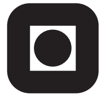


NTNU – Trondheim
Norwegian University of
Science and Technology

Ontology Design for Representation of mathematical Models

Arne Tobias Elve

Chemical Engineering and Biotechnology
Submission date: February 2015
Supervisor: Heinz A. Preisig, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Abstract

Process models are used in many model-based computational engineering activities, including process design, optimization, process control and simulation. The construction of mathematical models is generally regarded as difficult and time consuming, and is therefore often handled by modelling experts. If a tool were available that provided a systematic and safe approach for handling model complexity, consistent and correct models could be created and would eliminate the need for the expert. This project is part of an effort to further advance such tools.

This thesis presents an information structure from which mathematical models can be extracted. The hypothesis is that the information structure will facilitate the development of a structured modelling procedure that can be used in our computer-aided modelling tool. The information structure is in this thesis defined as the model ontology. The ontology is based on a hierarchical decomposition of processes into model objects that can be described individually. The first hierarchical level is a directed graph that consists of nodes and arcs. Nodes represent capacities that contain the fundamental extensive quantities such as mass, energy and information. Arcs represent the transfer of extensive quantities between the nodes. The model objects are increasingly refined for each level in the hierarchical structure based on different characteristics such as phase and time scales. The modelling tool utilising the modelling ontology will then extract building blocks from the hierarchical structure used for generating specific process models.

The rules for generating the structure are implemented in a graphical editor, called the "Ontology Editor". The "Ontology Editor" is designed for constructing consistent ontologies for representation of mathematical models. The results gained from a case study are promising. The ontology allow

for extraction of building blocks and variables that can be implemented in a our modelling tool that provide a systematic and safe approach for creating consistent and correct mathematical models.

Sammendrag

Prosessmodeller brukes i mange modelbaserte ingeniøraktiviteter som for eksempel prosessutforming, optimalisering, prosessregulering og simulering. Det å konstruere en prosessmodell er generelt ansett for å være både vanskelig og tidkrevende, og håndteres derfor ofte av modelleringseksperter. Et modelleringsverktøy som gav struktur til modelleringsprosessen og håndterte kompleksitet i modellen på en oversiktelig måte kunne ha eliminert behovet for modelleringseksperten. Denne avhandlingen er en del av arbeid mot å utvikle et slikt verktøy.

Denne avhandlingen presenterer en strategi for å strukturere informasjon til bruk for representasjon av matematiske modeller. Hypotesen er at denne strategien vil føre til en forenklet modelleringsprosedyre for å modellere prosesser og vil samtidig legge til rette for enklere gjenbruk av modellen. Strategien for å strukturere informasjon er i denne avhandlingen betegnet som en ontologi.

Ontologien er basert på en hierarkisk dekomponering av prosesser til modellobjekter, som kan beskrives individuelt. Det første nivået i hierarkiet er en rettet graf bestående av noder og kanter. Noder representerer kapasiteter som kan inneholde de fundamentale ekstensive mengdene slik som masse, energi og informasjon. Kanter representerer overføringen av disse mengdene mellom nodene. Modellobjektene blir videre sortert for hvert nivå i ontologien basert på ulike karakteristikker av systemer, slik som faser eller tidsskalaer.

Reglene for å generere denne informasjonsstrukturen ble implementert i en grafisk verktøy som er kalt "The Ontology Editor". Verktøyet er utviklet for å hjelpe en modelldesigner i å lage konsistente ontologier for å representere matematiske modeller. Resultatene fra implementeringen av ontologien

er lovende, og muliggjør for definisjon av konsistente prosessmodeller. Ontologien tillater for ekstraksjon av modellobjekter som kan implementeres i modelleringsverktøyet vårt og pålegger en systematisk og trygg metode for å konstruere matematiske modeller.

Preface

This thesis is part of the degree of Master Of Science at Norwegian University of Science and Technology (NTNU). The work presented her was carried out during the period from October 2014 to February 2015 at the Institute of Chemical Engineering. My supervisor has been Professor Heinz A Preisig.

This work was initialized by some ideas on how to improve and evolve Professor Preisig's already existing modelling tool. The modelling tool is designed on the principles of ontologies and is intended to produce executable code. The result is a prototype of what is called the ontology modeller. The "Ontology Editor" , is a prototype of an integrated computer environment for modelling of chemical engineering systems. "Ontology Editor" is implemented in Python and with a user interface designed using PyQt.

This thesis consists of three parts. The first part evolves on the scientific background. The second part describes the developed ontologies and the implementation of them into an ontology generation tool called the "Ontology Editor" . The third and last part evolves on the usage of "Ontology Editor" , results, discussion and suggestions on further work.

The first part is in no way meant to cover the whole area of chemical engineering modelling, rather to illustrate some important aspects about modelling of chemical engineering systems and to present some guidelines for a systematic modelling activity.

References are given with the first author's last name together with the year it was published.

A glossary is included on page xv. This glossary explain the terms used in the constructed ontologies.

Finally, I would like to express my gratitude towards the people that

have helped me during this master thesis. First of all I have to express my sincere gratitude towards to my supervisor, professor Heinz A Preisig. His enthusiasm for this field of study has been an inspiration ever since I started this thesis. This project has been a fun project because of him and his guidance and comments on my work have been of great help. I would also like to thank him for every single "coffee break" and discussions we have had this semester.

My colleagues in the Process Systems Engineering Group also deserves gratitude for feedback and discussions over the year and for making the reading room a fun place to learn. I would especially extend a thank to Ingrid Nyeng, Sigve Karolius and Kjetil Sonerud not only for being wonderful people, but also for proofreading my thesis.

I would also like to thank Christine Blom for great help with the layout of this thesis, and generally for being who you are. Finally I have to thank my family for believing in me and for encouraging me in what I do. Thanks.

Declaration of Compliance

I hereby declare that this thesis is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).
Arne Tobias Elve
Trondheim, February 28, 2015

Contents

Abstract **i**

Sammendrag **iii**

Preface **v**

Contents **vii**

List of Figures **xi**

List of Tables **xiii**

List of definitions **xiv**

List of special terms **xv**

Nomenclature **xxi**

1 Introduction **1**
- 1.1 Motivation and goals 1
- 1.2 Overview of related work 2
- 1.3 Outline of this thesis 3

I Scientific background **5**

2 Models, modelling and ontologies **7**
- 2.1 Ontology 7

2.1.1 Ontologies in computer science 7
2.1.2 What an ontology is 9
2.1.3 Constructing an ontology 9
2.2 Models 11
2.2.1 Mathematical model - behaviour 11
2.3 Modelling 11
2.3.1 Tokens 13
2.3.2 The token mass 14
2.3.3 The token energy 14
2.3.4 The token information 14
2.3.5 Attributes 15
2.4 Physical topology 15
2.4.1 The building blocks 16
2.4.2 Decomposition of a process 19

II The model ontology and implementation into the "Ontology Editor" 23

3 Overview of the implementation 25
3.1 Implementation detail 26

4 Architecture of the ontologies 27
4.1 The basic structure 27
4.1.1 File format 28
4.2 Frame & containment 29
4.3 Token 31
4.3.1 Physical model ontology 31
4.3.2 Information model ontology 32
4.3.3 Interaction model ontology for tokens 33
4.4 Types & rules 33
4.5 Colors & domains 34
4.6 Index structures 35
4.7 The variables and equations 35
4.7.1 Completeness 35
4.7.2 Behaviour - The variable types 36
4.7.3 The variable objects 36
4.8 Language definition 39

5 Implementation of the "Ontology Editor" 41
5.1 Networks 41

5.2 Rules . . . 42
5.2.1 Rules based on the model ontology . . . 43
5.2.2 Units and consistency check . . . 43
5.2.3 Index structure realization . . . 45
5.2.4 Operators . . . 47
5.3 The user interface . . . 47
5.3.1 The parser implementation . . . 47
5.3.2 The equation dialog . . . 53

6 Case study 59
6.1 Study on the dynamic flash tank . . . 59
6.1.1 Description . . . 59
6.1.2 Mathematical system description . . . 60
6.1.3 Balance equations . . . 63
6.1.4 Transport . . . 63
6.1.5 Transposition . . . 64
6.1.6 Secondary states . . . 64
6.2 Dynamic flash implementation . . . 67
6.2.1 The building blocks . . . 67
6.2.2 Frame . . . 67
6.2.3 State equations . . . 68
6.2.4 Incidence matrix . . . 68
6.2.5 Reactions and reaction systems . . . 69
6.2.6 Parameters . . . 70
6.2.7 Secondary state equations . . . 71
6.2.8 Transport equations . . . 72
6.3 Adding control . . . 73
6.3.1 The building blocks . . . 73
6.3.2 The interaction variable space . . . 74

III Discussion and conclusion 77

7 Discussion and unresolved issues 79
7.1 Unresolved issues . . . 79
7.1.1 Construction of dependent equation set . . . 79
7.1.2 Construction of artificial units . . . 80
7.2 Discussion . . . 81
7.2.1 Results of the case study . . . 81
7.2.2 User interface . . . 81
7.2.3 Ontologies . . . 82

7.2.4 Implementations 83
7.3 Further work 85

8 Conclusion 87

Bibliography 89

A Operator definitions code examples 93
A.1 Add 93
A.2 Khatri-Rao product 94
A.3 Matrix product 95
A.4 Exponential expressions 96
A.5 Implicit equations 97
A.6 Unitary functions 98
A.7 Integral 99
A.8 Total differential 100
A.9 Inverse 100
A.10 Partial differential 101

B Physical variable space 103

C Interaction variable space 109

List of Figures

2.1 This figure illustrates the two different types of ontologies used in computer science described by (Marquardt et al. 2010) and (Gruber 1993). 9
2.2 This figure illustrates the grand scheme of modelling presented in (Preisig 2010). The first step is the balance equations, the second step is transport equations and reactions. The last step is the state variable transportations. This version of the grand scheme was printed in (Preisig 2013). 12
2.3 Illustration of a simple basic graph 12
2.4 Illustration on possible separation of nodes based on time 16
2.5 Illustration of a shell and tube heat exchanger 18
2.6 This is a illustration of a possible topology for the heat exchanger illustrated in figure 2.5. 18
2.7 Stepwise decomposition of a distillation column 20

4.1 This figure illustrates the model ontology, and how the universe is broken down in a hierarchical structure. The first layer in the structure is the frame and containment. The next layer separates on the nature of the universe. The universe consist of physical objects and information and interaction between the physical objects and information. Types are combinations of the hierarchical structure and are extracted out of the model ontology as building blocks. 28

5.1 Figure illustrates two network types and the internal interconnection 42

5.2 This figure is a screen shot of user dialogue used for definition of units. In this example the base SI-units are used to express energy which is measured in joule $\left[\frac{kgm^2}{s^2}\right]$ 45
5.3 This is a screen shot of the dialogue used for index selection. . . 47
5.4 The figure illustrates the abstract syntax tree and how it is constructed by the parser for the example string 'A*B+C' 52
5.5 The figure illustrates the abstract syntax tree and how it would be constructed by the parser for the example string 'A*B+C' if the sequence in the parser changed 52
5.6 This figure illustrates the abstract syntax tree constructed by the parser for equation (5.2) 53
5.7 Screen shot of equation dialogue. 54
5.8 This figure exhibit a screen shot of the user interface for the operator buttons. If one of these buttons is activated, a template for the operator appears in the textbox where the equations are written. 55
5.9 This figure exhibit a screen shot of the section where a new variable is defined. 56
5.10 This figure exhibit a screen shot of the dialogue that appears when the change symbol action is triggered. 56

6.1 Illustration of a flash tank and corresponding topology 60
6.2 This is an illustration of the topology of the flash tank with control structure added 73

List of Tables

2.1 Explanations for figure 2.1 8
2.2 Attributes in nodes 16
2.3 Attributes in arcs 16

4.1 Physical variable types explanation 37
4.2 Informational variable types explained 38

5.1 Possible operators for equation definition in "Ontology Editor" . 48
5.2 Toy generator parser language definitions 49
5.3 Toy generator parser language definitions. The sequence of the parser follows the order in which the expressions are explained. . 51

6.1 Primary states and frame definitions 60
6.2 Secondary states 65

List of definitions

4.1 INI file format used for configuration files 29
4.2 Frame & Containment 30
4.3 Physical ontology 32
4.4 Information ontology 33
4.5 Interaction ontology for tokens 33
4.6 Type combinations for physical modelling interaction 34
4.7 Type combinations for information modelling interaction . . . 34
4.8 Type combinations for information modelling interaction . . . 34
4.9 Index structure definitions 35
4.10 Variable space example 38
4.11 Language definition of the language design for representation of mathematical expressions in EBNF 39
5.1 Unit definition 44
5.2 Parser code example 49
7.1 example of variable representation 83
B.1 Physical variable space 103
C.1 Interaction variable space 109

Special terms

0D

Acronym for zero- dimension

1D

Acronym for one- dimension

2D

Acronym for two- dimension

3D

Acronym for three- dimension

Action

Action is used to group what can be done to the different building blocks.

Add

Tokens can be added to a node.

Arc

Arcs represent the transfer of extensive quantities between nodes.

Behaviour

Behaviour provide the link between the mathematical equation and the meaning a variable have in the realization of the model. The behaviour is used to determine where in a model that variable can be extracted.

Bi-directional

Tokens can transfer in both directions over an arc.

Connect

An arc can be connected to a node.

Constant

Constant means that the quantities are constant in the time frame of the model.

Continuous

A continuous signal not discrete.

Control system

Node representation of information system. The control system can only contain information.

Convert

A token can convert. Typical example is a energy that can convert from heat energy to internal energy.

Delete

Token can be deleted from a node.

Distributed 1D

Distributed nodes have distributed intensive properties in one dimension.

Distributed 2D

Distributed nodes have distributed intensive properties in two dimensions.

Distributed 3D

Distributed nodes have distributed intensive properties in three dimensions.

Dynamic

Dynamic means that the changes in the system changes during the lifespan of the model.

Energy

Token used for representing energy, e.g Heat and Work.

Event

A event signal is a signal that is sent only once. For example a light switch.

Event-dynamic

Event -dynamic means that a an event is carried out in such a short time scale that is infinity fast compared to the other processes.

Experimental parameter

An experimental parameter is derived as a fitting variable based on measurements and parameter optimization. This kind of parameters often have experimentally made up units and could potentially change in the future based on new measurements and new parameter optimization. An example of this kind of parameter is a valve constant which have no physical interpretation, but is fitted to the model

Frame

Frame is the collective term for geometrical space, signal space and time.

Gas

Phase description of a physical node of gaseous phase.

Geometrical space

Physical space. Used to described the physical space a model occupies.

Graph

A graph is used for representation of the network of nodes and arcs.

Incidence matrix

A incidence matrix is a matrix for representing the graph and describes which nodes are connected to each other.

Information

Information is a non-physical token and is represented as a signal

Information network

Information network describe signals.

Inject

A token can be injected into a node.

Input

Describes the transfer of signal from a measurement to a control system.

Liquid

Phase description of a physical node of liquid phase.

Lumped

A building block for describing systems of dynamic behaviour with uniform intensive properties within the node.

Manipulator

A manipulator can transfer a signal from a control system to a parameter.

Mass

Mass is a token used for representing mass.

Measurement

A measurement building block can transfer a secondary state in a node over to a signal.

Node

Nodes represent capacities that are able to contain the fundamental extensive quantities such as mass, energy and information.

Numerical parameter

This class of parameters are without units and represent a numerical value. Examples of this kind of parameter is π, 2 and $\frac{3}{4}$. This kind of parameters are typically used in geometric calculations or as exponentials in a power function. The name of the parameter can be noted as the number itself.

Output

Describes the transfer of a signal from a control system to a manipulator.

Parameter

Model characteristic constants. Also represented in parameters are numerical constants to be used for example in context to geometrical shapes.

Physical

Physical network means that the building blocks exist in a physical space.

Re-connect

If one of the connections is broken in an arc, the arc can be re-connected to a node.

Reservoir

Reservoirs is an infinity large source of extensive quantity with constant intensive properties

Sampled

A sampled signal is a discrete signal.

Secondary state

State dependent quantities such as intensive quantities, conjugates of potentials, volume or geometrical quantities .

Signal

Transfer of information between two nodes.

Signal space

A signal space describes the space of a signal.

Solid

Phase description of a physical node of solid phase.

Species

Token for representing chemical species. In close connection to mass.

State

Conserved quantity associated with the respective token, for a physical space this represent either mass or energy.

Structure

Structure is used to denote all the building blocks defined in the model ontology.

Token

A token is minimal state representation that describes the extensive quantity that is held and transported within a network.

Transfer

A token can be transferred between nodes.

Transport

Represent the behaviour of flow of extensive properties between nodes. A time-scale assumption of high conductivity of the respective token has been made. This behaviour can only be related to arcs.

Transposition

Represent kinetic behaviour internally in nodes. Reactions and phase transitions are also included here .

Typing

Typing is used to denote attributes that will be used when defining types of building blocks. It is a specialisation or refinement of the building block. It is used as e keyword in the model ontology.

Uni-directional

Token is transferred strictly from one node to another.

Universal parameter

An universal parameter is constant and have a physical interpretation. This kind of constant have units and will not change in the future. An example of this kind of parameter is the gas constant. The gas constant has an exact value with a fully developed set of units, which will never change

Nomenclature

List of operators:

Operator		Description
$\underline{\underline{\mathbf{A}}}$	::	Bold double underline denotes a matrix
$\hat{A}_{i\|j}$	::	Hat denotes a flow of quantity A from system i to j
$\dot{A}$	::	Dot denotes the time differential
$\tilde{A}_i$	::	Tilde denotes the internal changes of quantity A
$\underline{A}$	::	Underline denotes a vector of quantity A

List of symbols:

Symbol		Description
T	::	Temperature
p	::	Pressure
n	::	Mole
m	::	Mass
c	::	Concentration
c_P	::	Heat capacity constant pressure
μ	::	Chemical potential
E	::	Total energy
U	::	Internal energy
q	::	Heat
w	::	Work
V	::	Volume
t	::	Time
H	::	Enthalpy
k	::	Valve constant mass flow
$D_{i\|j}$	::	The diffusion coefficient from i to j
A	::	Area
x	::	Fraction
l	::	Level
h	::	Specific enthalpy
$\mathbf{N}$	::	Stoichiometric matrix for all systems
$\mathbf{F}$	::	Flow matrix for all species in system

Chapter 1

Introduction

This study has been conducted at the Department of Chemical Engineering at Norwegian University of Science and Technology, as a master thesis for the degree of Master of Science from October 2014 until February 2015. The thesis has been carried out with professor Heinz A Preisig as supervisor.

1.1 Motivation and goals

A model of a process is a system of mathematical equations. The solutions to these equations reflect the behaviour of the process, which is to be modelled (Aris 1978). The effort of modelling a physical-chemical-biological (PCB) system remains high, due to the large variations of chemical process units, The effort required is also increased by modelling of physical phenomena as well as increasing level of complexity that will have to be met by the mathematical models. Furthermore, a family of models, with varying degree of detail, is required in order to support the application of model-based techniques during the whole process life cycle. The purpose of these models is to provide information on the behaviour, which is required for operations such as analysis, control, design, optimization and simulation.

The representation of a PCB process in form of a mathematical model is the key for solving many engineering problems. This has been formulated by many researches, among others (Ogunnaike & Ray 1994), (Cellier 1991) and (Hangos & Cameron 2001). Modelling of chemical processes requires the use of all the basic principles of chemical engineering, including: thermodynamics, reaction kinetics and transport phenomena. The modelling procedure

should therefore be approached with care and thoughtfulness (Stephanopoulos 1984).

The objective of this work is to formulate ontologies to be used in conjunction with a computer aided modelling tool. This modelling tool has been an ongoing project for professor Preisig for a long time. An ontology in it simplest form can be said to be structure of information. The idea is to design ontologies that allow for model extraction and storage and still facilitate for the principles introduced above to be apprehended. The secondary objective is to assist professor Preisig in the implementation of these ontologies into his modelling tool, which goes under the name "Process Modeller".

The focus of this work is on modelling and model formulation, not problem solving. Most of the currently available modelling tools and simulation packages focus on model manipulation, specification, analysis and solution. Many of them even leave out the modelling part. It is generally assumed that the mathematical model of the process under investigation is known or easily accessible. The construction of process models is, however, slow and time consuming. A modelling tool focusing on the modelling part is therefore of great importance and could potentially reduce the time consumption and the amount of errors done in the modelling procedure dramatically.

1.2 Overview of related work

The concept of a modelling tool that focuses on modelling development is not new. Research groups have been working on developing modelling tools and languages for a long time. Stephanopulous and his co-workers presented the **MODEL.LA** environment (Stephanopoulos et al. 1990). MODEL.LA was the first modelling tool implementing a modelling language specific for the domain of chemical engineering. The reimplementation of MODEL.LA following Bieszczad's thesis (Bieszczad 2000) provided a physical-chemical modelling language for representing chemical process models and a modelling logic of construction the underlying model. **ASCEND** is an equation-based environment for solving small to very large mathematical models (Piela et al. 1991). In more recent time the development of modelling tool has been focusing on code generation of symbolic expression. This is a primary focus in the development of **Mobatec** (Westerweele & Laurens 2008), **Modeller** (Westerweele 2003) and **MOSAIC** (Kuntsche et al. 2011). The first modelling tool based on the principles of ontologies was **MODKIT** (Bogusch et al. 2001) and (Yang & Marquardt 2004), which was implemented using **OntoCAPE** (Marquardt et al. 2010).

1.3 Outline of this thesis

The thesis it built up by three parts.

The first part of this thesis evolves on the theoretical principles of model, ontologies and modelling. It presents design principles of ontologies that are used in the construction of the ontologies presented later in this thesis. Moreover, it also elaborates on the principles of modelling and models, the terms and ideas that are essential for this work.

The second part focuses on the constructed ontologies and the implementation of them. The first chapter in this part presents the basic idea of the ontology and provide an introduction to the implementations. The next chapter presents the ontologies developed. The ontologies are presented in detail, both the internal design, and how they build on each other. The next chapter, chapter 5, describe the implementation of the ontologies and the user interface developed to control the model implementation. The last chapter of this part provide a case study, which describes procedure the extraction of a model for a flash tank from the ontologies and the definition of variables and equations.

In the last part, the implementation and the ontologies will be discussed. The final chapter contains the main conclusions, summarizes the main contributions and gives some suggestions for further research.

Part I

Scientific background

Chapter 2

Models, modelling and ontologies

This chapter will present the scientific background related to the construction of ontologies, the modelling procedure and models in general as defined in this thesis.

2.1 Ontology

Originally, Ontology is a philosophical discipline concerning the study of what exists, what the existing things are and their relations. The term "Ontology" originates from Greek and can be translated to "The study of being"[1]. Greek philosophers, such as Parmenides of Elea and Aristotle, introduced the philosophical discipline of ontology during the 4th century BC (Austin 1986).

2.1.1 Ontologies in computer science

In the last decades, computer scientists have borrowed the term ontology, firstly in the field of artificial intelligence (AI). Within computer scientists, the term is used with a more specific context, compared to philosophy, where it denotes an explicit specification of a conceptualization (Gruber 1993). A

[1](Ontos), the genitive of (On), means "of being"; the suffix - (-logica) denotes a science, study, or theory. So originally, the word signifies "theory of being"

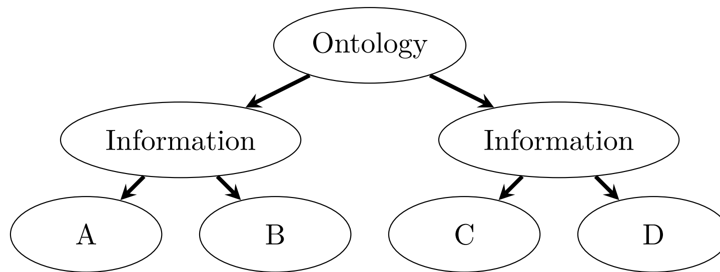
conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

According to (Gruber 1995) there are two usages of ontology in computer science.

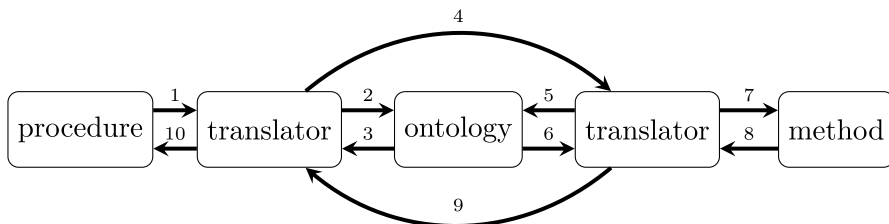
- The first type of usage is when an ontology serves as a library of information to efficiently build intelligent systems. To this aim, the generic ontology is to be transformed (i.e., extended and customized) into a knowledge base according to the requirements of the respective application. This type of ontology is illustrated in figure 2.1 (a).
- The second type of usage is as a shared vocabulary for communication between interacting human and/or software agents. According to their respective functions, the communicating agents may have different knowledge bases, but all the knowledge bases must be consistent with the ontology. This type of ontology is illustrated in figure 2.1 (b).

Table 2.1 – Explanations for figure 2.1

Number	Explanation
1	First software ask for a procedure
2	A translator passes the call for a procedure on to an ontology
3	The ontology returns the procedure as a process
4	The procedure translator sends the process to the translator for the method library
5	Process is sent to the ontology
6	The ontology returns the call for which method
7	The translator asks the method library for method of the process
8	The method is returned
9	The process method is returned to the first program
10	The procedure with all required information is returned



(a) this figure illustrates the first kind of ontologies, namely an ontology used as a knowledge-base. Information groups are stored in a tree.



(b) This figure illustrates the second type of ontologies, when an ontology is used as an integrator between different software tools. The term produce, used by one tool is translated into the term method used by another tool via the ontology. For explanations see table 2.1.

Figure 2.1 – This figure illustrates the two different types of ontologies used in computer science described by (Marquardt et al. 2010) and (Gruber 1993).

2.1.2 What an ontology is

The term ontology is more and more being used in an inflationary manner. It denotes all types of structures used for representation of information and knowledge. It is difficult to put a precise definition to the word. In thesis, ontologies are primarily seen as means to efficiently build and store information in a structure.

2.1.3 Constructing an ontology

There have been written many articles on how to properly construct ontologies and the procedure. In Grubers article from 1993 (Gruber 1993) he described a guideline on how to build ontologies. These design principles were later reused and slightly modified by (Uschold & Gruninger 1996)

and (Marquardt et al. 2010). Their principles are extracted and can be summarized in the following points:

Clarity

An ontology should effectively communicate the intended meaning of defined terms. This means to state exact and unambiguous definitions for all ontological terms in order to effectively communicate the intended semantics. The definitions should be objective and independent of context, meaning that the definitions should not be explicitly dependent on social or computational context.

Coherence

An ontology should be coherent. That is, terms used in the ontology should have one and only one meaning and must be coherent to the real world.

Extendibility

An ontology should be designed to anticipate the uses of the shared vocabulary. It should offer a conceptual foundation for a range of anticipated tasks, and the representation should be crafted so that one can extend and specialize the ontology monotonously. That is one should be able to define new terms for special uses based on the existing terms in a way that does not require a revision of the existing definitions.

Customizability

During the lifetime of an ontology, new applications for the usage is likely to appear, which were not anticipated during ontology development. Since different applications often imply different views on the world (Noy & Klein 2004), the applications will have demands on the ontology and will therefore require different conceptualizations. This means that an ontology must be able to new application requirements.

Minimal encoding and ontological commitments

An ontology should be as small as possible both in terms of size and the information required to use an ontology. For the sake of reuse of an ontology it is important to keep the amount of keywords as small as possible

2.2 Models

The terms model and modelling are used in a wide variety of interpretation and applications. The applications of models span from natural and social sciences, engineering, economics, and arts. Even in process systems engineering the terms are used to denote different concepts and methods.

2.2.1 Mathematical model - behaviour

A mathematical model is described by (Polderman & Willems 1998) as:

A mathematical model is a pair $(\mathbb{U}, \mathfrak{B})$ *with* $\mathbb{U}$ *a set, called the universum-its elements are called outcomes-and* $\mathfrak{B}$ *a subset of* $\mathbb{U}$*, called the behaviour*

In other words, a mathematical model is a system of mathematical equations that represent the nature of biological, physical, and/or chemical process, or even processes that occur in other science or non-science discipline. The equations involves variables that are restricted by equations and relations.

2.3 Modelling

"*Models* are constructed for a purpose, a specific application" (Preisig 2010). When varying the application, the model varies and may result in many different models for the same physical object. A complete model description requires three basic steps. These steps are shown in figure 2.2.

Figure 2.2 illustrates the four main components in a model and indicates the sequence in which they are established. The variables denote

- $\mathbf{N}$ is the stoichiometric matrix for all nodes
- $\mathbf{F}$ is the flow matrix for all species in a graph
- $\hat{x}$ denote the transport between nodes
- $\tilde{\xi}$ denote the internal changes to a system.
- $\dot{x}$ is the state time differential form flows and transpositions
- $\mathbf{x}$ denotes the new time integrated system.

The modelling procedure is initiated with making a directed basic graph that represent the capacities and how they interact. In this thesis a basic graph will be used to visualize the aspects of a model. Each process is

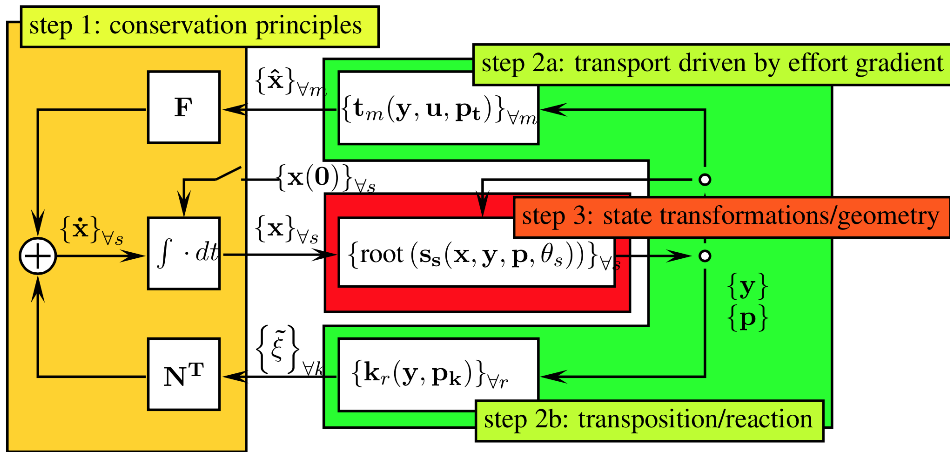


Figure 2.2 – This figure illustrates the grand scheme of modelling presented in (Preisig 2010). The first step is the balance equations, the second step is transport equations and reactions. The last step is the state variable transportations. This version of the grand scheme was printed in (Preisig 2013).

seen as a set of control volumes (nodes) and the communication between the control volumes are denoted as arcs. A node is visualized as a round circle and an arc is visualized as a line between two communicating nodes. A small example of a basic graph with nodes and arcs is provided in figure 2.3. The decisions taken when the control volumes are defined are critical, because the structure chosen is determining the application and contents of the model. What is not considered when structuring the basic graph will not be considered later and not captured by the model.

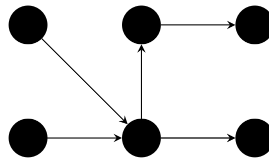


Figure 2.3 – Illustration of a simple basic graph

The basic graph gives rise to the first step, namely the balance equations. For a mass system the balance equations would have the form as described in equation 2.1.

$$\underline{\dot{n}}_s = \mathbf{F}_s^n \underline{\hat{n}}_s + \mathbf{N}_s^T \underline{\tilde{n}}_s \tag{2.1}$$

For this equation the time differential ($\underline{\dot{n}}_s$) is equal to the sum of flows ($\mathbf{F}_s^n \underline{\hat{n}}_s$) and the transposition of species in the system itself ($\mathbf{N}_s^T \underline{\tilde{n}}_s$). $\mathbf{F}_s^n$ is the directionality matrix for the species in the graph, $\mathbf{N}_s^T$ is the stoichiometric matrix.

The next step is determining the flow rates ($\underline{\hat{n}}_s$) and the transposition rates ($\underline{\tilde{n}}_s$). They are dependent on variables from the energy function, such as temperature (T), pressure (p) and chemical potential (μ).

The variables being introduced by the flow rates and the transposition rates are all functions of state (Haug-Warberg 2006). The last step in the procedure of making a complete model description is to provide these links between the primary states in the energy functions and the secondary state variables that are the variables derived from the transfer- and reaction rates. The primary state variables are often referred to in literature as internal state variables. The internal state variables are the smallest possible subset of system variables that can represent the entire state of the system at any given time (Nise 2013). In other words the minimal state space representation required to compute the future given the current input.

2.3.1 Tokens

The basic graph is established on the background on what each node and connection represents in the sense of what a node contains and what it transfers. The model is an abstraction of the model universe as a network and what is moving around is referred to as tokens. A token is minimal state representation that describes the extensive quantity that is held and transported within a unit. The network is the description and illustration of how tokens are being accumulated, transferred and modified. There are four main tokens, namely *Mass*, *Energy*, *Species* and *Information*. Every connection transfer a token, and since a token is being transferred from one node to another, each node also got tokens.

Each token have some dependencies. If a system contains *mass*, there should be a mass balance over the system. Since mass is able to carry both energy and momentum, these tokens could also be present. Mass could carry heat and momentum and could potentially then also receive energy token and momentum token if energy and momentum are to be 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 token gives a differential equation for internal energy.

2.3.2 The token mass

A mass token in a node represents the phase and species in that node. Different species and phases have different abilities and attributes.

In terms of transportation, the token mass could be diffusion. Some of the principle methodologies for mass transportation are (Welty et al. 1976):

- Mass can be transferred due to pressure difference and pressure gradients (pressure diffusion)
- Forced diffusion by external forces
- Mass can be transferred to even temperature distribution in a system. (Thermal diffusion)
- Diffusion can occur due to difference in chemical potential.

2.3.3 The token energy

Energy token in a node represents internal energy. In transport processes the token energy can represent convection, conduction, radiation, work or moving electrons when transporting electricity. Transportation of internal energy is mass transport, and is therefore not mentioned in this section. When transporting energy from one node to another the energy will have to undergo two transformations. First, the token energy would be represented as internal energy, then it would have another form in the connection and later go back to internal energy again in the other node. In order to set up the energy balances for each node one need to include all heat streams, work streams and also mass streams. Mass carries internal energy. Movement of mass adds kinetic energy. Potential energy may also be considered if the plant is exposed to a gravitational field. Movement of mass across boundaries also adds volumetric work to the balance. Since mass induces energy, it is possible to have mass balances in isolation, but it is not possible to have isolated energy balances in a system that exchange mass. The energy balance of a system can be described by equation (2.2) (Preisig 2013).

$$\frac{dE}{dt} = \mathbf{F}^m \hat{E}_m + \mathbf{F}^q \hat{q}_q + \mathbf{F}^w \hat{w}_w \tag{2.2}$$

2.3.4 The token information

There are many definitions of signals. A popular definition is "A signal is a function that conveys information about the behaviour or attributes of some

phenomenon" (Priemer 1991). Signals can occur naturally or they can be manually injected. Generally the signal is handled as a flow of information, for example the output of a pH-meter or the temperature in a column. These samples are electrical and are typical for what it the regular way to think of signals. A perfect modelling tool also needs to be able to handle non-electrical signals like an acoustic wave or an electromagnetic wave, a mass pulse throughout a pipe. These signals are also information.

When handling signals in terms of tokens it is important to be aware of that token information is not physical. What distinguishes information from the other tokens is that it is not a conserved quantity, meaning that its not necessary to do any calculations to check the validity of the signal and there are no balance equations for them. The signals can be handled at a different layer. It is not depended on steady state calculations simply just on the event-dynamic part.

2.3.5 Attributes

Nodes and arcs are elements in a directed graph. A directed graph itself does not give any meaning except for giving a network structure. In order to use the network structure it is important to assign attributes to nodes and arcs. The attributes give context to the nodes and arcs. The Oxford dictionary (Hawkins 1986) define attributes as "quality ascribed to or characteristic of a person or a thing".

When setting up the structure that holds the attributes there are two main goals. First of all, it needs to give sufficient information about the nodes and arcs in the model. Secondly, the structure of attribute combinations should be reasonable to what combinations it should reflect. For example, an arc is not constant, dynamic or event-dynamic. It does not need to specify that attribute. A node is not able to transport anything and does not need to structure attributes for transport. In order to solve the second goal it is possible split the attributes and specify them separately for nodes and arcs. The node attributes are displayed in table 2.2 and the connection attributes are displayed in table 2.3.

2.4 Physical topology

The first step in a modelling procedure is to break down the process who is to be modelled into a set of subprocesses. These subprocesses may again be broken down into subprocesses, and so on. In the end, a process consists of may subprocesses where each are small enough to be handled individually,

Table 2.2 – Attributes in nodes

Set	**Attributes**
Nature	Physical , Information
Dynamics	Constant , Dynamic , Event
Morphology	Distributed , Lumped
Token	Mass , Energy, Species , Information
Phase	Gas , Liquid , Solid
Dimensionality	0-D, 1-D, 2-D, 3-D

Table 2.3 – Attributes in arcs

Set	**Attributes**
Nature	Physical , Information
Token	Mass , Energy , Information
Directionality	Uni-directional , Bi-directional
Mechanisms	Heat , Work , Electric
Phase	Gas, liquid, Solid

meaning that the process is divided into a network of connected volume elements. The network of these subprocesses describes the physical structure of the process and shall in this report be referred to as the physical topology.

2.4.1 The building blocks

The two basic building blocks in a physical topology are, as mentioned, nodes and arcs. The nodes are able to contain token, while the arcs describe the transportation of token between two nodes. By introducing different time scales, and sizes to the node, it is possible divide the nodes into three different dynamic behaviours, namely constant, dynamic and event-dynamic. In figure 2.4 the node classification based on time-scale is illustrated.

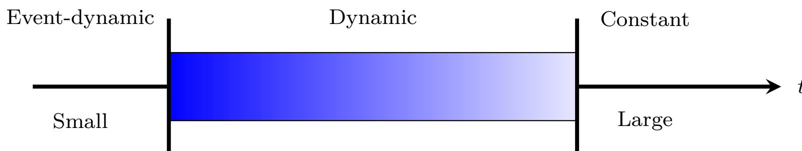


Figure 2.4 – Illustration on possible separation of nodes based on time

The nodes that represent systems with constant dynamic behaviour will from here on be referred to as reservoirs. A reservoir is defined as a node that has constant state, in other words, it has no dynamics. (Preisig 2013) define the reservoir as: "Reservoirs is an infinity large source of extensive quantity with constant intensive properties". Nodes that are event-dynamic have no capacity, so there is no accumulation of token in such a node. This reduces the differential algebraic equation (DAE) into an algebraic equation because the differences in primary states are constant in and out of the node. Presig define the event-dynamic systems as "surfaces with no capacity, thus exhibiting event dynamics" (Preisig 2013). Nodes that are representing systems of dynamic behaviour can be divided into two groups depending on the distribution of intensive properties within the system. A dynamic lumped system is a system where uniform intensive properties within the boundaries of the node, are assumed. A dynamic distributed system does not enforce uniform intensive properties within the system.

Arcs represent the interactions between two nodes. An arc describes the exchange of token from one system to another systems across a boundary separating the two systems. Therefore, an arc cannot exist without being connected to nodes in both ends. This definition also imposes that the arcs are directed. The actual flow of token across a boundary is caused by a difference in states for the two systems. For example the driving force in heat flow is temperature difference. In some cases the transfer of token happens in some other medium. This medium might also have capacities. To capture these capacities transport systems are introduced. The transport system can for example be used to model heat flow through a solid mass wall.

Once the building blocks are established, it is possible to start building topologies.

Example: Physical topology of heat exchanger

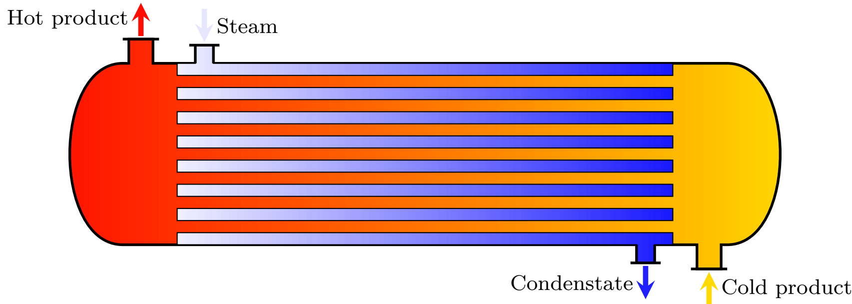


Figure 2.5 – Illustration of a shell and tube heat exchanger

Figure 2.5 illustrates a general shell and tube heat exchanger with high pressure steam which condensate as cooling medium. This heat exchanger could be a part of a larger plant. The heat exchange could be separated into three parts, namely the condensation of steam, the heat transfer in the wall and the heating of the product. At this stage only the topology is of concern, the actual contents of the heat exchanger and the volumes are not to be considered at this first stage of model construction. This topology will form the basis of the rest of the modelling process. A possible topology for this heat exchanger is illustrated in figure 2.6.

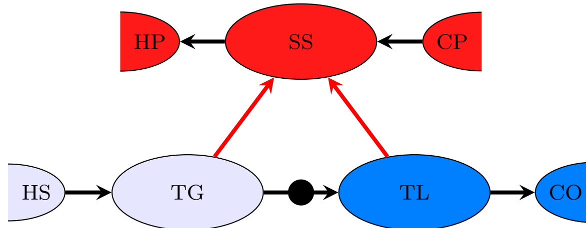


Figure 2.6 – This is a illustration of a possible topology for the heat exchanger illustrated in figure 2.5.

The half circles represents reservoirs, the ellipses represents distributed systems, and the phase-transition of the steam is represented with a black circle which is the boundary with no capacities. The steam is injected to the reservoir at bottom left corner of the topology. This steam is then transferred to the distributed system in the tube side in gas phase (TG). Heat (red arrow) is transferred from the gas phase shell side of the heat exchanger (SS). The vapour, condensates as the heat is transferred still at

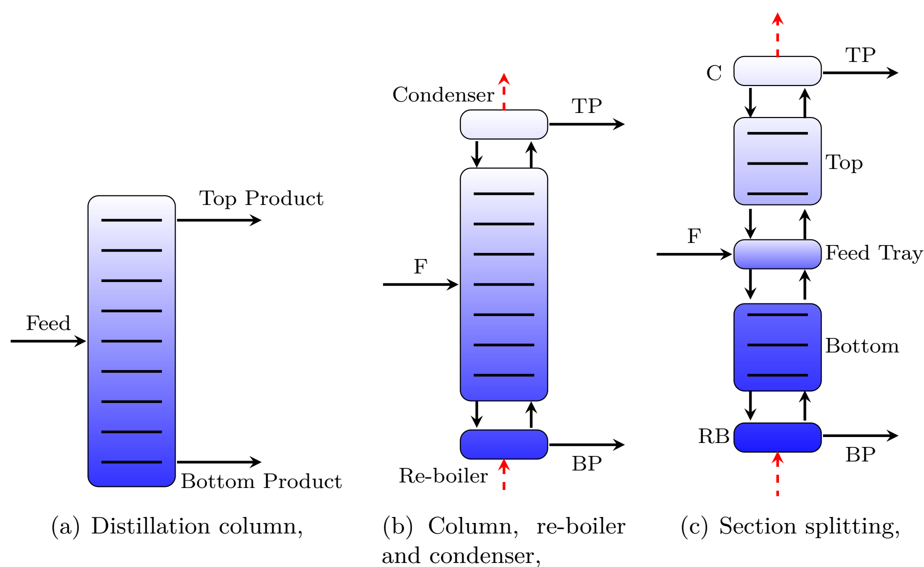
the tube side but in liquid phase (TL). This liquid has a high temperature and can transfer heat to the shell side, and some mass leaves the tube side to the condensate drain (CO). The cold product (CP) enters the shell side at the right top corner in the topology and is then transferred to the distributed shell (SS) side. Here it is heated and the heated product (HP) leaves the shell side.

2.4.2 Decomposition of a process

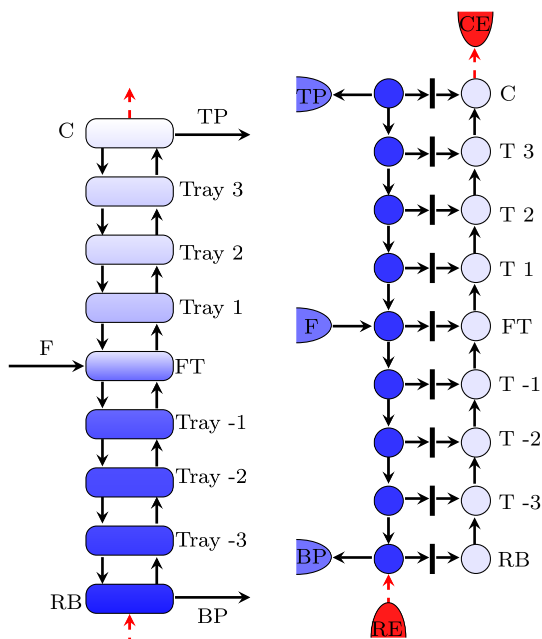
As defined earlier a physical topology is an abstract representation of a physical process based on the physical connections of the process. This process is initiated by breaking the process up into subprocesses. The decisions made during the process of defining subprocesses are largely based on the phase. A node can only consist of one phase in order to exhibit uniform and distributed properties. This was also illustrated in figure 2.6 where a phase transition occurred, the two phases were represented by two separate nodes. A second decision that has to be considered is the size of the system the node should represent. The systems that the nodes is going to represent must be small enough to have uniform quantities. The third decision is connection based and on how one defines the systems.

Example: Decomposition of a distillation process

In figure 2.7 a comic for a possible decomposition of a distillation column is illustrated. A distillation column is a multi-staged column where chemical components are separated due to differences in vapour pressure. A feed stream comes in and two product streams come out. This splitting of feed stream is illustrated in figure 2.7(a). Most distillation columns consists of a column and a reboiler to supply the necessary energy to vaporize, and a condenser to remove the energy at the top so the vapour can condensate. This is illustrated in figure 2.7(b) were the reboiler and the condenser now two separate nodes from the rest of the column. If there is a mid placed feed entry, the middle section would behave a bit different from the rest of the column. To capture this case with feed stream it could be a good idea to separate this into one node (figure 2.7(c)). Each tray in the column would have different abilities and properties from each other. If the objective of the modelling is to capture an effect at a certain tray, each tray should be decomposed. The decomposition of each tray is illustrated in figure 2.7(d). If one is to model the amount of fluid in tray 2, one needs to capture the behaviour of fluid in tray 2. It is then possible to split each tray into two



(a) Distillation column, (b) Column, re-boiler and condenser, (c) Section splitting,



(d) Tray decomposition, (e) Phase decompostion,

Figure 2.7 – Stepwise decomposition of a distillation column

phases with a boundary between them. A suggested splitting of each tray is illustrated in figure 2.7(e).

Entire figure 2.7 shows a possible way to decompose the distillation process. This finally leads out to a flat topology of the process (figure 2.7(e)), which consists of simple nodes and arcs. The following assumptions have been made during the construction of the physical topology as represented.

For the construction of the physical topology as represented the following assumptions have been made.

- Each tray is assumed to consist of two homogeneous phases, liquid and vapour. Between the two phases there is a boundary that separates the two homogeneous systems.
- The condenser and the reboiler can be decomposed into two homogeneous phases with a boundary between them.
- Heat loss to surrounding environment is assumend to be negligible
- The momentum balances are neglected

Part II

The model ontology and implementation into the "Ontology Editor"

Chapter 3

Overview of the implementation

Having established the scientific background of ontology engineering and modelling it is now time to present the ontology-based modelling tool that goes under the name "Process Modeller".

The "Process Modeller"is a complete modelling tool that professor Preisig has been working on for many years. Integrated in "Process Modeller" is a framework for constructing ontologies. This framework is called the "Ontology Editor" and is the focus of this thesis. "Ontology Editor" consists of several independent sub-ontologies, which will be presented in the chapter 4, and a user interface that allow for construction of ontologies for variable representation to be used in mathematical models. The constructed ontologies resulting from the "Ontology Editor" is denoted as variable spaces. The ontology describing the break-down of the universe into small building blocks will be refereed to as the model ontology.

This parts starts with describing the architecture of the model ontology from which the models are extracted. The model ontologies are based on a top down procedure first describing the basic building block and then start refining the building blocks and from the refinement extract types of building blocks. This chapter also describe the architecture of a variable space and include the presentation of a modelling language, which is designed for construction variable spaces with equations.

The next chapter present how the all ontologies are included in the "Ontology Editor" and present the rules that are included to guide the user to

construct consistent variable spaces. This involves presentation and explanation of the user interface, how it functions and what it is able to do. The implementation of the modelling language is also described in this chapter. The operators and how units and index structures are inherited over operations is also included in the modelling language implementation.

Finally, the implementation and construction of variable space, which form the basis of a mathematical model, is presented by a case study. This case study is a modelling example of a dynamic flash tank. This involves both implementation of the physical model to describe the physics and chemistry behind and the implementation of a signal space to control the physical model. This chapter describe how the "Ontology Editor" can be used for constructing a variable space and how building blocks can be extracted from model ontologies. The example is only intended as a prototype on how a variable space could be defined. The solving of the model and the production of executable computer code is at the present state not included.

3.1 Implementation detail

This part describes how the implementation works and not how it was implemented. Although the actual implementation required a lot of work and was far from trivial, the implementation details was not considered to be an important contribution to this thesis. Some of the most important details are worth spending some time on, especially the basic language and software used for the implementation, since these form the basis for the entire program.

The implementation of the modelling tool is done in Python[1]. Professor Preisig selected this programming language, since it provides the foundation for creativity during programming and is a flexible language. Since the main challenge with the modelling tool is organisation, and not speed of the calculations, it was considered to be a good language for implementation.

The construction of the graphical user interface was done using PyQt. PyQt is a Python binding of the cross-platform GUI toolkit Qt. PyQt contains a library of construction of GUI and provides translation of a GUI design done using Qt-designer into python code.

[1] Web page of the Python software foundation: `https://www.python.org/`

Chapter 4

Architecture of the ontologies

The principles on which one could design an ontology was first introduced by Gruber (Gruber 1995), as presented in chapter 2 and used as inspiration for these ontologies. Although the idea of ontologies and design of ontologies was presented with Gruber, the idea of making an ontology of ontologies has nothing to do with Gruber's definition. The objective is to construct an ontology from which physical-chemical-biological models can be extracted.

4.1 The basic structure

The ontologies presented in this thesis are dependent on each other in a layered structure, which forms a lower-trigonal structure [1]. The first ontology defines the basic set of definitions. The following ontologies are defined by systematically adding information to the first definitions by branching and specializing and then finally, recursively defining the overall ontology. This ontology construction approach is highly structural and forms a minimal representation, but differs from the attempts of ontology construction previously presented by other research groups. An illustration of the entire ontology is shown in figure 4.1.

As explained in chapter 3, the ontologies have inherited elements and also share a syntax. As a consequence, the definitions are expanded to

[1] Lower -trigonal structure means that an element in the structure build on previously defined elements. The elements are inherited from one defined element to the next

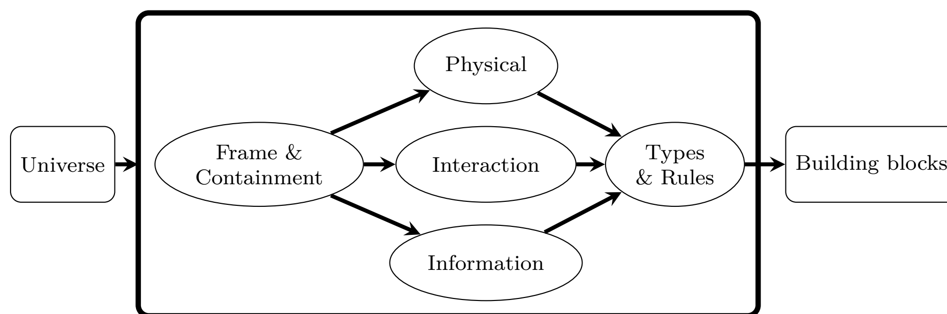


Figure 4.1 – This figure illustrates the model ontology, and how the universe is broken down in a hierarchical structure. The first layer in the structure is the frame and containment. The next layer separates on the nature of the universe. The universe consist of physical objects and information and interaction between the physical objects and information. Types are combinations of the hierarchical structure and are extracted out of the model ontology as building blocks.

capture the necessary attributes connected to a behaviour. This ensures that the ontologies are kept to a minimum, but still compatible with each other.

There are four key terms within each model ontology.

Definitions in *structure* are building blocks. A second term is *behaviour*. The behaviour term reflect to the behaviour of the variables and the signals sent between the variable space and the ontologies. *Actions* are descriptions on what is possible to do with the defined objects and finally, *typing* is used for specialisation of building blocks.

4.1.1 File format

The implementation of the all ontologies uses one of the simplest file formats, namely the INI file format for configuration files. The language itself is called the INI-language. In the ontology definition, it is used for specifying configurations of tasks. Meaning that it fits the purpose of the ontologies as a base structure. The language itself is build up by three objects. The first object is the section, which is a keyword enclosed by rectangular brackets.

```
[<section>]
```

The sections are used to group properties that consist of a property name and property value

```
<property name> = <property value>
```

An example on an ontology file is provided in definition 4.1.

Definition 4.1 – INI file format used for configuration files

```
[<section_1>]
    <property name_1.2> = <property value_1.2>
    <property name_1.2> = <property value_1.2>
    <property name_1.3> = <property value_1.3>

; Comments are initiated with a colon ;

[<section_2>]
    <property name_2.2> = <property value_2.2>
    <property name_2.2> = <property value_2.2>
    <property name_2.3> = <property value_2.3>
```

This file format is essentially representing a dictionary [2]. All the ontologies are represented using this file format. The ontologies are initialized by defining the frame, the containment and the tokens. Next a special purpose editor imposes rules for the construction of variables end equations.

4.2 Frame & containment

The frame introduces a concept of a coordinate system, which serves as a frame of reference for the observation of system properties. The frame is the anchoring point to the real world and is used for definition in the model.

Information is processed by control systems. The information is gained from measurements in the process and can then be translated by the control system. Since the processing of information is independent of the geometrical space, information is considered to be dependent on only time. Mass and energy are, on the other hand, conserved quantities, which not necessary have uniform quantities in space and that can convert over time. This imposes that physical properties is dependent on both time and geometrical space. The frame can then be considered to be context dependent in a geometrical space, while time is shared by both physical and information networks.

The containment is where systems are represented. The containment must be generic since it represents potentially large systems that contain tokens, and where the tokens communicate by physical connections or signals. The containment in the ontologies is represented using a directed graph, which is denoted in the ontologies as `graph`. The connection between the containment and a mathematical representation is denoted as the behaviour. The behaviour provides the link to the variables and equations

[2] For example: Python uses an ordered dictionary when loading and saving these file formats

used as the mathematical representation captured in the model ontology. The first layer in the model ontology to be defined is the related to the containment:

```
[structure] ; building blocks
   graph = ['node', 'arc']
   frame = ['time']
```

The mathematical representation of the graph is in the form of an incidence matrix:

```
[behaviour] ; link to mathematical description
   graph = ['incidence_matrix']
```

The building blocks are specialised more by adding typing. The typing imposes specialising on the different objects of the building blocks. This specialisation is closely related to the attributes described in the scientific background. The graph consist of both nodes and arcs. The specialization to the graph applies to both nodes and arcs while specialization to node or arc only applies to that object. The specific typing to frame and containment is:

```
[typing] ; specialisation of building block
   graph = [ 'physical', 'information']
   node = ['event','dynamic', 'constant']
   arc = ['uni-directional', 'bi-directional']
```

Possible actions are dependent on the building block and is therefore separated into node actions and actions possible for arcs:

```
[action] ; enabled actions for building blocks
   node = ['add', 'delete']
   arc = [ 'connect', 're-connect']
```

The first layer in the model ontology for representation of frame and containment is defined in definition 4.2.

Definition 4.2 – Frame & Containment

```
[structure] ; building blocks
   graph = ['node', 'arc']
   frame = ['time']

[behaviour] ; link to mathematical description
   graph = ['incidence_matrix']

[typing] ; specialisation of building block
   graph = [ 'physical', 'information']
   node = ['event','dynamic', 'constant']
   arc = ['uni-directional', 'bi-directional']

[action] ; enabled actions for building blocks
   node = ['add', 'delete']
   arc = [ 'connect', 're-connect']
```

4.3 Token

The next layer in the model ontology is context dependent, with the context defined and inherited from the first layer described in definition 4.2. The second layer in the model ontology for tokens is branched into three separate parts. One part for the physical representation, one part for representation of information and the last part for describing the interactions between information and physical properties.

4.3.1 Physical model ontology

In the physical ontology, the extensive quantities are defined as the tokens, and the frame on definition 4.2 is extended by the ability to represent properties in a geometrical space. This geometrical space can represent different dimensions in a coordinate system. This coordinate system may vary according to the objective of the definition. The structure definition is therefore adding token as an attribute and extend the frame definition:

```
[structure]
    token = ["mass","energy","species"]
    frame = ["geometrical_space"]
```

The behaviour for a physical token is extended by defining specific sets of signals applicable to define the mathematical representation of physical systems. Some of the behaviours are specific to nodes and some a specific to arcs and parameters can exist for both nodes and arcs:

```
[behaviour]
    node = ['state', 'secondary_state', 'transposition', 'parameter']
    arc = ['transport', 'parameter']
```

The structures used for representation of the physical tokens are separated into different phases, and the frame can have different dimensions for representing a coordinate system:

```
[typing]
    graph = ['liquid', 'solid','gas']
    geometrical_space = ['0D', '1D', '2D','3D']
```

Implementation of these tokens require actions that can be taken for the different objects. For the physical ontology the actions defined are:

```
[actions]
    token := [inject, add, delete, convert]
```

The complete physical ontology is described by definition 4.3.

Definition 4.3 – Physical ontology

```
[structure]
    token = ["mass","energy","species"]
    frame = ["geometrical_space"]

[behaviour]
    node = ['state', 'secondary_state', 'transposition', 'parameter']
    arc = ['transport', 'parameter']

[typing]
    graph = ['liquid', 'solid','gas']
    geometrical_space = ['0D', '1D', '2D','3D']

[action]
    token = ['inject', 'transfer', 'convert', 'delete']
```

4.3.2 Information model ontology

The ontology part designed for information is similar to the physical ontology, but has a simpler form since there is fewer alternatives in token definition. For an information system, only one token is added to the structure and signals communicate in a signal space. The structure definition then becomes:

```
[structure]
    token = ["information"]
    frame = ['signal_space']
```

The mathematical representation of information extends the possible behaviours applicable to information. Also in information there are specific attributes related to nodes and to arcs:

```
[behaviour]
    node = ['state']
    arc = ['input', 'output']
```

The typing of information relates to the type of the signal:

```
[typing]
    graph = ['sampled', 'continuous','event']
```

In order to use the information objects later some actions need to be available. For information token these actions are:

```
[action]
    token = ['add', 'delete']
```

The entire information token ontology is written out in form described in definition 4.4.

Definition 4.4 – Information ontology

```
[structure]
   token = ["information"]
   frame = ['signal_space']

[behaviour]
   node = ['state']
   arc = ['input', 'output']

[typing]
   graph = ['sampled', 'continuous','event']

[action]
   token = ['add', 'delete']
```

4.3.3 Interaction model ontology for tokens

In order to have communication between the two networks of tokens, an interaction ontology must be defined for allowing parings over the different networks. This ontology is essentially the union of the two ontologies defined in definition 4.4 and in definition 4.3, since it must represent both information and physical attributes. This ontology is used for measurements of physical properties in nodes and the manipulation of the physical flows. In connection to the behaviour term, physical properties are mathematically represented using secondary states, and manipulation of flows are represented with parameters. Therefore only the most essential parts of the two ontologies were extracted from the information ontology and the physical ontology to form the interaction ontology, which is included in definition 4.5.

Definition 4.5 – Interaction ontology for tokens

```
[structure]
   token := ["information", "mass", "species", "energy"]
   frame := ['signal space', 'geometrical space']
[behaviour]
   node := ['parameter', 'secondary state']
   arc := ['input', 'output']
```

Since this ontology only is used for interaction between physical token and information token, all the types and actions are defined in the other ontologies.

4.4 Types & rules

The next layer introduces types, which are used to control the definitions and the combinations of the definitions. Types of nodes and types of arcs are introduced. These types are combinations of the parts already defined in

the model ontology that are used in a modelling environment. The types are context dependent, and will therefore add an additional layer to the model ontology. The includes types for a physical system are given in definition 4.6.

Definition 4.6 – Type combinations for physical modelling interaction

```
[node]
    lumped = [dynamic] + [0D]
    distributed_1 = [dynamic] + [1D]
    distributed_2 = [dynamic] + [2D]
    distributed_3 = [dynamic] + [3D]
    reservoir = [constant]
    boundary = [event]

[arc]
    energy_transfer = [energy]
    mass_transfer = [mass]
    species_transfer = [species]
```

For the information systems, the type definitions are listed in definition 4.7.

Definition 4.7 – Type combinations for information modelling interaction

```
[node]
    control system = [event] + [information state]
[arc]
    signal = [input, output] + [uni-directional]
```

For interactions between physical token and information the following types are defined in definition 4.8.

Definition 4.8 – Type combinations for information modelling interaction

```
[node]
    measurement = [event] + [secondary state, input]
    manipulator = [event] + [parameter, output]
```

4.5 Colors & domains

In order to enable the specification of the context dependences in more detail, attributes are introduced to several objects, in particular to graph, node, arc and token. The term colour have been chosen is this context since the attributes add descriptive properties to the objects which can be related to colours. The attributes provide the specification of the objects and the attributes provide the context in which variables and equations are defined in the network. The colour distribution in the containment is calculated by a set of rules, such as tokens persist, which implies that defining an arc that transfer tokens implies that the same token must exist in both connected nodes. In the ontology the colours are added as attributes to the graph, tokens and the tokens attributes:

```
colours = ["physical","information", "mass","energy", "species", "
    liquid","solid","gas"]
```

The intention is to implement the colours into the graphical user interface. The colours can there be used to visualize where in the graph the coloured attributes are present or communicated.

4.6 Index structures

The index structures are used to determine the size and shape of the objects in the variable space. An index structure only makes sense in connection to the other objects. For example: The structure of a model could be given by the number of chemical species defined for each node and arc and must therefore be reflected in the dimension of the objects. The index structures associate the equations to the objects, nodes and/or arcs, and will therefore contain a running index. The index structure could also be indexed with connected with a token. This token determine the dimensionality of the running index. The incidence matrix is defined by nodes and arcs so the index combination 'nodes&arcs' is added. The different index structures included in "Ontology Editor" are listed in definition 4.9.

Definition 4.9 – Index structure definitions

```
[index structures]
   base = [nodes, arcs, nodes&arcs]
   nodes = [nodes&mass, nodes&energy, nodes&species]
   arcs = [arcs&mass, arcs&energy, arcs&species]
```

The introduction of an index structure also imposes a control of construction of a consistent variable space. When writing an equation using the operators, the index structures have to match with the used operator.

4.7 The variables and equations

Since the equations and variables form a bipartite graph, the overall mathematical representation is a super bipartite graph. In order to keep track of the mathematical representation, variable objects are introduced, which are separable into an ontology called the variable space.

4.7.1 Completeness

A complete variable space consists of a properly defined set of equations and variables. Equation and variables are defined using a set of differential

equations and these differential equations are supported and defined by a set of algebraic equations. The completeness of the variable space is checked by looking at the variable space as a simulation problem. The variable is properly specified if the equation set has no degrees of freedom given initial conditions, boundary conditions and parameters.

As already mentioned, a rule is imposed stating that the equations must be defined in a lower-triangular manner. That is, any new equation or variable must be defined using already defined variables. For example if a variable is dependent on two other variables as illustrated in equation (4.1).

$$\underline{\dot{x}} = \underline{\underline{\mathbf{A}}}\underline{x} \tag{4.1}$$

In order to define the new variable $\underline{\dot{x}}$, $\underline{\underline{\mathbf{A}}}$ and $\underline{x}$ must be defined.

Since the equations and variables are defined strictly in a lower-triangular manner, the variable and equation combination is strictly diagonal. By stating that the relationship between variable and equation are strictly diagonal, it is implied that for each dependent variable there exists a corresponding equation. This makes it possible to guarantee for the completeness of any model that is extracted from the variable spaces.

4.7.2 Behaviour - The variable types

Mathematical system theory defines behaviour as a subset of all possible events in the defined signal space (Willems 2007). This definition is used as foundation for making a set of signals to connect the types from the model ontology to the variable space. The variable type represent the context the variable primarily is intended and is the variable's connection to the building blocks defined in the model ontology. Since the variable types operate in two different networks they are separated in physical and informational variable types. The physical variable types are explained in table 4.1.

Information is also represented with variables. The variable framework for information is designed for signal processing. The informational variable types are explained in table 4.2.

4.7.3 The variable objects

The variable object is used to form variables together with operators, expressions and equations. The variable object has a number of default attributes:

- symbol: Unique ID, The symbol within the variable environment has to be unique.

[3]relative to the other capacities

Table 4.1 – Physical variable types explanation

Variable type	**Description**
Frame	Frame variable type represent the geometrical space or a time space.
Graph	Variable type for representing the graph objects
State	Conserved quantity associated with the respective token, for a physical space this represent either mass or energy
Secondary state	State dependent quantities such as intensive quantities, conjugates of potentials, volume or geometrical quantities
Transport	Represent the behaviour of flow of extensive properties between nodes. A time-scale assumption of high conductivity of the respective token has been made [3]. This variable type can only be related to arcs
Transposition	Represent kinetic behaviour internally in nodes. Reactions and phase transitions are also included here
Parameter	Model characteristic constants. Also included here are numerical constants to be used for example in context to geometrical shapes.

- doc: documentation string, written a documentation of the variable
- units: representation of physical units
- index_structs: list of index structures
- type: Variable type
- equations: list of rhs expressions for the variable together with a unique equation name and equation number

An example of a variable definition is shown in definition 4.10. This particular example is an example of time.

Table 4.2 – Informational variable types explained

Variable type	Description
Frame	Informational frame variables represent time. Time is common for physical and informational space. Does not include geometrical space
State	Represent the conserved quantity. Related to the informational token, namely information
Input	Represent the formation of information which is measured
Output	Represent the retrieval of signals

Definition 4.10 – Variable space example

```
[t]
var_type = frame
index_structures = ['nil']
equations = [('time', '_empty_', 'Eq000')]
units = [1, 0, 0, 0, 0, 0, 0, 0]
doc = time
```

An equation is defined for a specific variable, but each variable can have more than one equation describing the variable. A good example of alternative equation is equations if state for thermodynamic models.

$$p = \frac{nRT}{V} \tag{4.2a}$$

$$p = \frac{nRT}{V - Nb} - \frac{n^2 a}{V^2} \tag{4.2b}$$

As can be observed in equation (4.2a) and in equation (4.2b), there are two alternative state equations expressed for pressure, ideal gas and van der Waals equation respectively. Both these equations have the same description and the same units, and is therefore two alternatives for expressing the same quantity.

The equation is represented as a string. The rules for the expression are captured in the definition of a small language for which the definition is to attached to the variable space to facilitate compilation into different target languages.

The string is read by a parser implementing the language definition, and build an abstract syntax tree for each of the expressions. This abstract syntax tree can then be used to generate different target code using the templates for the target language.

4.8 Language definition

In the process of finding a minimal representation, the main challenge is to find the optimal structure of the ontology. This also extends to the language used to represent the mathematical expressions. At current state, "Ontology Editor" is designed for creating variable spaces for capturing macroscopic models. The language shown in definition 4.11 is designed for representing the macroscopic mathematical models. The language is defined using regular expressions and is written in Extended Backus Naur form [4].

Definition 4.11 – Language definition of the language design for representation of mathematical expressions in EBNF

```
token UFunc : '\b(sqrt|exp|log|ln|sin|cos|tan|asin|acos|atan)\b';
token Root : '\b(root)\b';
token VarID : '\w[\w]*'; #matches first word charfollowed by word
token IndID : '\|.*\|'

expr ::= term { ( "+" | "-" ) term }
term ::= fact { ( "." index "." | "*" | "\ " ) fact }
fact ::= atom "^" ( "+" | "-" ) VarID
atom ::= func | "(" expr ")"| VarID
func ::= unitary_func "(" expr ")"
          | root "(" expr ")"
          | integral "(" VarID "," VarID "," VarID "," VarID ")"
          | differential "(" VarID "," VarID ")"
```

The `unitary_func` items are the standard unitary functions which all are without units, except the `sqrt` in which the argument can have units. The `root` keyword is used to represent implicit equations. Integrals are represented using the `integral` keyword. An integral also includes arguments denoting the different inputs to the function. The integral is used in the form described below:

```
integral(derivative,integration variable, initial condition, upper
    limit)
```

The `differential` keyword represent differentials, the two arguments include the function and the variable, respectively.

The variable definition, `VarID` is composed of characters and numbers. The `IndID` represent index structures. In order to separate an index structure from a variable, the index structures have to be framed by square brackets. There are one operation dependent on the index structure, namely the product, `.|IndID|.` . The syntax is including the index structure over which the operator is reducing. The Khatri-Rao product is represented using a `*` notation. Since the Khatri-Rao product, in contrast to the regular

[4]Extended Backaus Naur Form (EBNF) is a family of metasyntax notation design to express context-free grammar

product, is expanding the index structure, no index structure have to be included.

All of the definitions above will trigger an action which generates a new variable in the implementation. This new variable is a result of the operation keeping track of the variables, and that applies the rules of the units and the index structures. For the addition, the result is an object which is the sum of two variables being added. In order to add two variables, both the units and the index structures must be equal for the two variables. These rules of both index structures and unit operations will be thoroughly presented in chapter 5.

By following this procedure, one build an abstract syntax tree. This abstract syntax tree can be evaluated for any output language. In order to evaluate the tree one only need a given a set of templates one for each of the actions and operators.

Chapter 5

Implementation of the "Ontology Editor"

When the ontological system has been defined, the "Ontology Editor" can be more closely introduced. This chapter will describe how the ontologies where implemented and how the "Ontology Editor" facilitates for the creation of variable space that can be used for creation of mathematical models.

5.1 Networks

Networks are defined for physical related variables and for information variables. As defined in the model ontology, a physical network type has a different refinement than an information network. Some of the refinements are shared, for example they operate over the same time frame, but most of the refinements are unique to only one of the networks. Both networks usually have to co-exist in a model. Since both networks are present and dependent on each other, there will be an interaction between the networks. This connection is called interconnection.

Each network is divided into more sub-networks. The physical network is divided into sub-networks based on phases. The information network is divided into sub-networks based on the signal type. There will also be communication between the different sub-networks, for example if a species is transformed from one phase to another. The communication between the sub-networks is called intraconnections. An overview of this structure is illustrated in figure 5.1.

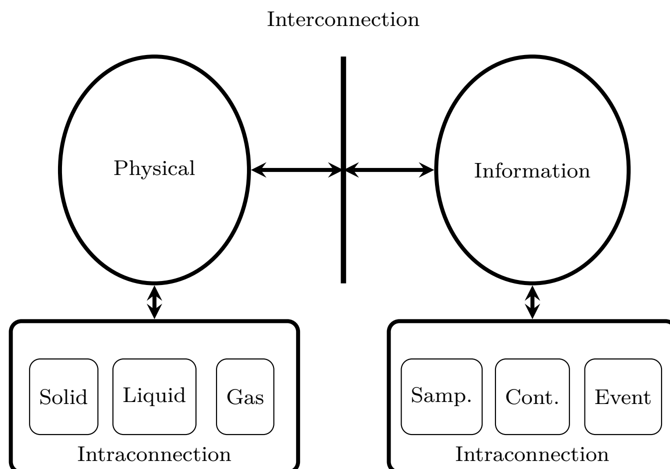


Figure 5.1 – Figure illustrates two network types and the internal interconnection

The different networks are in the modelling tool handled completely separately. This means that a variable space needs to be constructed for each of the networks (physical and information) and another variable space for the interconnection. The interconnection variable space will include equations and variables for the measurements, the dynamics of the measurement device and the translation of the measurement to a signal. The physical variable space will include physical variables, and will also include equations to describe the intraconnections. An example of such an equation is a chemical reaction involving a phase transition. The information variable space includes the control equations and the equations for translation between sampled, continuous and event time signals (Preisig 1996).

5.2 Rules

Rules control the process of model construction. The rules are separated based on the function they have during the construction of the models. Some rules are based on units and how the units are calculated. These rules are presented in section 5.2.2. These rules explain how the units for different mathematical operates are calculated. Some of the rules are based on mathematics. These mathematical rules are closely related to the structure of the indices. The last and most important rules are based on the modelling procedure for extraction of models from the ontologies. These rules are closely related to model definition and how we define the

containment, token and the realization of making a dynamic model. These rules are based on the model ontology and will be presented first.

5.2.1 Rules based on the model ontology

In a modelling environment some behaviours cannot be described by equations. Rules are used to define the actions possible for an object.

- Tokens are persistent. If a token is propagated in an arc, both connected nodes must contain the transferred token.
- Physical tokens can be added, deleted, converted or injected.
- Information tokens can be added or deleted.
- Variables can be defined, deleted or modified.
- Every variable must have units, index sets, a unique symbol and a type that represents the behaviour.

These rules are controlled and executed by the ontologies and in the user interface. The implementation of the rules is described in section 5.3.

5.2.2 Units and consistency check

All programming elements, including variables, parameters and expressions have physical units defined. This was declared in the rules based on the model ontology. When initiating a new variable space, the user have to assign scientific units to the basic elements. The basic elements of the variable space are considered to be the state and frame variables, the parameters and the system matrix - which is extracted from the graph editor in the "Process Modeller". For all the variables defined by expressions and equations within the "Ontology Editor" , the units are calculated based on the units of the elements and operators within the equations. The units of the equations are checked every time a new variable, equation or alternative equation is created, and an error is raised if something is found to be inconsistent. It is impossible to check if the basic elements are given the correct units, but since these elements, and the other variables defined by them, are used throughout the entire variable space, it is practically impossible to create a consistent variable space with the wrong units defined in the state variables. There are rules for calculating the units for most of the operations, but exception is the implicit equations. At the present state of the "Ontology Editor", the user must assign physical units to the implicit equations.

The units are checked for consistency by a set of rules, which applies for different mathematical operations.

- The basic elements must have user defined units.
- Summation operations require that both sides of the operator have identical units.
- Multiplication operations, which include product, Khatri-Rao product and integral operations, forms a new object. The units of the new object are the union of the units of the two objects that the operation handles.
- Division operations include differential operations when calculating units. The new units are calculated by subtracting the units of the variable that the function is divided by from the variable that represent the function. This is the opposite operation form the multiplication operation.
- Power operations cannot have units in the exponent.
- Square root unitary functions can have units.
- All other unitary functions cannot have units in the object.
- The user must assign units to the implicit variables
- When adding alternative equations, the units of the alternative equation must match the units already assigned to the variable

The units are represented using the base SI-units[1]. In the variable space, the seven base SI-units is represented in a list, which have eight slots. One slot assigned for each of the seven base units and one slot for assigning no units. An example of the unit representation and an explanation to the units is provided in definition 5.1.

Definition 5.1 – Unit definition

```
[units]
    vector.numbers = [0, 0, 0, 0, 0, 0, 0, 0]
    vector.letters = [s, m, mol, kg, K, A, cd, nil]
    s = time
    m = length
    mol = amount
    kg = mass
```

[1]SI is originally a French system and stand for "Le Système International d'Unités" which can be translated into "International system for units"

```
K = temperature
A = current
cd = light
nil = no units
```

When defining the units for the basic elements a dialogue window appears. A screen shot of this user dialogue is included in figure 5.2. For these particular units, energy $[J]$ is written out as base SI-units.

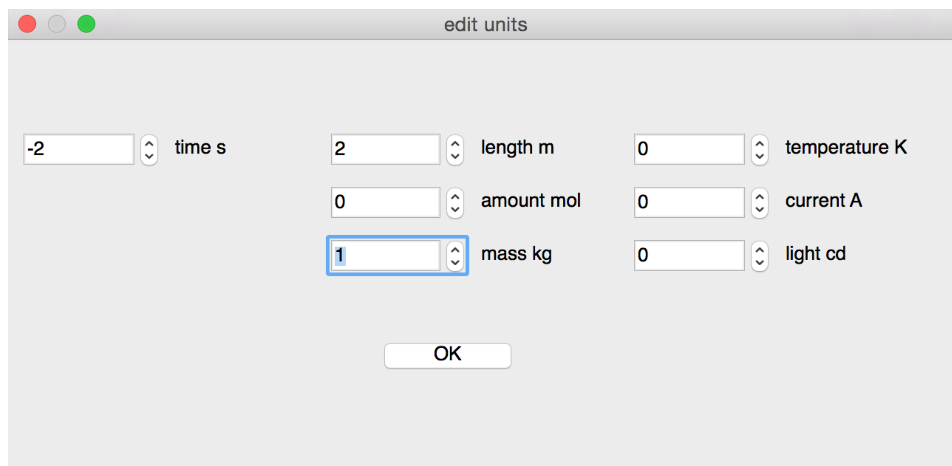


Figure 5.2 – This figure is a screen shot of user dialogue used for definition of units. In this example the base SI-units are used to express energy which is measured in joule $\left[\frac{kgm^2}{s^2}\right]$

.

It was considered sufficient to only allow for definition of units based on whole integers. In other words, it is not possible to create units with decimal numbers.

5.2.3 Index structure realization

Index structures are, as explained in section 4.6, used to describe the dimensionality of the variable. All elements have index structure for determining which building blocks the variable is defined for. For example the chemical potential is internal energy. This means that since the chemical potential is directly representing mass and species, it must be present in nodes. In the nodes there are species. Each of the species in the node has a chemical potential. This imposes that the chemical potential can be indexed in nodes & species. To get the total chemical potential in a node the chemical

potential is multiplied with the amount of species in the particular node as illustrated in equation (5.1).

$$\mu_{tot} = \underline{\mu} \times \underline{n} \tag{5.1}$$

Here the amount vector is also indexed with nodes & species. The multiplication reduces over species, that is the matrix product reduces the dimensionality of the final product over species. As a result of this vector multiplication, the total chemical potential is indexed in nodes.

The different operators have different indexing rules and requirements.

- The basic elements in a model must be given index structures.
- Summation operations requires the equal index structure on both sides of the operator.
- The inner product will impose a reduction in the index structure. The user must give the reducing index.
- The outer product, or Khatri-Rao product, will expand the index structure by making combination of the index structure of the objects. There are no mathematical restrictions to the expansion of the Khatri-Rao product, but for practical reasons expansion of the index structure to more than two dimensions was considered unnecessary.
- In a differential operation, the index structure of the new object will be inherited from the variable representing the function.
- In implicit equations, the variable inherit the index structure of the variable that is the basis of the implicit equation.
- Power functions return the index structure of the left hand side variable.
- Unitary functions return the index structure of the argument.
- When adding an alternative equation, the index structure of the alternative must be identical to the index structure already assigned to the variable.

The index structures of the basic elements are selected using a separate index selecting window. A screen shot of this user dialogue is included in figure 5.3.

Each variable can have more than one index structure and the user dialogue provide the possibility to select more than one index structure.

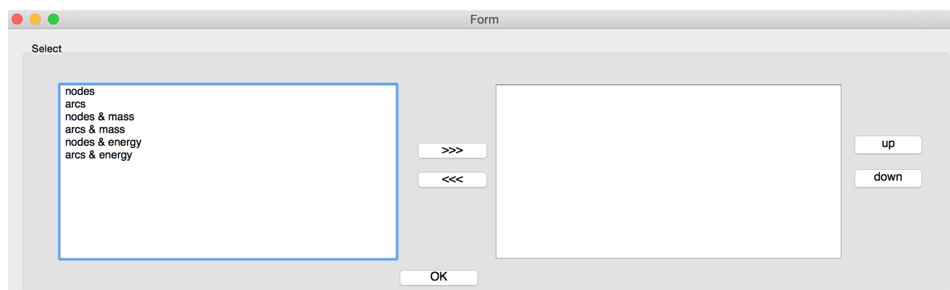


Figure 5.3 – This is a screen shot of the dialogue used for index selection.

5.2.4 Operators

Included in the "Ontology Editor" are ten operators. These operators are used to write equations. The operators are explained in table 5.1. The program code written in Python for construction of operator objects is included in appendix A.

5.3 The user interface

The user interface of the "Ontology Editor" is used to control and capture all the information provided by the user. The user provides input manually, either by writing a simple text sting into assigned text fields or by pressing buttons in the graphical user interface. The user interface is used to translate the manual inputs into objects, which are stored using the INI-format.

5.3.1 The parser implementation

The parser is the compiler of the grammar described in definition 4.11. It translates the strings given by the user in the text box into an equation written out in the INI-format. For this project the "Toy Generator Parser"[3], which will be denoted by TGP, was chosen to create the parser. The TGP was chosen for several reasons. Among the reasons are: provides a simple way of implementing self made parsers, it is free and written in Python.

[2]Khatri-Rao product is a block-by-block Kronecker product. For more information see (Khatri & Rao 1968)

[3]Web page of the Toy generator parser: `http://cdsoft.fr/tpg/`

Table 5.1 – Possible operators for equation definition in "Ontology Editor"

Name	Sym.	Use	Description
Plus	$+$	`A+B`	Adding variables together.
Minus	$-$	`A-B`	Subtracting one variable from another.
Dot	.	`A.|index|.B`	Multiplication of two variables with index reduction as normally used in matrix inner product. The index, which is being reduced, is written between the bars.
Star	$*$	`A*B`	Multiplication of two variables with the Khatri-Rao [2] product.
Power	ˆ	`A^B`	Power function. A to the power of B.
Total differential	tot dif	`Diff{A,B}`	Total differential of A with respect to B, $\frac{\partial A}{\partial B}$.
Partial differential	par dif	`diff{A,B}`	Partial differential of A with respect to B, $\frac{dA}{dB}$.
Integral	int	`int{dA::dt in [t0,t]}`	Integral of A from initial time to time, $\int_{t0}^{t} A\ dt$.
Inverse	inv	`inv{A}`	The inverse of A, equivalent to A^{-1}.
Root	root	`root(A)`	Implicit version defined from root expression of variable.

Before the parser designed for the "Ontology Editor" can be presented some key expressions and operators of the TGP must be introduced. These

key expressions and operators are presented in table 5.2.

Table 5.2 – Toy generator parser language definitions

Match	Description
`START`	Represent the start of the created object
`token`	Describes the matching of the keywords in the defined language
`()`	Parentheses are used for gropings. If something is enclosed by parenthesis, the parser starts a new syntax tree
`|`	The bar represent alternatives
`->`	Points to the next object in the sequence
`*`	The star after an object recognizes zero or more of the object. This is an inherited attribute from regular expressions.
`+`	The plus after an object recognizes one or more of the object. This is an inherited attribute from regular expressions.
`?`	The question mark after an object recognizes zero or one of the object. This is an inherited attribute from regular expressions.
`$`	The dollar sign construct objects. The objects are constructed using python code and are the objects assigned to the operators described in previous section.

The first step in implementing the parser is to define the keywords defined by tokens in TGP. These tokens must be unique and is used to match different operators in the equations. The token keyword is also defined for variable definition and the index structures. For definition of the tokens, regular expressions were used. The parser as implemented in the "Ontology Editor" is included in definition 5.2.

Definition 5.2 – Parser code example

```
class Expression(tpg.VerboseParser):
r'''
token UFunc : '\b(sqrt|exp|log|ln|sin|cos|tan|asin|acos|atan)\b';
token Root : '\b(root)\b';
separator spaces: '\s+' ;
token VarID : '\w[\w]*';
token IndID : '\|.*\|';
token sum : '[+-]';
token star : '\*';
token power : '\^';
token dot : '\.';
token ddot : ':';
START/e -> EXPR/e
;
EXPR/e -> TERM/e( sum/op TERM/t $e=Add(op,e,t,1, self.space)
    )*
```

```
;
TERM/t -> FACT/t (
    dot/op IndID/k dot FACT/f $t=ReduceProduct(op,k,t,f,2, self.space)
    |star/op FACT/f $t=KhatriRao(op,t,f,2, self.space)
)*
;
FACT/f -> ssATOM/f ( power/op ssATOM/e $f=Power('^',f,e,3, self.space)
             )?
;
ssATOM/ss -> sum/zz ATOM/a $ss = a
            | ATOM/a $ss = a
;
ATOM/a -> Func/fu $a = fu
    | '\(' EXPR/a '\)'
    | VarID/s $a=self.space.getPhysicalVariable(s)
;
Func/fu -> UFunc/s '\(' EXPR/a '\)' $fu=UFunc(s,a, self.space)
    | Root/s '\(' EXPR/a '\)' $fu=Implicit(s,a, self.space)
    | 'integral'/f
     '\{' TERM/dx '::'
        TERM/s 'in' '\['VarID/ll ',' VarID/ul '\]'
     '\}' $fu=Integral(dx,s,ll,ul, self.space)
    | 'Diff'/f
     '\{' EXPR/x ',' EXPR/y '\}' $fu=TotDifferential(x,y, self.space)
    | 'diff'/f
     '\{' EXPR/x ',' EXPR/y '\}' $fu=ParDifferential(x,y, self.space)
;
'''

verbose = 0


def __init__(self, space):
    self.space = space
    tpg.VerboseParser.__init__(self)
    self.space.eq_variable_incidence_list = []
```

In the "Ontology Editor" , this parser is used to translate a string into an abstract syntax tree that consists of objects. When a user first has written an equation, the entire string is sent to the starting point and creates an object. This object is then run into the parser that looks for matches in the string with the tokens defined in the parser. When the parser recognises a match, it creates an object and splits the string in two. Both of these strings are then run through the parser starting at the entry point, `START`. The parser works on the string from left to right and searches the entire string for each operator in the sequence defined. The sequence, and the grammar explanation of the parser, is explained in table 5.3.

Having established the key terms and operators in the parser, the parser can be explained in more detail towards what it does in the "Ontology Editor" . This is done by examples. For example look at the string `'A*B+C'`. First, the parser recognises the + sign which is matched by the token sum.

Table 5.3 – Toy generator parser language definitions. The sequence of the parser follows the order in which the expressions are explained.

Match	Description
START	Is the entry point. After the entry point the parser searches for an expression
EXPR	An expression is a term eventually followed with a + or a - sign
TERM	A term is a factor followed by a * or a .|IndID|. sign
FACT	A factor is a ssATOM followed by an optional power function
ssATOM	Is an ATOM with or without an optional sign matched by the token sum
ATOM	An ATOM is either a Func, an EXPR or a variables matched by the token `VarID`
Func	A function is either a unitary function, a root expression, an integral or one of the two differential operators

The parser creates an add object which then splits the string into two separate parts. The first part is `'A*B'` and the second part is simply `'C'`. The first part cannot find any matches to the token sum, but the parser match on the star operator. A Khatri-Rao object is then created, and the string is again split into two smaller strings, namely `'A'` and `'B'`. There are now three separate small stings consisting of only one single character. All these strings matches on the ATOM keyword in the variable identification. When the variable is matched, the variable object itself is returned. The matching of the variable ID does not have a following sequence in the parser. This means that the parsing of that string object is terminated and will stand alone as an object. The abstract syntax tree of this small example is illustrated in figure 5.4.

As can be observed in the provided example, the sequence in the parser is important for the correct translation of the string. The idea is to use the sequence of parsing to create the correct expressions instead of making specific rules. If the sequence of the expression and the term definition changed places in the parser, the star operator would be matched first and then the add operator. In the provided example, `'A*B+C'`, the parser would translate the sting into an abstract syntax tree as illustrated in figure 5.5. The result of this translation would not be allowed since the units of the object would not be equal. This is the tree that the string `'A*(B+C)'` would translate into and this string does not have the same meaning as `'A*B+C'`.

The parser, in the correct sequence, has been thoroughly tested. A test

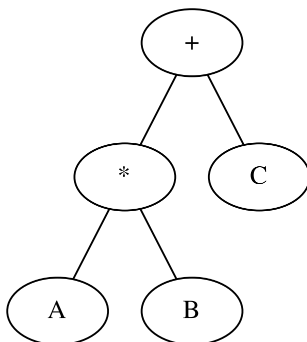


Figure 5.4 – The figure illustrates the abstract syntax tree and how it is constructed by the parser for the example string `'A*B+C'`

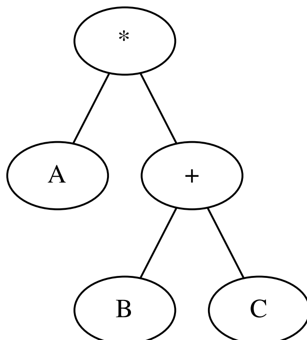


Figure 5.5 – The figure illustrates the abstract syntax tree and how it would be constructed by the parser for the example string `'A*B+C'` if the sequence in the parser changed

example was the variable displayed in equation (5.2).

$$Var = A \times B + \sin\left(C \times D\right)^{(-2)} + \frac{\partial\left(E \times (F - G)\right)}{\partial H} \tag{5.2}$$

The expression from equation (5.2) would have this form as a string:

```
'A*B+sin(C*D)^ (-2)+diff{E*(F-G),H}'
```

The abstract syntax tree of this example string is illustrated in figure 5.6. As can be observed in figure 5.6, the parser provides the correct translation of the example string into an abstract syntax tree. All the variables are at the

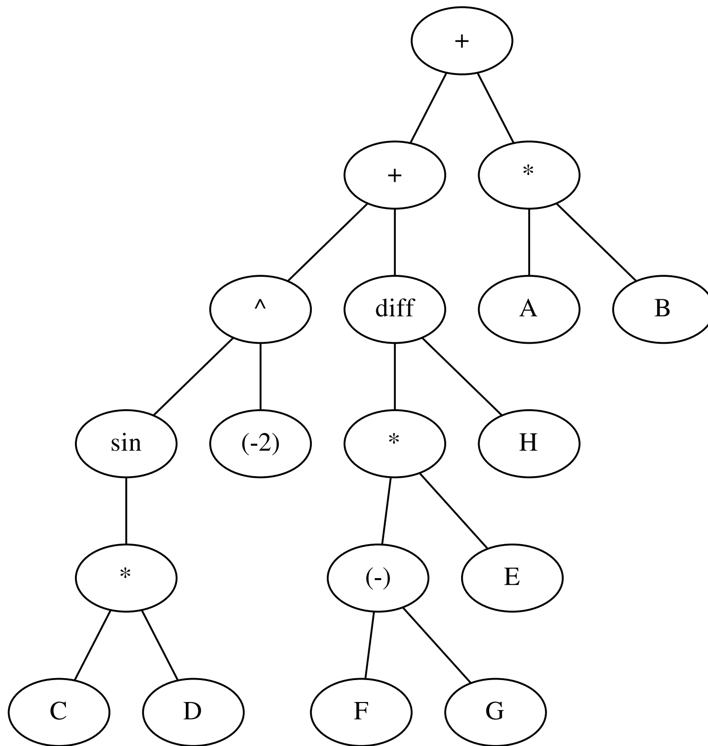


Figure 5.6 – This figure illustrates the abstract syntax tree constructed by the parser for equation (5.2)

.

leaf nodes and above the leave nodes the operator-objects are located. The correct operators are placed next to each other and the grouping operators create groups that are handled individually. The groupings done with the parenthesis initiate a new syntax tree. Worth noticing is the `'(-2)'` node located under the power operator. The minus sign is not matched by the token sum since the expression requires an object on both sides of the token. `-2` therefore matches first the `ssATOM` with the optional token sum match. The sum token is in this case included in the variable.

5.3.2 The equation dialog

The equation dialogue is where the user controls the operations around the different variables for creating a variable space. A screen shot of the equation editor is included in figure 5.7.

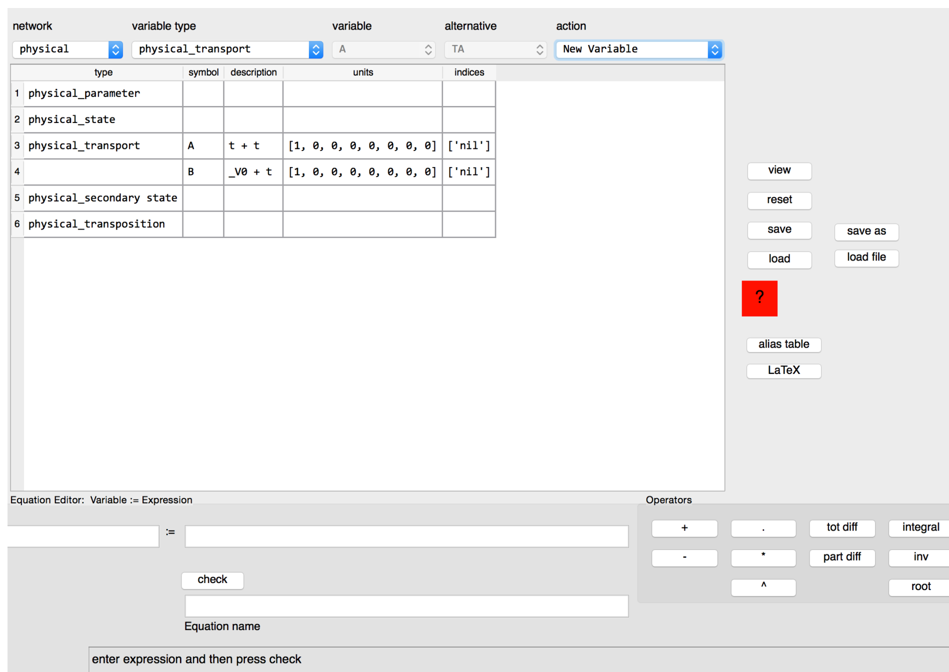


Figure 5.7 – Screen shot of equation dialogue.

The selection of variables in the equation editor is done using four comboboxes[4]. The first combobox selects the network. This select which variable space the equation will be written for. The next combobox is used for selecting the variable type. The third combobox is used to select the variable of the already selected network and variable type, and the last combobox is used to select alternative equation.

In order to create and manipulate the variables, six variable actions are included. These variable actions are explained below and include:

- New variable
- Change symbol
- Delete variable
- Alternative equation
- Edit equation

[4] A combobox is a selecting box in the graphical user interface

- Delete equation

New Variable

This action is used when defining a completely new variable. This variable type is available after a variable type is selected. When defining a new variable, the first thing to do is to give the variable a unique symbol within the network. An equation is then written into a textbox. This equation is written using the already defined variables and operator definitions already described in table 5.1. Templates for writing the operators according to the language definition are included in the user interface and available through buttons in the graphical user interface as illustrated in the screen shot included in figure 5.8.

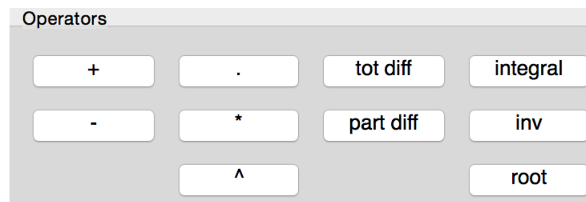


Figure 5.8 – This figure exhibit a screen shot of the user interface for the operator buttons. If one of these buttons is activated, a template for the operator appears in the textbox where the equations are written.

Then the equation and the symbol will be checked by the user interface. The equation is checked for correct definition of the abstract syntax tree, all variables used are defined and the rules for both units and index structures are being adhered. The symbol is checked for the uniqueness of the name. If the symbol and the equation are accepted, the equation must be given a unique name. This unique equation name is used for identifying that equation in the combobox. A screen shot of the new variable definition is included in figure 5.9.

If the name of the equation is accepted, the variable can be accepted, and will then be included to the variable space. If any of the tests fails, an error message appears explaining what is wrong with the definition of the variable and which test that failed.

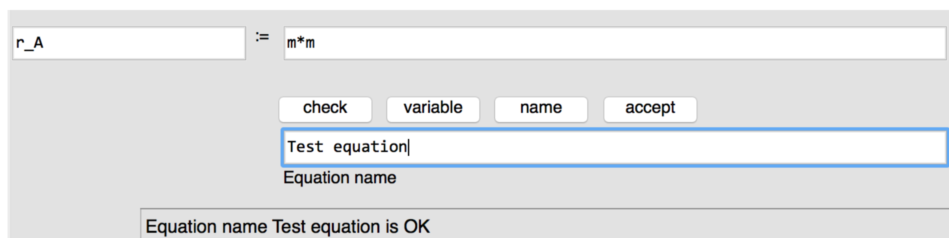


Figure 5.9 – This figure exhibit a screen shot of the section where a new variable is defined.

Change symbol

If the symbol of the variable is misleading or the documentation of the variable is unsatisfactory, the change symbol action is used. This action becomes available when a variable is selected. If the action is selected, a new window will open that allow for change of symbol, and redefinition of the variable documentation. A screen shot of the change symbol dialogue is included in figure 5.10.

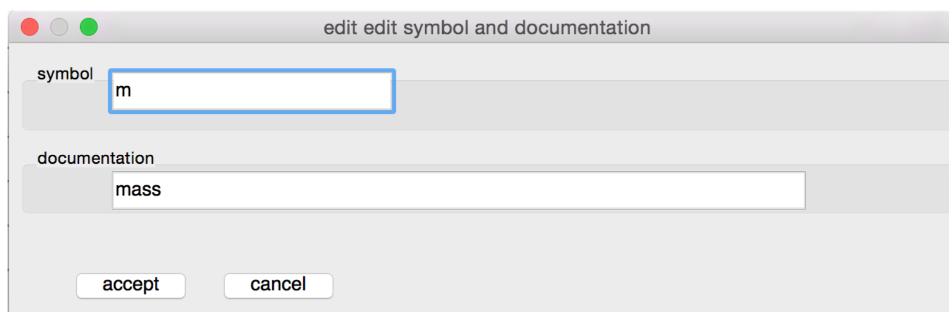


Figure 5.10 – This figure exhibit a screen shot of the dialogue that appears when the change symbol action is triggered.

If the new variable symbol is unique, the new symbol and documentation is updated in the variable space. The update of the symbol will also change the symbol in every equation dependent on that variable.

Delete variable

The deletion of a variable is a complex operation. When a variable is removed, all the dependent variables of the deleted are undefined in the syntax

tree. The same happens to the variables that are dependent on the dependent variables. When this option is selected a new version of the variable definition appears. This version has a delete button included. If the delete button is pressed, the variable is removed from the variable space and then the program reloads the variable space without the newly deleted variable. If a variable cannot be reproduced due to the removed variable, this variable will not be included in the new variable space.

Alternative equation

This action is used if there exist more than one alternative equation assigned to the same variable, as described in section 4.7.3. The variable editing section now opens with the variable symbol locked. The new alternative equation is written in the same manner as for a new variable. When this new variable is checked for correct definition of the abstract syntax tree, all variables used are defined, and the rules for both units and index structures are being adhered. The new equation must also have the same units and index structure as the one already defined in the variable. If the new equation fails in one of these tests an error is raised explaining what is wrong.

Edit equation

This action is available if an equation is selected, and allow for editing of that equation. Editing of an already existing equation is dangerous. As a rule, the newly written equation must have the same units and index structure as the equation already defined. If the equation has wrong units or wrong index structure, the only option is to make a new variable, and then delete the old variable.

Delete equation

This action is only available if there is more than one alternative equation. When deleting an alternative equation, the alternative equation is simply removed from the variable definition.

Chapter 6

Case study

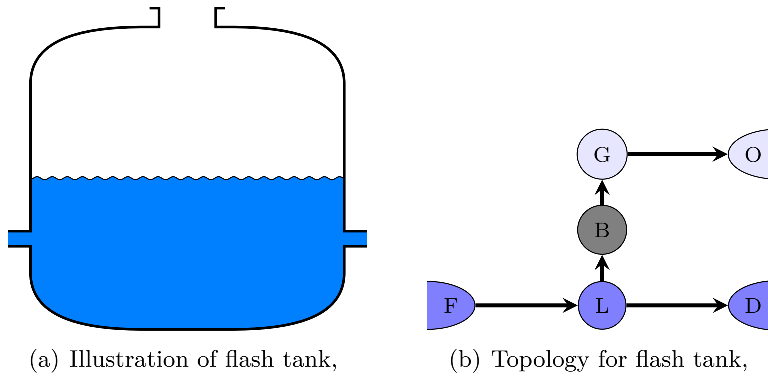
In order to verify the ontology representation from which a model is extracted a case study have been provided. The intention behind this case study is not to solve the model itself, but to show that it is possible to extracted models from the ontologies. At the present state there are still issues that need to be solved in the modelling tool. The intention behind this case study is not to create solvable models but to provide an example for how the "Process Modeller" can be used for defining equations, and how the equations will be represented in the variables space.

6.1 Study on the dynamic flash tank

6.1.1 Description

The layout for the flash tank is illustrated in figure 6.1. The tank has a liquid feed illustrated with reference F in the topology. The liquid and gas phase in the tank is denoted by L and G, and the boundary between the liquid and the gas phase is denoted by B. The liquid drain and outlet of gas outflow is denoted by D and O, respectively. Assumed is uniform properties within the tank, meaning that the model operates only in the time frame. The primary states are set to component mass and internal energy as shown in table 6.1.

Species in system is denoted by the A, B, C and c. C and c are the same species in liquid and gaseous phase respectively. Species A and B enters the tank from reservoir F and reacts to C which can exist in both liquid and gaseous phase.



(a) Illustration of flash tank, (b) Topology for flash tank,

Figure 6.1 – Illustration of a flash tank and corresponding topology

Table 6.1 – Primary states and frame definitions

Symbol	Documentations	Function
$\underline{n}$	Component mass vector	Primary state
U	Internal energy	Primary state
t	Time	Frame

6.1.2 Mathematical system description

The graph gives raise to the following incidence matrix:

	$F\|L$	$L\|D$	$L\|B$	$B\|G$	$G\|O$
F	-1	0	0	0	0
L	1	-1	-1	0	0
B	0	0	1	-1	0
G	0	0	0	1	-1
D	0	1	0	0	0
O	0	0	0	0	1

By removing the reservoirs is reduced:

	$F\|L$	$L\|D$	$L\|B$	$B\|G$	$G\|O$
L	1	-1	-1	0	0
B	0	0	1	-1	0
G	0	0	0	1	-1

In simple matrix form the incidence matrix would have the form described in equation (6.1).

$$\underline{\underline{\mathbf{F}}} = \begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{6.1}$$

The entire system have four possible species that can propagate throughout the model. The species vector for the flash tank have the form described by equation (6.2).

$$\underline{S}_s = \begin{bmatrix} A & B & C & c \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.2}$$

The vector for the species available in the liquid phase is described by equation (6.3).

$$\underline{S}_L = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix} \tag{6.3}$$

Possible species transferred from the feed reservoir to the liquid phase are described by equation (6.4).

$$\underline{S}_{F|L} = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \tag{6.4}$$

The minimal species flow matrix over connection can be found by diagonalize the species vectors and take the inner product as described by equation (6.5).

$$\underline{\underline{\mathbf{S}}}_{F,L|F} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{6.5}$$

The same procedure are repeated for every connection and node. All species flow matrix products are:

$$\underline{\underline{\mathbf{S}}}_{L,F|L} = \begin{bmatrix} 1\,0 \\ 0\,1 \\ 0\,0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{L,L|D} = \begin{bmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{L,L|B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{L,L|G} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{L,L|O} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\underline{\underline{\mathbf{S}}}_{B,L|B} = \begin{bmatrix} 0\,0 \\ 0\,0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{B,B|D} = \begin{bmatrix} 0\,0\,1 \\ 0\,0\,0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{B,B|B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{B,B|G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{B,B|O} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\underline{\underline{\mathbf{S}}}_{G,G|B} = \begin{bmatrix} 0\,0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{G,G|D} = \begin{bmatrix} 0\,0\,0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{G,G|B} = \begin{bmatrix} 0 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{G,G|G} = \begin{bmatrix} 1 \end{bmatrix} \quad \underline{\underline{\mathbf{S}}}_{G,G|O} = \begin{bmatrix} 1 \end{bmatrix}$$

All the diagonalized species matrices are inserted into the system species matrix S:

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

The final species flow matrix can be found taking the block-by-block Kronecker also called the Khatri-Rao product of $\underline{\underline{\mathbf{S}}}$ and $\underline{\underline{\mathbf{F}}}$, which is described by equation (6.6).

$$\underline{\underline{\mathbf{F}}}_S = \underline{\underline{\mathbf{S}}} * \underline{\underline{\mathbf{F}}} \tag{6.6}$$

$$\underline{\underline{\mathbf{F}}}_S = \left[\begin{array}{cc|ccc|c|c|c} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] * \begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

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

The generated species flow matrix is used in the balance equations of mass and energy.

6.1.3 Balance equations

The mass balance equations: The following mass balance can be written out as described in equations (6.7a) to (6.7d).

$$\dot{\underline{n}}_L = \hat{\underline{n}}_{F|L} - \hat{\underline{n}}_{L|D} - \hat{\underline{n}}_{L|B} + V_R \underline{\underline{\mathbf{N}}}^T \tilde{\underline{n}}_L \quad (6.7a)$$

$$\dot{\underline{n}}_B = \hat{\underline{n}}_{L|B} - \hat{\underline{n}}_{B|G} + V_R \underline{\underline{\mathbf{N}}}^T \tilde{\underline{n}}_B \quad (6.7b)$$

$$\dot{\underline{n}}_G = \hat{\underline{n}}_{B|G} - \hat{\underline{n}}_{G|O} \quad (6.7c)$$

$$\dot{n}_{i|j} = 0 \ \forall (i,j) \quad (6.7d)$$

The energy balances can be written as described in equations (6.8a) to (6.8d) by assuming that all nodes are static.

$$\dot{U}_L = \hat{H}_{F|L} - \hat{H}_{L|D} - \hat{H}_{L|B} - \hat{q}_{L|B} - \hat{w}_{L|B} \quad (6.8a)$$

$$\dot{U}_B = \hat{H}_{L|B} - \hat{H}_{B|G} + \hat{q}_{L|B} + \hat{w}_{L|B} - \hat{q}_{B|G} - \hat{w}_{B|G} \quad (6.8b)$$

$$\dot{U}_G = \hat{H}_{B|G} - \hat{H}_{G|O} + \hat{q}_{B|G} + \hat{w}_{B|G} \quad (6.8c)$$

$$\dot{U}_{i|j} = 0 \ \forall (i,j) \quad (6.8d)$$

6.1.4 Transport

Mass transport

For the flash tank there are diffusive transport between liquid phase and boundary, and between boundary and gas phase. This is described by equations (6.9a) and (6.9b).

$$\hat{\underline{n}}_{L|B} = -D_{L|B} \left(\underline{\mu}_B - \underline{\mu}_L \right) \quad (6.9a)$$

$$\hat{\underline{n}}_{B|G} = -D_{B|G} \left(\underline{\mu}_G - \underline{\mu}_B \right) \quad (6.9b)$$

For the reservoirs the mass transport is assumed to be of a convective nature as described in equations (6.10a) to (6.10c).

$$\hat{\underline{n}}_{F|L} = \underline{c}_{F|L} \hat{V}_{F|L} = \underline{c}_{F|L} \left(-\beta_{F|L} \mathrm{sgn}\left(p_L - p_F\right) \sqrt{|p_L - p_F|} \right) \quad (6.10a)$$

$$\hat{\underline{n}}_{L|D} = \underline{c}_{L|D} \hat{V}_{L|D} = \underline{c}_{L|D} \left(-\beta_{L|D} \mathrm{sgn}\left(p_D - p_L\right) \sqrt{|p_D - p_L|} \right) \quad (6.10b)$$

$$\hat{\underline{n}}_{G|O} = \underline{c}_{G|O} \hat{V}_{G|O} = \underline{c}_{G|O} \left(-\beta_{G|O} \mathrm{sgn}\left(p_O - p_G\right) \sqrt{|p_G - p_O|} \right) \quad (6.10c)$$

Energy transport

The transportation of energy follows the flow of mass and including the heat transformation in the phase transition as described in equations (6.11a) to (6.11h).

$$\hat{q}_{L|B} = -k_{L|B}\left(T_B - T_L\right) \tag{6.11a}$$

$$\hat{q}_{B|G} = -k_{B|G}\left(T_G - T_B\right) \tag{6.11b}$$

$$\hat{H}_{F|L} = \underline{h}_F \underline{\hat{n}}_{F|L} \tag{6.11c}$$

$$\hat{H}_{L|B} = \underline{h}_L \underline{\hat{n}}_{L|B} \tag{6.11d}$$

$$\hat{H}_{L|D} = \underline{h}_L \underline{\hat{n}}_{L|D} \tag{6.11e}$$

$$\hat{H}_{B|G} = \underline{h}_B \underline{\hat{n}}_{B|G} \tag{6.11f}$$

$$\hat{H}_{G|O} = \underline{h}_G \underline{\hat{n}}_{G|O} \tag{6.11g}$$

$$\underline{h}_i = \underline{h}_0 + \underline{c}_P\left(T_i - T_r ef\right) \tag{6.11h}$$

6.1.5 Transposition

Phase transposition

The boundary has no capacity meaning that the boundary itself does not contain mass or energy as described in equations (6.12a) and (6.12b).

$$\underline{n}_B = 0 \tag{6.12a}$$

$$U_B = 0 \tag{6.12b}$$

And the concentations of species is equal up to the boundary as described in equations (6.12c) to (6.12d).

$$c_{C_L} = c_{C_B} \tag{6.12c}$$

$$c_{c_B} = c_{c_G} \tag{6.12d}$$

6.1.6 Secondary states

The secondary state variables are listed in table 6.2.

To calculate the chemical potential at the boundary the two species, C which is in liquid phase and c which is in gaseous phase, must be treated as the same species. The chemical potential can be calculated from the steady state assumptions at the boundary as described in 6.13a–**??**.

$$\underline{\dot{n}}_B = \underline{\hat{n}}_{L|B} - \underline{\hat{n}}_{B|G} = 0 \tag{6.13a}$$

Table 6.2 – Secondary states

Symbol	Equation	Description
A	$U - TS$	Helmholtz energy
μ	$\frac{\partial U}{\partial n}$	Chemical potential
T	$\frac{\partial U}{\partial S}$	Temperature
$\underline{p}_i$	$\frac{\underline{n}_i RT}{V}$	Partial pressure
p	$\underline{e}^T \underline{p}_i$	Total pressure
V	$\frac{\partial U}{\partial p}, \frac{n_{tot}}{\rho_{tot}}$	Volumne
c	$\frac{\underline{n}_s}{V_s}$	Concentration
$\underline{h}$	$n_o + c_P \left(T - T_{ref}\right)$	Molar enthalpy
h	$\frac{V}{A}$	Height
S	$-\left(\frac{\partial A}{\partial T}\right)_{V,\underline{n}}$	Entropy
c_P	$\left(\frac{\partial H}{\partial T}\right)_p$	Heat capacity

From equation (6.9a) and equation (6.9b) :

$$-D_{L|B}\left(\underline{\mu}_B - \underline{\mu}_L\right) - \left(-D_{L|B}\left(\underline{\mu}_B - \underline{\mu}_L\right)\right) = 0 \tag{6.13b}$$

$$\underline{\mu}_B = -\frac{D_{L|B}\underline{\mu}_L - D_{B|G}\underline{\mu}_B}{D_{L|B} + D_{B|G}} \tag{6.13c}$$

The chemical reactions for this system is described by equations (6.14) and (6.15).

$$\mathrm{A} + \mathrm{B} \rightarrow \mathrm{C} \tag{6.14}$$

$$\mathrm{C} \leftrightarrow \mathrm{c} \tag{6.15}$$

The matrix representation this reaction system can be defined by equation (6.16).

$$\underline{\underline{\mathbf{N}}} = \begin{bmatrix} -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.16}$$

The reaction in the liquid phase is defined by equation (6.17).

$$r_L = k_{AB} C_A C_B - k_C C_C \tag{6.17}$$

In B defined by the diffusion of C into the boundary as described in equation (6.18).

$$r_B = \underline{\hat{n}}_{L|B} = -D_{L|B}\left(\underline{\mu}_B - \underline{\mu}_L\right) \tag{6.18}$$

The rection vector is the defined by equation (6.19).

$$\underline{r} = \begin{bmatrix} k_{AB} C_A C_B - k_C C_C \\ -D_{L|B}\left(\underline{\mu}_B - \underline{\mu}_L\right) \\ 0 \end{bmatrix} \tag{6.19}$$

The total mass balance can then be written out as described in equation (6.20)

$$\underline{\dot{n}} = \underline{\underline{\mathbf{F}}}_S \underline{\hat{n}} + \underline{\underline{\mathbf{N}}}^T \underline{r} \tag{6.20}$$

6.2 Dynamic flash implementation

Having done the theoretical background on the flash it would be interesting looking at the implementation in the modelling tool. All definitions are done using the user interface of the "Ontology Editor" .

6.2.1 The building blocks

The types of physical building blocks are described by definition 4.6, which is repeated below:

```
[node]
   lumped = [dynamic] + [0D]
   distributed_1 = [dynamic] + [1D]
   distributed_2 = [dynamic] + [2D]
   distributed_3 = [dynamic] + [3D]
   reservoir = [constant]
   boundary = [event]

[arc]
   energy_transfer = [energy]
   mass_transfer = [mass]
   species_transfer = [species]
```

The reservoirs F, D and O are represented using the type reservoir. The gas and liquid, G and L, phase are represented using the type lumped. The boundary is represented by the type boundary, which is used to describe event-dynamic behaviour. The arcs are all species_transfer and energy_transfer.

6.2.2 Frame

Frame for a physical network is time, t, and geometrical space, x. The following variables are then defined:

```
[t]
var_type = frame
doc = time
index_structures = ['nil']
equations = [('time', '_empty_', 'Eq000')]
units = [1, 0, 0, 0, 0, 0, 0, 0]

[geometrical space]
var_type = frame
doc = geometrical space
index_structures = ['nil']
equations = [('geometrical space', '_empty_', 'Eq001')]
units = [0, 1, 0, 0, 0, 0, 0, 0]
```

6.2.3 State equations

State equations are implemented by default. In our definition we use species, n, and energy, U. We choose to use internal energy as energy basis. Also included, by default is the mass definition. The following variables are included:

```
[m]
var_type = physical_state
doc = mass
index_structures = ['nodes & mass']
equations = [('mass', '_empty_', 'Eq002')]
units = [0, 0, 0, 1, 0, 0, 0, 0]

[n]
var_type = physical_state
doc = species
index_structures = ['nodes & species']
equations = [('species', '_empty_', 'Eq003')]
units = [0, 0, 1, 0, 0, 0, 0, 0]

[U]
var_type = physical_state
doc = Internal energy
index_structures = ['nodes & energy']
equations = [('Internal energy', '_empty_', 'Eq004')]
units = [-2, 2, 0, 1, 0, 0, 0, 0]
```

6.2.4 Incidence matrix

The incidence matrix represent the systems and the networks in the model. In the present state the modelling tool is not able to extract the incidence matrix from the graph editor. For now we introduce the incidence matrix manually in the same matter as it will be by the modelling tool. The incidence matrix take the following form in the variable space. The entire physical variable space is included in

```
[F]
var_type = incidence_matrix
doc = Incidence matrix for physical system
index_structures = ['nodes','arcs']
equations = [('Incidence matrix', '_empty_', 'Eq006')]
units = [0, 0, 0, 0, 0, 0, 0, 0]
```

The same applies also to the block species matrix which will be extracted from the graph editor and will have the following form as a variable:

```
[S]
var_type = incidence_matrix
doc = Species matrix for the system
index_structures = ['nodes & species','arcs & species']
equations = [('Incidence matrix', '_empty_', 'Eq006')]
```

```
units = [0, 0, 0, 0, 0, 0, 0, 0]
```

Now it is possible construct the species flow matrix:

```
[F_S]
var_type = None
doc = F * S
index_structures = ['nodes & species', 'arcs & species']
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Species block by block matrix', 'F*S', 'Eq027')]
```

Also matrices for energy and work transport can be extracted from "Process Modeller".

```
[F_e]
doc = Incidence matrix for energy physical system
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Incidence matrix for energy physical system', '_empty_',
    'Eq005')]
var_type = incidence_matrix
index_structures = ['nodes', 'arcs']

[F_w]
doc = Incidence matrix for work physical system
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Incidence matrix for work physical system', '_empty_', '
    Eq006')]
var_type = incidence_matrix
index_structures = ['nodes', 'arcs']
```

6.2.5 Reactions and reaction systems

Reactions and reaction systems are not implemented. Ideally the reaction systems should be extracted from a database containing possible reactions for the available species. The modelling tool will have to calculate the reaction matrix, it would then have the form described below:

```
[N]
var_type = incidence_matrix
doc = Reaction matrix
index_structures = ['nodes & species', 'nodes']
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Reaction matrix', '_empty_', 'Eq018')]

[r]
var_type = incidence_matrix
doc = kinetics
index_structures = ['nodes & species', 'nodes']
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('kinetics', '_empty_', 'Eq019')]
```

6.2.6 Parameters

A lot of parameters need to be defined in order to produce the current equations. The valve constant have units that are impossible to create using the modelling tool. Therefore, the squared beta was defined.

```
[Beta2]
doc = Squared valve constant
units = [0, 7, 0, -1, 0, 0, 0, 0]
equations = [('Squared valve constant', '_new_', 'Eq023')]
var_type = physical_parameter
index_structures = ['arcs', 'nodes']
```

Taking the square root of this parameter will produce de correct units:

```
[Beta]
doc = sqrt(Squared valve constant)
units = [0.0, 3.5, 0.0, -0.5, 0.0, 0.0, 0.0, 0]
equations = [('Valve constant', 'sqrt(Beta2)', 'Eq024')]
var_type = physical_secondary state
index_structures = ['arcs', 'nodes']
```

The rest of the parameters were produced without any problems:

```
[R]
doc = Gas constant
units = [-2, 2, -1, 1, -1, 0, 0, 0]
equations = [('Gas constant', '_new_', 'Eq010')]
var_type = physical_parameter
index_structures = ['nil']

[Cp]
doc = Heat capacity, constant pressure
units = [-2, 2, -1, 1, -1, 0, 0, 0]
equations = [('Heat capacity, constant pressure', '_new_', 'Eq011')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[D]
doc = Diffusivity Liquid to boundary
units = [1, -2, 2, -1, 0, 0, 0, 0]
equations = [('Diffusivity constant', '_new_', 'Eq012')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[T_0]
doc = Standard temperature
units = [0, 0, 0, 0, 1, 0, 0, 0]
equations = [('Standard temperature', '_new_', 'Eq013')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[k]
doc = Heat diffusion constant
units = [-1, 0, 1, 0, 0, 0, 0, 0]
equations = [('Heat diffusion constant', '_new_', 'Eq015')]
var_type = physical_parameter
index_structures = ['nodes','arcs']
```

```
[deltah_0]
doc = Enthalpy STP
units = [-2, 2, -1, 1, 0, 0, 0, 0]
equations = [('Enthalpy STP', '_new_', 'Eq016')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[s_0]
doc = entropy STP
units = [-2, 2, -1, 1, -1, 0, 0, 0]
equations = [('entropy STP', '_new_', 'Eq017')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[1]
doc = 1 numerical constant
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('-1 numerical constant', '_new_', 'Eq018')]
var_type = physical_parameter
index_structures = ['nil']
```

6.2.7 Secondary state equations

The secondary state variables are those represent the derivatives of the state equations. The secondary state equations defined for this system include:

```
[mu]
doc = Chemical potential
units = [-2, 2, -1, 1, 0, 0, 0, 0]
equations = [('Chemical Potential', 'diff{U,n}', 'Eq009')]
var_type = physical_secondary state
index_structures = ['nodes']

[T]
doc = root
units = [0, 0, 0, 0, 1, 0, 0, 0]
equations = [('Temperature', 'root(U)', 'Eq014')]
var_type = physical_secondary state
index_structures = ['nodes']

[p]
doc = root
units = [-2, -1, 0, 1, 0, 0, 0, 0]
equations = [('Pressure', 'root(U)', 'Eq025')]
var_type = physical_secondary state
index_structures = ['nodes']

[n_dot]
doc = F_S . r
units = [-1, 0, 1, 0, 0, 0, 0, 0]
equations = [('Mass balance', 'F_S.|arcs|.n_hat+N.|arcs|.r', 'Eq028')]
var_type = physical_secondary state
index_structures = ['nodes']

[V]
```

```
doc = root
units = [0, 3, 0, 0, 0, 0, 0, 0]
equations = [('Volume', 'root(U)', 'Eq029')]
var_type = physical_secondary state
index_structures = ['nodes']

[h]
doc = deltah_0 + _V1
units = [-2, 2, -1, 1, 0, 0, 0, 0]
equations = [('Enhalpi', 'deltah_0+Cp.|arcs|.(T-T_0)', 'Eq030')]
var_type = physical_secondary state
index_structures = ['nodes']

[U_dot]
doc = F . q_hat
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Energy balance', 'F.|arcs|.H_hat+F_e.|arcs|.q_hat', '
    Eq032')]
var_type = physical_secondary state
index_structures = ['nodes']

[c]
doc = root
equations = [('consentration, volumentric', 'root(n)', 'Eq055')]
units = [0, 0, 1, 0, 0, 0, 0, 0]
var_type = physical_secondary state
index_structures = ['nodes']
```

6.2.8 Transport equations

For species flow there exist more than one alternative equation. One for diffusion and one for advection.

```
[n_hat]
doc = D . _V0
equations = [('Mass diffusion', 'D.|nodes|.(mu-mu)', 'Eq022'),('Mass
    advection', 'c.|nodes|.V_hat', 'Eq040')]
units = [-1, 0, 1, 0, 0, 0, 0, 0]
var_type = physical_transport
index_structures = ['arcs']
```

The rest of the transport equations are written out using equations:

```
[V_hat]
doc = Beta . _V1
units = [-1.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0]
equations = [('Volumentric flow, advection', '-Beta.|nodes|.sqrt(p-p)',
    'Eq026')]
var_type = physical_transport
index_structures = ['arcs']

[q_hat]
doc = k . _V0
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Heat transport diffusion', 'k.|nodes|.(mu-mu)', 'Eq027')]
var_type = physical_transport
```

```
index_structures = ['arcs']

[H_hat]
doc = h . n_hat
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Energy trasport', 'h.|nodes|.n_hat', 'Eq031')]
var_type = physical_transport
index_structures = ['arcs']
```

6.3 Adding control

In order to have stable operation of the flash tank control must be added. The pressure in the tank must be kept constant and a liquid level in the tank must also be held constant. The flow of gas is chosen to control the pressure in the tank and the liquid level is controlled by the liquid flow out of the tank. Required for this control structure is a measurement of the pressure in the gas phase, a measurement of the liquid level in the liquid phase, two controllers and two manipulators to manipulate the flows out of the tank. This control structure is illustrated in figure 6.2.

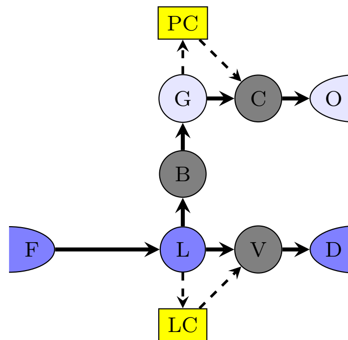


Figure 6.2 – This is an illustration of the topology of the flash tank with control structure added

The control equations are not included in this case study.

6.3.1 The building blocks

The building blocks for the control system are extracted from the types defined interactions between the physical and information listed in definition 4.8 and the types defined internal representation in the information network listed in definition 4.7. These types are repeated below:

```
[node]
```

```
    control system = [event] + [information state]
[arc]
    signal = [input, output] + [uni-directional]
```

For interactions between physical token and information the following types are defined:

```
[node]
    measurement = [event] + [secondary state, input]
    manipulator = [event] + [parameter, output]
```

The pressure controller, PC, and the level controller, LC, is represented by the type name control system. The measurements of the level in L and the pressure in G are represented by the type measurements. The valves V and C are represented by the type manipulator. The arcs connected to the control systems are all classified as signals. These signals are all uni-directional.

6.3.2 The interaction variable space

Measurements are of the secondary state. In G, the measurement is of pressure. This is already described in the the physical variable space. The measurement of the level can be derived from the volume of the liquid phase provided the area. This can be described by equation (6.21).

$$l = \frac{V}{A} \tag{6.21}$$

A parameter must be given to the interaction variable space for the area before the equation can be written.

```
[p]
doc = root
units = [-2, -1, 0, 1, 0, 0, 0, 0]
equations = [('Pressure', 'root(U)', 'Eq040')]
var_type = physical_secondary state
index_structures = ['nodes']

[V]
doc = root
units = [0, 3, 0, 0, 0, 0, 0, 0]
equations = [('Volume', 'root(U)', 'Eq041')]
var_type = physical_secondary state
index_structures = ['nodes']

[A]
doc = Area of liquid phase
units = [0, 2, 0, 0, 0, 0, 0, 0]
equations = [('Area', 'root(U)', 'Eq042')]
var_type = parameter
index_structures = ['nodes']
```

```
[l]
doc = root
units = [0, 1, 0, 0, 0, 0, 0, 0]
equations = [('level', 'root(V)', 'Eq043')]
var_type = physical_secondary state
index_structures = ['nodes']
```

The manipulators, C and V, are controlled using the valve constant β.

```
[Beta]
doc = sqrt(Squared valve constant)
units = [0.0, 3.5, 0.0, -0.5, 0.0, 0.0, 0.0, 0]
equations = [('Valve constant', 'sqrt(Beta2)', 'Eq024')]
var_type = physical_secondary state
index_structures = ['arcs', 'nodes']
```

The entire interaction variable space is printed in appendix C.

Part III

Discussion and conclusion

Chapter 7

Discussion and unresolved issues

This chapter will present some unresolved issues that have to be implemented before the "Ontology Editor" can be used for generating variable spaces. Also included in this chapter is the discussion on the formulated ontologies, the structure of the ontologies, the implementation problems and the results form the case study.

7.1 Unresolved issues

There are still issues to be resolved before the "Ontology Editor" can be integrated as a fully operating part of the "Process Modeller". These problems are important to fix, but was considered not to be urgent and considered to be solvable. Included in this section is suggestions for possible solutions to these unresolved issues.

7.1.1 Construction of dependent equation set

If one have two equations defined and dependent on each other as in the form:

$$y_1 = f_1\left(\underline{x}, y_1, y_2\right) \tag{7.1}$$

$$y_2 = f_2\left(\underline{x}, y_1, y_2\right) \tag{7.2}$$

At the present state the equation generator checks the variables used to express the new variable. The variables used in an expression must be

already defined. Because of this the two equations above can not be defined. A possible solution to this problem is to allow for block equations as shown under in equation (7.3).

$$(y_1, y_2) = \left[f_1\left(\underline{x}, y_1, y_2\right), f_2\left(\underline{x}, y_1, y_2\right)\right] \tag{7.3}$$

Alternatively one could allow for vector blocks as shown in equation (7.4):

$$\underline{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} f_1\left(\underline{x}, y_1, y_2\right) \\ f_2\left(\underline{x}, y_1, y_2\right) \end{bmatrix} \tag{7.4}$$

7.1.2 Construction of artificial units

When defining the units only by usage of integers, some of the parameters used in classical modelling becomes impossible to define. For example, pressure driven convection volume flow from A to B is usually driven by the following equation which involves a valve constant:

$$\hat{V}_{A|B} = \beta_{A|B}\,\mathrm{sgn}\left(p_B - p_A\right)\sqrt{|p_B - p_A|} \tag{7.5}$$

The units for the valve constant becomes somewhat artificial:

$$\beta\ \left[\frac{m^{3.5}}{kg^{0.5}}\right] \tag{7.6}$$

In reality, the valve constant, β is certainly not constant as the valve changes the cross section when manipulating the flow.

A fix to this problem is to define the parameter as a squared version of the wanted parameter, and then use it with as the square root of the parameter.

$$\zeta\ \left[\frac{m^7}{kg}\right] = \left(\beta\ \left[\frac{m^{3.5}}{kg^{0.5}}\right]\right)^2 \tag{7.7}$$

The unit dialogue would then look as shown in figure 5.2 and the unit list would take the following shape:

```
units = [ 0, 7, 0, -1, 0, 0, 0, 0]
```

Equation (7.5) would then be written as:

$$\hat{V}_{A|B} = \sqrt{\zeta_{A|B}}\,\mathrm{sgn}\left(p_B - p_A\right)\sqrt{|p_B - p_A|} \tag{7.8}$$

7.2 Discussion

7.2.1 Results of the case study

The case study described the procedure that a modeller would have to undergo for construction of a variable space. Not all details of the "Ontology Editor" are at the present state implemented and resolved. Despite of this, the case study provide an indication on the effort and complexity behind creating a variable space. And the fact that the modelling tool is incomplete help to illustrate one of the main benefits by using an ontology for variable space, namely the ability for reuse an ontology in another application than what it was designed for.

Reuse

The variables and equations that were impossible to construct, were loaded directly into the variable space. A user could also do the same. If a variable is defined in another model, the variable could be loaded in to the new model as long as all the other equations are defined. Reuse of the variables and models is one of the most important benefits by using this modelling procedure.

Parameters

In the case study a lot of parameters had to be defined. The parameters that were included as flow determination are essential to control and definition of the transport equations. These parameters vary from model to model and from modeller to modeller, and must be defined every time. On the other hand, the numerical constants, such as π and numbers, are fixed. These parameters would not change meaning during from model to model. It is possible to argue that one could include numbers as constants recognized by the parser. These constants would then have to be a part of the abstract syntax tree and created as separate objects.

7.2.2 User interface

The user interface of the "Ontology Editor" is designed for construction a variable space. The functionality needs to be improved before the software can be released. For example it is not possible to look at the equations already written. just the name of the equation, the symbol of the variable, units and indices are shown. Being able to look at the actual equations would improve the functionality.

Actions

The actions available for generating of variables are: 'New Variable', 'Change Symbol', 'Delete Variable', 'Alternative Equation', 'Edit Equation' and 'Delete Equation'. These actions were sufficient for creating a variable space. Six different actions are not too many, but some of the actions could perhaps be combined. For example one could combine change symbol and delete variable into one action named edit variable, and delete equation could be a part of edit equation. The actions could then be reduced to the following:

- *New Variable*, initiate a new variable

- *Edit Variable*, edit the symbol and documentation of an already defined variable and allow for deletion of the variable.

- *Alternative Equation*, construct an alternative equation to an already defined variable.

- *Edit equation*, edit the selected equations related to the variable, or delete the entire equation if more than one alternative is defined.

By reducing the amount of available actions from six to four one could consider making buttons instead of combo-boxes for selecting actions.

7.2.3 Ontologies

The basic structure of the entire ontology was presented in chapter 4. Ontology design will require making trade-offs among the criteria defined in chapter 2. In this ontology the most important design criterion was considered to be minimal representation. This minimal representation criterion was complied by allowing for extendibility, which is another of the design criterion. Since the ontologies defined allow for extendibility, the criterion of minimal ontological commitment is contradicted. When allowing for extendibility the ontologies requires more defined keywords. This make the ontologies more committed to each-other and requires a shared vocabulary. This conflict between the extendibility and the minimal ontological commitment has been discussed before among others in (Aitken 1998) and in (Gruber 1993). Gruber refer to the conflict between the criterion of extendibility and the criterion of minimal ontological representation as a trade-off.

7.2.4 Implementations

This subsection includes discussion on the most important features of the implementation of the ontologies into the modelling tool.

The equations and variables

Variables in the variable space are represented by the format exemplified in definition 7.1.

Definition 7.1 – example of variable representation

```
[q_hat]
doc = k . _V0
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Heat transport diffusion', 'k.|nodes|.(mu-mu)', 'Eq027')]
var_type = physical_transport
index_structures = ['nodes & species']
```

This representation of a variable is compact, but still clear and readable. The user is not required to give documentation to the variables that are created. If the user does not include documentation on the variable, the first objects in the abstract syntax tree is assigned as documentation. For the example given in definition 7.1 the documentation is generated using an object stored in the syntax tree. The last object, '_V0' is an object created inside the abstract syntax tree. These objects are given temporary names in the structure, and are not intendant for a user to see. A solution to this problem would be to require the user to provide documentation to the variable. This would impose more work for a user, but would result in an even more readable variable space.

The representation of the units in form of a list with eight numbers is unreadable for a human, but easily readable for a computer provided that the sequence of the numbers and how they should be interpreted is provided. In the implementation of the units many other alternatives for storage was considered. Among others expanding each slot in the list to a tuple containing unit identification and number. The example provided in definition 7.1 would then have the form illustrated below:

```
[q_hat]
doc = k . _V0
units = [('s',-3), ('m',2), ('mol',0), ('kg',1), ('K',0), ('A',0), ('cd'
    ,0), ('nil',0)]
equations = [('Heat transport diffusion', 'k.|nodes|.(mu-mu)', 'Eq027')]
var_type = physical_transport
index_structures = ['nodes & species']
```

Despite this alternative way of formulation the variable format being much more readable for a human, the other alternative was preferred because of simplicity of the other format.

Also worth mentioning is the eighth and last slot in the representation of the units. The intention behind this slot was the representation of no units. This is never used, instead no units are represented by only zeros in the other slots representing the base units. The eight slot could then be removed.

Index structure rules and implementation

The rules for the implementation of the index structure are under development. Especially, since it is difficult to have control over the index structures of every single object. The index structures need to be implemented at the state variables. That happens at the initiation of the modelling procedure. When doing detailed modelling with equations, it is difficult at all time to have control over the index structure. A presently unclear situation is the index structure of transport variables. Transport variables are only represented in arcs, but are directly dependent on variables valid for nodes. The inheritance of index structures from the node defined variables added together and multiplied with parameter need more thinking.

The operators that changes the index structure is the inner product (matrix multiplication product) and the Khatri-Rao product. The inner product is in reality more used, but more complex to use correctly in the definition of an equation. The Khatri-Rao product is used for expanding the index structure. This is rarely done in the modelling equations, and the only example found is when expanding the incidence matrix, which is indexed with nodes and arcs with the species vector, which is indexed with nodes&species and arcs&species. Notation for the Khatri-Rao product is a star,`*`, while the notation for inner product is, `.|index|.`. The star operator is often used for inner products in other programming languages[1]. In order to reduce the amount of overuse of the Khatri-Rao product, a possible solution could be to change the notation. For example one could consider writing the Khatri- Rao product as `KRp{A,B}` were `A` and `B` denote the objects being multiplied.

Another issue that have to be discussed is the rules of what have to be index. At the moment, indexing happens over containment and token. One should also index over the intraconnections in the network. For the physical network one would have to expand the defined index structures with phase. Within the information network one would then also index over the signal type.

[1] For example: MATLAB

7.3 Further work

The use of this ontology facilitate for a modelling procedure described in (Preisig 2010). The most important challenge for a modelling tool using this modelling procedure is to get the modelling procedure known and accepted. The best way getting this modelling procedure known is to create a modelling tool with a structured and safe modelling methodology for construction of good process models.

Chapter 8

Conclusion

In this study, ontologies intended for structuring information of physical-chemical-biological processes has been presented. The ontologies was implemented in a tool, called the "Ontology Editor", that facilitate for construction of another ontology for representation of a variable space. The implementation of the ontology was presented and a case study using the "Ontology Editor" was carried out for the purpose of being an illustration of concept.

The ontology presented provide structure to the classification of physical-chemical-biological systems. The entire classification consist of only three files. A file that classify nodes and arcs and two files from refining the nodes and arcs in a physical and informational nature. Each of these files consist of keywords with associated lists. The entire classification of physical-chemical-biological systems are done in 33 lines of code. The model objects extracted from this ontology as types can be used as building blocks of a model and the variable spaces created using the "Ontology Editor" can be used for representation of mathematical models. and the "Ontology Editor" provide a variable space that mathematical models can be extracted from. This provided structure could be the bases of a more structured modelling procedure that provide a safe approach for handling model complexity.

Bibliography

Aitken, S. (1998), 'Extending the HPKB-Upper-Level Ontology : Experiences and Observations'.

Aris, R. (1978), *Mathematical modelling techniques* , Pitman, London.

Austin, S. (1986), 'Parmenides: being, bounds and logic '.

Bieszczad, J. (2000), A framework for the language and logic of computer-aided phenomena-based process modeling, PhD thesis.

Bogusch, R., Lohmann, B. & Marquardt, W. (2001), 'Computer-aided process modeling with ModKit', *Computers and Chemical Engineering* **2**5, 963–995.

Cellier, F. E. (1991), *Continuous system modeling* , Springer, New York.

Gruber, T. (1995), 'Toward principles for the design of ontologies used for knowledge sharing', *International Journal of Human-Computer Studies* **4**3, 907–928.
URL: *http://dx.doi.org/10.1006/ijhc.1995.1081*

Gruber, T. R. (1993), 'A translation approach to portable ontology specifications', *Knowledge Acquisition* **5**, 199–220.
URL: *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.7493*

Hangos, K. M. & Cameron, I. T. (2001), *Process modelling and model analysis* , Academic Press, San Diego.

Haug-Warberg, T. (2006), 'Den termodynamiske arbeidsboken: -for de to hundre hjem '.

Hawkins, J. (1986), 'The Oxford reference dictionary '.

Khatri, C. G. & Rao, C. R. (1968), 'Solutions to Some Functional Equations and Their Applications to Characterization of Probability Distributions', *Sankhya* **3**0(2), 167–180.

Kuntsche, S., Barz, T., Kraus, R., Arellano-Garcia, H. & Wozny, G. (2011), 'MOSAIC a web-based modeling environment for code generation', *Computers & Chemical Engineering* **3**5(11), 2257–2273.
URL: *http://www.sciencedirect.com/science/article/pii/S0098135411001128*

Marquardt, W., Morbach, J., Wiesner, A. & Yang, A. (2010), 'OntoCAPE: A Re-Usable Ontology for Chemical Process Engineering '.

Nise, N. S. (2013), 'Control systems engineering '.

Noy, N. F. & Klein, M. (2004), 'Ontology evolution: Not the same as schema evolution', *Knowledge and information systems* **6**(4), 428–440.

Ogunnaike, B. A. & Ray, W. H. (1994), *Process dynamics, modeling, and control* , Oxford University Press, New York.

Piela, P., Epperly, T., Westerberg, K. & Westerberg, A. (1991), 'ASCEND: an object-oriented computer environment for modeling and analysis: The modeling language', *Computers & Chemical Engineering* **1**5(1), 53–72.
URL: *http://www.sciencedirect.com/science/article/pii/009813549187006U*

Polderman, J. W. & Willems, J. C. (1998), 'Introduction to Mathematical Systems Theory: A Behavioral Approach '.

Preisig, H. A. (1996), 'Computer-aided modelling — Two paradigms on control', *Computers & Chemical Engineering* **2**0, Supplement 2(0), S981–S986.
URL: *http://www.sciencedirect.com/science/article/pii/0098135496001718*

Preisig, H. A. (2010), 'Constructing and maintaining proper process models', *Computers & Chemical Engineering* **3**4(9), 1543–1555.
URL: *http://www.sciencedirect.com/science/article/pii/S0098135410000669*

Preisig, H. A. (2013), 'The ABC of Process Modelling'.

Priemer, R. (1991), 'Introductory signal processing '.

Stephanopoulos, G. (1984), *Chemical process control: an introduction to theory and practice* , Prentice-Hall, Englewood Cliffs, N.J.

Stephanopoulos, G., Henning, G. & Leone, H. (1990), 'MODEL.LA. A modeling language for process engineering—I. The formal framework', *Computers & Chemical Engineering* **14**(8), 813–846.
URL: *http://www.sciencedirect.com/science/article/pii/009813549087040V*

Uschold, M. & Gruninger, M. (1996), 'Ontologies : Principles , Methods and Applications', (February).

Welty, J. R., Wicks, C. E. & Wilson, R. E. (1976), 'Fundamentals of momentum, heat, and mass transfer '.

Westerweele, M. R. (2003), *Five Steps for Building Consistent Dynamic Process Models and Their Implementation in the Computer Tool Modeller*, Technische Universiteit Eindhoven.
URL: *http://books.google.no/books?id=T3T8NwAACAAJ*

Westerweele, M. R. & Laurens, J. (2008), *18th European Symposium on Computer Aided Process Engineering*, Vol. 25 of *Computer Aided Chemical Engineering*, Elsevier.
URL: *http://www.sciencedirect.com/science/article/pii/S1570794608801800*

Willems, J. C. (2007), 'The Behavioral Approach to Open and Interconnected Systems', *IEEE Control Systems Magazine* **27**(December), 46–99.

Yang, A. & Marquardt, W. (2004), *European Symposium on Computer-Aided Process Engineering-14, 37th European Symposium of the Working Party on Computer-Aided Process Engineering*, Vol. 18 of *Computer Aided Chemical Engineering*, Elsevier.
URL: *http://www.sciencedirect.com/science/article/pii/S1570794604802591*

Appendix A

Operator definitions code examples

A.1 Add

```
class Add(PhysicalVariable):
    def __init__(self, op, a, b, prec, space):
        '''
        Binary operator
        operator:
        + | - :: units must fit, index structures must fit

        @param op: string:: the operator
        @param a: variable:: left one
        @param b: variable:: right one
        @param prec: precedence
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )

        self.op = op
        self.a = a
        self.b = b
        self.prec = prec

        self.doc = TEMPLATES[op] % (a.symbol, b.symbol)
        self.units = a.units + b.units
        #
        # rule for index structures
        if a.index_structures == b.index_structures:
            self.index_structures = a.index_structures
```

```
        else:
            raise IndexStructureError('add incompatible index structures %s'
                                      % self.doc)

    def __str__(self):
        language = self.space.language
# if not TempVarID.match(self.symbol): return self.symbol

        if self.a.prec < self.prec: self.a = "((%s))" % self.a
        if self.b.prec < self.prec: self.b = "((%s))" % self.b
        LOGGER.info('language %s' % language)
        # if str(self.a) in self.space.equation_alternatives and str(self.b)
             in self.space.equation_alternatives:
        # print('hei')
# print(self.a+'+'+self.b)
        s = CODE[language]['binary'] % (self.a, self.op, self.b)
        return s
```

A.2 Khatri-Rao product

```
class KhatriRao(PhysicalVariable):
    # KhatriRao(op,t,f,2, self.space)
    def __init__(self, op, a, b, prec, space):
        '''
        Binary operator
        operator:
        + | - :: units must fit, index structures must fit
        * :: Khatri-Rao product
        # .index. :: matrix product reducing over the index

        @param op: string:: the operator
        @param a: variable:: left one
        @param b: variable:: right one
        @param prec: precedence

        This is not a universal Khatri-Rao product. This version is limited
            to be on practical form
        and to be usable in current indices
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )
        self.op = op
        self.a = a
        self.b = b
        print ('self.a = ',self.a)
        self.prec = prec
        self.doc = TEMPLATES['*'] % (a.symbol, b.symbol)
        self.units = a.units * b.units

        indexa = [IS.split(' & ') for IS in a.index_structures]
        indexb = [IS.split(' & ') for IS in b.index_structures]

        # HERE ADD HEINZ EXCEPTION PROBLEM
```

```
        chinda,chindb = [t[0] for t in indexa],[t[0] for t in indexb]
        if len(chinda) == 1 and len(chinda) == 1 and chinda != chindb:
            self.index_structures = a.index_structures+b.index_structures
        else:
            for ind in chinda:
                if ind not in chindb:
                    raise IndexStructureError('KhatriRao - index %s in %s not
                          in or not compatible too %s'
                                  % (ind, a.symbol, b.symbol))
            self.index_structures = []
            for aind in indexa:
                for bind in indexb:
                    if aind[0] == bind[0] and len(aind) == len(bind) and aind
                          != bind:
                        # Crash if types are nodes & species and nodes &
                            energy tries to mix
                        raise IndexStructureError('KhatriRao - index %s in %s
                            not in or not found in %s'
                                  % (aind[0], a.symbol, b.symbol ))
                    if aind[0] == bind[0] and len(aind) > len(bind):
                        self.index_structures.append(str(aind[0]+' & '+aind
                            [1]))
                        bind = 'used'
                    elif aind[0] == bind[0] and len(aind) == 1 and len(bind)
                         == 1: # If objects of same type and 1D
                        self.index_structures.append(str(bind[0]))
                        bind = 'used'
                    elif aind[0] == bind[0]:
                        self.index_structures.append(str(bind[0]+' & '+bind
                            [1]))
                        bind = 'used'


    def __str__(self):
        language = self.space.language
        print ('language = ',language)
        if self.a.prec < self.prec: self.a = "((%s))" % self.a
        if self.b.prec < self.prec: self.b = "((%s))" % self.b

        s = CODE[language]['khatriraho'].format(self.a, self.index, self.b)
        return s
```

A.3 Matrix product

```
class ReduceProduct(PhysicalVariable):
    def __init__(self, op, index, a, b, prec, space):
        '''
        Binary operator
        operator:
        .index. :: matrix product reducing over the index

        @param op: string:: the operator
        @param a: variable:: left one
        @param b: variable:: right one
```

```
        @param prec: precedence
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )

        self.op = op
        self.index = index
        self.a = a
        self.b = b
        self.prec = prec

        self.doc = TEMPLATES['.'] % (a.symbol, b.symbol)
        self.units = a.units * b.units
        index = index.strip('|')
        print(a.index_structures, b.index_structures)

        i_a = a.index_structures.index(index) # first occurance
        i_b = b.index_structures.index(index) # first occurance

        if i_a < 0:
            raise IndexStructureError('index structure %s not in the list of
                %s'
                                      % (index, a.doc))
        if i_b < 0:
            raise IndexStructureError('index structure %s not in the list of
                %s'
                                        % (index, b.doc))
        self.index_structures = []
        for i in range(len(a.index_structures)):
            if i != i_a :
                self.index_structures.append(a.index_structures[i])

        for i in range(len(b.index_structures)):
            if i != i_b :
                self.index_structures.append(b.index_structures[i])

    def __str__(self):
        language = self.space.language

        if self.a.prec < self.prec: self.a = "((%s))" % self.a
        if self.b.prec < self.prec: self.b = "((%s))" % self.b

        s = CODE[language]['reducedproduct'].format(self.a, self.index, self
            .b)
        return s
```

A.4 Exponential expressions

```
class Power(PhysicalVariable):
    def __init__(self, op, a, b, prec, space):
        '''
        Binary operator
        operator:
```

```
        ^ :: the exponent,b , must have no units

        @param op: string:: the operator
        @param a: variable:: left one
        @param b: variable:: right one
        @param prec: precedence
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )

        self.op = op
        self.a = a
        self.b = b
        self.prec = prec

        self.doc = TEMPLATES[op] % (a.symbol, b.symbol)

        # units of the exponent, b, must be zero
        if not b.units.isZero():
            raise UnitError('units of the exponent must be zero', a.units, b.
                units)
        self.units = a.units

        # rule for index structures
        self.index_structures = a.index_structures

    def __str__(self):
        language = self.space.language
        if self.a.prec < self.prec: self.a = "((%s))" % self.a
        if self.b.prec < self.prec: self.b = "((%s))" % self.b
        s = str(self.a) + CODE[language]['^'] + str(self.b)
        return s
```

A.5 Implicit equations

```
class Implicit(PhysicalVariable):
    def __init__(self, fct, arg, space):
        # TODO: implement implicit equation this is sqrt.

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )

        self.space = space
        self.args = arg
        self.prec = 99
        self.fct = fct

        self.doc = fct

        # get variable defined as lhs - must appear on the rhs
        # if variable exists -- no worries
        # if not then things are difficult x = ax for example:
```

```
        # what should be the units ? no hands on them neither the indexing.

        self.units = arg.units
        if fct == 'sqrt':
            self.units = copy.copy(arg.units)
            self.units.__pow__(0.5)
            self.doc = 'sqrt(%s)' % arg.doc
        elif fct in UNITARY_NO_UNITS:
            for i in arg.units.asVector():
                if i != 0:
                    raise UnitError('%s expression must have no units'
                                    % fct, arg, '-')

        self.index_structures = arg.index_structures

    def __str__(self):
        return "%s(%s)" % (self.fct, self.args)
```

A.6 Unitary functions

```
class UFunc(PhysicalVariable):
    def __init__(self, fct, arg, space):
        '''
        Unitary functions such as sin cos etc.
        arguments may be an expression, but must have no units
        @param symbol: symbol representing
        @param fct: function name
        TODO: needs some work here such as variable name generated etc
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )

        self.space = space
        self.args = arg
        self.prec = 99
        self.fct = fct

        self.doc = fct # __add__ returns doc string if succ

        self.units = arg.units
        if fct == 'sqrt':
            self.units = copy.copy(arg.units)
            self.units.__pow__(0.5)
            self.doc = 'sqrt(%s)' % arg.doc
        elif fct in UNITARY_NO_UNITS:
            for i in arg.units.asVector():
                if i != 0:
                    raise UnitError('%s expression must have no units'
                                    % fct, arg, '-')

        self.index_structures = arg.index_structures
```

```
    def __str__(self):
        return "%s(%s)" % (self.fct, self.args)

    def getPy(self):
        return self.__str__()
```

A.7 Integral

```
class Integral(PhysicalVariable):
    def __init__(self, y, x, xl, xu, space):
        '''
        implements an integral definition
        @param y: derivative
        @param x: integration variable
        @param xl: lower limit of integration variable
        @param xu: upper limit of integration variable
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )
        # why like this: physical units is the problem.
        self.y = y
        self.x = x
        self.xl = xl
        self.xu = xu
        self.space = space
        self.prec = 99

        xunits = Units.asVector(x.units)
        yunits = Units.asVector(y.units)
        units = [xunits[i]+yunits[i] for i in range(len(yunits))]
        self.units = units
# self.symbol = self.space.newAutoVar()


    def __str__(self):
        language = self.space.language
        LOGGER.info('language %s, variables y,x,xl,xu:%s, %s, %s, %s'
                    % (language, self.y, self.x, self.xl, self.xu))
        t_y = self.space.translate(self.y, language)
        t_x = self.space.translate(self.x, language)
        t_xl = self.space.translate(self.xl, language)
        t_xu = self.space.translate(self.xu, language)
        LOGGER.info('language %s, variables y,x,xl,xu:%s, %s, %s, %s'
                    % (language, t_y, t_x, t_xl, t_xu))

        y = CODE[language]['var'] % t_y
        x = CODE[language]['var'] % t_x
        xl = CODE[language]['var'] % t_xl
        xu = CODE[language]['var'] % t_xu

        s = CODE[self.space.language]['integral'].format(y, x, xl, xu)
            # (self.y, self.x, self.xl, self.xu)
```

```
        return s
```

A.8 Total differential

```
class TotDifferential(PhysicalVariable):
    def __init__(self, x, y, space):
        '''
        implements total differential definition
        @param x: dx
        @param y: dy
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )
        self.x = x
        self.y = y
        self.space = space
        self.prec = 99
#       self.symbol = self.space.newAutoVar()
        xunits = Units.asVector(x.units)
        yunits = Units.asVector(y.units)
        units = [xunits[i]-yunits[i] for i in range(len(yunits))]
        self.units = units

    def __str__(self):
        language = self.space.language
        x = CODE[language]['var'] % self.space.translate(self.x, language)
        y = CODE[language]['var'] % self.space.translate(self.y, language)
        return CODE[self.space.language]['Diff'] % (x, y)
#       return CODE[self.space.language]['Diff'] % (self.x, self.y)
```

A.9 Inverse

```
class Inverse(PhysicalVariable):
    def __init__(self, x, space):
        '''
        implements inverse matrix representation
        @param x: 1/x
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )
        self.x = x
        self.space = space
        self.prec = 99
#       self.symbol = self.space.newAutoVar()
        xunits = Units.asVector(x.units)
        units = [-1*xunits[i] for i in range(len(xunits))]
        self.units = units

    def __str__(self):
```

```
        language = self.space.language
        x = CODE[language]['var'] % self.space.translate(self.x, language)
        return CODE[self.space.language]['Inv'] % (x)
```

A.10 Partial differential

```
class ParDifferential(PhysicalVariable):
    def __init__(self, x, y, space):
        '''
        implements partial differential definition
        @param x: dx
        @param y: dy
        '''

        symbol = space.newTempPhysicalVariableName()
        PhysicalVariable.__init__(self, space, symbol, equation_rhs=EMPTY_EQ
            )

        self.x = x
        self.y = y
        self.space = space
        self.prec = 99

        xunits = Units.asVector(x.units)
        yunits = Units.asVector(y.units)
        units = [xunits[i]-yunits[i] for i in range(len(yunits))]
        self.units = units
#       self.symbol = self.space.newAutoVar()

    def __str__(self):
        language = self.space.language
        x = CODE[language]['var'] % self.space.translate(self.x, language)
        y = CODE[language]['var'] % self.space.translate(self.y, language)
        return CODE[self.space.language]['diff'] % (x, y)
#       return CODE[self.space.language]['diff'] % (self.x, self.y)
```

Appendix B

Physical variable space

Included in this appendix is the entire variable space for the case study presented in chapter 6.

Definition B.1 – Physical variable space

```
[t]
var_type = frame
doc = time
index_structures = ['nil']
equations = [('time', '_empty_', 'Eq000')]
units = [1, 0, 0, 0, 0, 0, 0, 0]

[geometrical space]
var_type = frame
doc = geometrical space
index_structures = ['nil']
equations = [('geometrical space', '_empty_', 'Eq001')]
units = [0, 1, 0, 0, 0, 0, 0, 0]

[m]
var_type = physical_state
doc = mass
index_structures = ['nodes & mass']
equations = [('mass', '_empty_', 'Eq002')]
units = [0, 0, 0, 1, 0, 0, 0, 0]

[n]
var_type = physical_state
doc = species
index_structures = ['nodes & species']
equations = [('species', '_empty_', 'Eq003')]
units = [0, 0, 1, 0, 0, 0, 0, 0]

[U]
```

```
var_type = physical_state
doc = Internal energy
index_structures = ['nodes & energy']
equations = [('Internal energy', '_empty_', 'Eq004')]
units = [-2, 2, 0, 1, 0, 0, 0, 0]

[F]
var_type = incidence_matrix
doc = Incidence matrix for physical system
index_structures = ['nodes','arcs']
equations = [('Incidence matrix', '_empty_', 'Eq006')]
units = [0, 0, 0, 0, 0, 0, 0, 0]

[S]
var_type = incidence_matrix
doc = Species matrix for the system
index_structures = ['nodes & species','arcs & species']
equations = [('Incidence matrix', '_empty_', 'Eq006')]
units = [0, 0, 0, 0, 0, 0, 0, 0]

[F_S]
var_type = None
doc = F * S
index_structures = ['nodes & species', 'arcs & species']
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Species block by block matrix', 'F*S', 'Eq027')]

[F_e]
doc = Incidence matrix for energy physical system
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Incidence matrix for energy physical system', '_empty_',
    'Eq005')]
var_type = incidence_matrix
index_structures = ['nodes', 'arcs']

[F_w]
doc = Incidence matrix for work physical system
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Incidence matrix for work physical system', '_empty_', '
    Eq006')]
var_type = incidence_matrix
index_structures = ['nodes', 'arcs']

[N]
var_type = incidence_matrix
doc = Reaction matrix
index_structures = ['nodes & species', 'nodes']
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('Reaction matrix', '_empty_', 'Eq018')]

[r]
var_type = incidence_matrix
doc = kinetics
index_structures = ['nodes & species', 'nodes']
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('kinetics', '_empty_', 'Eq019')]
```

```
[Beta2]
doc = Squared valve constant
units = [0, 7, 0, -1, 0, 0, 0, 0]
equations = [('Squared valve constant', '_new_', 'Eq023')]
var_type = physical_parameter
index_structures = ['arcs', 'nodes']

[Beta]
doc = sqrt(Squared valve constant)
units = [0.0, 3.5, 0.0, -0.5, 0.0, 0.0, 0.0, 0]
equations = [('Valve constant', 'sqrt(Beta2)', 'Eq024')]
var_type = physical_secondary state
index_structures = ['arcs', 'nodes']

[R]
doc = Gas constant
units = [-2, 2, -1, 1, -1, 0, 0, 0]
equations = [('Gas constant', '_new_', 'Eq010')]
var_type = physical_parameter
index_structures = ['nil']

[Cp]
doc = Heat capacity, constant pressure
units = [-2, 2, -1, 1, -1, 0, 0, 0]
equations = [('Heat capacity, constant pressure', '_new_', 'Eq011')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[D]
doc = Diffusivity Liquid to boundary
units = [1, -2, 2, -1, 0, 0, 0, 0]
equations = [('Diffusivity constant', '_new_', 'Eq012')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[T_0]
doc = Standard temperature
units = [0, 0, 0, 0, 1, 0, 0, 0]
equations = [('Standard temperature', '_new_', 'Eq013')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[k]
doc = Heat diffusion constant
units = [-1, 0, 1, 0, 0, 0, 0, 0]
equations = [('Heat diffusion constant', '_new_', 'Eq015')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[deltah_0]
doc = Enthalpy STP
units = [-2, 2, -1, 1, 0, 0, 0, 0]
equations = [('Enthalpy STP', '_new_', 'Eq016')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[s_0]
```

```
doc = entropy STP
units = [-2, 2, -1, 1, -1, 0, 0, 0]
equations = [('entropy STP', '_new_', 'Eq017')]
var_type = physical_parameter
index_structures = ['nodes','arcs']

[1]
doc = 1 numerical constant
units = [0, 0, 0, 0, 0, 0, 0, 0]
equations = [('-1 numerical constant', '_new_', 'Eq018')]
var_type = physical_parameter
index_structures = ['nil']

[mu]
doc = Chemical potential
units = [-2, 2, -1, 1, 0, 0, 0, 0]
equations = [('Chemical Potential', 'diff{U,n}', 'Eq009')]
var_type = physical_secondary state
index_structures = ['nodes']

[T]
doc = root
units = [0, 0, 0, 0, 1, 0, 0, 0]
equations = [('Temperature', 'root(U)', 'Eq014')]
var_type = physical_secondary state
index_structures = ['nodes']

[p]
doc = root
units = [-2, -1, 0, 1, 0, 0, 0, 0]
equations = [('Pressure', 'root(U)', 'Eq025')]
var_type = physical_secondary state
index_structures = ['nodes']

[n_dot]
doc = F_S . r
units = [-1, 0, 1, 0, 0, 0, 0, 0]
equations = [('Mass balance', 'F_S.|arcs|.n_hat+N.|arcs|.r', 'Eq028')]
var_type = physical_secondary state
index_structures = ['nodes']

[V]
doc = root
units = [0, 3, 0, 0, 0, 0, 0, 0]
equations = [('Volume', 'root(U)', 'Eq029')]
var_type = physical_secondary state
index_structures = ['nodes']

[h]
doc = deltah_0 + _V1
units = [-2, 2, -1, 1, 0, 0, 0, 0]
equations = [('Enhalpi', 'deltah_0+Cp.|arcs|.(T-T_0)', 'Eq030')]
var_type = physical_secondary state
index_structures = ['nodes']

[U_dot]
doc = F . q_hat
```

```
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Energy balance', 'F.|arcs|.H_hat+F_e.|arcs|.q_hat', '
    Eq032')]
var_type = physical_secondary state
index_structures = ['nodes']

[c]
doc = root
equations = [('consentration, volumentric', 'root(n)', 'Eq055')]
units = [0, 0, 1, 0, 0, 0, 0, 0]
var_type = physical_secondary state
index_structures = ['nodes']

[n_hat]
doc = D . _V0
equations = [('Mass diffusion', 'D.|nodes|.(mu-mu)', 'Eq022'),('Mass
    advection', 'c.|nodes|.V_hat', 'Eq040')]
units = [-1, 0, 1, 0, 0, 0, 0, 0]
var_type = physical_transport
index_structures = ['arcs']

[V_hat]
doc = Beta . _V1
units = [-1.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0]
equations = [('Volumentric flow, advection', '-Beta.|nodes|.sqrt(p-p)',
    'Eq026')]
var_type = physical_transport
index_structures = ['arcs']

[q_hat]
doc = k . _V0
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Heat transport diffusion', 'k.|nodes|.(mu-mu)', 'Eq027')]
var_type = physical_transport
index_structures = ['arcs']

[H_hat]
doc = h . n_hat
units = [-3, 2, 0, 1, 0, 0, 0, 0]
equations = [('Energy trasport', 'h.|nodes|.n_hat', 'Eq031')]
var_type = physical_transport
index_structures = ['arcs']
```

Appendix C

Interaction variable space

Included in this appendix is the entire interaction variable space for the case study presented in chapter 6.

Definition C.1 – Interaction variable space

```
[p]
doc = root
units = [-2, -1, 0, 1, 0, 0, 0, 0]
equations = [('Pressure', 'root(U)', 'Eq040')]
var_type = physical_secondary state
index_structures = ['nodes']

[V]
doc = root
units = [0, 3, 0, 0, 0, 0, 0, 0]
equations = [('Volume', 'root(U)', 'Eq041')]
var_type = physical_secondary state
index_structures = ['nodes']

[A]
doc = Area of liquid phase
units = [0, 2, 0, 0, 0, 0, 0, 0]
equations = [('Area', 'root(U)', 'Eq042')]
var_type = parameter
index_structures = ['nodes']

[l]
doc = root
units = [0, 1, 0, 0, 0, 0, 0, 0]
equations = [('level', 'root(V)', 'Eq043')]
var_type = physical_secondary state
index_structures = ['nodes']

[Beta]
```

```
doc = sqrt(Squared valve constant)
units = [0.0, 3.5, 0.0, -0.5, 0.0, 0.0, 0.0, 0]
equations = [('Valve constant', 'sqrt(Beta2)', 'Eq024')]
var_type = physical_secondary state
index_structures = ['arcs', 'nodes']
```