3")
            nice.add_node(b)
            for j in listB[key][i]:
                digraph.add_edge(pydot.Edge(named, "var"+str(listA.index(j))))

    elif type(listB[key][0]) is str:
        for eqs in listB[key]:
            a = pydot.Node("eq"+str(key), label=str(key), shape = "hexagon", \
            style="filled", fillcolor="red", width="3")
            clusterEqs.add_node(a)
            digraph.add_edge(pydot.Edge("eq"+str(key), "var"+str(listA.index(eqs))))

digraph.add_subgraph(clusterEqs)
digraph.add_subgraph(clusterVars)
clusterEqs.add_subgraph(nice)

digraph.write_raw('examplethings.dot')
digraph.write_png(filename+".png", prog='dot')
```

Appendix D

Installation procedure

Python is a popular scripting language. In addition to the standard library, there is a myriad of external packages available for download throughout the internet. External packages are most easily installed in Linux through a software user interface such as the Advanced Packaging Tool (apt):

```
sudo apt-get install <package>
```

In windows, installation is usually performed by downloading and extracting a zipped file that contains the necessary files. Developers often put a Python file called *setup.py*. Installing is done in the command line:

```
python setup.py install
```

The most popular scientific distribution for Python is Pythonxy. This distribution includes the necessary packages, including PyQt, except for the following external packages that were added and used during the project.

D.1 Pydot

Pydot is a Python package that extends the Graphviz software to the Python environment. Graphviz is a graph visualization software that can be generated by code in several different languages. This enables a program

to automatically draw graphs based on dynamic information within the program. Graphviz for Python requires the module *pyparser.py* and *pydot.py*.

Graphviz can be used in the context of this project as visualization of the attribute tree. This visualization can be showed upon request or placed in an automatically generated pdf-document. Documentation for the modelled system, including graphical information, would be beneficial for the user. Graphviz is free software licensed under the Eclipse Public License.

Graphviz is used for the modeller software in two different ways: visualizing node and arc attribute trees, and constructing a bipartite graph showing the relationship between equations and variables.

The following code generates a tree graph representation of all possible node attributes.

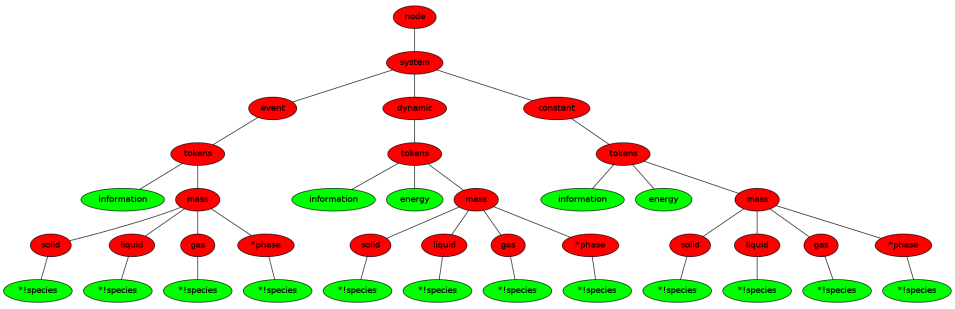

Figure D.1: Attribute tree visualized using Pydot.

Pydot is also used to generate a bipartite graph between equations and the equation variables.

The code for creating these graphics are found in Appendix C

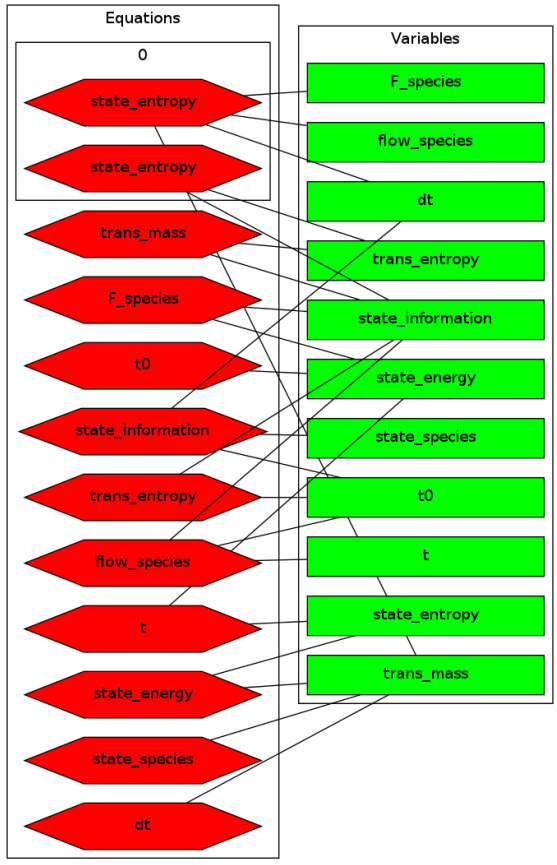

Figure D.2: Bipartite graph showing relationships between equations and variables.

D.2 Py2exe

The py2exe package converts Python scripts to Windows executables, enabling a user to run Python scripts without needing to install a Python interpreter. py2exe scans the main python script recursively for dependencies, and add them all in the same folder as the generated executable.

This package is particularly useful for this software project, as it uses PyQt4, Pyparser, Graphviz and Pydot for graphical GUI and data representation. These packages must be installed, in addition to Python itself, for the program to run. It is more convenient to send a Windows executable so that the modeller program can be run without any prerequisites. The output of the

convertion is a folder containing the executable (.exe), Windows libraries and Python modules (.pyd, .dll), a zip file with all pure source modules, the Python interpreter library (python27.dll).

The script *theory_ wizard_ task.py* was converted as a proof of concept. This was done by creating a setup.py file:

```
from distutils.core import setup
import py2exe
setup(windows=[{"script":"theory_wizard_task.py"}],
          options={"py2exe":{"includes":["sip"]}})
```

Note that sip is required when dealing with programs produced with PyQt4. The folder system is generated with the following command in Window's command line:

```
python setup.py py2exe
```

It was necessary to have a specific library file in the source folder, called msvcp90.dll. This is a dll for Visual Studio 2008. It is recommended to not download this file from an untrusted source. file was obtained by renaming an excisting file in the system32 folder called msvcp90.dll.