

TONY HÜRLIMANN

COMPUTER-BASED MATHEMATICAL MODELING

**An Essay for the Design
of Computer-Based Modeling Tools**

COMPUTER-BASED MATHEMATICAL MODELING

**An Essay for the Design
of Computer-Based Modeling Tools**

TONY HÜRLIMANN

Institute of Informatics

Site Regina Mundi

University of Fribourg

Computer-Based Mathematical Modeling

An Essay for the Design of Computer-Based Modeling Tools

Von der Wirtschafts- und Sozialwissenschaftlichen Fakultät der
Universität Freiburg genehmigte Habilitationsschrift zur Erlangung der
Venia Legendi für das Fach "Informatik".

Vorgelegt von
Dr. rer. pol. Tony Hürlimann
aus Walchwil/ZG

Gutachter: Prof. Dr. Jacques Pasquier,

Institut für Informatik, Universität Freiburg, 1700 Freiburg, Schweiz

Zweiter Gutachter: Prof. Dr. Johannes J. Bisschop,

Faculteit der Toegepaste Wiskunde, Universiteit Twente, 7500 AE Enschede, The Netherlands

Habitationsabgabe: 21. Juli 1997

Annahme der Habilitation: 12. Februar 1998

Denna bok är tillägnad
min hustru Kajsa
och mina döttrar Malika och Selma

Preface

Computer-based mathematical modeling – the technique of representing and managing models in machine-readable form – is still in its infancy despite the many powerful mathematical software packages already available which can solve astonishingly complex and large models. On the one hand, using mathematical and logical notation, we can formulate models which cannot be solved by any computer in reasonable time – or which cannot even be solved by *any* method. On the other hand, we can *solve* certain classes of much larger models than we can practically *handle* and manipulate without heavy programming. This is especially true in operations research where it is common to solve models with many thousands of variables. Even today, there are no general modeling tools that accompany the whole modeling process from start to finish, that is to say, from model creation to report writing.

This book proposes a framework for computer-based modeling. More precisely, it puts forward a modeling language as a kernel representation for mathematical models. It presents a general specification for modeling tools. The book does *not* expose any solution methods or algorithms which may be useful in solving models, neither is it a treatise on how to build them. No help is intended here for the modeler by giving practical modeling exercises, although several models will be presented in order to illustrate the framework. Nevertheless, a short introduction to the modeling process is given in order to expound the necessary background for the proposed modeling framework.

Therefore, this work is not primarily intended for the model user or the modeler – the mathematician, physicist, or operation research model builder who use software to solve a given problem. Ultimately, this research is done for *them*, of course, and I hope I have created and implemented a useful software tool (LPL) to this end – at least for educational purposes.

Nor is this book about implementation topics. I do not describe here how I have concretely implemented my own modeling tools. This would have considerably augmented the number of pages and it would have been of scarce interest, since there are many excellent books on compiler construction

already available.

This work offers *concepts* and a *general framework for modeling* and is mainly intended to help other designers of modeling tools. It details my views on ‘why and how’ we should create such tools and which utilities and functionalities they should contain in order to be useful for “practitioners”. In that sense, this work is more a research programme than the presentation of an already established domain. I think that we are only at the beginning of a new development in “modeling with the aid of computers” and that I am able to barely scratch the surface of this new and exciting research field. I am, however, firmly convinced that there is a bright future for this topic – just as there was for word processors and spreadsheets fifteen years ago; or just as there was forty years ago when the first high level programming language was developed.

To grasp the main ideas, the reader should have some background in applied mathematics, logic, operations research, and computer science, as I rarely explain basic notions from these disciplines (like, for example, the concepts of “computational complexity” from the theoretical computer science, “cutting planes” for the polyhedral theory, and others), although these notions are fundamental to this book. I assume that the reader is familiar with these concepts. However, further references are always given.

My own background is, in order of importance, computer science, operations research and economics. This ordering also corresponds to the weight the three fields take in this book. Developing *ideas* merely on how modeling tools should be implemented is either easy or too far away from what is really needed; at least it is only a small part of the task as a whole to elaborate on all kinds of neat concepts without actually going through the thorny work of implementation. Only the process of implementation teaches you what works and what does not work.

My motivation for this research came initially from practical modeling: a research group under the direction of Dr. Hättenschwiler and Prof. Kohlas at the Institute of Informatics of the University of Fribourg (IIUF) had to build and maintain – at that time – large LP models. Although I was never directly engaged in the management of these models, I followed closely what they did

and what they needed, and I created a first version of LPL (Linear Programming Language) which enabled them to formulate their models better. That was 1987 and LPL was in its infancy. Nevertheless I got a Ph.D. for it and it encouraged me to extend and generalize it in several directions. Since then I have worked more or less intensely on the language. This book is the result of this “project LPL”.

The book is made up of three parts. In Part I, the concept of *model* and related notions are defined, the modeling life cycle is outlined, and different model types and paradigms are presented. Part II gives an overview of what actually exists in the line of computer-based modeling tools, explains why we need more, and presents a general modeling framework. Part III describes my own concrete contribution to this field: the modeling language LPL. A more complete survey of this book is given at the end of the *Introduction*.

Acknowledgements

There are many people who have directly or indirectly participated in the successful completion of this book. I particularly wish to thank the individuals who were invaluable in the process of making this work a reality:

Prof. Dr. Jürg Kohlas was the initiator of LPL. He proposed to me to develop a declarative language which would allow us to formulate concisely large LP models. Figure 0-1 is a sketch he made during a conference in the Swiss Alps (3ème Cycle d'Informatique, 1985). He has also supported and still supports actively my research since then.

The image shows a handwritten sketch of LPL syntax. It is enclosed in a rectangular box. The text is written in a cursive, handwritten style. It starts with 'In class' followed by 'Variables' and 'Restrictions'. Under 'Variables', it lists 'X(J), J FROM 1 TO N STEP 1' and 'Y(J), J FROM 1 TO N'. Under 'Restrictions', it lists 'BALANCE(K), K FROM 1 TO N'. Below this, it says 'INCIDENCE' and then 'BALANCE(K): CONTAINS X(J); ALL OF X(.) WITH COEFF. A; Y(K)'. There are some corrections and scribbles in the original image, such as a crossed-out 'K' and a 'J' above 'K' in the incidence line.

Figure 0-1: The Birth of LPL (Kohlas J.)

Prof. Dr. Ambros Lüthi was always around in the most critical moments of my life as a researcher. He gave me invaluable advice and supported my research project actively. Without him my career as a researcher would never have begun and would have ended quite some time ago.

Prof. Dr. Jacques Pasquier is not only the supervisor of my habilitation, he was actively engaged in the modeling management from the hypertext and software engineering point of view. As a supervisor, he did a great job and read earlier drafts of this manuscript several times and gave me many invaluable tips. He also eagerly supported my project and has thus allowed me

to pursue my research an additional two years.

Prof. Dr. Pius Hättenschwiler, the project manager of several large LPs, was and is still an intense interlocutor when it come to the practical use of LPL. Without him, LPL would be quite different or would probably even vegetate as a theoretical tool without much practical use. Marco Moresino, working under the direction of Pius, was and is probably the most intense and fiercest user of LPL. He found many subtle bugs.

Rare are the people in one's life who – when you meet them for the first time – seem to be like old friends who you have known years. Prof. Dr. Johannes Bisschop from the University Twente of Enschede is one of thee people for me. This is probably because we share a similar dream of a modeling system. Not only is he the second referee for my habilitation, but he also encouraged me to continue with my small (compared to the titans available on the markets) LPL implementation project when I was at the point of abandoning the whole thing (in Budapest 1994). A seemingly minor event might be of special interest in this context. When I first met Johannes, I asked him what he thought about the expressive power of modeling languages, whether they should include the whole power of a Turing Machine or not. His answer was a simple yes, and had an inestimable influence on what I now think should be a modeling language (the answer is in Chapter 7).

Prof. Dr. Arthur Geoffrion of UCLA allowed me to stay with him for one year. I owe him a great deal of credit when it comes to many of my ideas in modeling management systems.

Daniel Raemi, my old friend, has given extravagantly of his time to read earlier drafts of this book. His many corrections and suggestions have been very valuable. I also wish to thank Simon Lacey for his help in proof-reading the finished manuscript. Mitra Packham did a excellent job in reading the finished manuscript and greatly improved my English.

I also owe a depth of gratitude to many other colleagues and teachers from whom I have learned so much over the years. I thank them all.

Finally, I thank my wife Kajsa for her love and patience when I was busy on writing at evenings and weekends while we could have spent enjoyable hours together with our daughter Malika. I hope to make amends.

None of the mentioned (or not mentioned) persons, however, is responsible for the defects in what follows. I alone am to blame for all and any errors which may remain or for the type setting of this document.¹

Finally, I wish to thank the following institutions for their financial support and the use of their infrastructure. The first is the Swiss National Science Foundation who financed my stay at UCLA in Los Angeles with Prof. Arthur Geoffrion for one year in 1989, and still supports my research (under Project No. 1217-45922.95). The second is the Institute of Informatics at the University of Fribourg in Switzerland (IIUF) where I developed and implemented most of my ideas in a favourable and agreeable atmosphere of collaboration. The Institute also let me use all the necessary equipment over many years to accomplish this work.

*Tony Hürlimann
Freiburg, Switzerland
July 1997*

¹ This document was written on different Macintoshes using the word processor of MSWord 5.0 (not 6.0 – which I never will use!), trademarked by MicroSoft. With exception of the formula editor, I was quite happy with this software, but I would use Latex2e if I had to begin again.

Availability of the LPL System

Part III of this book presents software called **LPL**.² LPL is my own contribution to the field of computer-based mathematical modeling and is available free of charge from the Institute of Informatics at the University of Fribourg (IIUF), Switzerland. Currently, there are implementations on MS/DOS, Windows 95 and NT and Macintosh PowerPC. The version used in this book is 4.25.

LPL – the software – together with documentation containing the Reference Manual, several papers and many models written in LPL syntax, can be obtained via anonymous Internet file transfer *ftp*. The address is:

ftp-iiuf.unifr.ch/pub/lpl

In a WWW browser, one can type the following URL address to get to this source (called the *LPL-site*):

ftp://ftp-iiuf.unifr.ch/pub/lpl

The LPL-site – documentation and software – is updated periodically. For further information or suggestions, the author can be contacted by email at

tony.huerlimann@unifr.ch

Using a WWW browser, the author's homepage can be found at the following URL address:

http://www2-iiuf.unifr.ch/tcs/tlpl/

From this link, the reader can also find the software of LPL and instructions on how to obtain it from the ftp server.

² Initially, **LPL** was an abbreviation of *Linear Programming Language*, since it was designed exclusively for Linear Programs. During the intervening years, LPL's capability in logical modeling has become so important that one could also call it *Logical Programming Language*. My intention, however, is to offer a tool for full model documentation too. Therefore, in the futur it may be called *Literate Programming Language* (see [Knuth 1984]).

Table of Contents

1. Introduction	1
1.1. Models and their Functions	2
1.2. The Advent of the Computer	5
1.3. New Scientific Branches Emerge	6
1.4. Mathematical Modeling – the Consequences	10
1.5. Computer-Based Modeling Management	13
1.6. About this Book	15
PART I: FOUNDATIONS OF MODELING	17
2. What is Modeling?	19
2.1. Model: a Definition	19
2.1. Mathematical Models	27
2.2. Model Theory	30
2.3. Models and Interpretations	33
2.4. Related Concepts	35
2.5. Declarative versus Procedural Knowledge	38
3. The Modeling Life Cycle	43
3.1. Stage 1: Specification of the Real Problem	45
3.2. Stage 2: Formulation of the Mathematical Model	46
3.3. Stage 3: Solution of the Model	51
3.4. Stage 4: Validation of the Model and its Solution	52
3.4.1. Logical Consistency	58
3.4.2. Data Type Consistency	59
3.4.3. Unit Type Consistency	60
3.4.4. User Defined Data Checking	61
3.4.5. Simplicity Considerations	61
3.4.6. Solvability Checking	62
3.4.7. Numerical Stability and Sensitivity Analysis	62
3.4.8. Checking the Correspondence	64
3.5. Stage 5: Writing a Report	66
3.6. Two Case Studies	67
4. Model Paradigms	77
4.1. Model Types	78

4.1.1. Optimization Models	78
4.1.2. Symbolical — Numerical Models	78
4.1.3. Linear — Nonlinear Models	79
4.1.4. Continuous — Discrete Models	79
4.1.5. Deterministic — Stochastic Models	80
4.1.6. Analytic — Simulation Models	81
4.2. Models and their Purposes	82
4.3. Models in their Research Communities	84
4.3.1. Differential Equation Models	84
4.3.2. Operations Research	84
4.3.3. Artificial Intelligence	87
4.3.3.1. Search Techniques	88
4.3.3.2. Heuristics	91
4.3.3.3. Knowledge Representation	91
4.4. Modeling Uncertainty	92
4.4.1. Mathematical Models and Uncertainty	93
4.4.2. General Approaches in Modeling Uncertainty	95
4.4.3. Classical Approaches in OR	97
4.4.4. Approaches in Logical Models	100
4.4.5. Fuzzy Set Modeling	107
4.4.6. Outlook	111

PART II: A GENERAL MODELING FRAMEWORK 113

5. Problems and Concepts 115

5.1. Present Situation in MMS	116
5.2. What MMS is not	117
5.3. MMS, what for?	121
5.4. Models versus Programs	124

6. An Overview of Approaches 131

6.1. Spreadsheet	131
6.2. Relational Database Systems	136
6.3. Graphical Modeling	143
6.4. Constraint Logic Programming Languages (CLP)	149
6.5. Algebraic Languages	158
6.5.1. AIMMS	160
6.5.2. AMPL	166
6.5.3. Summary	169
6.6. General Remarks	171
6.6.1. Structured Modeling	172
6.6.2. Embedded Language Technique	174
6.6.3. Multi-view Architecture	174
6.6.4. A Model Construction and Browsing Tool	175

6.6.5. Conclusion	176
7. A Modeling Framework	177
7.1. The Requirements Catalogue	178
7.1.1. Declarative and Procedural Knowledge	178
7.1.2. The Modeling Environment	180
7.1.3. Informal Knowledge	182
7.1.4. Summary	183
7.2. The Modeling Language	184
7.2.1. The Adopted Approach	186
7.2.2. The Overall Structure of the Modeling Language	192
7.2.3. Entities and Attributes	198
7.2.4. Index-sets	203
7.2.5. Expression	210
7.2.6. The Instruction Entities	212
7.2.7. The Complete Syntax Specification	213
7.2.8. Semantic Interpretation	215
7.2.8. Summary	217
7.3. Four Examples	218
7.4. Modeling Tools	230
7.4.1. A Textual-Based Tool	231
7.4.2. A Tool Based on Graphs	233
7.5. Outlook	234
PART III: LPL – AN IMPLEMENTED FRAMEWORK	237
8. The Definition of the Language	239
8.1. Introduction	239
8.2. An Overview of the LPL-Language	240
8.2.1. The Entities and the Attributes	240
8.2.2. Index-Sets	243
8.2.3. Data	244
8.2.3.1. LPL's own Data Format	244
8.2.3.2. The Import Generator	247
8.2.3.2. The Report Generator	249
8.2.4. Expressions	250
8.2.5. Logical Modeling	251
8.2.5.1. Predicate Variables	254
8.2.5.2. Logical Constraints	255
8.2.5.3. Proceeding the T1–T6 rules	257
8.3. The Backus-Naur Specification of LPL (4.25)	264
9. The Implementation	267
9.1. The Kernel	268

9.2. The Environment (User Interface)	269
9.3. The Text Browser	270
9.4. The Graphical Browser	275
10. Selected Applications	283
10.1. General LP-, MIP-, and QP-Models	283
10.2. Goal Programming	294
10.3. LP's with logical constraints	297
10.5. Problems with Discontinuous Functions	316
10.6. Modeling Uncertainty	318
11. Conclusion	325
 APPENDICES	 331
 References	 333
 Glossary	 353
 Index	 359

Model Examples

Example 2-1: The Intersection Problem	22
Example 3-1: The Frustum Problem	47
Example 3-2: Theory of Learning	49
Example 3-3: The 3-jug Problem	56
Example 3-4: Cooling a Thermometer	68
Example 3-5: A Production Problem	69
Example 4-1: A Letter Game Problem	89
Example 4-2: A PSAT Problem	102
Example 4-3: Probabilistic Entailment	103
Example 4-4: PSAT versus ATMS	105
Example 4-5: Modeling Dynamic Systems	109
Example 4-6: Fuzzy LPs	111
Example 5-1: An Energy Import Problem	128
Example 6-1: A Portfolio Problem	132
Example 6-2: A Transshipment-Assignment Problem	137
Example 6-3: The Transportation Problem	145
Example 6-4: The Car Sequencing Problem	155
Example 6-5: The Cutting Stock Problem	160
Example 6-6: The Diet Problem	167
Example 7-1: The n-Queen Problem	218
Example 7-2: The Cutting Stock Problem (again)	221
Example 7-3: The n-Bit-Adder	223
Example 7-4: A Budget Allocation Problem	228
Example 10-1: Determine Workforce Level	283
Example 10-2: "Rhythmed" Flow-Shop	287
Example 10-3: Assignment of Players to Teams	290
Example 10-4: Portfolio Investment	293
Example 10-5: Book Acquirements for a Library	294
Example 10-6: The Intersection Problem	298
Example 10-7: A Satisfiability Problem (SAT)	302
Example 10-8: How to Assemble a Radio	304
Example 10-9: An Energy Import Problem	305
Example 10-10: Finding Magic Squares	311
Example 10-11: The Capacitated Facility Location Problem	313
Example 10-12: A Two-Persons Zero-Sum Game	316
Example 10.13. A Stochastic Model	318
Example 10.13. A Model Using Fuzzy-sets	320

Figures

Figure 0-1: The Birth of LPL (Kohlas J.)	IV
Figure 2-1: El torro	21
Figure 2-2: The Intersection Problem	22
Figure 2-3: Topological Deformation	22
Figure 2-4: Solution to the Intersection Problem	23
Figure 2-5: The Intersection Problem, an Interpretation	27
Figure 2-6: Similarities between Theories	34
Figure 2-7: Model Structure versus Model Instance	38
Figure 3-1 : The Model Life Cycle	44
Figure 3-2: The Frustum Problem	48
Figure 3-3: The Frustrum Problem, an Intermediate Step	48
Figure 3-4: The Frustrum Problem, the Solution	48
Figure 3-5: A Learning Model based on Markov Chain	50
Figure 3-6: Validation of a Model	53
Figure 3-7: The 3-jug Problem	56
Figure 3-8: The 3-jug Problem, the Search Path	57
Figure 3-9: Sensitivity Analysis	63
Figure 3-10: The Correspondence Problem	65
Figure 3-11: The Temperature Plot	69
Figure 4-1: Different Optimizing Criteria	85
Figure 4-2: The Search Path of the Letter Game Problem	90
Figure 4-3: Fuzzy Sets for small, medium, and tall	109
Figure 4-4: The Inverted Pendulum	110
Figure 5-1: Different Model Representations	123
Figure 5-2: Different Representations with many Links	123
Figure 6-1: A Plot of the Reinvestment Flow	133
Figure 6-2: A Spreadsheet for the Portfolio Model	134
Figure 6-3: Product Flow in the Transshipment-Assignment Problem	139
Figure 6-4: Netform of a Transportation Model	147
Figure 6-5: Aggregated Netform for the Transportation Model	148
Figure 6-6: Multi-view Architecture	175
Figure 7-1: An Architecture for Modeling Tools	184
Figure 7-2: Modeling Language Embedding	185
Figure 7-3: An Index-tree	205
Figure 7-4: An Index-tree with Collapsing Paths	206
Figure 7-5: An Index-tree Viewed as a Compound Index-set	206
Figure 7-6: A tagged Index-tree	207
Figure 7-7: A Marked Index-tree	209
Figure 7-8: A Solution for the 4- and the 8-Queen Problem	219
Figure 7-9: The n-Bit-Adder	223
Figure 7-10: AND-, OR-, NOT-Gates	223

Figure 7-11: The XOR-Gate	224
Figure 7-12: A Half-Adder	224
Figure 7-13: A Full-Adder	225
Figure 7-14: A 3-bit Adder	226
Figure 7-15: Architecture of a Text Browser/Editor	232
Figure 9-1: Overall Architecture	267
Figure 9-2: The LPL Environment	270
Figure 9-3: LPL's text browser, a Variable Entity	274
Figure 9-4: The Text Browser, a Parameter Entity	275
Figure 9-5: The A-C Graph of EGYPT.LPL	276
Figure 9-6: The Value-dependency Graph of EGYPT.LPL	278
Figure 9-7: The Index-dependency Graph of EGYPT.LPL	280
Figure 10-1: The Intersection Problem	298
Figure 10-2: Initial Assignment of Colours	298
Table 10-5: The Intersection Model	299
Figure 10-3: Crossing Paths	300
Figure 10-4: A Solution to the Intersection Problem	301

Tables

Table 3-1: A Production Problem	71
Table 5-1: The Energy Import Problem	129
Table 6-1: A Portfolio Model	132
Table 6-2: A Model for the Transshipment-Assignment Problem	138
Table 6-3: A Model for the Transportation Problem	146
Table 6-4: A Model for the Car Sequencing Problem	157
Table 6-5: A Model for the Declarative Part of the Cutting Stock Problem	161
Table 6-6: An Algorithm for the Column Generation	162
Table 7-1: The n-Queen (declarative) Model	219
Table 7-2: The Half-Adder Model	225
Table 7-3: The Full-Adder Model	225
Table 7-4: The n-Bit-Adder Model	226
Table 7-5: The Budget Allocation Model	229
Table 8-1: Logical Operators in LPL	252
Table 8-2: Binary Boolean Connectors	253
Table 8-2: Algorithm for Transforming a Logical Constraint	256
Table 8-3: T1-rules: Eliminate Operators at Parse-Time	256
Table 8-4: T2-rules: Eliminate Operators at Evaluation-Time	257
Table 8-5: T3-rules: Push the NOT Operator	258
Table 8-6: T4-rules: Pull ANDs outwards	260
Table 8-7: Parent/Child Pair for a Detachable Subexpression	261
Table 8-8: T6-rules	263
Table 10-1: The Workforce Level Model	285
Table 10-2: The “Rhythmed” Flow-Shop Model	288
Table 10-3: The Portfolio Model	293
Table 10-4: Acquisitions for a Library, a Model	295
Table 10-6: A Small SAT Model	303
Table 10-7: The Radio Model	305
Table 10-8: The Magic-Square Model	312
Table 10-9: The Capacitated Facility Location Model	313
Table 10-10: A 2-Persons-Zero-Sum Game	317
Table 10-11: A Stochastic Model	319

1. INTRODUCTION

“Dass alle unsere Erkenntnis mit der Erfahrung anfangt, daran ist gar kein Zweifel.
... Wenn aber gleich alle unsere Erkenntnis *mit* der Erfahrung anhebt, so entspringt sie darum
doch nicht eben alle *aus* der Erfahrung.”

— Kant I., Kritik der reinen Vernunft, 1877.

“Model-building is the essence of the operations research approach.”

— Wagner H.M., Principles of Operations Research, 1975.

“Today mathematicians are generally part of a project team – as such they develop expertise
about the process of system being analysed rather than act merely as solvers of mathematical
equations.”

— Cross/Moscardini, 1985.

Observation is the ultimate basis for our understanding of the world around us. But observation alone only gives information about particular events; it provides little help for dealing with new situations. Our ability and aptitude to recognize similarities in different events, to distil the important factors for a specific purpose, and to generalize our experience enables us to operate effectively in new environments. The result of this skill is *knowledge*, an essential resource for any intelligent agent.

Knowledge varies in sophistication from simple classification to understanding and comes in the form of principles and models. A *principle* is simply a general assertion and is expressed in a variety of ways ranging from saws to equations. «Virtue is its own reward», and « $E=mc^2$ » are examples of principles. They can vary in their validity and their precision. A *model* is, roughly speaking, an analogy for a certain object, process, or phenomenon of interest. It is used to explain, to predict, or to control an event or a process. For example, a miniature replica of a car, placed in a wind tunnel, allows us to

predict the air resistance or air eddy of a real car; a globe of the world allows us to estimate distances between locations; a graph consisting of nodes (corresponding to locations) and edges (corresponding to streets between the locations) enables us to find the shortest path between any two locations without actually driving between them; and a differential equation system enables us to balance an inverted pendulum by calculating at short intervals the speed and direction of the car on which the pendulum is fixed.

A model is a powerful means of structuring knowledge, of presenting information in an easily assimilated and concise form, of providing a convenient method for performing certain computations, of investigating and predicting new events. The ultimate goal is to make decisions, to control our environment, to predict events, or just to explain a phenomenon.

1.1. Models and their Functions

Models can be classified in several ways. Their characteristics vary according to different dimensions: function, explicitness, relevance, formalization. They are used in scientific theories or in a more pragmatic context. Here are some examples classified by function, but also varying in other aspects. They illustrate the countless multitude of models and their importance in our life.

Models can *explain phenomena*. Einstein's special relativity explains the Michelson–Morley experiment of 1887 in a marvellously simple way and overruled the ether model in physics. Economists introduced the IS-LM or rational expectation models to describe a macroeconomical equilibrium. Biologists build mathematical growth models to explain and describe the development of populations. Modern cosmologists use the big-bang model to explain the origin of our world, etc.

There are also models *to control our environment*. A human operator, e.g., controls the heat process in a kiln by opening and closing several valves. He or she knows how to do this thanks to a learned pattern (model); this pattern could be formulated as a list of instructions as follows: “IF the flame is bluish at the entry port, THEN open valve 34 slightly”. The model is not normally

explicitly described, but it was learnt implicitly from another operator and maybe improved, through trial and error, by the operator herself. The resulting experience and know-how is sometimes difficult to put into words; it is a kind of *tacit* knowledge. Nevertheless one could say that the operator acts on the basis of a model she has in mind.

On the other hand, the procedure for aeroplane maintenance, according to a detailed checklist, is thoroughly *explicit*. The model is possibly a huge guide that instructs the maintenance staff on how to proceed in each and every situation.

Chemical processes can be controlled and explained using complex mathematical models. They often contain a set of differential equations which are difficult to solve (see [Rabinovich 1992]). These models are also explicit and written in a *formalized* language.

Other models are used to control a social environment and often contain *normative* components. Brokers often try, with more or less success, to use guidelines and principles such as «If the FED publicizes a high government deficit provision, the dollar will come under pressure, so sell immediately». Such guidelines often don't have their roots in a sophisticated economic theory; they just prove true because many follow them. Many models in social processes are of that type. We all follow certain principles, rules, standards, or maxims which control or influence our behaviour.

Still other models constitute the basis *for making decisions*. The famous waterfall model in the software development cycle says that the implementation of a new software has to proceed in stages: analysis, specification, implementation, installation and maintenance. It gives software developers a general idea of how to proceed when writing complex software and offers a rudimentary tool to help them decide in which order the tasks should be done. It does not say anything about how long the software team have to remain at any given stage, nor what they should do if earlier tasks have to be revised: It represents a *rule-of-thumb*.

An example of a more *formal* and complex decision-making-model would be a mathematical production model consisting typically of thousands of constraints and variables as used in the petroleum industry to decide how to transform crude oil into petrol and fuel. The constraints – written as

mathematical equations (or inequalities) – are the capacity limitations, the availability of raw materials etc. The variables are the unknown quantities of the various intermediate and end products to be produced. The goal is to assign numerical values to the variables so that the profit is maximized or some other goal is attained.

Both models are tools in the hand of an intelligent agent and guide her in her activities and support her in her decisions. The two models are very different in form and expression; the waterfall model contains only an informal list of actions to be taken, the production model, on the other hand, is a highly sophisticated mathematical model with thousands of variables which needs to be solved by a computer. But the degree of formality or complexity is not necessarily an indication of the “usefulness” of the model, although a more formal model is normally more precise, more concise, and more consistent. Verbal and pictorial models, on the other hand, give only a crude view of the real situation.

Models may or may not be *pertinent* for some aspects of reality; they may or may not *correspond* to reality, which means that models can be misleading. The medieval model of human reproduction suggesting that babies develop from homunculi – fully developed bodies within the woman's womb – leads to the absurd conclusion that the human race would become extinct after a finite number of generations (unless there is an infinite number of homunculi nested within each other). The model of a flat, disk-shaped earth prevented many navigators from exploring the oceans beyond the nearby coastal regions because they were afraid of “falling off” at the edge of the earth disk. The model of the falling profit rate in Marx's economic theory predicted the self-destruction of capitalism, since the progress of productivity is reflected in a decreasing number of labour hours relative to the capital. According to this theory, labour is the only factor that adds plus-value to the products. Schumpeter agreed on Marx's prediction, but based his theory on a very different model: Capitalism will produce less and less innovative entrepreneurs who create profits! The last two examples show that very different sophisticated models can sometimes lead to the same conclusions.

In neurology, artificial neural networks, consisting of a connection weight matrix, could be used as models for the functioning of the brain. Of course,

such a model abstracts from all aspects except the connectivity that takes place within the brain. However, some neurologists believe that only 5%(!) of information passes through the synapses. If this turned out to be true, artificial neural nets would indeed be inappropriate models for the functioning of the brain.

One can see from these examples that models are ubiquitous and omnipresent in our lives. “The whole history of man, even in his most non-scientific activities, shows that he is essentially a model-building animal” [Rivett 1980, p. 1]. We live with “good” and “bad”, with “correct” and “incorrect” models. They govern our behaviour, our beliefs, and our understanding of the world around us. Essentially, we see the world by means of the models we have in mind. The value of a model can be measured by the degree to which it enables us to answer questions, to solve problems, and to make correct predictions. Better models allow us to make better decisions, and better decisions lead us to better adaptation – the ultimate “goal” of every being.

1.2. The Advent of the Computer

This work is not about models in general, their variety of functions and characteristics. It is about a special class thereof: mathematical models. Mathematics has always played a fundamental role in representing and formulating our knowledge. As sciences advance they become increasingly mathematical. This tendency can be observed in all scientific areas irrespective of whether they are application- or theory-oriented. But it was not until this century that formal models were used in a systematic way to solve practical problems. Many problems were formulated mathematically long ago, of course. But often they failed to be solved because of the amount of calculation involved. The analysis of the problem – from a practical point of view at least – was usually limited to considering small and simple instances only.

The computer has radically changed this. Since a computer can calculate extremely rapidly, we are spurred on to cast problems in a form which they can manipulate and solve. This has led to a continuous and accelerated pressure to formalize our problems. The rapid and still ongoing development of computer technologies, the emergence of powerful user environment software for geometric modeling and other visualizations, and the development of

numerical and algebraic manipulation on computers are the main factors in making modeling – and especially mathematical modeling – an accessible tool not only for the sciences but for industry and commerce as well.

Of course, this does not mean that by using the computer we can solve every problem – the computer has only pushed the limit between practically solvable and practically unsolvable ones a little bit further. The bulk of practical problems which we still cannot, and never will be able, to solve efficiently, even by using the most powerful parallel machine, is overwhelming. Nevertheless, one can say that many of the models solved routinely today would not be thinkable without the computer. Almost all of the manipulation techniques in mathematics, currently taught in high-schools and universities, can now be executed both more quickly and more accurately on even cheap machines – this is true not only for arithmetic calculations, but also for algebraic manipulations, statistics and graphics. It is fairly clear that all of these manipulations will become standard tools on every desktop machine in the very near future. Twenty years ago, the hand calculator replaced the slide rule and the logarithm tables, now the computer replaces most of those mathematical manipulations which we learnt in high-school and even at university.

1.3. New Scientific Branches Emerge

Some research communities such as *operations research* (OR) owe their very existence to the development of the computer. Their history is intrinsically linked to the development of methods applicable on a computer. The driving force behind this development in the late 1940s was the Air Force and their Project SCOOP. Numerical methods for linear programming (LP) were stimulated by two problems they had to solve: one was a diet problem. Its objective is to find the minimum cost of providing the daily requirement of nine nutrients from a selection of seventy-seven different foods. Initially, the calculations were carried out by five computers³ in 21 days using electromechanical desk calculators. The simplex method for linear

³ Human calculators carrying out extended reckoning were called “computers” until the end of the 1940's. Many research laboratories utilized often poorly paid human calculators – most of them were women.

programming (LP), discovered and developed by Dantzig in 1947⁴, was and still is one of the greatest successes in OR. Together with good presolve techniques we can now solve almost any LP problems up to 250,000 variables and 300,000 constraints (or alternatively 1,000,000 variables and 100,000 constraints) by a direct method. Problems with 800 constraints and 12 million variables have been solved using column generation methods. Problem belonging to the particular class of transportation problems and consisting of 10's of thousands of constraints and 20 million variables are routinely solved today. LPs containing special structures with 10^{50} variables are solved frequently.⁵ Unfortunately, this success story does not prove true for most other interesting problems. On the contrary, it was discovered that almost all integer and combinatorial problems are algorithmically hard to solve. No efficient procedure is yet known despite intensive research over the last 40 years. There seems to be little hope of ever finding an efficient one. There are even problems that cannot be solved, neither efficiently nor inefficiently, but that is another story...

Artificial intelligence (AI) is also thoroughly dependent on the progress in computer science. Initially, many outstanding researchers in this domain believed that it would only be a question of decades before the intelligence of a human being could be matched by machines. Even Turing was confident that it would be possible for a machine to pass the Turing Test by the end of the century. Some scientists believe that we are not far from that point. Even though most problems in AI turned out to be algorithmically hard, since they are closely related to combinatorial problems. This led to an intensive research of heuristics and “soft” procedures – methods we humans use daily to solve complex problems. The combination of these methods and the computer's extraordinary speed in symbolic manipulation produces a powerful means to

⁴ See: Dantzig G.B. [1991], Linear Programming, in: History of Mathematical Programming, A Collection of Personal Reminiscences, edited by Lenstra J.K., Rinnooy Kann A.H.G., Schrijver A., CWI, North-Holland, Amsterdam, 1991.

⁵ See: Infanger G., [1992], Planning under Uncertainty: solving large-scale stochastic linear programs, Techn. Univ. Wien. See also: Hansen P., [1991], Column Generation Methods for Probabilistic Logic, in: ORSA Journal on Computing, Vol. 3, No. 2, pp. 135–148.

implement complex problems in AI.

Other scientific communities had already developed highly efficient procedures for solving sophisticated numerical problems before the first computer was built. This is especially true in physics and engineering. For example, Eugène Delaunay (1816–1872) made a heroic effort to calculate the moon's orbit. He dedicated 20 years to this pursuit, starting in 1847. During the first ten years he carried out the hand calculation by expressing the differential system as a lengthy algebraic expression in power series, then during the second ten years he checked the calculations. His work made it possible to predict the moon's position at any given time with greater precision than ever before, but it still failed to match the accuracy of observational data from ancient Greece. A hundred year later, in 1970, André Deprit, Jacques Henrard and Arnold Rom revised Delaunay's calculation using a computer-algebra system. It took 20 hours of computer time to duplicate Delaunay's effort. Surprisingly, they found only 3 minor errors in his entire work. Calculations in celestial mechanics was the most challenging task in the 19th century and no engineer's education was complete without a course in it.⁶

Many numerical algorithms, such as the Runge-Kutta algorithm and the Fast Fourier Transform (FFT), were already known – the later even by Gauss⁷ – before the invention of the computer and they were much used by human “computers”. But for many problems these efforts were hopeless, for the simple reason that the human computer was too slow to execute the simple but lengthy arithmetics.

An interesting illustration of this point is the origin of numerical meteorology.⁸ Prior to World War II, weather forecasting was more of an art, depending on

⁶ See: Peterson I, [1993], *Newton's Clock, Chaos in the Solar System*, W.H. Freeman, New York, pp. 215, see also: Pavelle R., Rothstein M., Fitch J., [1981], *Computer Algebra*, in: *Scientific American*, Dec. 1981, p. 151.

⁷ See: Cooley J.W., [1990], *How the FFT Gained Acceptance*, in: Nash S.G. (ed.), *A History of Scientific Computing*, ACM Press, New York, pp. 133–140. See also [Heideman al. 1985].

⁸ An excellent account of its origins is given in Chapter 6 of: Aspray W., [1990], *John von Neumann and the Origins of Modern Computing*, The MIT Press, Cambridge.

subjective judgement, than a science. Although, in 1904, Vilhelm Bjerknes had already elaborated a system of 6 nonlinear partial differential equations, based on hydro- and thermodynamic laws, to describe the behaviour of the atmosphere, he recognized that it would take at least three months to calculate three hours of weather. He hoped that methods would be found to speed up this calculation. The state of the art did not fundamentally change until 1949 when a team of meteorologists – encouraged by John von Neumann, who regarded their work as a crucial test of the usefulness of computers – fed the ENIAC⁹ with a model and got a 24-hour “forecast”, after 36 hours of calculations, which turned out to be surprisingly good. Four years later, the Joint Numerical Weather Prediction Unit (JNWPU) was officially established; they bought the most powerful computer available at that time, the IBM 701, to calculate their weather predictions. Since then, many more complex models have been introduced. The computer has transformed meteorology into a mathematical science.¹⁰

In still other scientific communities, until very recently, mathematical modeling was not even a topic or was used in a purely academic manner. Economics is a good example of this. Economists produced many nice models without practical implications. Sometimes, such models have even been developed just to give more credence to policy proposals. But mathematical models are not more credible simply because they are expressed in a mathematical way. On the other hand, important theoretical frameworks –

⁹ ENIAC (Electronic Numerical Integrator and Computer) was one of the first electronic general-purpose computers built at the Moore School of Engineering, University of Pennsylvania, in 1946.

¹⁰ Meteorology is only one representative of a large problem class that can be formulated as differential systems, fluid dynamics being another. These applications are still regarded as "grand challenge" problems in contemporary supercomputing. It is estimated that a direct simulation of air flow past a complete aircraft will require at least an exaflop (10^{18}) computer. The supercomputer Cray YMP is running at 200 megaflops (10^6). See: COVENEY P., HIGHFIELD R., [1995], *Frontiers of Complexity*, Fawcett Columbine, New York, p. 68-69. However, Intel has announced (December 1996) the first teraflop (10^{12}) computer consisting of more than 7000 Pentium Pro processors.

such as game theory going back to the late twenties when John von Neumann published his first article on this topic – have been developed. Realistic n -person games of this theory cannot be solved analytically. They need to be simulated on computers. So the attitude towards these formal methods is also gradually changing in these other sciences as well. Today, no portfolio manager works without optimizing software. Branches such as evolutionary biology which have been more philosophical, are also increasingly penetrated by mathematical models and methods.

1.4. Mathematical Modeling – the Consequences

We should not underestimate the significance of the development of computers for mathematical modeling. Their capacity to solve mathematical problems has already changed the way in which we deal with and teach applied mathematics. The relative importance of skills for arithmetic and symbolic manipulation will further decrease. This does not mean that we will no longer need applied mathematicians. On the contrary, we will need even more people qualified to translate real-life problems into formal language, and what activity is more rewarding and intellectually more challenging in applied mathematics than – *modeling*? While relatively fewer mathematicians are needed to solve a system of differential equations or to manipulate certain mathematical objects, more and more are needed who are skilled and expert in modeling.

This development is by no means confined to science. Since World War II, a growing interest has been shown in formulating mathematical models representing physical processes. These kinds of models are also beginning to pervade our industrial and economical processes. In several key industries, such as chip production or flight traffic, optimizing software is an integral part of their daily activities. In many other industrial sectors, companies are beginning to formalize their operations. *GeoRoute*, for instance, a transportation company in the United States who delivers more than 10 millions items a year all over the USA using 500 trucks, is literally built on a computer program, called NETOPT, which optimizes the deliveries in real-time. Mathematical models are beginning to be an essential part of our highly developed society.

But the use of models in an industrial context still demands a highly

specialized team of experts; experts in operations research, in computer science as well as in management. These teams are costly. Is it imaginable that for many modeling tasks these teams could one day be replaced by a small group or even a single person who uses a highly sophisticated computer-based modeling tool? The question seems to be a little bit naive! But one has to look to the problem in another way: In practice, many operations are *not* formalized, and highly developed optimizing software is *not* used, *because the set-up of the mathematical model is too costly*. So a large amount of the precious time of these highly qualified people is wasted to express the model in a way that the solver can use.

The dream of such a miraculous software has already got a name: “Expert Systems” or “Decision Support Systems (DSS)”. According to the philosophy of these new technologies, the idea that only experts can perform complex work is outdated. Certainly, experts are needed to assure progress in their special fields. But in order to solve more and more complex problems efficiently, experts can be replaced by generalists. The argument is: “... the real value of expert systems technology lies in its allowing relatively unskilled people to operate at nearly the level of highly trained experts. ... Generalists supported by integrated systems can do the work of many specialists, and this fact has profound implications for the ways in which we can structure work.” [Hammer/Champy 1994, p. 93]. While the treatise of Hammer/Champy is not about DSS, the quotation expresses very well the underlying philosophy. There is a vast amount of literature about DSS. But, as is often the case, quantity is not necessarily a sign of quality. Too much expectation has been sown, and too few results have been harvested. The main limitation of most DSS – as built and used today – is that they are *not general* tools for modeling, but can only be applied in a restrictive way only and for narrowly specified problems. Nonetheless, the need for such tools is evident.

An important prerequisite for the widespread use of DSS tools is a change in the mathematical curriculum in school: A greater part of the manipulation of mathematical structures should be left to the machine, but more has to be learnt about how to recognize a mathematical structure when analyzing a particular problem. It should be an important goal in applied mathematics to foster creative attitudes towards solving problem and to encourage the students' acquisition and understanding of mathematical concepts rather than

drumming purely mechanical calculation into their heads. Only in this way can the student be prepared for practical applications and modeling.¹¹

But how can modeling be learnt? Problems, in practice, do not come neatly packaged and expressed in mathematical notation; they turn up in messy, confused ways, often expressed, if at all, in somebody else's terminology. "Whereas problem solving can generally be approached by more or less well-defined techniques, there is seldom such order in the problem posing mode" [Thompton 1989, p. 2]. Therefore, a modeler needs to learn a number of skills. She must have a good grasp of the system or the situation which she is trying to model; she has to choose the appropriate mathematical tools to represent the problem formally; she must use software tools to solve the models; and finally, she should be able to communicate the solutions and the results to an audience, who is not necessarily skilled in mathematics.

It is often said that modeling skills can only be acquired in a process of learning-by-doing; like learning to ride a bike can only be achieved by getting on the saddle. It is true that the case study approach is most helpful, and many university courses in (mathematical) modeling use this approach [Clements 1989], [Ersoy 1994]. But it is also true – once some basic skills have been acquired – that theoretical knowledge about the mechanics of bicycles can deepen our understanding and enlarge our faculty to ride it. This is even more important in modeling industrial processes. It is not enough to exercise these skills, one should also acquire methodologies and the theoretical background to modeling. In applied mathematics, more time than is currently spent, should be given to the study of discovery, expression and formulation of the problem, initially in non-mathematical terms.

So, the novice needs first to be an observer and then, very quickly, a do-er. Modeling is not learnt only by watching others build models, but also by being actively and personally involved in the modeling process (case study approach).

¹¹ A desperate teacher of applied mathematics wrote: "If one fights for applied mathematics teaching, one can actually promise only 'blood, sweat and tears' and, moreover, must already fear one more wave (the computer wave) which will be supported as widely and extensively as that first" [Schupp H., in: Blum al., 1989, p. 36].

As a consequence of these needs, which became evident in the seventies, a “movement of model-building” in applied mathematics education has emerged, marked by an increasing number of publications, see [Clements 1989, p. 10]. The journal *Teaching Mathematics and Its Applications* was founded in 1982 (as a successor of the *UMAP Journal*), and regular conferences have been organized in this field since 1983.

The use of computer modeling tools to simulate, visualize, and analyze mathematical structures has spread steadily. In the eighties, the “micro-computer” – a word which disappeared as quickly as it emerged – was the archetype of a whole generation of self-made quick and dirty models. Everyone produced their own simulation tool, mathematical toolbox, etc. This phenomena has now almost vanished. We have powerful packages such as Mathematica [Wolfram 1991], Maple [Char et al., 1991], MatLab [MatLab 1993], Axiom [Jenks, 1992], the NAG-library [NAG], to mention but a few, for solving complex problems. But the modeling process still needs more than easy-to-use solving tools: It also needs tools to *integrate different tasks*: such as data manipulation tools, representation tools for the model structure, viewing tools to represent the data and structure in different ways, reporting tools, etc. Some tasks can be done by different software packages: data is best manipulated in databases and spreadsheets, and is best viewed by different graphic tools.

1.5. Computer-Based Modeling Management

This brings us to the heart of this work: Modeling is still an art with tools and methods built in an ad hoc way for a specific context and for a particular type of model. Whilst I believe that modeling will always be an art in the sense of a creative process, I also firmly believe that we can build *general* computer-based modeling tools that can be used not only to help find the solution to a mathematical model, but also *to support the modeling process itself*. The extraordinary advances in science, on one hand, and computing performance, on the other hand, leave a cruel gap that can be bridged by general and efficient *modeling management systems*. Only when we reach this point, will the new technologies be fully exploited.

There are three main reasons, in my opinion, why DSS have not fulfilled their

promise.

- The first reason has to do with the modeling process itself. Every problem needs its own model specification, and making good models *is* difficult. No modeling management system however sophisticated can do the job on its own. A major difficulty in developing such a system is that the modeling of knowledge representation involves a wide range of activities. Modeling is done by people with very different backgrounds and in various contexts, and it is difficult to develop modeling tools which can be used by almost everyone.

Therefore, most modelers still develop and use their own ad hoc tools to manage their models. There are important disadvantages in doing so. Models are difficult to maintain with a changing crew. Model transparency may suffer, and portability to different environments is limited. Often a model, or parts of it, could also be used in another context, but reusability is almost impossible. Operations research and artificial intelligence journals are full of articles describing a special implementation of a model and its environment. The cost of developing such ad hoc tools should not be underestimated. This is one of the main reasons why decision makers still tend to use rules-of-thumb rather than the troublesome path of model building.

- The second reason has to do with the solution complexity of many problems. While the specification of the problem is sometimes straightforward, the solution is not. This is especially true for discrete problems. In this case, we need tools which assist the modeler in reformulating and translating the model in such a way that different methods or heuristics can be used to solve it. The model structure must be easy to manipulate. We need flexibility in manipulating model structures in the same way as we are able to manipulate data – using databases for instance. None of the tools available today are suited for these tasks.
- The third reason concerns coping with *uncertainty* and *inconsistency*. In real problems, uncertainty is ubiquitous, and eliminating inconsistencies from a model is one of the modeler's main task. Both aspects have not received the attention they deserve, at least in the field of mathematical modeling. Chapter 4 explains, to some extent, why I think that these aspects are important.

It would be of great use, if decision makers owned some *universally usable* modeling tools and methods to do their job. Not only are we sorely in need of them, but – I am convinced – they are also feasible. Actually, there *are* important research activities going on in the realm of modeling management systems.

1.6. About this Book

This book does *not* propose (computer-based) solution methods for mathematical models, *nor* does it give suggestions on how to build mathematical models. It proposes a framework for representing and specifying mathematical models on computers in machine-readable form. It suggests a *modeling language* in contrast to a *programming language*.

The “ultimate” modeling language is in many senses a superset of a programming language: it specifies *what* the model is in a declarative way *and* it encodes *how* the model has to be solved in a procedural (algorithmic) way. A programming language only represents the second, algorithmic part. I shall concentrate on the first, declarative part of how mathematical models can be specified.

Part I explores fundamental features of models. It consists of three chapters. Chapter 2 defines the notion of “mathematical model” and other related concepts. It begins with general considerations about *models* and ends with a few thoughts about declarative and procedural knowledge. Chapter 3 gives a comprehensive overview of the modeling life cycle, wherein the different steps from building to solving models are traced. Chapter 4 summarizes different paradigms of models, different model types and their purposes. It also suggests a short introduction to modeling uncertainty – an important and all too often neglected topic in real life modeling.

Part II presents more thorough justifications for the need for modeling management systems, and especially for a modeling language. It also contains three chapters. Chapter 5 presents the current situation in model management systems as well as the problems linked to it. Different approaches and systems

in computer-based modeling are exposed in Chapter 6. The five most frequently used methodologies are reviewed. Finally, Chapter 7 proposes my own view of a general framework in computer-based modeling. This Chapter is the core of this book.

Part III displays my own concrete contribution to this field of research: the modeling language LPL. Chapter 8 shows how the general framework exposed in the previous chapter is implemented. An exact syntax notation of the language is given in extended Backus-Naur form. Chapter 9 exposes in which modeling environment the language has been embedded. Model browser and different graphical tools are presented. Finally, Chapter 10 gives different applications of the framework. Several examples used in previous chapters are restated in LPL, others explain various aspects such as units, modeling of logical constraints, and others.

The glossary summarizes the specific vocabulary used in this book.

Many examples are used in this text to illustrate different aspects of modeling. They are all clearly numbered and listed after the table of contents. The symbol character \boxtimes is used to indicate in the text where the example ends.

