

ACCEL : a tool for supporting concept generation in the early design phase

Citation for published version (APA):

Ivashkov, M. (2004). *ACCEL : a tool for supporting concept generation in the early design phase*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Built Environment]. Technische Universiteit Eindhoven.
<https://doi.org/10.6100/IR578193>

DOI:

[10.6100/IR578193](https://doi.org/10.6100/IR578193)

Document status and date:

Published: 01/01/2004

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

ACCEL: a Tool for Supporting Concept Generation in the Early Design Phase

PROEFONTWERP

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. R.A. van Santen,
voor een commissie aangewezen door het College voor Promoties
in het openbaar te verdedigen op
dinsdag 15 juni 2004 om 16.00 uur

door

Maxim Ivashkov

geboren te Minsk, Wit-Rusland

De documentatie van het proefontwerp is goedgekeurd door
de promotoren:

prof.ir. P.G.S. Rutten

en

prof.dr.ir. B. de Vries

Copromotor:

dr.ir. C.W.A.M. van Overveld

“ACCEL: a Tool Supporting Concept Generation in the Early
Design Phase.”

ISBN 90-6814-581-9

Copyright © Maxim Ivashkov 2004

All rights reserved.

Printed by the Eindhoven University Press, Eindhoven, The Netherlands.

Published as issue 84 in the *Bouwstenen* series of the Faculty of
Architecture, Building and Planning of the Eindhoven University of
Technology.

Contents

LIST OF ACRONYMS	1
LIST OF SYMBOLS	3
CHAPTER 1. INTRODUCTION.....	5
1.1 PROBLEM DEFINITION.....	6
1.2 THE STAKEHOLDERS	8
1.3 MODELING APPROACH	10
1.3.1 <i>Knowledge management</i>	10
1.3.2 <i>Linguistics</i>	11
1.3.3 <i>The mechanism of questions and answers</i>	12
1.4 THE DOMAIN OF THE EARLY DESIGN PHASE	13
1.4.1 <i>The organization domain</i>	14
Design team.....	14
Stakeholders	15
1.4.2 <i>The product domain</i>	16
The design problem.....	16
Solutions.....	18
1.4.3 <i>The process domain</i>	18
1.5 OUTLINE OF THE RESULTS OF THIS THESIS	19
CHAPTER 2. AN OPERATIONAL MODEL OF THE DESIGN PROCESS	21
2.1 INTRODUCTION.....	21
2.1.1 <i>The basic assumptions</i>	21
2.1.2 <i>Intention of the model and the outline</i>	21
2.1.3 <i>Position of the model</i>	22
2.2 DESCRIPTIVE PURPOSE	23
2.2.1 <i>State-transition model</i>	23
2.2.2 <i>A structure for the product</i>	25
2.3 A FORMALISM FOR PRODUCT KNOWLEDGE.....	27
2.3.1 <i>Introduction</i>	27
2.3.2 <i>Basic ingredients of the formalism</i>	28
The set of concepts.....	28
The set of solution concepts.....	29
The set of attributes	29
Attribute types.....	32
Structures for concepts	33
2.3.3 <i>The decision space V_D</i>	34
2.3.4 <i>The objective space V_O</i>	37
2.3.5 <i>The contextual space V_C</i>	38
2.3.6 <i>The auxiliary space V_A</i>	39
2.3.7 <i>Summary</i>	39
2.4 PRODUCT MODELS.....	40
2.4.1 <i>Basic definitions</i>	40
2.4.2 <i>Definition of the product model</i>	41
2.4.3 <i>Classes of product models and the scope of our formalism</i>	43
2.4.4 <i>Development of product models</i>	44
2.4.5 <i>Summary</i>	45

2.5	REFLECTIONS	46
2.5.1	<i>Formal definition of questions and answers</i>	46
2.5.2	<i>Categories of questions</i>	47
2.5.3	<i>Example</i>	48
2.6	COMPUTATIONAL OPERATIONS	50
2.6.1	<i>Introduction</i>	50
2.6.2	<i>Mathematical programming problems</i>	50
2.7	GENETIC-BASED OPTIMIZATION: SPEA	52
2.7.1	<i>Pareto optimality</i>	53
2.7.2	<i>Fitness assignment</i>	54
2.7.3	<i>Diversification strategy</i>	55
2.7.4	<i>Density preservation</i>	56
2.7.5	<i>Elitism strategy</i>	56
CHAPTER 3. ACCEL		59
3.1	INTRODUCTION	59
3.1.1	<i>The tool structure</i>	60
3.1.2	<i>Top-down approach</i>	61
3.1.3	<i>Bottom-up approach</i>	61
3.1.4	<i>Outline</i>	63
3.2	REQUIREMENTS ANALYSIS	63
3.2.1	<i>Functional requirements</i>	63
3.2.2	<i>Additional functional requirements</i>	64
3.2.3	<i>The choice of C++</i>	65
3.2.4	<i>Context and constraints</i>	65
3.3	ACCEL DOCUMENT: OBJECT ORIENTED DATA MODEL	66
3.3.1	<i>Syntax</i>	66
3.3.2	<i>Everything is a function</i>	67
3.3.3	<i>Data structure</i>	67
3.3.4	<i>XML format of data storage</i>	68
	XML Constructs	69
	XML example	69
3.4	ACCEL MANAGER: EVALUATION SEMANTICS	70
3.4.1	<i>Parser</i>	70
3.4.2	<i>Factory of functions</i>	71
3.4.3	<i>Evaluation of expressions</i>	72
3.4.4	<i>Error handling</i>	73
3.5	SPECIAL FUNCTIONS	74
3.5.1	<i>Function ALT()</i>	74
3.5.2	<i>Function ASKUSER()</i>	74
3.5.3	<i>Reference function</i>	75
3.6	ACCEL GUI: GRAPHICAL USER INTERFACE	77
3.6.1	<i>Spreadsheet</i>	79
3.6.2	<i>Properties view</i>	82
3.6.3	<i>AskUser dialog</i>	83
3.6.4	<i>Sensitivity dialog</i>	84
3.6.5	<i>Trace View</i>	85
3.7	OPTIMIZATION WITH ACCEL	87
3.7.1	<i>Optimization dialog</i>	87
	Top part	87

Central part.....	88
Bottom part	88
3.7.2 <i>Tightening constraints method</i>	88
3.7.3 <i>Optimization example</i>	89
CHAPTER 4. CASE STUDIES.....	91
4.1 INTRODUCTION.....	91
4.2 AIR-CONDITIONING SYSTEM FOR A SWIMMING POOL	92
4.2.1 <i>Introduction</i>	92
4.2.2 <i>Design solutions</i>	93
4.2.3 <i>Objectives</i>	94
Prices.....	95
Price over 5 years.....	96
Air distribution quality.....	97
4.2.4 <i>Contextual information</i>	98
4.2.5 <i>Auxiliary (intermediate) information</i>	98
4.2.6 <i>Representation in ACCEL and optimization</i>	99
4.2.7 <i>Analysis of the model</i>	100
Range analysis.....	100
Analytical sensitivity analysis.....	100
Graphical sensitivity analysis.....	101
4.2.8 <i>Conclusions</i>	102
Our conclusions.....	102
Conclusions of our expert	103
4.3 TRANSPORTATION SYSTEM FOR WTC	103
4.3.1 <i>Introduction</i>	103
4.3.2 <i>Manual generation of solutions</i>	104
Group 1	105
Group 2	106
Group 3	106
4.3.3 <i>Objective values</i>	107
Contextual values.....	108
4.3.4 <i>Analysis of the results</i>	108
4.3.5 <i>Conclusions</i>	109
4.4 EARLY EXPERIMENTS	109
4.4.1 <i>The second year</i>	110
First experiment	110
Second experiment.....	110
Third experiment.....	110
4.4.2 <i>The third year</i>	112
CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS.....	113
5.1 RESULTS	113
5.2 COMPLIANCE WITH INITIAL OBJECTIVES.....	114
5.3 RECOMMENDATIONS FOR DESIGN COURSES.....	114
5.4 RECOMMENDATIONS FOR SOFTWARE DEVELOPMENT	115
5.5 EPILOG	115
APPENDIX A. SPEA ALGORITHM.....	117
APPENDIX B. SYNTAX OF ACCEL.....	118

APPENDIX C. DESIGNING WITH ACCEL	121
APPENDIX D. RESULTS OF WTC PROJECT	123
APPENDIX E. COFFEE PROBLEM.....	125
BIBLIOGRAPHY.....	130
SUMMARY	135
SAMENVATTING	137
ACKNOWLEDGEMENT	139
BIOGRAPHICAL NOTE	140

List of Acronyms

ACCEL	A tttributes C oncepts C onstraints E valuation L anguage
CAD	C omputer A ided D esign
D	D efinition
DAG	D irected A cylic G raph
Ex	E xample
FHS	F ully H ierarchical S tructure
FOS	F ully O rthogonal S tructure
GA	G enetic A lgorithm
GPS	G eneral P roblem S olving
GUI	G raphical U ser I nterface
HVAC	H eating V entilation A ir- C onditioning
MDI	M ultiple D ocument I nterface
OOD	O bject- O riented D esign
PPO	P roduct P rocess O rganization model
R	R emark
SPEA	S trength P areto E volutionary A lgorithm
VC++	V isual C++
XML	e Xtensible M arkup L anguage

List of Symbols

a_j	an attribute
A	the set of currently considered attributes
c_i	a concept
$c_i.a_j$	a variable
C	the set of all considered concepts in the product domain P
f	functional dependency
F	product model
$\phi(c_i)$	signature of a concept c_i
$\phi'(c_i)$	extended signature
$\psi(a_j)$	extension of an attribute a_j
g	constraint function
l_f	left constraint of a function f
P	the product domain: all knowledge about a product
Q	the set of defined operations
q_{ij}	an operation to obtain values of a variable $c_i.a_j$
r_f	right constraint of a function f
R_j	the range of an attribute a_j
R'_j	extended range
s_i	a solution concept
s_i^d	decision variables
s_i^o	objective variables
s_i^c	contextual variables
s_i^a	auxiliary variables
S	the set of currently considered solutions
v_{ij}	the value of a variable $c_i.a_j$
V_D	the decision space: all knowledge that designer can decide upon
V_O	the objective space: all knowledge that enables selection of the solutions
V_C	the context space: all knowledge that constitutes measured or observed fact
V_A	the auxiliary space: all intermediate knowledge
v_i^*	the tuple of values $(v_{i0}, v_{i1}, v_{i2}, \dots, v_{ij})$ which represents a concept c_i
v_i^d	decision values
v_i^o	objective values
v_i^c	contextual values
v_i^a	auxiliary values

CHAPTER 1. INTRODUCTION

The life cycle of any product begins with one or more people expressing needs that require a new product. We will refer to such people ‘*stakeholders*’. Stakeholders express their needs to an ‘*organization*’, which initiates a design process in order to deliver a new product. We will refer to people who participate in the design process and create a future product as ‘*designers*’. The design process is a complex procedure involving intermediate phases. Many of the well-known design process models identify the phases of ‘*briefing*’ (the analysis of the problem [French, 1971][Hubka, 1992], clarifying objectives [Cross, 1991]) and ‘*conceptual design*’, which we consider as the early design phase because of tight coupling and an intensive interaction between them. During the briefing phase, the designers collaborate with the stakeholders in order to translate the needs of the stakeholders into the ‘*design problem*’. Design problems are ‘*ill-defined*’, i.e. there is no the single best definition of the problem and a single valid solution. Therefore during the conceptual phase, the designers simultaneously reformulate the design problem and generate one or more ‘*solution concepts*’ for the problem. The next phase is the ‘*detailed design phase*’, where selected solutions are engineered and validated or tested. Advanced computer tools such as CAD tools (Computer Aided Design), prototyping and simulation can be used to support this phase. This is followed by the manufacturing phase, where one or more instances of the final product solution are manufactured. The life cycle of any one product usually ends with the recycling phase.

In this thesis, we focus on tools to support generation of solution concepts during the early design phase. The early design phase is characterized by the intensive processes of ‘*communication*’ and ‘*decision-making*’, which lead to a build up of knowledge about the product. Design decisions have a long-term impact and imply significant investments and the dedication of costs for many years. For instance, building products may have a life cycle lasting decades. Early design decisions therefore need to simultaneously take into account wide-ranging considerations such as energy consumption, environmental aspects, appearance, legislation, manufacturing and recycling. Broad considerations allow to satisfy the needs of the stakeholders more efficiently and effectively than could be done with a few limited, sequential considerations. Any design decisions changed at a later design phase, e.g. during the detailed design phase or the production phase, will be expensive. The decisions taken during the early design phase can strongly influence the range of possible product designs and will reduce manufacturing options. Ill-considered decision-making or miscommunication can therefore result in inefficient solutions for the product. The design decisions that cover conceptual design issues are often directive and, at the same time, restrictive and irreversible, [Rutten, 1998].

1.1 Problem definition

The communication and decision-making processes of the early design phase are based on ‘*natural tools*’ such as natural languages, intuition, creativity and common sense. According to Kleban [2001], “in the early design, people tend to rely more on ad hoc representations (sketches, short memos, etc) and communication (e-mail, hallway conversations, etc)”. Only at the end of the conceptual phase or at the beginning of the detailed design phase, available computer support tools become suitable. Early design decisions are therefore made with both the lack of consistent knowledge and computational support. Given the complexity and impact of the early decisions, the natural tools do not provide a sufficient means with which to analyze and validate the decisions made. We believe that as the amount of consistent knowledge increases, so the impact of the decisions taken decreases. We therefore have a computational support in the design process after the moment D and we need to shift this moment back to an earlier moments in the design process, see Figure 1.

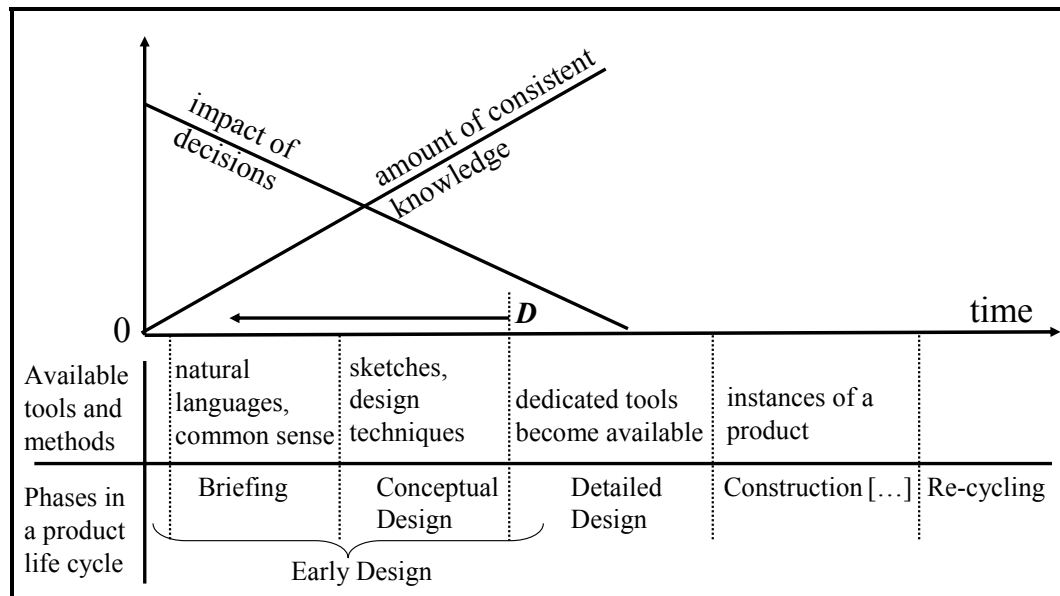


Figure 1. The impact of decisions versus the availability of consistent knowledge

Most CAD tools focus on the detailed design phase (engineering phase) after the moment D , i.e. after the early design phase; it is difficult to adapt these computational tools to the earlier phases. It is generally accepted that dedicated design tools are too inflexible to be adapted to growing needs. According to Akman [1990] “CADs do not support crucial ingredients of design, namely interaction, creativity”. However, there is a need for highly expressive computational tools in the early design phase; these would shift the moment D to the early phases of the design process.

There are various non-computational design methods, which we will call ‘techniques’. Techniques (e.g. creativity techniques [Glover, 1989], [Bono, 1970], and techniques for re-engineering such as QFD [Prasad, 1998] and LCA [Berg, 1995]) can be used to support designers during the early design phase. These techniques are easy to use. However, they are also limited to certain types of decisions. The formalisms of techniques limit the possibilities of the designers to express the thoughts behind the decisions. This in turn means that learning, assessment and operations with the obtained results become problematic. C. Jones [1992] writes the following about Alexanders’

method [Alexander, 1971], “There is the expression that private thoughts are being classified to the point where the thinker can make a decision but not to the point where the results can be tested and understood by others”. From the variety of the available techniques and methods very few are suitable for the early design phase. These few techniques are the only artificial tools that the designers have. The management of complexity is not part of these techniques and is usually non-computational and limited, e.g. “house of quality” in QFD technique.

Skilled designers rely on their past experience to manage the complexity; we could say that they operate with large chunks of *‘implicit knowledge’*. We will define various kinds of knowledge in section 1.3.1. New designers do not yet have this type of experience. Because skilled designers often appear to work in a rather unsystematic way, some people claim that it does not help designers to learn a systematic approach. Design literature contains many examples of the resistance of designers to apply systematic procedures in the early design phase. The designers’ traditional approach is to try to move fairly quickly to a potential solution, or a set of potential solutions, and to use this as a means of further defining and understanding the problem. Traditional approaches to engineering [Pahl, 1984] assume a set of defined functional requirements for the product. In the early design phase, the problem formulation is not fixed and solutions with different functionalities can therefore be proposed. Even if the stakeholders specify a list of functions, they may be vague and incomplete.

According to Hurlimann [1999], “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. 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”. According to Cross [1994], “Many designers are suspicious of rational methods, fearing that they are a straitjacket, or that they stifle creativity. This is a misunderstanding [...]. Creative methods and rational methods are complementary aspects of a systematic approach to design. Rather than a straitjacket, they should be seen as a life jacket, helping the designers – especially the student designer – to keep afloat”. Several research studies show that a systematic approach can be helpful to students and that systematic procedures correlate positively with both the quantity and the quality of the students’ design results [Radcliffe, 1989].

A demonstration of the effectiveness of systematic approaches in the early design phase is complicated by the lack of computational tools capable of efficiently dealing with the complexity caused by these approaches. In this thesis we therefore look at both the development of a systematic procedure and a computational support tool. We define the addressed design problem as follows:

- D. 1** (Problem definition) Develop a systematic procedure and a tool to support generating concepts in the early design phase:
- to evaluate the application of systematic procedures in the early design phase,
 - to increase the complexity of design problems that can be considered during design courses and to approach the complexity of realistic design problems in industry,
 - to improve the clarity of the decision-making process and the process of communication as it takes place in the early design phase.

The addressed problem is too complex to allow us to research all the aspects related to the early design phase. We have therefore narrowed down the subject to considerations

of knowledge build up during the design process. We will limit the range of possible solutions for support to the following means:

1. An operational model of design processes.
2. A software tool for the early design phase, based on a model.

Here, an '*operational model*' means that we will develop support for the early design phase using a model that is '*systematic*' and '*computational*'. The term '*systematic*' means that a model should explain how solution concepts are generated in an incremental step-by-step manner. Traditional systematic procedures prescribe several phases such as analysis, synthesis, evaluation, refinement, and development [Cross, 1991], [Ullman, 1986]. However, these models do not computationally support these phases. The term '*computational*' means that the application of the model is supported by computational operations, such as knowledge evaluation, generation and optimization of solutions. The operational models require the following two ingredients to be developed:

1. A state-transitions model (STM) of the considered process. We therefore need to define the states in the early design phase and to describe transitions from one state to another.
2. A formalism (formal language) that should allow STM to be formally defined, so that computerized support is applicable.

The development of software support begins with functional requirements. We position our target support in the niche between the dedicated design tools and the techniques. It is unclear which functional requirements a tool in this niche needs to comply with. Standard approaches to define a list of functional requirements, such as interviewing, protocol analysis, situational analysis [Sauter, 1997], are of limited use and do not guarantee the completeness of the list. The support for an early design cannot only be based on the opinions of few people. Even if many interviews were conducted, this would not guarantee the completeness of the list. Existing functional requirements, such as those in Kleban [2001], have the following typical problems:

1. The requirements are introduced without sufficient reasoning and it is therefore not clear how to analyze them.
2. The requirements lack precision, which is necessary for the development of the specification.

Therefore, one of the purposes of our operational model is to produce a list of functional requirements for a support tool. In the next section, we introduce the stakeholders of this project and we define a set of objectives.

1.2 The stakeholders

In recent years, industry has stressed the need for graduates who are capable of solving design problems. The Stan Ackermans Institute (SAI) ¹ at the Eindhoven University of Technology (TU/e) organizes eight two-year multidisciplinary full-time programs leading to the degree of Master of Technological Design. These programs are the second part of a two-tiered system for training engineers in the Netherlands. The SAI

¹ The SAI website can be found at the URL address: <http://www.sai.tue.nl/>

initiated this PhD design project with the general problem formulation, “Improve the understanding of the existing design processes by proposing methods and tools”. After two and a half years, the project was transferred to the department of Building and Architecture at the TU/e, where one of the eight SAI programs is located. Here the project was embedded in the Center for Building and Systems TNO-TU/e.

The Center for Building and Systems TNO-TU/e² is a joint venture between the TU/e and TNO. TNO is one of the largest research and technology organizations in Europe. Its mission is to make scientific knowledge applicable, and to strengthen the innovative capacity of business and government³. Both the TU/e and TNO cooperate according to the diagram shown in Figure 2. Within the Center for Building and Systems the project joined the ‘*Strategic Design and IFD*’ cluster, and was embedded in the following research field:

- The strategic Design Support Tools for decisions: the research in this direction focuses on conceptual models for decision-making and strategic criteria for design teams for different building concepts and different users of buildings.

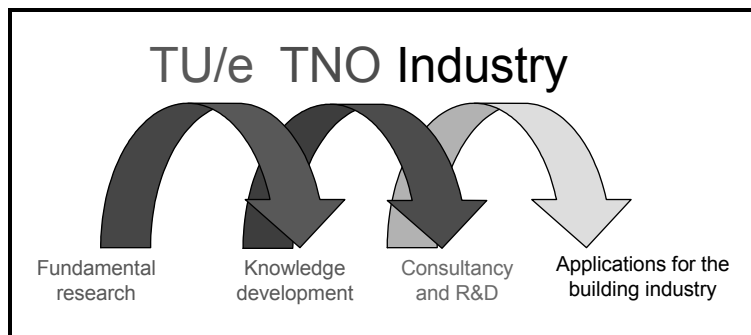


Figure 2. Cooperation between TUE and TNO in the context of the Center for Building and Systems

The ‘*initial set of objectives*’ for this PhD project were arrived at by considering the initial assumptions and the requirements of the stakeholders. These objectives are characterized as SMART according to the following definitions:

- Systematic – a support tool should provide a systematic procedure for concepts generation in the early design phase
- Multidisciplinary – a support tool should be domain independent and therefore be applicable in multidisciplinary teams
- Acceptable - a support tool should be approved by the stakeholders and be likely to be accepted by the designers with multidisciplinary backgrounds, e.g. students, practicing designers, researchers
- Relevant – a support tool should enable meaningful and useful operations with the product, which should facilitate the design process
- Teachable – the support should be acceptable for the design education community and be accompanied by a methodology and examples

² The website of the Knowledge Centre can be found at the URL address: <http://www.kcbs.nl/>

³ The TNO website can be found at the URL address: <http://www.tno.nl/>

1.3 Modeling approach

We limit our considerations of the early design phase to knowledge dynamics during this process. Therefore, the two purposes of this section are 1) to give a definition of knowledge and consider aspects of knowledge that we need to bring into our model and 2) to introduce a mechanism capable of explaining the dynamics of knowledge in a systematic way. The formalization of this mechanism forms the core of our modeling approach to describe, study and support the early design phase.

1.3.1 Knowledge management

We define the term '*knowledge*' using Knowledge Management science. In its domain, it is usual to distinguish levels of knowledge, i.e. data, information, understanding and knowledge [Nooteboom, 1996], where Knowledge Management provides support for each of the levels [Rosenberg, 2001].

- *Data*. "External sign material produced by events"
- *Information*. "Interpretation entails the production of meaning, which transforms data into information"
- *Understanding*. "Connects and transforms information into beliefs or claims of causal or deductive insight"
- *Knowledge*. "A meaningfully ordered stock of information (interpreted data), and understanding, plus ability to transform it into actions, which yields performance".

It is common to distinguish between several dimensions of knowledge, i.e. the social or organizational dimension and the cognitive dimension. These dimensions are summarized in the following definition of knowledge, "knowledge is information combined with experience, context, interpretation, and reflection" [Davenport, 1998]. Dimensions of knowledge are considered to be useful and capable of facilitating knowledge acquisition. However, at this point it is not yet clear which useful knowledge dimensions or structures could underlie knowledge in the early design phase. We therefore need to consider the domain of the early design phase in more detail; we do this in section 1.4.

The distinction between the explicit and implicit knowledge also brings some clarity into the application of the natural tools and knowledge acquisition. '*Explicit knowledge*' refers to knowledge that is transmittable in formal, systematic language. There is also '*implicit knowledge*', which is personal and therefore hard to formalize and communicate. The following phrase demonstrates the mechanism of formalization of knowledge, "we can know more than we can tell" [Polanyi, 1966]. According to Snowden [Andrews, 2002] there are three rules of Knowledge Management which govern knowledge discovery:

1. Knowledge can only be volunteered; it can never be conscripted.
2. We only know what we know when we need to know it.
3. We always know more than we can say, and we can always say more than we can write down.

Whereas explicit knowledge can easily be recorded in databases, it is more difficult to record implicit knowledge. This problem has been addressed by the expert systems experience. Making the knowledge more objective always means a transformation of the current knowledge [Hatchuel, 1992]. For this reason, the process of transformation of implicit knowledge into explicit knowledge is often problematic, because tacit knowledge is not structured before its extraction [Vinck, 1997]. We aim to address this problem by developing a knowledge structure and a formalism. One way of doing this is to consider linguistics, which addresses the problems of using natural languages.

1.3.2 Linguistics

Because language is used as a tool to explain thoughts to others, we can consider problems identified within the domain of linguistics. The main purpose of this section is to find a foundation on which we can base a formalism and propose improvements for the application of natural language in the early design phase.

Linguists are concerned with problems similar to design problems. The same text can be put in different forms that make the text easier or more difficult to understand. In a similar way as in design problems, linguists see written text as an ill-defined problem. The quality of the text does not only depend on the text itself, but also on the purpose of the text. Several equally suitable texts can be produced for the same purpose. Within the domain of linguistics, language is considered to be a tool to control behavior. Within our domain of study we consider language to be a tool for better communication and the systematic generation of consistent and convincing solutions. We present the linguistics' view as given in a number of essays written by Eugene Ionesco, the author of "The Bald Soprano".

- (Problem of translating thoughts into a language representation) "If we cannot even begin with a best way of representing the information contained in that problem, we are certainly no better off regarding any method of solving the problem. How do we go about determining what counts as an assumption, what assumptions are important, what kinds of critique are relevant? Where do we begin? Put more plainly, where is the step-by-step kind of approach that worked so well for us when we were adding⁴?"
- (Relation between thoughts and the language) "Because language and thought do not share a one-to-one relation, yet clearly influence each other, we might best describe them as loosely coupled. They are connected like two links in a chain, rather than fused like two pieces of metal welded together. Imagine, if you will, the way in which two chain links interconnect. If we pull on one link, the other will tend to be drawn in the same direction as the one it's connected to; however, even though it will match the first link's angle and position fairly closely, this other link may not arrive at precisely the same angle. In other words, the second link will approximate the position of the first within certain limits. Such is a useful way of thinking about the relationship between thought and language: they strongly influence each other, but with some room for play."
- (Limitations of natural languages) "Natural languages are extremely flexible and allow to express nearly any thoughts in terms of the sentences. Consider this:

⁴ Here the addition of numbers is considered to be a systematic procedure.

even if a person can and does have thoughts for which no words are available, to what extent are solitary thoughts useful? For example, most of us have experienced the frustration of trying to say something but being unable to "find the right words" to express what we mean. At that moment, how useful are those inexpressible thoughts?"

- (Limitations of artificial languages) "At present, artificial languages, such as those developed to program computers, cannot undergo natural change, and can only "express" functions or processes incorporated into their design. We may think similarly of mathematics, another artificial language developed expressly for solving certain problems and limiting the ways in which ideas may be expressed. In both cases, these artificial languages are expressly geared to preclude the kinds of debates over meaning and word change that characterize natural languages such as English." [End of the quote]

1.3.3 The mechanism of questions and answers

Both Knowledge Management and linguistics address the problem of translating knowledge from its implicit state to its explicit state. We therefore choose a mechanism that is capable of explaining this transition in a systematic way, so that it is suitable for further formalization and systematic usage. The purpose of this section is to explain our choice and provide a link to operations which we will define later on.

We have therefore chosen the mechanism of questions and answers. In daily life, both in professional and private settings, people are constantly asking questions and exchanging explanations. Asking questions and giving explanations are special instances of communication. According to Stempfle [2002], "in a heterogeneous group in which group members have different levels of understanding, it is highly likely that solution ideas will not be understood by everyone in the team right away. This will provoke questions, thus causing the group to go into analysis prior to evaluation". "If the group lacks a shared mental model, such a mental model must be built" [Klimoski, 1994]. "This building of a mental model will take place through questioning" [Mohammed, 2001]. "Even if questions are not meant to challenge a solution idea, but simply aim at filling in facts, thinking in detail about the problem may still cause previously unseen things to come up during the discussion" [Stempfle, 2002]. "Very few questions can be answered definitively in the early stage of the design process. Most answers immediately ask for follow-up questions" [Stempfle, 2002]. Here we aim to enhance natural way of asking and answering questions with a formalism and a knowledge structure.

In the previous section we considered natural languages and artificial languages as two extreme boundaries for the definition of a formalism. According to Veth [1987], knowledge can be represented by an amalgam of intuitionistic logic, modal logic, temporal logic, inheritance, and situational calculus. According to Mili [2001], "the level of abstraction and the expressive power of the modeling language used to describe decision-making processes dictate the ease with which multiple solution alternatives can be generated and the insight which can be obtained from them". Brown [1994] states that "effectiveness and efficiency of decisions are a direct function of the modeling language used to represent the decisions". Kleban [2001] says that "because so little is known about the product in early design, and because so many possibilities remain open

to the design team, the processes and information used in early design resist formal characterization and offer little structure for coordinating group activity”.

We argue that artificial languages are an aid in the early design phase; however, their application is limited due to their low flexibility and the complexity of directly transforming implicit knowledge into an explicit form. In our opinion, an intermediate step is necessary to connect flexible natural languages and precise artificial languages. We therefore aim to fill in this gap by finding a balance between natural languages and the strict formalism of artificial languages. From the previous section, we can conclude that a formalism needs to be positioned between natural languages, in order to enable sufficient flexibility, and artificial languages, in order to provide operational support. Another requirement is that such a formalism needs to be rooted in the domain of early design and must allow the dynamics of knowledge in this domain to be expressed. We will therefore consider the domain of the early design phase in the next section.

1.4 The domain of the early design phase

We introduce the domain of our study using the Product-Process-Organization model (the PPO model), which was developed in our group (see section 1.2). The PPO model is shown in Figure 3. We use this model to describe and to plan the life cycle of design situations. According to the PPO model [Friedl], four essential domains of concerns in the design situation can be distinguished: the ‘*product domain*’, the ‘*process domain*’, the ‘*organization domain*’, and the ‘*context domain*’. The dynamics of these domains constitute the life cycle of the design situation. We therefore need to define the knowledge content and the major characteristics of the dynamics for these domains during the early design phase.

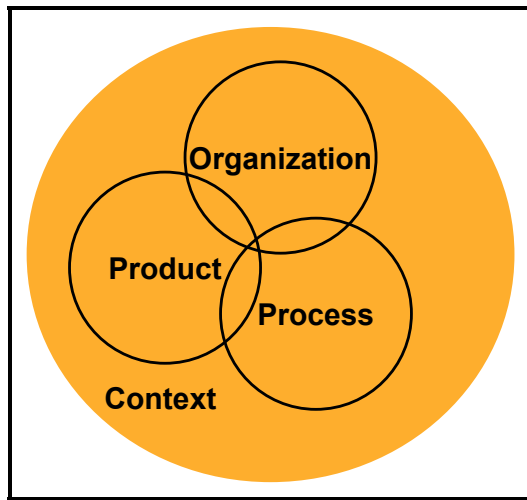


Figure 3. PPO model for the life cycle of the design situation

The organization domain contains people involved in the design process, as described in section 1.4.1. Here we will consider two types of people who possess implicit knowledge, i.e. the stakeholders and the designers. The product domain contains explicit knowledge about a product to be designed, which we will call the product. The product domain includes knowledge about the design problem and knowledge about currently considered solutions for the problem. This domain is introduced in section

1.4.2. In the process domain, we consider two paradigms that aim to explain the phenomenon of designing as described in 1.4.3. The context domain constrains the fixed world knowledge e.g. laws of the nature, domain knowledge, technologies, regulations⁵. We will not consider this domain separately; later on we will include this domain in the product domain.

1.4.1 The organization domain

We consider two types of participants in the organization, namely the team of designers and the group of stakeholders, who do not necessarily form a team. We assume that all the participants share common goals and therefore possess knowledge that is important, thus creating the necessity to communicate this knowledge. The purpose of this section is to consider the motives of communication; this will help us to understand how knowledge is updated.

Design team

We consider design teams that are composed of designers from multiple discipline domains.

- D. 2 (Designer) A designer is a participant of the organization who makes and evaluates design decisions about the future product.

Design methodologists generally agree that teams of designers with multidisciplinary backgrounds allow to enhance each other's vision and quickly compensate for any lack of knowledge. We assume that product knowledge is initially packaged implicitly in the participants, existing as a result of imagination, experience, knowledge about design theories, earlier design projects etc. In the course of the design process, the designers communicate and make decisions in order to identify, represent and connect chunks of knowledge that are relevant for the product.

Successful collaboration in a design team depends much more on the level of negotiation and agreement than the formal co-ordination and transfer of information between team members [Austin, 2002]. Some design decisions are the concern of more than one discipline, e.g. the design of environmental control is the concern of both the architect and the building service engineer. Substantial communication is therefore required to enable the design team to function. Lewis [1963] gave a precise description of an experiment in group communication. This experiment had shown how the members could be unaware of a misunderstanding that arose between them.

According to Kleban [2001], in the early design phases design teams are only partially formed. Manufacturing process engineers, reliability engineers, quality control personal and other specialists typically join the team as the project moves into later phases, e.g. the detailed design phase. Early design teams face more than just a problem of organizing large amounts of explicit knowledge: they must also deal with different human agendas, implicit assumptions and points of view.

⁵ We assume that contextual knowledge is fixed relatively to rapidly changing knowledge about the solutions. Contextual knowledge may change but on a larger time-scale, e.g. the time scale of a design phase.

Stakeholders

The stakeholders are another type of participant in the organization. They are people who express their needs and pay for the final product. Here the payment is not necessarily financial; it can also be a payment in time or in inconvenience. The designers may not know who all the stakeholders are beforehand⁶.

- D. 3** The *stakeholders* are the participants in the organization who influence the product in the following two ways: 1) stakeholders evaluate the product and make the final judgment about the solution for the product, 2) stakeholders constrain possible solutions for the product via decisions, which are fixed for the designers.

Stakeholders can be a part of the organization, but this is not necessarily so. Customers and users are examples of ‘*external stakeholders*’ who are physically not part of the organization. In the architectural design domain, the external stakeholders could be a commissioner, the legislative government and contractors.

One of the points where miscommunication often occurs is when the stakeholders’ needs are translated into a formulation of the design problem. In our domain of study, we assume that the needs of the stakeholders are not precise, i.e. the needs express *what should be done* and not *how it should be done*. In some situations the stakeholders can constrain a part of the solution for the product by making decisions about the future product. In such a case, the stakeholders can commit all possible solutions to a single engineering domain. The advantage of the design approach is that the needs can be looked at from a broader perspective, so that solutions can be found equally well in different engineering domains, in a combination of engineering domains, or outside any engineering domain. The early commitment to a certain range of solutions can introduce risk of overlooking simple but efficient solutions. This can result in a solution for the problem being overlooked for many years.

- Ex. 1** (Ferry disaster) As an example of a stakeholder decision, we now consider the Estonia ferry disaster [Killander, 1995], which happened in the fall of 1994 in the Baltic Sea. During a storm, the bow door of the ferry broke and water poured into the cargo deck. As a consequence, the buoyancy center of the ferry shifted, the vessel lost stability and sank. After the accident, the Nordic Marine Board decided to form sealed compartments in the cargo decks like in traditional ships, to ensure safety during the crossing. This need was expressed to several groups of engineers. In the following year several variations of the partitioning system were generated. However, all these solutions lead to the same problems, such as an inability to store bulky cargo and an increased cargo loading time due to the need to seal the compartments. The stakeholder’s decision constrained all the solutions to the mechanical principle of partitioning, and therefore to a single engineering domain. Later on, more advanced options were identified and presented in Killander [1995], such as partitioning using plastic foam, see Figure 4. During any emergency, this foam is injected into the cargo deck and prevents the access of water. This solution was independently identified some years later and successful research was carried out on its applicability in the TNO Bouw research laboratories [Bruijn, 2003]. This example illustrates how the premature definition of the design problem and commitment to the mechanical principle of partitioning excluded the consideration of other solution principles.

⁶ The designers may also overlook a stakeholder; identification of the stakeholders is one of the reasons for communication.

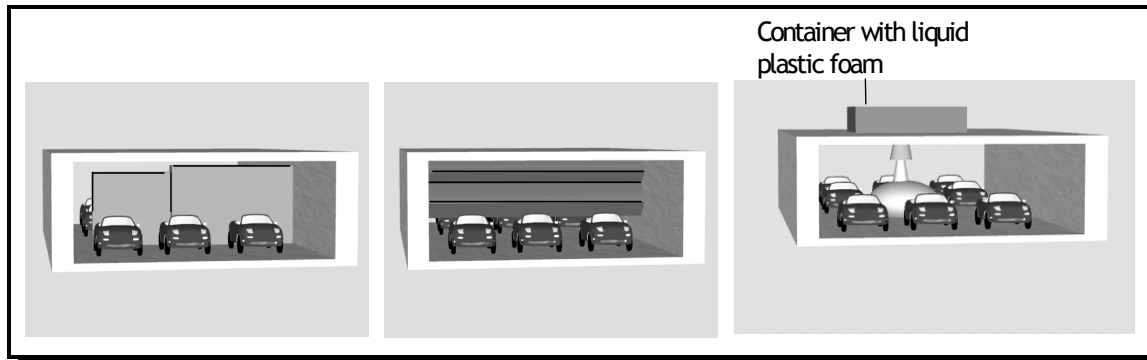


Figure 4. Various principles of partitioning

In design literature it is common to represent the stakeholders' needs using a set of '*objectives*' and '*constraints*'. According to Cross [1994], "the design objectives might also be called client requirements, user needs or product purpose. Whatever they are called they are the mixture of abstract and concrete aims that the design must try to satisfy or achieve". Because this mixture can be very complex, we have to distinguish between the needs that define the design problem from the needs that already suggest solutions for the problem. This is the topic of the next section.

1.4.2 The product domain

From the definitions of the early design phase, we identify two different ways in which knowledge about the product develops. Knowledge about the design problem seems to result from the stakeholders. Knowledge about the solutions seems to result from the designers. At this point a formal distinction that underlies these differences is not yet clear. The purpose of this section is to identify characteristics of knowledge about the product and thus to approach formalization of the participants' activities and of knowledge.

The design problem

The life cycle of any product begins with the needs of the stakeholders that are expressed as the design problem.

D. 4 (Design problem) The term '*design problem*' refers to the expression of the stakeholders' needs.

Design problems are considered to be 'ill-defined'. Characteristics of ill-defined problems are presented in Cross [1994]; Rittel [1984] describes ten properties of design problems. Here we present some of them:

- There is no definite formulation of the design problem. The information needed to formulate the design problem depends upon one's ideas of the design boundary.
- Design problems have no stopping rule.
- Solutions to design problems are not true or false, but better or worse. Various stakeholders may make different judgments about the design objectives.

In Table 1 we compare design problems with well-defined problems which occur in engineering.

Table 1. Types of problems

Problem	Formulation	Solution process
1) Well-defined	Mathematically defined	Systematic; all possible solutions are fixed within the boundaries of engineering domains or theories; fast convergence to a single best solution with the fixed product structure; the process is computationally supported
2) Ill-defined	Assumes causality, semi-unstructured, different formulations are possible	Iterative; adaptation of the available knowledge from different discipline domains; converges to several solution options; solutions may not belong to engineering domains or theories; very limited computational support; there is no single best way to proceed

Ex. 2 (Example of a design problem - The hats⁷) The first cinemas were introduced at the beginning of the 19th century. The owners of the cinemas faced a problem. The ladies of that time wore big hats that obstructed the screen from other people. The ladies refused to remove their hats, as this might jeopardize their beauty. This is an example of an ill-defined problem, as the problem can be formulated in many ways. For example, 1) how can you improve the visibility of the screen? Solutions could be to raise the screen or lower the front chairs. Another formulation could be 2) how can you keep the frustrated customers? The solution here could be to give customers without hats a reward or position these customers in the first rows. Yet another formulation of the problem could be 3) how do you encourage customers to remove hats? The cinema owners found an original solution to this formulation: at the entrance all visitors were met by the sign “Only elderly ladies are permitted to wear hats”.

In the early design phase, the product does not yet exist physically. In this thesis we therefore assume that the product is identical to the knowledge about the product. The design literature explains the appearance of the product as the simultaneous development of the design problem and solutions for the product. Design practitioners recommend formulating the design problem without formulating the solution to this problem. This approach does not restrict the initial range of possible solutions and removes biases that pull the designers towards the standard solutions. We consider this separation to be a strategy to reduce mental biases and to allow less-standard solutions to be considered. The stakeholders initially express their needs as the mixture of objectives and constraints, so that it is difficult to separate them clearly.

According to Cross [1994], “the level at which the problem is defined for or by the designer is crucial. There is a big difference between being asked to ‘design a telephone handset’ and to ‘design a telecommunication system’ ”. In this example the terms ‘a telephone handset’ and ‘a telecommunication system’ refer to the level of the solution, i.e. it is a stakeholder decision that constrains the range of possible solutions. In the second case, the constraint allows a wider range of solutions to be considered. The design problem is still not defined because the objectives are not expressed.

⁷ This example was presented by V. Timohov at www.trizland.ru (in Russian)

Solutions

We assume that the designers propose a number of solutions for the product. In Lawson [1980], the following two characteristics of solutions are described, “There is an inexhaustible number of different solutions and there are no single optimal solutions to a design problem due to a number of competing objectives, which represent the design problem”.

Multiple objectives that express the needs of the stakeholders are often competing objectives such as performance and cost. Such competing objectives are also called ‘*conflicting objectives*’, because designers can often not clearly see how to satisfy these objectives equally well with feasible solutions. The solutions need to be analyzed and explored to resolve conflicts. One well-known theory that provides techniques for the systematic resolution of conflicts is ‘TRIZ’ (Theory of Innovative Problem Solving), as described in Altshuller [1991].

If we consider only one of these objectives, the design process can be clearly defined as a search procedure aimed at a single optimal solution, since the solutions can be ordered. Indeed, each solution is clearly more powerful or cheaper than another solution. This fact helps the fast convergence of the search procedures. The situation is different if we try to order solutions with respect to several objectives at the same time. In this case, the solutions are only partially ordered, since two solutions can be ‘*indifferent*’ to each other; this is also called ‘*trade-off*’. For instance, cars are more powerful than bicycles, but bicycles are safer than cars. This clearly shows that the generation of solutions in the presence of multiple objectives adds a further level of complexity compared to the single-objective case. The participants may have personal preferences with respect to importance of objectives. As a consequence, the design process cannot only be seen as a rational search procedure. Multiple views on the design process become relevant. In the next section we will consider two views of the design process.

1.4.3 The process domain

There are several paradigms that aim to explain the nature of the design process. We will consider two of them. One is the ‘*General Problem Solving*’ [Simon, 1984] design paradigm (GPS) and the other is the ‘*Reflexive Practice*’ [Schön, 1991] paradigm (RP). GPS is characterized as a fully rational search process, and is therefore suitable for introduction of supporting tools in the design process. GPS assumes that the product structure exists objectively. For ill-defined problems, the length of the search path increases. The role of a designer in PS is as a ‘goal-seeking information processing system’, operating in an *objective* and knowable reality [Simon, 1992]. Solutions for the product are obtained in a fully rational and thus objective way. In our opinion, PS is not well defined in situations with multiple objectives; we therefore consider RP.

In RP every design problem is regarded as being unique. The designer is not simply searching for solutions, but imposes his or her personal view on the design situation by means of ‘*reflections*’ on the design situation: the designer names the relevant factors in the situation, frames the problem in a certain way, makes moves toward a solution and evaluate those moves [Schön, 1983]. Because multiple participants form design views, RP explains multiple formulations of the design problem and the appearance of divergent (contrasting) solutions.

Dorst [1997] considered both paradigms to be compatible and able to enrich each other. Both paradigms can be carried out systematically; in *GPS* this is done by means of systematic search procedures, in *RP* it is done using systematic reflections [Reymen, 2001]. We tend to consider the design process to be a combination of these two paradigms. We identified the mechanism of questions and answers as a mechanism to underlie the design process. Formal definition of activities that will execute this mechanism in a systematic way is not yet clear.

1.5 Outline of the results of this thesis

This project was started on the 1st of May 2000 and was finished on the 1st of May 2004. The results presented in this thesis were obtained in an iterative way. The formulation of the addressed design problem in section 1.1 was refined during the first year. The defined set of initial objectives in section 1.2 determined our modeling approach, as it was described in section 1.3, and limited the range of possible solutions to an operational model of the design process and a support tool for the early design phase. The model was developed during the first year, as presented in [Ivashkov, 2001]. We claim that the design process can be modeled in terms of concepts, attributes and values. We formally define these terms in the next chapter. During the first and the second years early experiments with the model were performed [Ivashkov, 2002]. The model was applied to teach design methods in postgraduate design courses [Overveld, 2003]. From the moment that we introduced operations involving generation of solutions and optimization, a support tool became essential; these operations are too tedious for the designers to carry out, and are not supported by standard tools [Overveld, Ivashkov, 2003]. The tool was developed during the second and the third years of this project. The name of the tool is derived from the names of the basic ingredients of the model, i.e. Attributes Concepts Constraints⁸ Evaluation Language (ACCEL).

The results of this thesis support concept generation from the definition of the design problem to the automated generation of optimal solutions and a support of convergence to few solutions. This is done through the following methodological steps (see Appendix C): design problem → product model → mathematical programming problem → genetic algorithm → optimal solutions → few (single) optimal solution(s). We present the results of this PhD project in the following order:

Chapter 2 presents an operational model of the design process. The developed model has two major characteristics. First, it is a systematic model, which is based on a state-transitions view of the design process. Second, it is a computational model, which allows optimal solutions to be computed. In this chapter, we present the developed formalism and explain how our model can be used to develop product models through the mechanism of questions and answers, and how to link the product model to an optimization procedure. We will consider the application of the genetic algorithms to multi-objective optimization problems.

Chapter 3 introduces ACCEL. We developed this tool according to the functional requirements that we derived from the model in chapter 2. ACCEL enables the automation of operations with the product such as an update of the representation, an

⁸ In the early stages of the project values were called ‘constraints’. The name ‘ACCEL’ was created at that time, and has not been changed for compatibility reasons.

evaluation of knowledge, multi-objective optimization and more. We present a development approach and describe the major design decisions which underlie the tool's syntax, the data-oriented model, the functionality of the tool and the graphical user interface. As a part of the optimization dialog, we present a method to converge to few optimal solutions without prioritization of the objectives,

Chapter 4 presents case studies, where we evaluate the developed support. In the first case study we use our approach to re-engineer an air-conditioning unit for a swimming pool. In this study, we focus on the methodology of ACCEL and contrast a re-engineered solution with the original one. We were able to show that our method gives a significant improvement over traditional design methods In the second case study we describe how students applied our approach to obtain optimal solutions for a vertical transportation system for the new World Trade Center in New York. In the third case study we present our early experiments with our approach before ACCEL became available.

Chapter 5 presents our results, recommendations and the epilog. We have been using ACCEL in post-graduate design courses for several years. We have seen that the developed support tool can be used to effectively teach and support the systematic generation of optimal solutions for a product in the early design phase. This enables students to learn various design approaches quickly, to consider more alternative solutions, and eventually to be better prepared for realistic design problems.

CHAPTER 2. AN OPERATIONAL MODEL OF THE DESIGN PROCESS

“We believe that only through a clean design theory and formalization can one arrive at a testable conjectures of design and build computer models of it”, Akman, Hagen and Tomiyama [1990].

2.1 Introduction

According to [Evbuomwan, 1996] ‘*design models*’ are the representations of philosophies or strategies proposed to show what a design is and how it can be achieved. As with any other model, our model has assumptions (section 2.1.1), an intention (section 2.1.2), and a story that connects a formalism of the model with the intention. To better understand the story, we first need to define the scope of our model; we do this in section 2.1.3. The remainder of the chapter is outlined after the introduction of the scope.

2.1.1 The basic assumptions

The basic assumptions about the design process are as follows:

- Knowledge is distributed over the participants’ minds and multi-modal representations, e.g. notes, emails.
- Knowledge *can be structured*, i.e. represented in terms of elements and relations between the elements, and given a systematic procedure in terms of smaller and formally defined incremental steps.
- There are various advantages of imposing such a structure. These advantages include operational support, easier communication, less confusion, well-considered decision-making, stimulation of a more systematic generation of solutions.

2.1.2 Intention of the model and the outline

We formulate the basic intentions of the model in the following steps:

- Define state-transitions of knowledge (section 2.2).
- Define a formalism for generating concepts (section 2.3).
- Facilitate the development of a product model and define functional requirements for a support tool (section 2.4).

- Formalize and support reflections (section 2.5).
- Formally define operation with knowledge (section 2.6).

2.1.3 Position of the model

In this section, we define the scope of an operational model. For this purpose we use some dimensions that we obtained from the literature and the PPO model from the previous chapter. Three purposes of design models are known within the design research field [Evbuomwan, 1996]:

- Descriptive models are concerned with designers' actions and activities during the design process. These models usually emphasize the importance of generating solution concepts early on in the process.
- Prescriptive models tend to look at the design process from a global perspective, covering the procedural steps.
- Computational models put an emphasis on the use of numerical and qualitative computational techniques and artificial intelligence techniques, combined with modern computing technologies.

Given this classification of models, we define dimensions to describe the scope of various design models. We did not encounter an identical list of dimensions in the literature and therefore present it as one of our results.

D. 5 (Dimensions of design models)

- The 'purpose of a design model' can be descriptive or prescriptive.
- The 'field of a design model' can be in the product domain, in the organization domain or in the process domain.
- The 'formalism of a design model' can be computational (formal) or non-computational (informal icons, diagrams etc.). Formalism of computational models is well defined and enables operational procedures, which can be automated. Non-computational models do not introduce a well-defined formalism and therefore do not enable computational procedures.

The following list of values defines the scope of our model (also see Figure 5):

1. The purpose is both descriptive and prescriptive. We aim to both give a systematic description of the design process, and prescribe a formalism for knowledge build up, that we will couple with the design process.
2. The field is both in the product domain and in the process domain, since we consider dynamics of knowledge about the product, which results from the design process via the participants.
3. The model is operational, because we formally define operations with knowledge and demonstrate their support on the computer.

Note that our model does not prescribe the sequence of steps in the design process. It is systematic only with respect to representation of steps already taken by the designers. Our model is therefore compatible with existing design models and methods. We did observe a positive correlation between the systematic representation of steps in the design process and the design process itself. Given an overview of product knowledge,

more consideration can be given to planning of future steps in the design process. The model does not consider organizational aspects of the design process, such as the number of team members or their roles. Our model is computational with respect to the design process, as it allows the previous sequences of incremental steps to be kept during the process. The sequence of steps can be played back so that it becomes possible to compute the previous states of the design process and to reconsider the previously made decisions. The major benefits of the model are computational operations with the expressed product knowledge.

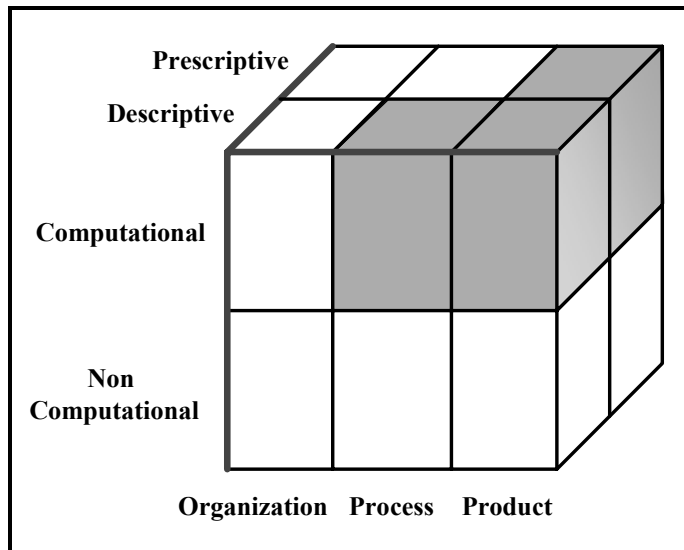


Figure 5. The scope of the operational model

We present our model in the following order. In section 2.2 we present the descriptive purpose of the model, which gives a systematic view on the design process and the relationship between the design process and knowledge build up. This section introduces a structure for product knowledge. In section 2.3 we derive a formalism for concept generation from the structure. In section 2.4 we describe how application of the formalism leads to development of product models. In section 2.5 we apply our formalism to describe a Questions Answers mechanism, and define eight ways of answering questions in design. We use these ways later on to define functional requirements for a support tool. In section 2.6 we define computational operations with the expressed knowledge. Finally, in section 2.7, we describe how a genetic algorithm enables these operations and leads to the generation of the set of optimal solutions for the design problem.

2.2 Descriptive purpose

We introduce a state-transition model of the design process in section 2.2.1. In section 2.2.2 we introduce a structure for product knowledge.

2.2.1 State-transition model

This section aims to describe the appearance of product knowledge in a systematic way. We introduce a well-known notion of the '*state of the product*' and that of the

‘transition’. In the field of design, the state of the design product and the state of the design process are usually distinguished [Reymen, 2001]. Here we introduce a somewhat unusual view of the state of the product. The usual assumption is that only design activities can cause transitions of the product from one state to another. This assumption excludes transitions of the product due to automated operations. Therefore transitions in the design process due to the reflections (section 2.5) are considered separately from transitions due to automated operations (section 2.6). Both types of transitions (operations) will be described using the same formalism, which makes the expressed knowledge understandable as for the designers as for representation in the computer.

The *state of the design process* PS_i refers to information about the design process at a particular moment in time, i.e. the current focus, the current state of the design method being executed etc. It is useful to distinguish the states of the design process, since they allow to track the progress and to come back to the previously made decisions. Transitions in the design process cause transitions of the product from one state to another. In our model the *state of the product* KS_i refers to the state of the available knowledge about the product at a particular moment in time.

The states of the design process and the states of the product are synchronized by regular translation of the most recent design activities into product transitions. Consider the state of the design process PS_i and the state of the product KS_i at some moment in time, as is depicted in Figure 6.

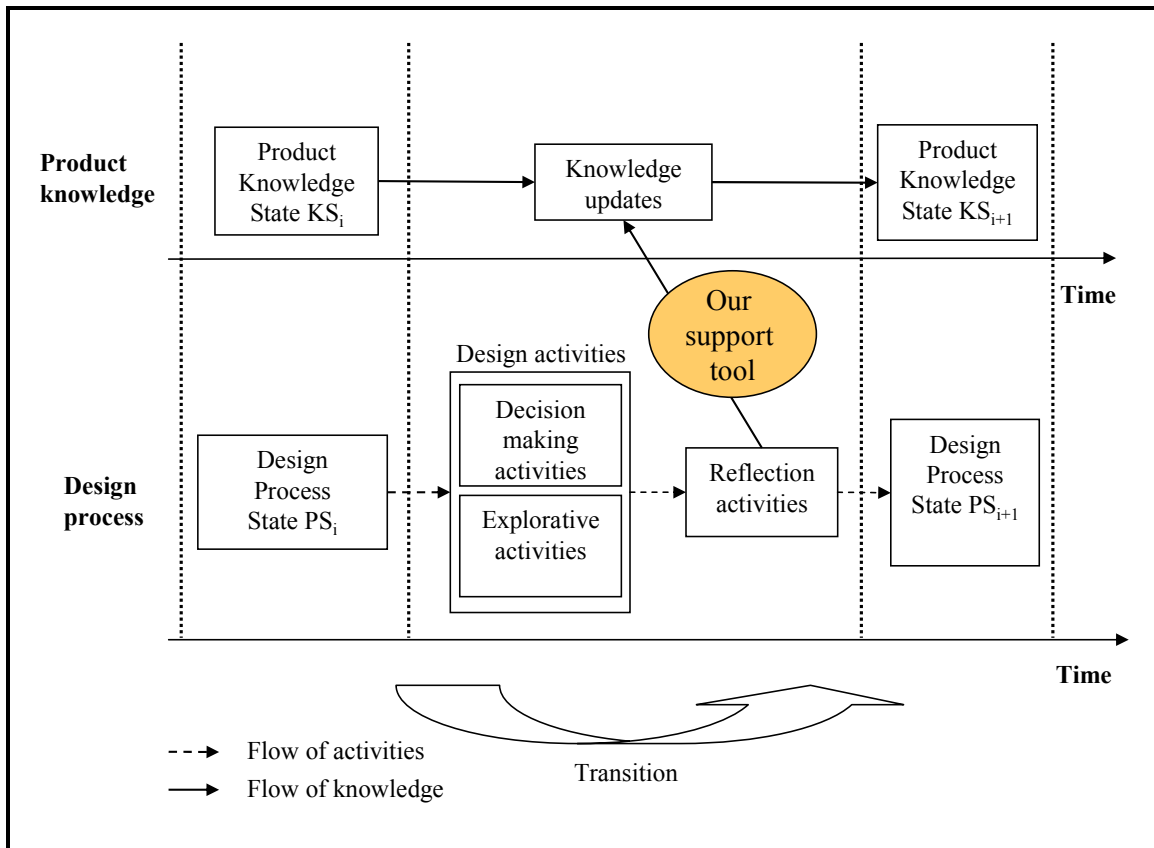


Figure 6. Relationship between the state-transitions of the design product and of the design process

Designers perform activities for some time, which can be prescribed by a design method or by common sense. At some point in time, designers stop the design activities and start to reflect (reflection activities) on the design activities performed since the PS_i . This defines a new state in the design process PS_{i+1} . By reflecting, designers aim to translate the recently performed design activities in terms of knowledge updates, which brings the product state KS_i into a new state KS_{i+1} . By modeling all the subsequent states of the design process and the product in a similar way, the design process can be seen as a sequence $PS_1, PS_2, \dots, PS_i, PS_{i+1}, PS_{i+2}, \dots$; the product can be seen as a sequence $KS_1, KS_2, \dots, KS_i, KS_{i+1}, KS_{i+2}, \dots$, where there is one-to-one mapping between PS_i and KS_i ⁹.

Our support tool will be used during the moments of reflection; it is not a tool that will be used for long periods of time, but systematically for short intervals. Our approach is complementary to other methods, such as other tools, imagination, communication and decision-making. Although we have seen that the systematic application of our support tool enables new, useful design activities, the main focus is the improved use of currently available explicit knowledge. In section 2.5 we prescribe how clarity of reflections can benefit from the formalized mechanism of questions and answers.

2.2.2 A structure for the product

In the introduction to this chapter we discussed the motivation for imposing a structure on product knowledge, such as the easier translation of implicit knowledge to explicit knowledge, the clarity of communication and the automatic generation and evaluation of solutions. In our model we therefore aim to identify different spaces of knowledge. The idea behind the representing product knowledge by means of spaces is based on General Problem Solving (GPS) paradigm, which was introduced in section 1.4.3. GPS allows us to deal with well-defined problems and we therefore used it as a starting point when defining our formalism.

Newell and Simon used the term ‘*problem space*’, defining it as the metaphoric space in which the problem solver moves from an initial state of the problem through intermediate states to a desired state by applying appropriate operators [Newell, 1972]. The problem space consists of all knowledge related to the problem, including knowledge about the solutions, the context, and the goals. Newell and Simon write, “In sum, we need to describe the space in which problem solving activities take place. This space is called therefore the ‘*problem space*’”. Different problem spaces can be defined for the same problem. The problem space is used to search for the solutions. In GPS the term ‘solution’ can refer to the sequence of expressions, the sequence of operators, or the terminal state of the process, e.g. a crossword puzzle. In GPS the problem and solutions are not clearly distinguished from each other, for example where the final state of the problem is the solution. According to Newell, the set of operations for ill-defined problems is not defined; we are therefore not able to define operations for ill-defined problems. In our research we aim to model operations (activities) of the designers by means of several categories of operations, which are specific to the design process. The major benefit is that based on these categories we distinguish between several knowledge spaces. These spaces lead to computational benefits, as it will be explained later on.

⁹ This is not always the case. Designers can also reflect on design activities related to organizational aspects.

We encountered some recent attempts to distinguish between knowledge spaces in the field of design. For instance, Puro [2001] gives the following motivation for the introduction of the design space, “development of a ‘solution’ i.e. a design, proceeds through proposals, expansions and challenges. These activities clearly represent moves within the design space and continue after an initial ‘goal’ state is reached – suggesting the existence of a separate design space”. The authors claim that activities in the problem space and the design space clearly represent a dimension orthogonal to the different activities that a designer may undertake. The authors mention that, “The significant overlap between the real world objects and the objects-modeled also lead to blurring of boundaries between the problem and the design space.” A possible reason for this ‘blurring’ could be an unclear definition of the spaces. Analogously, we aim to define the spaces based on the activities performed in these spaces and show that these spaces are orthogonal by imposing an orthogonal structure.

We are able to distinguish four different categories of operations and it is therefore meaningful to distinguish four different knowledge spaces. We propose the following knowledge spaces:

- the decision space
- the objective space
- the context space
- the auxiliary space

Formalization of these spaces is one of our major contributions. To clarify these spaces we consider an example of the design of a heating system for a house.

We define the **decision space** to consist of all the options the designers can consider to provide heating, such as different types of radiators, their possible positions, the available manufacturers, various delivery schedules and different maintenance options. Knowledge in this space appears from the imagination and existing technical possibilities. Knowledge in the decision space is operated by means of *decisions*, i.e. the designers’ can choose between these options and combine them into solutions.

Knowledge in the **objective space** is the result of evaluating the available solutions. This knowledge reflects the preferences of the stakeholders. For instance, the inside temperature should be within the preferred range; the energy consumption should be minimal. Constraints also fall in this space. The stakeholders may prefer heating by means of radiators as opposed to heating by the hot air-flow. The objective space is therefore meant to enable the ordering and selection of the solutions, while all knowledge is operated by *evaluation* procedures.

The outcome of a decision may depend on contextual parameters, which constitute the **contextual space**; for instance, the climate, the efficiency of radiators, and the type of the actual building could play a role. It is necessary for the designers to collect some contextual knowledge in order to evaluate their solutions. Knowledge in the contextual space is relatively fixed and exists objectively, because it is obtained by means of *measurement* operations.

Knowledge in the **auxiliary space** provides intermediate knowledge, which is necessary for the evaluation of the objectives. The designers or the stakeholders are not interested in this knowledge in itself. For instance, the evaluation of energy consumption and of the inside temperature is easier if the power of the radiators and the energy losses are known. Such auxiliary knowledge will require knowledge about the solutions and about

the context, such as the climate. The purpose of the auxiliary knowledge space is therefore to connect knowledge in the decision space and in the context space with the knowledge in the objective space. We can say that knowledge in the auxiliary space is operated using *modelling* (see Figure 7).

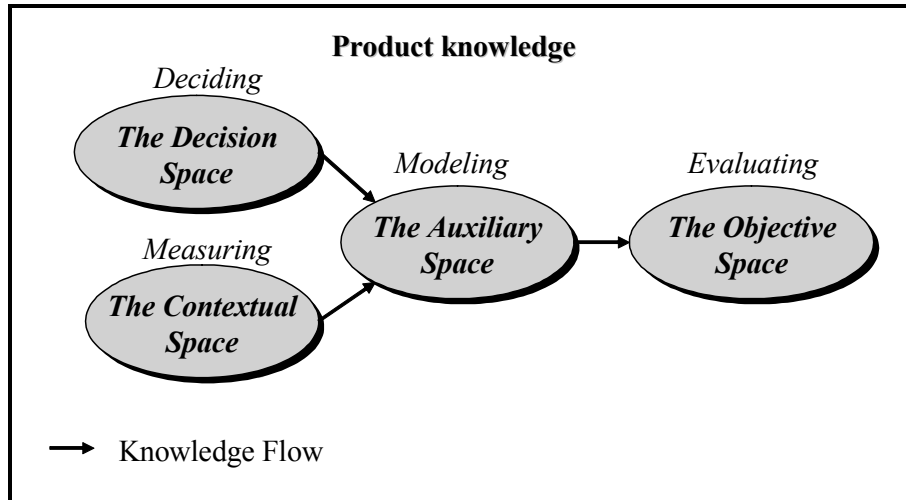


Figure 7. Interrelations between the four knowledge spaces of product knowledge

2.3 A formalism for product knowledge

“In general, whatever you are trying to learn, if you can imagine trying to explain it to a computer, then you learn what you do not know about the subject. It helps you ask the right questions. It is the ultimate test of what you know.” - Knuth D.E.

2.3.1 Introduction

It is certainly not easy to formalize implicit knowledge. The direct formalization of implicit knowledge into its mathematical form is beyond most people and it is likely to stifle natural way of working, creativity and communication. Formalization in terms of natural languages is easier and more flexible, but this flexibility is accompanied by imprecision and our continued inability to provide computational support. In this section we introduce a formalism that aims to provide a smoother translation from the designers’ thoughts to a formal representation. The characteristic feature of our formalism is that it resembles the way knowledge is stored in the human memory. Therefore our formalism is rooted in cognitive psychology. This is a necessary step for development of an acceptable support tool, which deals with knowledge. Snowden in [Andrews, 2002] addresses this step from the Knowledge Management point of view: “We see technology as a tool: If you pick it up and it fits the hand, then it’s useful. If you have to bio-reengineer your hand to fit your tool, it’s a waste of time. Too many Knowledge Management solutions – and a lot of Artificial Intelligence- require us to change the way we think to conform with a mechanical or process view of the world. Human beings don't work like that.”

Our contribution is that we provide the mathematical meaning to the formalism. The application of our formalism leads to a more compact representation of knowledge. In this sense the complexity of applying our formalism is comparable to writing a summary; it is not an easy task but it leads to the development of product models and computational operations such as the automated generation and evaluation of solutions.

Object-oriented formalism [Booch, 1993] is the most comparable to our own formalism. The direct application of the object-oriented formalism in the early design phase is problematic for a number of reasons, which we consider below.

- R.1 (Application of object-oriented formalism in the early design phase) Object-oriented formalism has been applied since the 1970s, to manage the complexity of large systems. According to Jones [1979], in the object-oriented approach the main accent is put on specific characteristics of a concrete or abstract system. An object possesses integrity which cannot be affected. The properties and the behavior that the object possesses are unchangeable. In the early design phase it is a serious limitation; ideas, which can be seen as objects of thoughts, are often combined with each other. In our formalism we will use the term ‘concept’ instead of ‘object’. The term ‘concept’ is more appropriate to describe vague and rapidly changing ideas in the early design phase. Our formalism could therefore also be called a ‘*concept-oriented*’ formalism, which is dedicated to the early design phase.

We will gradually describe our formalism in a sequence to show how a concept for the product could be generated. Knowledge related to the product is initially packaged in large and implicit chunks of knowledge. Our formalism enables this knowledge to be packaged so that it can be manipulated and analyzed, and will eventually enable the generation of solutions in the computer.

We will define product knowledge, i.e. the product domain P , by giving formal definitions of the four knowledge spaces. The decision space V_D will represent the knowledge of possible decisions, the objective space V_O will represent the knowledge of the stakeholders’ needs, the context space V_C will represent the context of the product, and the auxiliary space V_A will represent the intermediate knowledge. In section 2.3.2 we introduce the basic ingredients that will be used to define the space V_D in section 2.3.3, the space V_O in section 2.3.4, the space V_C in section 2.3.5 and the space V_A in section 2.3.6. As this is the most fundamental section of the thesis, we summarize our results in section 2.3.7.

2.3.2 Basic ingredients of the formalism

We will represent any chunk of knowledge in each of the spaces V_D, V_O, V_C, V_A in terms of concepts, attributes and their values. These terms are well-known in the fields of cognitive psychology and design, they are frequently used, and they are domain independent. Our contribution will be two fold: 1) to increase the precision of these terms and to relate them to well-defined mathematical terms such as ‘variables’ and ‘functions’ 2) to relate these terms to the four knowledge spaces.

The set of concepts

There is a considerable body of knowledge about the laws and processes of concept generation (formation). According to [Solso, 1998] the early definition of ‘*concept*’ was “mental images, ideas or processes.” For our purposes we use the word ‘*concept*’ in a generic way to refer to anything that is of interest to the designer(s): a solution, a

solution feature, a requirement, a stakeholder etc. A concept may refer to any issue that has or needs a name and that has to be formally defined or distinguished later. Many people in the field of design associate the word ‘concept’ with the solution concept, therefore it can be confusing for them to refer to a customer as a concept. However, the advantage here is that a customer is not excluded from the range of solutions to the problem. In the hats example in the previous chapter (see Ex. 2), the problem was solved by changing the attitude of the customers; the customer was a part of the solution concept. The concepts could also be called ‘things’ or ‘objects’. However, these terms imply some physical meaning and therefore carry certain mental biases.

- D. 6** (Concept) A concept may refer to any issue which has to be distinguished or defined. We will denote concepts as c_i , where i is the unique index of a concept. Each concept can also be referred to by its unique name as an alternative to referring to by the index. We will identify the names of the concepts typographically by underlining them.
- D. 7** (Set of concepts) The set $C : C = \{c_i\}$ is the set of concepts related to the design product that is currently being considered.
- R.2** (Sets) The notion of the set is one of the fundamental notions of mathematics. We will define a set of concepts to be any defined aggregate (collection) of concepts. The concepts of a set are called the elements of the set. We use the notation $c \in C$ to show that an element c belongs to a set C . Elements of the set are such that they can be distinguished from each other. Although it seems to be intuitively clear, if we look more closely it is not clear how to distinguish between elements, e.g. are α and a two different elements? Without additional procedures, it is also no trivial task to decide whether an element belongs to a set or not. Is 4562934529347 a prime number or not?

The set of solution concepts

In this section we will distinguish the solutions concepts from the rest of concepts. We assume that the design process starts with an initial set of ideas for the solution, which may be incomplete or infeasible. At a given moment in the design process, the set $S = \{s_i\}$ is the set of all currently considered solutions to a problem. Consider the following example, which we will develop in the course of this chapter.

- Ex. 3** (Transportation) In order to design a new product for transportation we generate the following initial set of solutions: $s_1 = \underline{\text{Freight train}}$, $s_2 = \underline{\text{Bicycle}}$, $s_3 = \underline{\text{Conveyor belt}}$.

Each initial solution s_i , and in fact any concept, starts off without any explicitly-represented knowledge; it does have a clear, implicit, non-articulated interpretation because it can be named and people can imagine what such a concept means. In other words, there is a large amount of implicit knowledge that is packaged in each of the suggested concepts. Useful knowledge needs to be identified and transformed into explicit knowledge. We do not yet know any model that would help us to distinguish between meaningful and non-meaningful knowledge about the concepts. The next step is to make elements of a possible knowledge structure about the concepts visible.

The set of attributes

Knowledge about concepts can be represented in different ways. Within the field of cognitive psychology, five major models of representing knowledge in the human memory are recognized [Solso, 1998]. We will use the ‘*Set-Theoretical Model*’ as a basis [Meyer, 1976]. This model is used as a basic model for several other models of

representational knowledge, e.g. '*Semantic Feature-Comparison Model*' [Smith, 1981]. According to the Set-Theoretical Model, concepts are represented in memory as sets of elements, or collections of information. The set can include instances of a concept (for example, the concept 'furniture' can include the instance of chair, table, couch) and also attributes, or properties, of a concept (for example, 'wood', 'sit'). According to Soslo [1998], many experiments were done to demonstrate that word meanings are represented in the human memory as a bundle of attributes [Wickens, 1970].

For any concept c_i , that the designer is explicitly aware of (i.e. a concept that can be named), we assume that all relevant knowledge that is contained in c_i is operated via one or more attributes that are meaningful for c_i . We will use attributes to generate knowledge about the concepts, since attributes make the elements of knowledge about concepts visible and operational. We therefore model attributes as functions that allow to operate with knowledge about the concepts, i.e. the presence of attributes makes generating concepts an operational procedure. We did not encounter elsewhere this interpretation of attribute, therefore this is our result.

- D. 8 (Attribute) An '*attribute*' a_j is defined as a function that returns a value v_{ij} depending on its argument, where this argument is a concept c_i ; $c_i \cdot a_j = v_{ij}$.
- D. 9 (Value) A value has a meaning of information, i.e. interpreted data.
- D. 10 (Operation) An *Operation* q_{ij} : $q_{ij} \in Q$ is an action directed towards obtaining of a value v_{ij} .
- D. 11 (Set of operations) $Q = \{decide, evaluate, measure, model\}$ is the set of operations with knowledge.
- D. 12 (Variable) The application of an attribute to a concept defines a variable $\langle c_i \cdot a_j, q_{ij} \rangle$.
- D. 13 ('Dot' notation) 'Dot' notation is equivalent to the functional notation used in mathematics, namely $a_j(c_i)$. 'Dot' notation is also symbolic. We represent variables graphically using a notation similar to the 'dot' notation

We will denote attributes using an index, which allows different attributes to be identified. The index also acts as a history. Attributes with a higher index are added later than attributes with a lower index (the same holds for the indexes of concepts). We assume that every attribute has a unique name, so that it is clear what the attribute means. We will identify the names of the attributes typographically by underlining them and placing a dot in front of the names.

- D. 14 (Set of attributes) The set $A : A = \{a_j\}$ is the set of attributes related to the design product, that is currently being considered.

The meaningfulness of attributes for a concept is subjective and cannot be formally defined. We represent this subjectivity using the notion of a '*signature set*'.

- D. 15 (Signature set) For a concept c_i we have the signature set $\phi(c_i) \subseteq A$ which is the set of all attributes that for any $a_j : a_j \in \phi(c_i)$ the designer can define q_{ij} .
- Ex. 4 (Signature set) For the concept Bicycle the signature set might contain the following attributes: $\phi(\text{Bicycle}) = \{ \text{.Height}, \text{.Number of speeds}, \text{.Type of Tires} \}$. For the concept Freight Train the signature set might contain the following attributes: $\phi(\text{Freight Train}) = \{ \text{.Power}, \text{.External Voltage}, \text{.Weight} \}$. Note that concepts can have different signature sets. For instance Type of tires $\notin \phi(\text{Freight Train})$ but Type of tires $\in \phi(\text{Bicycle})$. Note that definition of the signature set stimulates creative thinking. Some attributes may initially be not elements of the signature.

In a similar way, we define a set of concepts that are meaningful for an attribute using the notion of an ‘*extension set*’.

D. 16 (Extension set) For an attribute a_j we have the extension set $\psi(a_j) \subseteq C$ which is a set of concepts to which an attribute a_j can be applied by means of an operation q_{ij} .

Ex. 5 (Extension set) For the attribute .External Voltage the extension set contains the following elements: $\psi(\text{.External Voltage}) = \{\text{Freight Train}, \text{Conveyor Belt}\}$. For instance, Bicycle $\notin \psi(\text{.External Voltage})$.

In our approach, we assume that any concept c_i is described using a name, the tuple¹⁰ of variables with the corresponding tuple of values and the signature set $\phi(c_i)$:

D. 17 (Concept) $c_i = \langle \text{name}, (c_i.a_0, c_i.a_1, \dots, c_i.a_j), (v_{i0}, v_{i1}, \dots, v_{ij}), \phi(c_i) \rangle$, where $c_i \in \psi(a_j)$ and $a_j \in \phi(c_i)$ for all j .

We assume that any attribute a_j can be described by means of a name, the range R_i (which determines the attribute type as defined in the next section), and the extension set $\psi(a_j)$.

D. 18 (Attribute) $a_j = \langle \text{name}, R_i, \psi(a_j) \rangle$.

Consider the differences between the notion of the variable that is used in mathematics, the notion of the attribute and the notion of the value. In mathematics, computer languages, or engineering the variable has a name, for instance ‘ x ’, which can be assigned a value, e.g. ‘ $x=2$ ’. In our formalism, a *variable* always refers to some attribute a_j of a concept c_i , which we will denote as $c_i.a_j$. Therefore, in our notation a variable does not obscure the concept. Every variable needs to be assigned a value, in a similar way as it is done in mathematics (see Figure 8).

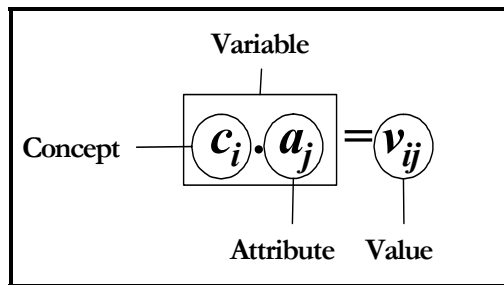


Figure 8. The difference between a variable, an attribute and a value

The assignment can be one of the four operations: ‘decide’, ‘evaluate’, ‘measure’, ‘model’. The values of each variable therefore can be obtained by one of the four operations that determine the knowledge space for a variable and the value. Using the introduced terminology we can give a definition to the term ‘*knowledge space*’. We will represent all knowledge about concepts $c_i; c_i \in C$ in a knowledge space V by means of tuples of variables $c_i.a$ with the corresponding tuples of values v_i and a operation $q \in Q$ that is used to obtain the values.

D. 19 (Knowledge space) $V = \langle \{(c_i.a, v_i)\}, q \rangle$, where $a \subseteq \phi(c_i)$.

We will come back to this issue in sections 2.3.3 to 2.3.6.

¹⁰ Tuples are also called vectors. Strictly speaking this is wrong: vectors are elements of a vector space, and tuples do not generally satisfy the conditions that are imposed on the vector space.

Attribute types

The range R_j contains all the allowed values that can result from the application of an a_j to any concept c_i . This means that any value v_{ij} is such that it is an element of the range R_j .

- Ex. 6** (Range) Note that a range is associated to an attribute, not to a variable. So even though pen is typically not very long i.e. the value of the variable `pen.length` may initially be assumed to vary between 3 and 15 centimeters, the range of the length $R_{length} \in [0, \infty)$. We can extend the conventional subset of values and consider the whole range of the corresponding attribute; in some cases this can facilitate innovative thinking. For instance, a pen with a length of 20 meters could mean ... , a laser point. For this reason, identifying attributes of any concept is a way to stimulate creativity.

It is very important to distinguish between different types of attributes since they enable different relations among the values. We will identify and support four basic types of the attributes, as shown in Table 2. The types we distinguish are similar to those in Boer [1999].

Table 2. Types of attributes

Type	Description	Examples
Rank 1. NOMINAL (Names, Uniqueness)	Values cannot be put in order, and it does not make sense to talk about “greater than” or “less than”. Equality and inequality are the only meaningful relations between two items of nominal data. Nominal values should be mutually exclusive, and a set of nominal values can be exhaustive.	$a_j = \text{Taste}$ has a nominal range, where $R_j = \{\text{sweet, sour, salty, bitter}\}$. Note that numeric data can be nominal as well, e.g. Belgian or German postal codes.
Rank 2. ORDINAL (Rank order) Relation “ \geq ”	Ordinal scales identify ordered magnitudes, but they do not quantify exact differences between values.	The classic example is rank scores. Tennis player A may rank 13 th in the world, B may rank 22 nd , C may rank 70 th . All this says is that A is officially better than B and B is better than C; it does not say anything about how much better.
Rank 3. INTERVAL (relative values without zero) Relation “ \geq ”, “-”	Interval scales are numeric, and differences between sequential values are also meaningful.	Temperatures on the Celsius or Fahrenheit scales are interval variables: there is a physical quantity associated to the temperature difference 13 °C-10 °C, and the same quantity is associated to the temperature difference 73 °C-70 °C (namely, the amount of heat needed to cause the temperature difference).
Rank 4. RATIO (relative values and zero) Relation “ \geq ”, “-”, “/”	Ratio scales include absolute numeric measurements (total counts) of continuous or discrete phenomena.	A temperature of 6 °C is not twice as hot as 3 °C, but a weight of 6 kilograms is exactly twice as heavy as a weight of 3 kilograms. The Kelvin temperature scale, in which 0 is genuinely absolute zero, has a ratio type. In other words, there is no physical quantity that is twice as large for the interval 0 °C->6 °C compared to the interval 0 °C->3 °C, whereas there is one for 0 K->6 K compared to 0 K->3 K.

- R.3** We have ranked the types of attributes according to a number of measurable relations between any two values from a corresponding range. For the nominal type it makes no sense to talk about these relations, while for the ratio type we can even talk about the ratio between two values, i.e. the question of how many times one value is greater than the other is meaningful.

Structures for concepts

Although no systematic methods exist to guarantee completeness, a common practical approach is to design a structure for concepts in order to ensure some degree of completeness. There are numerous examples of this problem, such as generating and selecting alternative solutions for a design problem, identifying stakeholders and assembling a product portfolio for a company. Some examples of structures can be found in Appendix E.

We introduced the ‘attribute’ in order to operate knowledge about individual concepts. Because the same attribute can be applicable to several concepts, attributes can be used to distinguish concepts from each other, i.e. structure sets of concepts. In this section we define two types of structures, namely the Fully Orthogonal Structure (FOS) and the fully hierarchical structure (FHS). In our opinion the two types of structure can suite two different purposes. FOS focuses on differences between the concepts and tends to represent concepts as independent, i.e. orthogonal. Hierarchical structures are based on some relation between the elements of this structure, such that the relation is reflexive, asymmetric and transitive. FHS therefore tends to consider concepts as related. Because of these differences, FOS can be used to study differences between the concepts, i.e. differentiate concepts. FHS can be used to study relations between the concepts, i.e. integrate concepts. We will show later that both types of structures can also be used in combinations. Both types of structures can lead to generation of new ideas. We will demonstrate mechanisms of generating concepts in the next section, which is dedicated to the decision space. The two types of structures are formally defined below.

Consider a set of concepts Y and a set of attributes Z .

- D. 20** (FOS) We will say that $(a_j: a_j \in Z)$ and $(a_m: a_m \in Z)$ form a fully orthogonal structure on Y iff¹¹ $\forall (c_i: c_i \in \psi(a_j) \wedge c_i \in Y : c_i \in \psi(a_m) \wedge c_i \cdot a_j \neq c_i \cdot a_m)$.

We will say that orthogonal attributes represent ‘*dimensions of the orthogonal structure*’ on Y .

We will now define the hierarchical structure based on “is-a” relation between concepts, which is also called the ‘inheritance relation’.

- D. 21** (is-a relation) $(c_i: c_i \in Y)$ is-a $(c_n: c_n \in Y)$ iff $\phi(c_i) \supseteq \phi(c_n)$, and $\forall (a_j: a_j \in \phi(c_n): c_i \cdot a_j \subseteq^{12} c_n \cdot a_j)$, where a_j will be called hierarchical attributes.
- D. 22** (FHS) We will say that Z forms a fully hierarchical structure of Y iff the is-a relation holds for all concepts in Y .

¹¹ iff is the abbreviation of ‘if and only if’.

¹² The relation “ \subseteq ” holds in cases of single valued attributes.

2.3.3 The decision space V_D

All knowledge contained in the decision space V_D is defined by the set of decision variables together with their values.

- D. 23 (Decision variable) A decision variable $\langle s_i, a_j, \text{decide} \rangle$ is a variable such that it constitutes a decision for which the designer has full authority. Thus the values of decision variables are operated by decisions.

Given this definition, every solution $s_i: s_i \in \mathcal{S}$ can be represented in the space V_D by the tuple of decision variables, which we denote as s_i^d . We will refer to the tuple of values that correspond to s_i^d as 'decision values' and will denote these values by v_i^d . The formal difference between s_i and any other concept c_k is that s_i has at least one decision value, while c_k has no decision values.

- D. 24 (Decision space) Decision space V_D is a knowledge space such that $V_D = \langle \{s_i^d, v_i^d\}, \text{decide} \rangle$.

In the following two examples we describe how the decision space can be structured. The structuring of this space results in useful, systematic design activities for the following reasons:

- (Identification of decisions) Solutions that are initially distinguished from each other intuitively become distinguished explicitly by decision values. Comparison of the solutions makes it easier to identify new attributes, which allow to define new decision values. If there are two identical solutions for the same tuple of decision values, but the solutions are intuitively different, then new decision values need to be found.
- (Generation) Segments of the decision space, which have no corresponding solutions, become visible. This result can be used to facilitate a brainstorming session to find new solutions that correspond to empty segments. The empty segments can be considered systematically.

- Ex. 7 (Application of orthogonal structures) We have introduced a number of solutions for the transportation product (see Ex. 3), which resulted from a brainstorming session. In order to impose FOS on the obtained solutions in the space V_s we need to introduce a number orthogonal attributes which would be meaningful for the solutions and would therefore produce new tuples of decision values. New attributes can be introduced by identifying the differences between the considered solutions. We can propose the following orthogonal attributes: $a_1 = \text{Energy source}$ with the range $R_1 = \{\text{diesel}, \text{electricity}, \text{human power}\}$ and $a_2 = \text{Media}$ with the range $R_2 = \{\text{air}, \text{ground}, \text{sea}\}$. The application of the defined attributes to the found solutions results in tuples of decision variables and the corresponding tuples of decision values. Each solution can be positioned in the FOS according to the tuple of decision values, as shown in Figure 9.

- Ex. 8 (Manual generation of solutions) Consider some empty segments in V_D . For instance the tuple $(\text{diesel}, \text{sea})$ inspires the new solution Boat, the tuple $(\text{diesel}, \text{ground})$ inspires the new solution Car. For $(\text{human power}, \text{air})$ there is no straightforward feasible interpretation. Designers therefore need to interpret what such a combination means. In this way the orthogonal structure inspires creative imagination. Indeed, it was long thought that it would be impossible for a human being to develop enough power to fly, until a solution which we will refer to as Manned flight, was actually built. Another solution for this segment is Parachute. Note that if we add both Manned flight and Parachute the structure is not fully orthogonal, because within this structure Manned flight = Parachute.

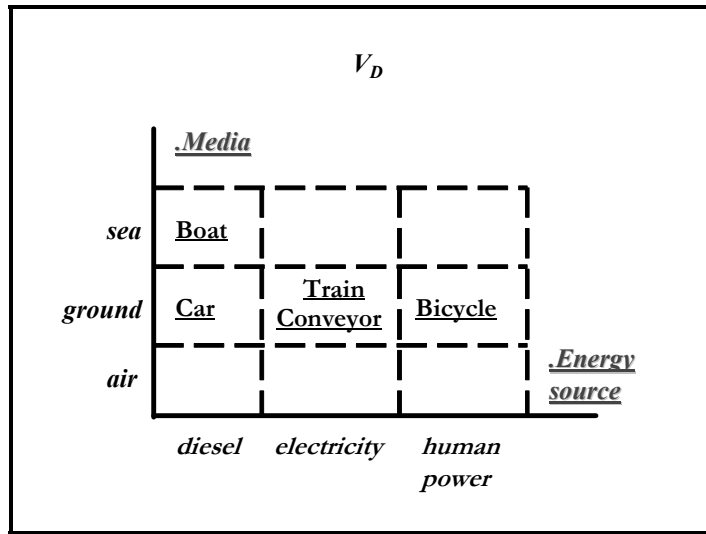


Figure 9. Structuring the decision space V_D using design attributes

- R.4** (Morphological box) The design literature refers to the described method of systematic generation of solution alternatives as the ‘morphological chart method’ or the ‘morphological box’. This method was first introduced by Zwicky [1969], and is one of the earliest systematic design methods. It is still regarded in the literature as one of the most useful methods in the early design phase.

For the reasons explained earlier it is advantageous to find orthogonal attributes, since this allows us to systematically and quickly focus on promising (or empty) regions in the decision space. In the following sections we will also demonstrate a computational usage of orthogonal structures. Orthogonal structures are not easy to build because they require identification of attributes applicable to all concepts that need to be structured. In order to address decision variables of specific solutions, it is often easier to impose a fully hierarchical structure on the set of solutions. A typical way to build an is-a hierarchical structure is to start from a moderately-sized set of concepts and to introduce new attributes until every concept inherits no more than one concept.

- Ex. 9** (Hierarchical structures) We impose a fully hierarchical structure on the decision space for the problem of finding a new way of transportation. At the top of the hierarchy is the concept Transport, which has a decision variable Transport.Energy Source (es), where $R_{es} = \{electricity, human\ power\}$. The attribute .Energy Source allows us to distinguish between therefore between two concepts. We will use Electric Transport to refer to concepts that have the value Transport.Energy source=electricity. In order to structure known solutions that are electric transports we introduce a new attribute .Electricity Supply (els), such that $R_{els} = \{generating, consuming\}$. In order to structure concepts that generate electricity themselves, we introduce a new attribute .Electro Effect (ee), such that $R_{ee} = \{chemical, mechanical\}$. We will refer to a concept that generates electricity chemically as a Battery Car. We will refer to a concept that generates electricity mechanically as a Hybrid Car. By continuing this reasoning we arrive at a fully hierarchical structure, as shown in Figure 10. Other introduced attributes are: .Source Position (sp) with the range $R_{sp} = \{external, internal\}$ and .Working Part (wp) with the range $R_{wp} = \{legs, arms\}$.

For computational reasons, it is necessary to treat hierarchical structures as orthogonal structures. We therefore need to assume that all solutions have the same signature. This

assumption can only be true in some specific cases. Most of the time solutions will have different signatures, as in example Ex. 10.

Ex. 10 (Hierarchical structures) Consider the bicycle solution for the transportation product. According to Figure 10 this solution is represented by the following variables: (Bicycle.Energy source¹³, Bicycle. Working part). The following tuple of values corresponds to these variables (*human power*, *legs*). Now assume that for some reason we want to assign a new value to the variable Bicycle.Energy source, for instance *electricity*. Note that the attribute .Energy source is hierarchical in this case, since Bicycle.Energy source \subset Muscle Transport.Energy source. The attribute .Working part may therefore become meaningless for the concept Bicycle. (We assume that in this case it does, since the attribute .Working part refers to a part of the human body, which is meaningless in the context of electrically powered transport).

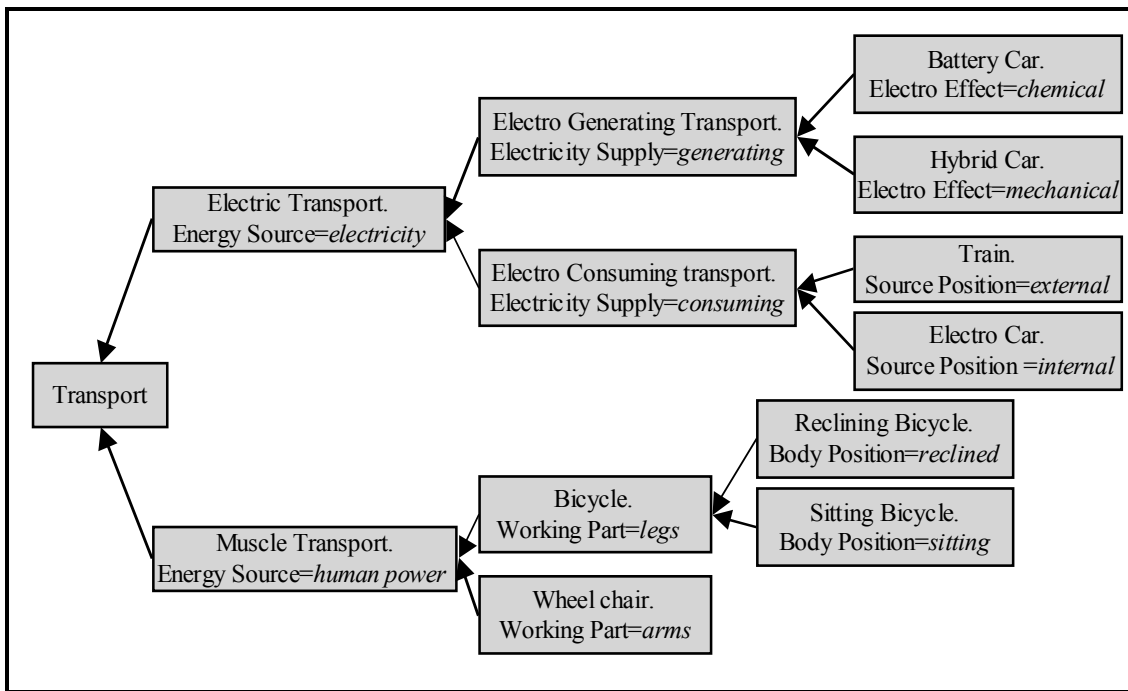


Figure 10. Hierarchy of solutions for the transportation product

In order to represent solutions in an orthogonal structure we extend the definition of the signature.

D. 25 (Extended signature) ‘*Extended signature*’ (ϕ') of a set of concepts Y is: $\phi'(Y) = \cup (c_i : c_i \in Y : \phi(c_i))$.

When an attribute $a_j : a_j \in \phi'(c_i) / \phi(c_i)$ is applied to a concept $c_i : c_i \in Y$ the meaning of a_j is not defined. The designer can therefore not assign a value to the variable $c_i.a_j$. Since we want to be able to apply such meaningless attributes, we introduce a new value ‘*not_apply*’. The range of all $a_j : a_j \in \phi'(C)$ is extended with this value such that $R'_j = R_j \cup \{not_apply\}$. We will call R'_j the ‘*extended range*’.

Ex. 11 (Extended signature) In the previous example the value *not_apply* needs to be assigned to the following variables: Solar Car.Body Part, Bicycle.Source Position etc.

¹³ This variable is inherited from the concept Transport, because Bicycle is-a Transport.

We notice that given the same set of attributes FOS is capable to produce more solutions than FHS allows doing. With n orthogonal attributes, each with at least m values, we get m^n segments. In other words, although the required mental effort for an attribute that has to be orthogonal (to all concepts in a set) is considerable, we need considerably fewer of them in order to subdivide them into a given number of sub-collections (i.e. of the order of $O(\log k)$ for k concepts). When constructing an orthogonal structure starting from a random set of concepts, we will encounter empty segments. Suppose that we have 25 concepts to start with. Suppose also that we only use attributes that are 2-valued (such as hot-cold, wet-dry, etc.). Then we need at least five attributes in order to have segments where every segment contains no more than one concept. However, this gives 32 segments ($2^5=32$), so seven of them are empty (note that we do not encounter empty segments when we build a hierarchy). These empty segments are very valuable, as they suggest extensions of the initial set of concepts. In other words, the method of finding orthogonal attributes ‘automatically’ generates new tuples, which can be interpreted as new concepts and can therefore be given a name.

- R.5 (Hybrid structures) If we cannot find all $O(\log n)$ attributes for n concepts in a set to make a fully orthogonal structure, we can start by building an orthogonal structure with a smaller number of attributes, such as m , $m < n$. As a result, some segments may contain several concepts. For these segments, provided there are not too many of them, we can attempt to build hierarchical structures. The know concepts for each particular segment can be associated to a root node of a tree to be build for that segment. So in this case we build an orthogonal structure with one or more encapsulated hierarchical structures.

2.3.4 The objective space V_O

When the set of solutions S has been created, the designer is faced with the problem of selecting the best ones, i.e. with imposing an order on S . Initially S is not ordered, i.e. the relation “ \geq ” is not defined between the solutions. We define that all knowledge that is necessary for selection of the solutions will be contained in the objective space V_O . V_O therefore enables the ordering, or at least partial ordering, of the solutions. All knowledge contained in the objective space V_O is defined by the set of objective variables together with their values.

- D. 26 (Objective variable) An objective variable $\langle s_i, a_j, evaluate \rangle$ is a variable such that it allows the evaluation of a solution (this means that the range R_j of the corresponding attribute should enable the relations ‘better than’ or ‘worse than’ between the values, i.e. R_j should at least be ordinal). The values of the objective variables are operated by means of evaluation.

Given this definition, every solution $s_i; s_i \in S$ can be represented in the space V_O by the tuple of objective variables, which we denote as s_i^o . We will refer to the tuple of values that correspond to s_i^o as objective values and will denote these values as v_i^o .

- D. 27 (Objective space) Objective space V_O is a knowledge space such that $V_O = \langle \{s_i^o, v_i^o\}, evaluate \rangle$.
- D. 28 (Corollary) All attributes a_j that allow the evaluation of solutions must be orthogonal on the set of solutions S . These attributes form an FOS of the space V_O . When an attribute is used to evaluate solutions, we will call it an ‘objective function’ or ‘objective’.

Every solution s_i can be positioned in the space V_o according to the tuple of objective values v_i^o .

Ex. 12 Consider two objectives .Power and .Safety, which allow the transportation means to be evaluated, i.e., for both attributes an extreme (maximum) value is required when these attributes are applied to the solutions concepts. We assume that the design problem is adequately represented by these two objectives. The necessary conditions are satisfied, indeed, .Safety $\in \phi(\text{Transportation})$ and .Power $\in \phi(\text{Transportation})$. Thus, according to D. 20, these two attributes form an FOS of V_o onto which all transportation means can be mapped. We can also define ordinal ranges for these attributes: $R_{power}=\{low, average, high\}$, $R_{safety}=\{safe, risky, dangerous\}$. Given this FOS, we can position all solutions in the space V_o as shown in Figure 11.

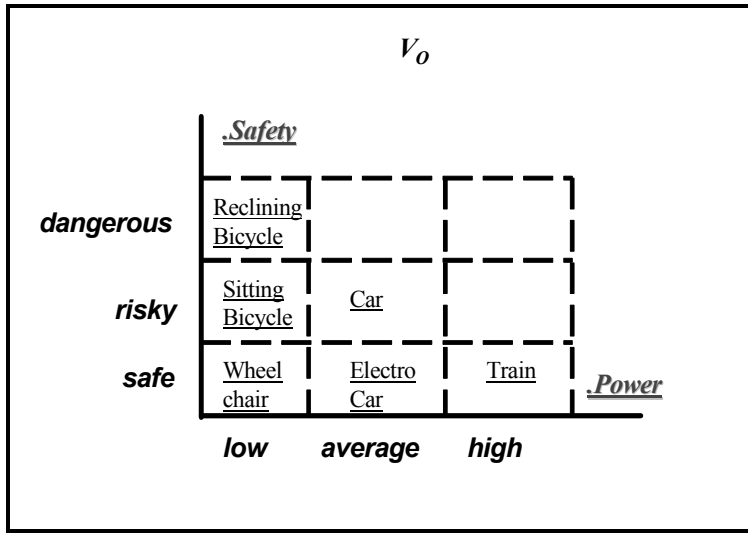


Figure 11. Orthogonal structure imposed on V_o

2.3.5 The contextual space V_C

We assume that all knowledge contained in the contextual space V_C will be defined by the set of contextual variables together with their values.

D. 29 (Contextual variable) A contextual variable $\langle c_i, a_j, measure \rangle$ is a variable such that its value constitutes a fact of the world about a concept c_i for which the designer has no control. Values of contextual variables are therefore operated using observations or measurements.

Contextual variables will return a unique value, for instance *petrol.specific_heat*=[2] kJ/kg K'. Unlike decision variables or objective variables, contextual variables are not necessarily related to solution concepts. For this reason we do not aim to prescribe any structure for V_C .

Any concept $c_i; c_i \in C$ can be represented in the space V_c by the tuple of contextual variables, which we denote as c_i^c . We will refer to the tuple of values that correspond to c_i^c as contextual values and will denote these values as v_i^c .

D. 30 (Contextual space) Contextual space V_C is a knowledge space, such that $V_C = \langle \{(c_i^c, v_i^c)\}, measure \rangle$.

2.3.6 The auxiliary space V_A

We assume that all intermediate knowledge will be contained in the space V_A and that it will be defined by the set of auxiliary variables together with their values.

- D. 31 (Auxiliary variable):** An auxiliary variable $\langle c_i, a_j, model \rangle$ is a variable such that $c_i^a \notin V_D \cup V_O \cup V_C$. Values of auxiliary variables are operated by means of modeling.

Thus, an auxiliary variable is a variable that cannot be classified as a decision variable, an objective variable, or a contextual variable. In a similar way as in the space V_C , the space V_A may contain variables of any concept and not necessarily only solution concepts. The space V_A is therefore unstructured. Any concept $c_i: c_i \in C$ can be represented in the space V_A by the tuple of auxiliary variables, which we denote as c_i^a . We will refer to the tuple of values that correspond to c_i^a as auxiliary values and will denote these values as v_i^a .

- D. 32 (Auxiliary space)** Auxiliary space V_A is a knowledge space such that $V_A = \langle \{c_i^a, v_i^a\}, model \rangle$.

Ex. 13 (Auxiliary variables) Auxiliary variables usually refer to internal or intermediate knowledge about the design product. They do not themselves express any desired feature of the product. For instance, vibrations in themselves do not add to the perceived success of a solution, therefore the solutions' vibration amplitude is a variable $\in V_A$, and not $\in V_O$. Another example is that the power of the heating system does not add to the perceived success of a solution, but it helps to compute the inside temperature and the energy consumption.

2.3.7 Summary

In our model we represent the state of product knowledge in terms of concepts and attributes. We model concepts by tuples of variables with their values ($c_i, a_0 = v_{i0}, c_i, a_1 = v_{i1}, \dots, c_i, a_j = v_{ij}$). We have described all knowledge as initially appearing in chunks, which are packaged in concepts; the names and values of the concepts therefore help to identify these chunks. By applying attributes to a concept c_i , the designer accesses and operates the implicit chunks of knowledge about the concept in a systematic way. Since variables can be operated by four categories of operations, the knowledge about a concept can be distributed between the four knowledge spaces V_D, V_O, V_C, V_A , such that $c_i = c_i^d \cup c_i^o \cup c_i^c \cup c_i^a$.

Distribution of knowledge over the four spaces has an important consequence. The same attribute may have four different meanings, which are formally identified by the four categories of operations. We can demonstrate this in an example, which is described in Table 3. We have introduced the formalism to represent product knowledge and we now need to explain how this formalism leads to systematic development of the product model to complete the description of the product state.

Table 3. Four different meanings for the same attribute name $a_j = \text{'length'}$

Category of a variable	Meaning	Operation
Design: $c_i, a_j \in V_D$	A designer can decide what the length of a concept c_i should be	<i>Decide</i>
Objective: $c_i, a_j \in V_O$	A concept c_i is preferred to other concepts based on the length. Let us assume that the longer the length is the better. Then if v_{ij} is bigger than v_{mj} , c_i is preferred to a concept c_m . Length can also be a constraint. For example if $v_{ij} > l_{\max}$, c_i is excluded from further considerations	<i>Evaluate</i>
Contextual: $c_i, a_j \in V_C$	Length represents an observation or a measurement of the fact about a concept c_i . The designer has no freedom to decide what the length should be	<i>Measure</i>
Auxiliary: $c_i, a_j \in V_A$	The length of a concept is an internal knowledge. For example, a piece of wire can be cut in n segments of an equal length. The length of a segment is an auxiliary variable which can be obtained through modeling.	<i>Model</i>

2.4 Product models

In this section we describe the meaning of operations from the category ‘model’ in the context of our formalism. Operations from this category lead to development of the product model. In section 2.4.1 we give basic definitions, which will be used to define the product model in section 2.4.2. In section 2.4.3 we consider different classes of product models and define the scope of product models that our formalism supports. Section 2.4.4 is dedicated to problems related to the development of the product model. In section 2.4.5 we present a summary.

2.4.1 Basic definitions

The basic notion that underlies the product model is ‘*functional dependency*’. Thus operation of modeling is done through development of functional dependencies. Consider a space X which consists of elements v_l , and a space Y which consists of elements v_m . If according to some rule f we can find a corresponding element v_m for any element v_l , then we can say that there is a function $v_m = f(v_l)$ with the domain in X and the range in Y . We depict this functional dependency as a graph in Figure 12.

We will call the element v_l ‘*independent*’ with respect to f , and will denote it graphically by a star. We will call the element v_m ‘*dependent*’ and we will denote it graphically by a circle. The functional dependency f will be denoted graphically as an arrow, going from an independent element to a dependent one. We will call such a graph ‘*a directed functional graph*’. An independent element v_l is called *an argument* of f . The dependent value will be called the ‘*outcome*’ of f .

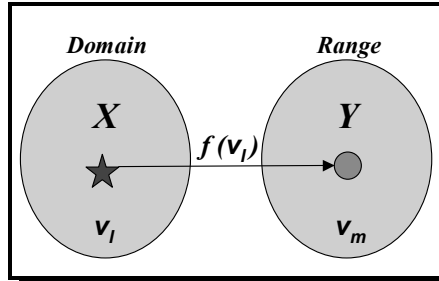


Figure 12. A functional dependency between two elements

We will now consider a functional dependency of n arguments:

$$v_m = f_m(v_1, v_2, \dots, v_n) = f_m(\mathbf{X}).$$

We can depict this situation graphically as shown in Figure 13 on the left.

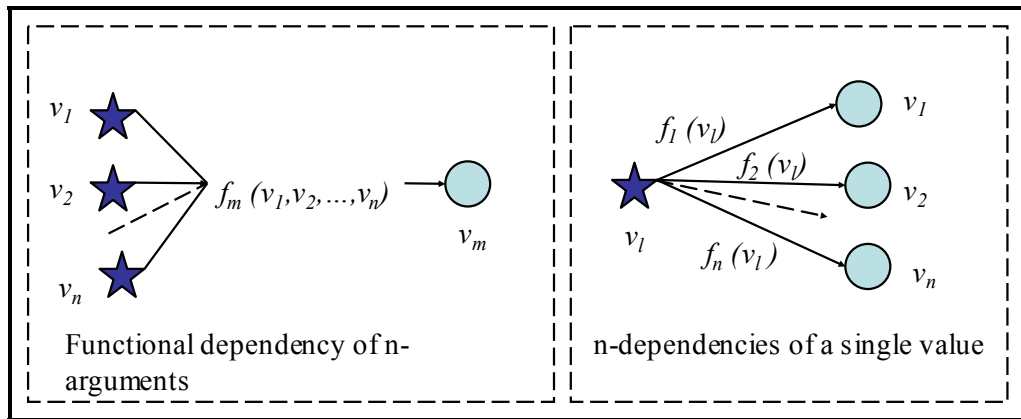


Figure 13. Functional dependency of n -arguments

A value v_l can play the role of an argument in n -functional dependencies f_1, f_2, \dots, f_n . We can graphically depict this situation as shown in Figure 13 on the right.

In terms of our formalism we will say that if a value v_{ij} of a variable $c_i.a_j$ is obtained by means of a functional dependency f_i , then f_i is a modeling operation such that $\langle c_i.a_j, f_i \rangle = v_{ij}$.

2.4.2 Definition of the product model

After we have defined a functional dependency, we can define the product model.

- D. 33 (Graphical definition of the product model) The product model is a directed acyclic graph (DAG) that connects values from the spaces V_D and V_C with the values in V_O , possibly via intermediate values in V_A .

We will consider an example of such a graph later on. The term ‘*acyclic*’ means that the graph does not have cycles, i.e. it is impossible to get into a value by following any sequence of outgoing arrows from the value. Several types of graphs are considered in the literature, such as index dependency graphs, value dependency graphs etc. DAG

corresponds to modular structures [Geofrion, 1988]: “the vertices are the model entities; an arc is defined if a model is nested within another model”¹⁴.

The definition of the product model has several consequences for the formalism, which we consider below:

- All values in V_D are independent, i.e. all decision variables s_i^d are independent variables whose values can be decided by the designer¹⁵.
- All values in V_O are dependent, i.e. all objective variables s_i^o are dependent variables whose values result from evaluation and need to be a minimum or maximum.
- All values in V_C are independent, i.e. all contextual variables c_i^c are independent variables whose values are measured and therefore they are constant.
- All values in V_A are dependent, i.e. all auxiliary variables c_i^a are dependent variables whose values are not interesting to the designer or the stakeholders in themselves.

The distinction between dependent and independent variables is well recognized in the design literature. According to Baarda [1995], “the variables that are singled out are ‘*dependent variables*’, the ones that are varied to study their impact are the ‘*independent variables*’, and those that are constant are ‘parameters’”. An example of a graph can be found on page 118 in Cross [1994]. The same publication contains the distinction between design variables, intermediate variables and performance metrics. This enables the link between our formalism and existing formalisms, which were developed within the field of engineering. Consider an example of the evaluation of a graph.

- Ex. 14** (Evaluation of the product model) In the design of the product for transportation (TR) we consider: the decision variables Tr.Els (Electricity source), Ext.Med (Media to transform electricity for externally powered transport) and Int.Btr (Type of battery for internally powered transport); the contextual variables Wire.R (Efficiency of the energy transmission via the wire), Air.R (Efficiency of energy transition via the air), Alc.Ah (Capacity of alkaline battery) and Ni.Ah (Capacity of Ni-Cd battery); the auxiliary variables Int.P (power of the internally powered transport), Ext.P (power of the externally powered transport) and Tr.P (Power of the transport) and an objective variable Tr.Sp (Speed of transport). Suppose we have all functional dependencies as depicted in the graph in Figure 14¹⁶. The evaluation of the product model goes from right to left. We compute Tr.Sp from Tr.P¹⁷. From Tr.P further evaluation can take one of the two paths i.e. 1 or 2, depending on the current value of Tr.Els. So the decision values switch the path in which the graph is evaluated.

¹⁴ In our terminology, vertices are the values and the arcs are functional dependencies.

¹⁵ Variation of design values is one of the design activities. In section 2.6.2 we consider algorithmic variation of these values.

¹⁶ We have developed a special colour scheme for the graphical representation of the spaces. For instance, the green colour of the decision space is associated by many people with creativity.

¹⁷ Tr.Sp also depends on the shape, mass etc. These considerations are omitted as they would complicate the graph.

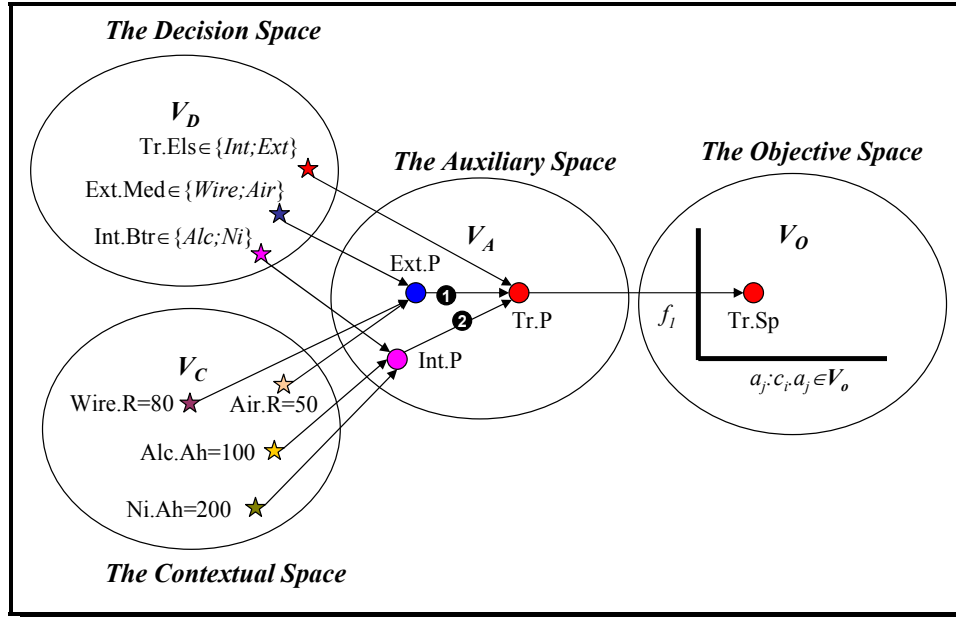


Figure 14. Product models as graph representations

Consider the functional definition of the product model in our terminology.

- D. 34 (Functional definition of the product model): A product model is a function $F=F(f_1(\mathbf{v}), f_2(\mathbf{v}), \dots, f_h(\mathbf{v}))$, where $\mathbf{v} \subset V_D \cup V_C \cup V_A$, that enables the mapping of decision values (\mathbf{v}^d) from n-dimensional space V_D to objective values (\mathbf{v}^o) in h-dimensional space V_O , taking into account m-independent contextual values \mathbf{v}^c , possibly via k-dependent auxiliary values \mathbf{v}^a , see Figure 15.

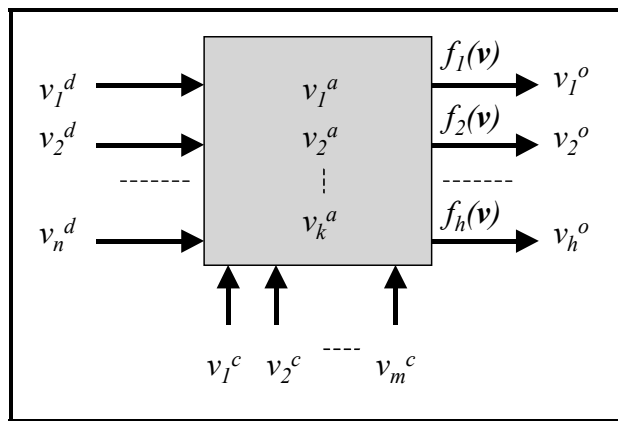


Figure 15. Definition of the product model

2.4.3 Classes of product models and the scope of our formalism

The designer can develop different ‘types of product models’. The variety of products and problems makes it impossible for us to precisely classify all product models. Models can differ with respect to their precision, representation, etc. Two classes of product models are distinguished in the literature: ‘structural models’ and ‘functional models’, both of which can be deterministic or stochastic. Product knowledge is unstructured and frequently updated; we therefore need to consider structural and deterministic models. We were guided in our approach by the following consideration:

if the product model can be expressed in a declarative way, then it should be. More advanced (stochastic) functional models can be built upon the developed product model in the later design stages. Because these models are more time consuming and present a new dimension of complexity, we first need to build a basic model and then decide what needs to be improved.

The purpose of structural models is to model the structural characteristics of the product. Structural models can be topological or geometrical. Topological models model the product as a set of elements with relations. These models are built to define the main ingredients of the product and the positions for them. Topological models can be represented by graphs or matrices. Geometric models focus on the geometry of the product.

2.4.4 Development of product models

The product models are provided by the designer. These models constitute functional dependencies such as the interpretation, the physical causality, economic causality, or assumed psychological causality. Our formalism describes how the product models can be developed in small incremental steps. Models can be built or found for many values, but not for all. Values for many objective variables, such as those related to aesthetics, usability, or the comfort of the end user, may depend on the subjective opinion of the stakeholders. These dependencies may not be available or known to the designer. We therefore need to distinguish between functional dependencies and interpretational dependencies.

- D. 35 (Interpretational dependency) A dependency is '*interpretational*' iff it cannot be expressed as a calculated functional dependency f .

Interpretational dependencies can be seen as links in the graph which cannot be expressed (at least immediately) in terms of mathematical functions. The role of the designer is therefore to allow such links to be filled in by interpreting the arguments of the missing dependency and providing the corresponding values.

- Ex. 15 (Interpretational dependency) Consider an evaluation of two objective attributes Power with the range $R_{\text{power}} = \{\text{low, average, high}\}$ and Safety with the range $R_{\text{safety}} = \{\text{safe, risky, dangerous}\}$. We will use these attributes to evaluate decisions from the decision space V_D . This space is structured using two orthogonal attributes, e.g. Energy Source and Media, and contains solutions that are represented by the following tuples of decision values: *(diesel, ground)*, *(human power, ground)*, *(human power, air)*. We assume that there are functions that relate Power to the media state and energy state. However, for Safety we may not have found such functional dependencies. We therefore have to rely on our own interpretation of safety for each particular solution. For instance, the designers' interpretation may result in the following order: $\text{Safety}(\text{human power, ground}) = \text{safe}$, $\text{Safety}(\text{diesel, ground}) = \text{risky}$, $\text{Safety}(\text{human power, air}) = \text{dangerous}$. The operation of evaluation here can be based on intuition and experience, but it can also be linked to measurements, for instance an estimated number of accidents.

2.4.5 Summary

The definition of the product model completes the description of the product state. It is convenient to summarize by relating our formalism to the levels of knowledge that were described in the context of knowledge management in the introductory chapter.

- D. 36 (Data) The meaning of a value as ‘data’ is not defined yet. A value is initially positioned explicitly in a document or implicitly in a human mind.
- D. 37 (Information) We will call a value that was interpreted and related to a variable ‘information’. The meaning of a value as information is defined and related to a specific but an isolated variable.
- D. 38 (Knowledge) If a value is referenced and is used in the product model, we will say that such a value is meaningful and represents an explicit chunk of knowledge.

We summarize these definitions in Figure 16.

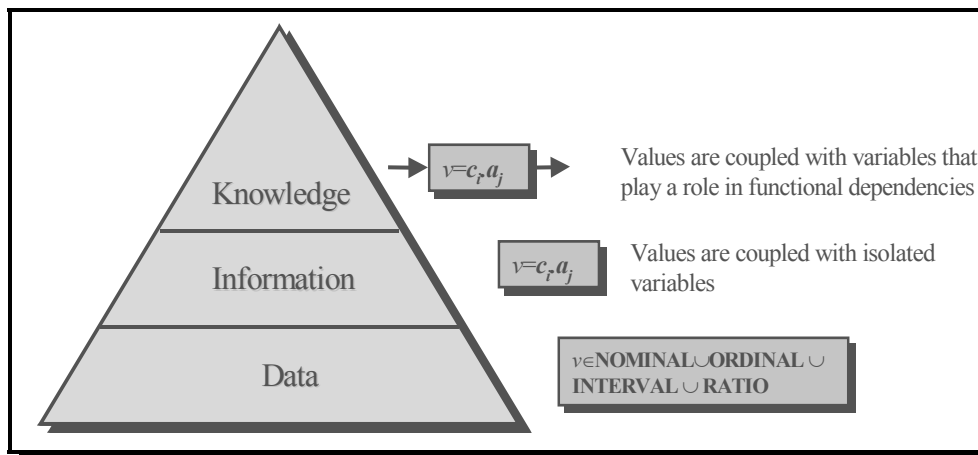


Figure 16. The distinction between data, information, and knowledge

In the context of Knowledge Management we stated that it is useful to identify dimensions of knowledge. For our purposes it is also necessary to show that the introduced knowledge spaces are orthogonal. On this basis we can state that the four operations that underlie the spaces are orthogonal as well. We already defined one orthogonal dimension for knowledge, namely $a_1 = \underline{\text{Dependency}}$, such that $\mathbf{R}_{a_1} = \{\text{dependent}, \text{independent}\}$. The new dimension that we consider is $a_2 = \underline{\text{Subjectivity}}$, such that $\mathbf{R}_{a_2} = \{\text{objective}, \text{subjective}\}$. These two dimensions form an orthogonal structure for the knowledge spaces and therefore for the four categories of operations with knowledge; it has four segments, with each knowledge space corresponding to one segment. Indeed, values in the decision space V_D are subjective with respect to the designers. Values in V_O are subjective with respect to the stakeholder¹⁸, i.e. $\underline{V_D}.\text{Subjectivity} = \text{subjective}$ and $\underline{V_O}.\text{Subjectivity} = \text{subjective}$. Values in the context space V_C are objective because they represent world knowledge; values in the auxiliary space are objective because they typically do not represent the personal interest of the participants. Thus, $\underline{V_C}.\text{Subjectivity} = \text{objective}$ and $\underline{V_A}.\text{Subjectivity} = \text{objective}$. Using a_1 and a_2 we impose a fully orthogonal structure on the knowledge spaces, as demonstrated in Table 4. This structure is one of our results that supports the completeness of the four

¹⁸ Many objectives can be evaluated by means of known functional dependencies. However, this does not guarantee the acceptability of the generated solution concepts by the stakeholders. Therefore the objective spaces is subjective with respect to the stakeholders.

knowledge spaces and therefore of the four categories of operations with product knowledge.

Table 4. The structure for the knowledge categories

	Independent	Dependent
Subjective	1. V_D : designers choices	2 V_O : objectives
Objective	3. V_C : contextual world knowledge	4. V_A : auxiliary

The next step in the definition of our model is to give a formal definition of the mechanism of questions and answers.

2.5 Reflections

In this section, we prescribe how reflections can be facilitated by means of the formalized mechanism of questions and answers. This mechanism we explain in terms of the introduced formalism in section 2.5.1. In section 2.5.2 we consider how our formalism allows us to distinguish eight different categories of questions and therefore eight different formats of giving the answers. We will present an example in section 2.5.3.

2.5.1 Formal definition of questions and answers

In this section we define the question and answer mechanism in terms of our formalism. The purpose of this formalization is two fold: 1) to relate our formalism to conventional ways the designers work, 2) to enable clearer thinking and communication during the moments of reflections.

We define any question to address an attribute a_j of some concept c_i . As an intermediate step, a question can simply bring a new concept into consideration. Thus, a new question defines a new variable $c_i.a_j$. The purpose of the question is therefore to obtain a value v_{ij} for this variable. For a dependent value v_{ij} , the answer conveys a functional dependency f , which allows us to compute a value by evaluating f . When a new question is asked, this may cause a transition of the product to a new state due to a new concept or a new attribute that is contained in the question. The designer can also disregard a question, i.e. consider it to be irrelevant. The deletion of a question might result in the deletion of a concept or an attribute from a state of the product; it would definitely result in the deletion of an answer, i.e. of the value. Table 5 presents the described transitions.

The modeling of questions provides designers with the following guidelines during the moments of reflections:

- Statements that are not questions or answers to the previously posed questions are filtered out and need to be interpreted.
- Such filtered-out statements can either be transformed into new questions or are considered to be no longer relevant and are excluded from further considerations.

In the next section we describe how questions can be analyzed and how eight different forms of answers to the same questions can be given.

Table 5. Knowledge transitions caused by the question and answer mechanism

Design activity	Knowledge updates
Asking a new question	Adding $c_i, a_j, c_i.a_j$
Answering a question	Assigning v_{ij} or adding a functional dependency f to obtain this value
Revising a previous question	Removing, changing c_i, a_j, v_{ij}
Revising a previous answer	Varying v_{ij}

2.5.2 Categories of questions

We identified an orthogonal structure for the four knowledge spaces. Asking and answering questions mechanism produces new knowledge. Therefore it is logical to distinguish four categories of questions, which correspond to this structure. Earlier in the chapter we considered two knowledge dimensions: $a_1 = \text{Dependency}$, and $a_2 = \text{Subjectivity}$. In order to distinguish between questions that can and cannot be answered, we now introduce a new orthogonal dimension $a_3 = \text{Answer availability}$, such that $R_{a_3} = \{available, unavailable\}$. Given these three orthogonal dimensions, we obtain a structure with eight segments, where each segment corresponds to a category of questions. By interpreting the corresponding tuple of values of every category and assigning a name to it, we end up with the structure shown in Table 6.

Table 6. Eight ways of answering questions in design

		Answer availability		
		Variable category	Available	Unavailable
Subjective	Independent	Design	1) Guessing Give a provisional answer (making an educated guess)	2) Deciding Give a (possibly multi-valued) answer, which combines alternative answers to a question
	Dependent	Objective	3) Parametrizing Composing a functional dependency taking into account preferences of the stakeholders	4) Asking Composing an interpretational dependency
Objective	Independent	Contextual	5) Defining Give a definitive, full and complete answer	6) Searching Searching for an answer
	Dependent	Auxiliary	7) Modeling Composing a functional dependency	8) Postponing Composing a functional dependency which will be evaluated later

We still do not know whether there is a fourth or fifth orthogonal dimension. For these three dimensions we already needed to conduct a brainstorming session in order to translate the tuples of values into meaningful names describing the various forms of answers. For some categories it was fairly easy, such as *(dependent, objective, available)=modeling*, *(independent, subjective, unavailable)=deciding*, *(independent, objective, available)=defining*. For some tuples we initially had vague ideas of what they could mean, e.g. *(independent, subjective, available)=guessing*, *(dependent, subjective, unavailable)=asking*. The addition of a new orthogonal attribute, even if it only had two values in the range, would require another eight forms of answers. In our opinion, it is not likely that we could define so many additional forms. We therefore consider this orthogonal structure to be complete, and we present it as one of our results.

We think that there are two applications of this orthogonal structure:

- Designers and in particular student designers, can use this structure to master the interpretation of questions and to develop an awareness of the various forms of giving answers and to improve clarity of the reflections. We demonstrate this application as an example in the next section.
- Developers of operational support tools for the early design phase can consider the eight forms of answering questions as a list of functional requirements for support tools. Characteristics of these requirements are that they are a) complete and b) formally expressed in terms of our formalism; this makes it straightforward to translate the requirements into a specification. In section 3.2.1 we will follow this guideline and introduce a set of functional requirements for a support tool.

2.5.3 Example

This section demonstrates an example to each of the possible ways of answering questions. Consider the design process of a chair and the question, “What material should the chair be made of?” Formalization of this question results in a new concept $c_I = \underline{\text{Chair}}$, and a new attribute $a_I = \underline{\text{Material}}$. Let us assume that $R_I = \{\text{plastic, steel, wood}\}$ ¹⁹. Our purpose is to explain how the question (variable) Chair.Material can be interpreted, and to assign a value v_{II} in eight different ways.

- 1) **Guessing.** The answer to the question is subjective, i.e. the designer has the freedom to decide what the answer should be. The answer is given in a definitive way, for one of two possible reasons: a) the available alternative answers are not applicable or b) alternative answers are not considered. It is because of b) that we call this category Guessing. For instance, the designer guesses that the material should be *wood*. If alternative answers are considered, then this question falls into the next category – deciding,
- 2) **Deciding.** This is similar to the previous category, but here the designer has several alternative answers (or no known answers at all). The designer therefore needs to consider (produce) alternative answers to the question, which is in fact a design decision. For instance, the material can be *wood, plastic* or *steel*. The set of possible alternatives can be obtained directly from the range of the

¹⁹ The set of questions about the possible material follows automatically from considerations of the range.

corresponding attribute. We can more clearly see why it is useful to explicitly identify attributes.

- 3) **Parametrizing.** The question addresses an objective variable and the answer therefore depends on the personal preferences of the stakeholder. If the preferences are well defined, a functional dependency could be used to compute an answer. If there is no such dependency, the preferences of the stakeholder with respect to the possible outcomes need to be acquired. We will refer to values that express the stakeholder's personal preferences as '*parameters*'. In this example, the question about the material requires the stakeholder to order the possible materials according to his or her personal preference, e.g. (*wood, steel, plastic*); in this case a wooden chair is better for the stakeholder than a steel chair. Note that the stakeholder's preferences can be conditional and can depend on some other values of the chair, in which case the question falls into the next category.
- 4) **Asking.** The stakeholder's preferences may depend on other values of the product. Suppose that the stakeholder's opinion about the material depends on a few values. It would be efficient to only ask the stakeholder about those values. For example, if the stakeholder's preference depends on the location of the chair, the designer might ask the stakeholder, "Would you like a wooden chair in the garden? And in the office?" In this case the values for the variable place of the chair are taken from the range of the attribute Place, i.e. $R_{place} = \{garden, office\}$,
- 5) **Defining.** The question addresses knowledge that we assume to be fixed, i.e. contextual knowledge. A definite answer can therefore be given. For example, a stakeholder has expressed the requirement to have a wooden chair.
- 6) **Searching.** This is similar to the previous category, but the answer is unknown or is missing from the documentation. The designer needs to search for the answer, to ask follow up questions, etc. For instance, the designer can ask a manufacturer what material the chair should be made of.
- 7) **Modeling.** The question is related to an intermediate knowledge, i.e. the material itself is not important. For instance, the designer considers the material to be only an indirect means of obtaining an objective value. The designer considers the question to be related to some existing knowledge, for instance to the usage of the chair. The material should be *wood* if the chair is used inside and the material should be *plastic* if the chair is used outside. The available answer represents a known functional dependency, which produces a value.
- 8) **Postponing.** This is similar to the previous category, but the answer cannot be obtained due to incomplete knowledge. In this case the designer expresses a function dependency which does not deliver a value immediately. In the future when the product model is complete, the expressed dependency will produce a value. For example, if Chair.Place=*garden* then Chair.Material=*plastic*, if Chair.Place=*office* then Chair.Material=*wooden*. We assume that the Chair.Place is unknown.

2.6 Computational operations

In the previous section we considered state-transitions due to operations performed by the designers. The topic of this section is state-transitions due to computational operations.

2.6.1 Introduction

Besides helping designers to gain a clearer way of thinking and communicating, the major benefit of formalising the product is the computational usage of formally defined knowledge about the product. The ability to automate meaningful operations with the product cannot be overestimated in the early design phase. It is generally agreed that it is much easier to produce and evaluate solutions if the alternatives are known. We therefore aim to enable modern computing technologies combined with artificial intelligence techniques²⁰ and numerical computational techniques in the early design phase; these are techniques which typically become available in the detailed design phase.

Operations with knowledge belong to the domain of a branch of mathematics known as ‘operations research’. The goals of operations with knowledge within this domain can be of two kinds, i.e. a ‘*direct goal*’ or an ‘*inverse goal*’.²¹ The direct goal is related to the development of the product model. This is the topic of ‘*mathematical modeling*’, which focuses on the development of mathematical models to obtain output characteristics of the product from the input characteristics of the product. In section 2.4 we described how our formalism supports the development of the product model. This section focuses on the inverse goal.

The inverse goal is the topic of ‘*mathematical programming*’, which aims to optimize the output characteristics of the product model by varying values of the product’s independent variables. The word ‘programming’ here means planning rather than implementation of computer support tools. In section 2.6.2 we define the problem of mathematical programming in terms of our formalism. In section 2.7 we introduce a *genetic algorithm* (GA) to deal with the mathematical programming problems.

2.6.2 Mathematical programming problems

We have defined product knowledge as consisting of sets A and C ; we have also defined the spaces V_D , V_C , V_O , and V_A , which are related by the product model F . The classical formulation of the mathematical programming problem (the inverse goal) is similar to the formulation of the product model (the direct goal) (D. 34), with the exception of the following differences (see Figure 17):

- optimal output values $\mathbf{v}^o = (v_1^o, \dots, v_n^o)$ need to be obtained
- $\mathbf{v}^d = (v_1^d, \dots, v_n^d)$ are varied by an algorithmic procedure

²⁰ One of the definitions of Artificial Intelligence is “the ability of machines to do things that require intelligence”.

²¹ In the literature, an inverse goal is also called an inverse problem or optimization problem. There are several kinds of inverse problems in mathematics. See Denn [1969] for some examples.

- a set of m -constraint functions is given: $g_1(\mathbf{v}), \dots, g_m(\mathbf{v})$ ²²

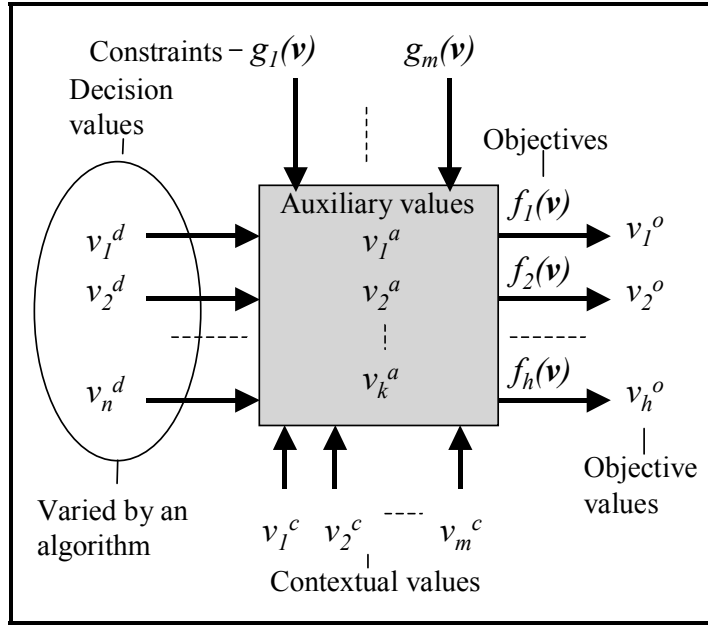


Figure 17. General formulation of the mathematical programming problem

We extend the classical definition in the following way to provide additional flexibility:

- \mathbf{v}^d can be manually varied (or fixed) by the designers
- \mathbf{v}^o can be constrained, so that every f can have the left constraint l_f and the right constraint r_f , which determine the range of acceptable outcomes. l_f and r_f are subjectively defined by the participants and can be further adjusted during the decision making.

The adjustment of the constraints on objectives enables convergence to a single optimal solution. We have therefore defined a method of tightening constraints, which is based on this mechanism of adjustment and which compensate disadvantages caused by the classical decision-making methods, which were described in section 2.7.2.

Historically there is a different terminology for describing problems in mathematical programming. We will therefore reformulate the conventionally used terms using the ingredients from our model.

- D. 39 (Operation) An *Operation* is an action directed towards the optimization of \mathbf{v}^d with respect to the set of h objectives.
- R.6 Note that s_i may include decision values of several concepts. Due to algorithmic variation, decision values related to hierarchical attributes may affect the signature of concepts, and therefore the signature of s_i . We therefore need to deal with the extended signature of the set of concepts (see D. 25 in section 2.3.3).
- D. 40 (Variation) Variation is an operation that leads to a different tuple of values \mathbf{v}^d , which can be varied within the range of the corresponding attributes.
- D. 41 (Solution) a solution s_i is a concept such that $s_i^d = \mathbf{v}^d$.

²² Constraint functions limit the range of acceptable solutions. The maximum or minimum values are not required for the constraint functions.

- D. 42 (Solution element) A decision variable $\langle s_i, a_j, decide \rangle$ of a solution will be called a 'solution element'.
- D. 43 (Inactive solution element) A solution element s_i, a_j is inactive iff $a_j \in \varphi'(s_i) / \varphi(s_i)$, than $s_i, a_j = not_apply$, where $not_apply \in R'_j$.
- R.7 (Evaluation of inactive elements) The evaluation of the product model can take different paths because of the variation. Using *not_apply* allows us to skip the evaluation of inactive elements.
- D. 44 (Number of solutions) By the systematic variation of values v^d , the total number of solutions that can be generated by n -decision variables is equal to $R'_1 \times R'_2 \times \dots \times R'_n$, where R'_1, \dots, R'_n are the extended ranges of the attributes related to v_n^d .
- D. 45 (Optimization) Optimization can mean finding s_i^d that produce a maximum of F with respect to all considered solutions, where decision values v_1^d, \dots, v_n^d are to be varied:

$$\begin{aligned} & \max F(\mathbf{v}), \\ & \text{subject to } \mathbf{g}(\mathbf{v}) > 0. \end{aligned}$$

- D. 46 (Set of optimal solutions) Solutions that were found by means of optimization.
- D. 47 (Problem of mathematical programming) The problem of mathematical programming is to find the set of optimal solutions.

If $\dim(F) = 1$ and F is a linear function, then the principle of *linear programming* [Denn 1969] can be applied; this offers well-developed procedures and computer programs. There are no general ways to solve problems of non-linear programming. In each specific case, the approach depends on F and the constraint functions. In the general case, when $\dim(F) > 1$, we do not know what *max* means. We postpone the definition of *max* until section 2.7.2.

Evolutionary algorithms have recently become established [Back, 1997] as an alternative to the classical methods of dealing with multiple objectives. These algorithms allow us i) to handle more complex problems and ii) to generate multiple alternative solutions in a single optimization run. They can also be implemented in a way that avoids some problems related to the classical methods.

2.7 Genetic-based optimization: SPEA

The term genetic algorithm (GA) stands for a class of optimization methods that simulate the process of natural evolution. In genetic algorithms, natural selection is simulated by a stochastic selection process. Selection represents the competition for resources among living beings; some are better than others and more likely to survive and to reproduce their genetic information. Each solution is given a chance to reproduce, depending on its quality. Quality is assessed by evaluating the individuals and assigning them scalar fitness values. The other principle, variation, imitates the natural capability of creating "new" living beings by recombination and mutation.

In general, any GA is characterized by three properties:

1. A set of solution candidates is maintained.
2. It undergoes a selection process.
3. It is manipulated by genetic operators, usually in the form of recombination and mutation.

Analogous to natural evolution, the solution candidates are called ‘*individuals*’ and the set of solution candidates is called the ‘*population*’. Each individual represents a possible solution, i.e. a vector of decision values, to the problem at hand. The set of all possible vectors constitutes the decision space V_D .

In the course of this PhD project we have looked at several genetic algorithms to enable the generation and evaluation of solutions in a support tool. We have chosen SPEA [Zitzler, 1999] since it is a well-documented and tested algorithm. SPEA is an acronym for Strength Pareto Evolutionary Algorithm. SPEA is a genetic algorithm, in which the selection process is based on the Pareto principle of optimality, as described in section 2.7.1. SPEA was one of the first algorithms that was extensively compared to several existing evolution-based methods. Due to its good performance, it has been used as a reference point by various researchers.

The design of any evolutionary algorithm strives to attain two objectives: a) the distance of the resulting set of solutions to the Pareto set is to be minimized and b) the achieved Pareto set should be as diverse as possible. The important decisions that underlie any GA are therefore: a) the choice of a fitness function (described in section 2.7.2), b) the choice of a diversification strategy (described in section 2.7.3) and c) the choice of an elitism strategy, i.e., different ways to prevent non-dominated solutions from being lost (described in section 2.7.5). The SPEA algorithm was implemented in the context of a developed software tool. We therefore present the steps of this algorithm with minor editing in Appendix A.

2.7.1 Pareto optimality

The consequence of multiple competing objectives is that there is usually no single optimal solution, but rather a set of partially-ordered solutions. The Pareto principle allows a reduction in the number of solutions that have to be considered (decided).

- D. 48 (Dominated solution) According to the Pareto principle, solutions that are worse with respect to all the objectives should not be considered. Such solutions are called *dominated solutions*.
- D. 49 (Pareto Optimal solutions) Solutions that are non-dominated are called Pareto optimal. The set of optimal solutions forms the *Pareto set Ps*, as shown in Figure 18. The Pareto set is also called the *Pareto front*.

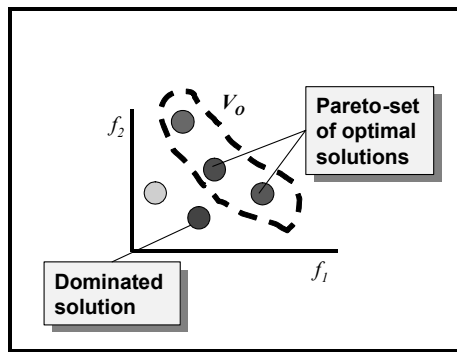


Figure 18. Definition of the Pareto set of optimal solutions

The selection of solutions based on the concept of Pareto optimality is only the first step in solving a multi-objective optimization problem. A decision making process is necessary to further converge to a single solution. We assume here that there are no

analytical means to prefer one Pareto optimal solution over another. Further selection is therefore *preference based*, i.e. some objectives are considered to make a bigger contribution to the success of the product than others. Here we assume that multi-objective optimization takes place before the decision-making, and therefore simplifies the task by reducing the number of solutions that have to be considered.

2.7.2 Fitness assignment

Here we will present some classical approaches of defining $\max \mathbf{F}(\mathbf{v})$. Classical approaches, such as the weighting method or the constraint method, transform a multi-objective problem into a single objective problem. According to the weighting method, each function $f_i; f_i \in \mathbf{F}$ is assigned a scalar weight w_i . The weights are used to compose the linear objective function y that is going to be maximized.

D. 50 (Weighting method)

$$\begin{aligned} &\text{maximize} && y = w_1 \cdot f_1(\mathbf{v}) + w_2 \cdot f_2(\mathbf{v}) + \dots + w_h \cdot f_h(\mathbf{v}), \\ &\text{subject to} && \mathbf{g}(\mathbf{v}) > 0. \end{aligned}$$

Besides some computational disadvantages, this method requires the designer to prioritize objectives using the weights w_1, w_2, \dots, w_h . The weights need to be guessed and the results therefore require additional analysis. According to the constraint method, $h-1$ objectives are transformed into constraints. The remaining objective, which can be chosen arbitrarily, is the resulting objective function that is going to be maximized.

D. 51 (Constraints method)

$$\begin{aligned} &\text{maximize} && y = f_h(\mathbf{v}), \\ &\text{subject to} && e_i = f_i(\mathbf{v}) \geq \varepsilon_i \\ &&& \mathbf{g}(\mathbf{v}) > 0. \end{aligned}$$

The lower bounds, ε_i , are the parameters that are varied in order to find multiple Pareto-optimal solutions. If the lower bounds are not chosen appropriately, there can be no solution to the corresponding problem. In order to avoid this situation, a suitable range of values for the e_i has to be known beforehand. We may therefore require problem knowledge that is not available.

The $\max \mathbf{F}(\mathbf{v})$ is defined in the context of GA using the notion of a fitness function; this allows solutions to be compared against each other in the presence of multiple objectives. A fitness function therefore enables the solutions to be ordered, and has to be either minimized or maximized, depending on a specific GA.

In the selection process, which can be either stochastic or completely deterministic, low-quality individuals are removed from the population, while high quality individuals are reproduced. The goal is to focus the search on particular portions of the decision space, and to increase the average quality within the population. The quality of an individual with respect to the set of objectives is represented by a scalar value, the so-called fitness, which is produced by the '*fitness function*'.

The fitness function in SPEA explores the idea of calculating a fitness of solutions on the basis of a Pareto set [Zitzler, 2002]. What is remarkable here is the fact that fitness is related to the whole population, while in other Pareto-based techniques an individual's raw fitness value is calculated independently of other individuals. SPEA uses two sets to compute fitness. The set \mathbf{P}_t , is the set of considered solutions at the step t , which has N solutions. $\underline{\mathbf{P}}_t$ is the externally-stored set of Pareto optimal solutions at

step t . For every non-dominated solution $s_i: s_i \in \underline{P}_t$, the strength st_i is computed, i.e. the number of solutions that s_i dominates in the set \underline{P}_t . The fitness of s_i is calculated as st_i/N . The fitness of a dominated solution $s_j: s_j \in \underline{P}_t$ is computed as $1 + \text{SUM}(st_i)$ for such s_i that dominate s_j . Consider the example of fitness assignment in Figure 19.

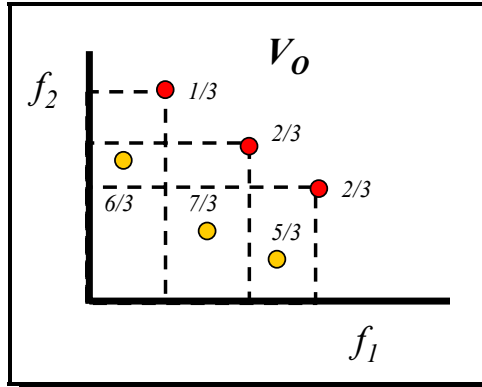


Figure 19. Fitness calculation based on the SPEA algorithm

Note that the fitness value of a dominated solution increases if it is dominated by more Pareto optimal solutions. The fitness function is therefore minimized, i.e. dominated solutions with smaller fitness values are preferred. This mechanism intuitively reflects the idea of preferring individuals near the Pareto-optimal front and simultaneously distributing them along the trade-off surface. The calculation of the fitness of dominated solutions as $1 + \text{SUM}(st_i)$ ensures that dominated solutions always have a fitness greater than Pareto optimal solutions.

2.7.3 Diversification strategy

The diversification of the population is provided by genetic operators, i.e. by recombination and mutation. These operators are applied to individuals in the population according to some probabilities that are parameters of the genetic algorithm. The application of these operators can lead to a fast convergence of a GA to some solution region. Genetic algorithms might therefore provide an additional means of improving the diversity of the resulting solutions.

- D. 52 (Mutation) Mutation is called an operation which randomly varies a single decision value of an individual. Mutations are identical to the random variations of values. Consider an individual s_i , such that $s_i.a_{j+3} = v_{ij+3} = r$, where $R_{j+3} = \{a, b, c, \dots, z\}$. The value v_{ij+3} is randomly mutated to another value from the attribute range, for instance to d , as shown in Figure 20 a).

The result of a mutation is the same individual but with a changed value. The mutation operation allows a single solution to be varied. However, the chance of a successful mutation is low. The mutation operation is usually applied with a low probability.

- D. 53 (Recombination) The operation of recombination operates with two individuals instead of one. Parts of individuals are randomly picked up and a new individual is created by combining the parts, as shown in Figure 20 b).

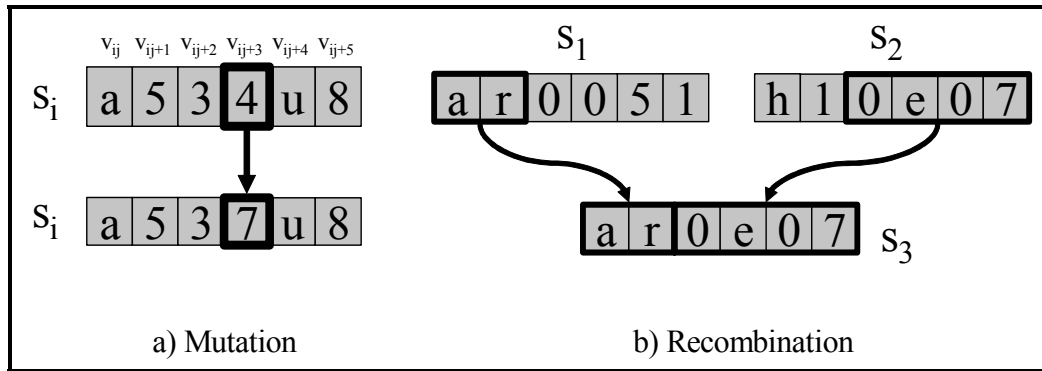


Figure 20. The mutation operation

The chance of creating a successful individual by recombination is considered to be higher than that by mutation, for the following reasons:

- more values of a solution are varied at the same time, and the chance is therefore greater that a new individual will escape a dominated region in the objective space
- the operation is not completely random because only the successful individuals are allowed to be recombined.

2.7.4 Density preservation

SPEA tries to maintain diversity in the current approximation of the Pareto set by incorporating density information into the selection process: an individual's chance of being selected is decreased if the density of individuals in its neighborhood is high. This issue is closely related to the estimation of probability density functions in statistics.

Nearest neighbor techniques [Silverman, 1986] take the distance of a given point to its k^{th} nearest neighbor into account in order to estimate the density in its neighborhood. The estimator is usually a function of the inverse of this distance. SPEA [Zitzler, 2002] calculates the distance value of each individual to the k^{th} nearest individual and adds this value to the raw fitness value (fitness is to be minimized). The distance between individuals is calculated with respect to the objective space.

2.7.5 Elitism strategy

Random effects (genetic operations, selection) can result in good solutions (solutions with a good fitness value) being lost from the population during the optimization process. A common way to deal with this problem is to maintain a secondary population, the so-called archive, to which non-dominated solutions in the population are copied from each generation. In SPEA the archive is integrated into the selection process by including archive members in the calculation of the fitness.

The size of the archive is usually restricted due to memory and run-time limitations. Criteria therefore have to be defined which would allow the size of the archive to be reduced (so-called pruning). The dominance criterion is most commonly used, i.e. dominated archive members are removed and the archive comprises only the current approximation of the Pareto set; density information (less crowded regions are preferred

to crowded regions) [Knowles 1999] and the time that has passed since the individual entered the archive [Rudolph 2000] can also be used (old individuals are preferred to new ones).

CHAPTER 3.ACCEL

“Almost everything follows in a natural way from a well-designed modeling language” [Hurlimann, 1999].

3.1 Introduction

The designers can use our model when thinking about the design and when putting their thoughts down on paper. The model helps them to identify meaningful product knowledge and transform it into the product model. In the early stages of the project we practiced the model using standard tools such as Microsoft Excel. We found that the most crucial ingredients of the model, i.e. design decisions, multiple objectives and knowledge categories, were not well supported. From the moment that we introduced operations involving the generation of solutions and optimization, a support tool became essential; these operations are too tedious for the designers to carry out, and are not supported by standard tools. We therefore decided to support these operations using a software tool. In this chapter we will mainly be concerned with the development process of the support tool and with a demonstration of its basic functionality. Examples of more advanced usage, such as the evaluation and methodology of designing with a tool, are covered in the next chapter.

We knew early on that we did not want to design a support tool packed with built-in features; from the literature we already had evidence that this type of tool does not guarantee usefulness. Instead, we wanted to design a tool that was as empty as possible while still being able to deliver useful operations supporting concepts generation. In this respect the developed tool is more an environment than a tool capable of direct support of decision-making. We needed to resist the temptation to provide structures for products, checklists, and all sorts of recommendations on how the design process should be done. In this way we made existing approaches compatible with our approach.

We developed a support tool according to our model of the design process, as presented in the previous chapter. The name of the tool is derived from the names of the basic ingredients of the model, i.e. Attributes Concepts Constraints²³ Evaluation Language (ACCEL). We use the term ‘*user*’ to refer to a person who operates ACCEL. This could be a designer, a stakeholder, an expert, etc. ACCEL provides the user with the following functionality:

- a graphical user interface for viewing and updating product knowledge according to our operational model
- a syntax for expressing knowledge

²³ In the early stages of the project values were called ‘constraints’. The name ‘ACCEL’ was created at that time, and has not been changed for compatibility reasons.

- evaluation of knowledge
- multi-objective optimization based on the described genetic algorithm
- a sensitivity analysis and more.

The development of a software tool for the early design phase is similar to the design situations that we considered in the introduction: many crucial decisions have to be taken on the way, there are many competing objectives and the design problem is ill-defined. There are many different ways of handling this complexity. Programming languages, software development methods and principles, high-level modeling languages and software tools to aid the development process, are all considered to be elements of the solution. However, none of the available methods and tools provide a ready-made solution. The developed operational model does not provide the solution either. The model allows us to derive *what* a support tool should do, but not *how* it should do it.

3.1.1 The tool structure

If we adhere to design principles, the tool should maintain a clean separation between the data model and the user interface. Software tools must provide facilities to allow the exploration and modification of data via a ‘*graphical user interface*’ (*GUI*). The tool needs to deal with the task of maintaining the data consistency between the data model and the GUI. It can be difficult to integrate the GUI with the data model to create a complete tool application, while at the same time keeping a clear distinction between the two. These are the considerations that we took into account when taking decisions about the structure of ACCEL. All the data is aggregated in the document object. The document object interacts with the GUI for two reasons: to show the data to the user and to update the data. Additional functionality is necessary for data management, for instance the parsing of the user input, the interpretation of the user actions etc. Such auxiliary functionality is delegated to the ‘*application manager*’. The structure of ACCEL is shown in Figure 21.

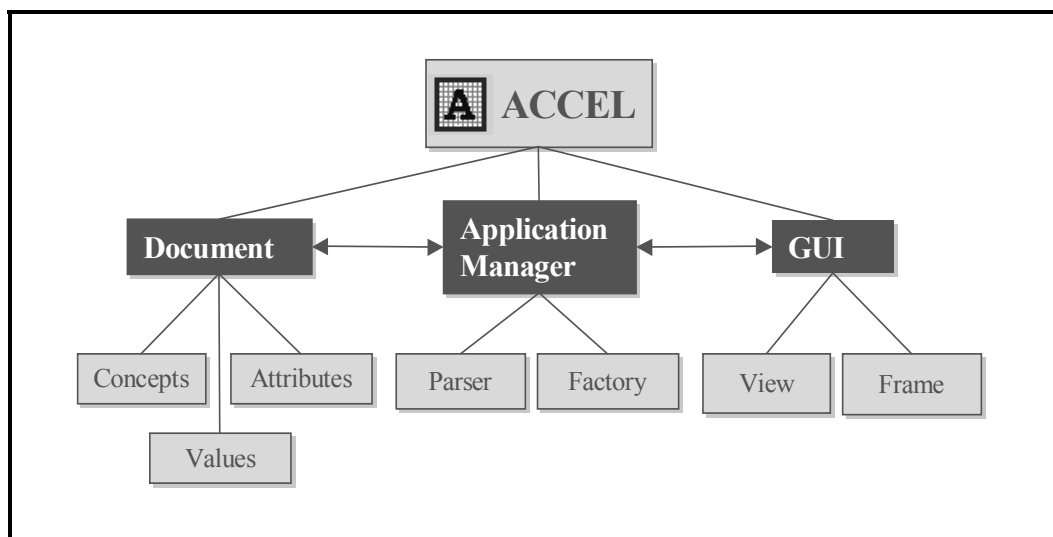


Figure 21. The structure of ACCEL

Our approach to design of a software tool (which we will call the ‘*development*’ process) is based on a mixture of the top-down and bottom-up approaches, both of which are common software development practices. Gilb [1988] refers to the top-down approach as an ‘*evolutionary delivery*’. This approach introduces major software development principles and helps the designer to avoid serious mistakes in software development projects. We present this approach in section 3.1.2. The bottom-up approach is based on ‘*Object-Oriented Design*’ (OOD). OOD enable the flexible management of software complexity, which results in extendable and maintainable software architectures. We present this approach in section 3.1.3. The remainder of this chapter is outlined in section 3.1.4.

3.1.2 Top-down approach

The essence of evolutionary delivery is to produce software systems in small useful steps as opposed to *revolutionary delivery*, where the whole system is delivered at the end of a project. The evolutionary approach prescribes several basic principles, which are regarded as a set of initial guidelines in software engineering management. The following definitions are given by Gilb [1988]:

- D. 54 “(The invisible target principle) All critical system attributes must be specified clearly. Invisible targets are usually hard to hit (except by chance).”

We used this principle to approach the formal definition of functional requirements and to define the set of initial objectives.

- D. 55 “(The all-the-holes-in-the-boat principle) Your design solutions must satisfy all critical attributes simultaneously.”

We aim to evaluate a solution with respect to all objectives, as described in the concluding chapter.

- D. 56 “(The learn-before-your-budget-is-used-up principle) Never attempt to deliver large, complex systems all at once; try to deliver them in many smaller increments, so that you can discover the problems and correct them early on.”

The developed tool was delivered in several phases; in each phase the functionality of the tool was extended with what we considered to be the most valuable functions.

- D. 57 “(The keep-pinching-yourself-to-see-if-you-are-dreaming principle) Don’t believe blindly in any one method; use your methods and common sense to measure the reality against your needs.”

We derived functional requirements for a tool using formal reasoning. We built a tool from scratch; only the basic graphical components were reused.

We need to clarify the term ‘*critical attribute*’. Gilb gives the following definition:

- D. 58 “(Critical attribute) A critical attribute is one which, if it got out of control, would threaten the viability of the whole solution.”

3.1.3 Bottom-up approach

Object orientation does not only provide the means to develop software. The same principles can be used to design software using modeling languages, which are based on the object-oriented approach to design (OOD approach). The structure of the software

systems can become very complex and barely manageable. There are two major approaches to manage the complexity: a) system decomposition in terms of algorithms (functional elements of systems) and b) system decomposition in terms of objects that interact with each other; the choice of a) or b) is a matter of personal preference. The same system can be structured using either approach (but not both at the same time). The advantage of b) is that systems designed using the OOD approach are more flexible and easier to evolve and update over time. Before we look into the application of an OOD approach to design of our tool, we first need to introduce the basic OOD terminology. The following definitions summarize several of the standard texts such as Booch [1993], Booch [1998] and Gamma [1995].

D. 59 (Class) A ‘*class*’ is a model of a group of similar objects (real or imaginary) that describes ‘*properties*’ and ‘*methods*’ that determine the behavior of the objects. We will use the following font to denote classes: `class A`.

D. 60 (Encapsulation) Encapsulation is known as aggregation in a class of properties, which are called *data members*, and methods, which provide operations with data members and provide secured access to them from other classes.

D. 61 (Types of methods) Different types of methods provide the security. Methods (properties) of a class can be ‘*public*’ (accessible by any other classes), ‘*protected*’ (accessible by the class itself, inherited classes or specially designated classes called ‘*friend*’ classes), or *private* (accessible by the class itself and friend’s classes).

D. 62 (Object) An object is an instance of a class.

We will underline the names of objects using the following font: object A.

D. 63 (Abstract class) A class is abstract if it is used to provide a common interface for the inherited classes via public data and public methods. New objects cannot be instantiated from the abstract classes.

Various relations can hold between different classes of objects, such as an ‘*association*’, ‘*aggregation*’, ‘*usage*’, etc.

D. 64 (“is-a” relation) A class A “is-a” class B if A is based on the model of B, i.e. properties and methods of B are meaningful for A and can be used by A. Class B is then called a *parent class* and class A is called a *child class*. In OOD, the “is-a” relation is denoted by an outgoing arrow from A to B.

Note that the above definition does not prohibit a class **A** from having different properties or methods from a class **B**. In fact, for some methods, for example with name *P* in class **A**, method *P* might have a different behavior in class **B**.

D. 65 (Polymorphism) Polymorphism refers to a variety of different behaviors inherited from other classes, but represented by the same method name.

Polymorphism is a complementary concept, which makes the “is-a” relation more practical. It enables the responsibility for executing the right method to be delegated to objects while executing the program. This is not trivial when dealing with objects of under-defined classes, i.e. the parent class of an object is known but the specific class of the object is unknown.

Another relationship between the classes that we will use is called *aggregation*. Aggregation is identical to the “part-of” relation between classes. When a class **A** is aggregated in a class **B**, it means that class **B** is a data member of class **A**.

D. 66 (Aggregation) A class A is “part-of” a class B if all data and methods that are encapsulated in A are encapsulated in B. We will denote the Part-of relation using an

arrow that goes from the container class to the contained class and contains a diamond at the container class.

In summary, we can say that encapsulation, inheritance and aggregation are considered to be the three major principles of the OOD approach to design and implementation.

3.1.4 Outline

We continue this chapter with the definition of functional and software requirements, which are described in section 3.2. In section 3.3 we present the data model, which is included in the ACCEL document. Section 3.4 describes the approach taken to interpret the user input and the evaluation of knowledge. In section 3.5 we describe the support of some specific forms of giving answers, such as asking, postponing and deciding. The major decisions underlying the graphical user interface are presented in section 3.6. Here we will provide short examples of ACCEL usage in order to give the reader an initial appreciation of the tool's functionality. More detailed examples are given in the next chapter, which describes several case-studies. In section 3.7 we present the most advanced operation that can be done using ACCEL. This section describes how a designer can optimize solutions in the early design phase. We present an optimization dialog (which gives the user control of the optimization problem), we present our method of decision making, and we provide an example.

3.2 Requirements analysis

This section presents a major link between the intuition of the designers and the operations that are formally defined and supported by ACCEL.

3.2.1 Functional requirements

The definition of the functional requirements of a software tool is the starting point for the development process. The success of the tool depends heavily on the completeness and accuracy of the requirements list. The operational model in chapter 2 provided a formal mechanism to distinguish eight categories of questions. We defined how to give answers for each category. In this section we translate these results into basic functional requirements, and give a reference to the section that describes how the requirement is addressed in the tool. We ensure the completeness of the list by considering three orthogonal dimensions, i.e. objectivity, dependency and the availability of the answer. The accuracy of the requirements is provided by the formal nature of the model. The functional requirements are shown in Table 7. We define these requirements as general requirements for the operational support of product knowledge.

It is critical for the tool that these functional requirements are satisfied. There are other functional requirements, which were initially considered to be secondary, such as the GUI functionality, feedback to the user etc.

Table 7. Basic design activities and the corresponding functional requirements

Design activity	Resulting functional requirements	Reference
Asking questions	Addition of concept, attributes and variables i.e. $c_i, a_j, c_i.a_j$	3.6.1
Categorizing a question	Applying a category for a variable	3.6.1, 3.6.2
Guessing: Give a provisional answer (making an educated guess)	Adding a value	3.6.1
Deciding: Give a multi-valued answer, which combines alternative answers to a question	Giving a multi-valued answer	3.5.1
Modeling: Answer is given as a functional dependency	Adding a functional dependency	3.6.1
Postponing: Postpone giving the answer	Adding undefined variable	3.5.3
Defining: Give a definitive, full and complete answer	Adding a value	3.6.1
Searching. Searching for an answer	Adding a functional dependency that refers to an external tool	Not supported
Parametrizing: Composing a functional expression that takes into account personal preferences of the stakeholder	Adding a functional dependency and values that express personal preferences of the stakeholder	3.5.3
Asking: Compose and delegate a question to the stakeholder	Adding a functional dependency that composes questions and interface with the user in order to benefit from interpretive abilities of the user at all necessary moments	3.5.2,3.6.3

3.2.2 Additional functional requirements

The set of additional requirements follows from the set of functional requirements and the definition of the model from chapter 2. The additional functional requirements are to:

- formally distinguish between four categories of variables (see sections 3.6.1 and 3.6.2)
- automate the optimization (generation and evaluation) of solutions (see section 3.7)

- support the four types of attributes, namely Nominal, Ordinal, Interval and Ratio (see section 3.6.2)
- provide syntax for encoding functional dependencies (see section 3.3.1)
- provide a syntactical mechanism for the evaluation of functional dependencies (section 3.4)
- enable error checking and feedback (see sections 3.4.4 and 3.6.5)
- provide a user interface for all functional requirements (see section 3.6)
- provide support in the form of a help system (see provided installation of ACCEL)
- automate sensitivity analysis (see section 3.6.4).

3.2.3 The choice of C++

The choice of the programming language and the development environment are fundamental to any software tool. In this project, the choice of the development software was determined by the OOD approach and the high demands for the computational efficiency. We therefore decided to implement the tool using Visual C++. Our choice was influenced by the following factors:

- **Popularity.** Visual C++ is popular and well supported. Most developers can understand this language. There is a powerful development environment that supports the C++ language, namely Microsoft Developer Studio 6. C++ is well documented; there are multiple libraries and controls that are capable of accelerating the development process.
- **Computational efficiency.** C++ is an extremely efficient language; the produced programs are computationally faster than programs written in Visual Basic or Java.
- **Familiarity.** We are familiar with C++.

3.2.4 Context and constraints

When developing a software tool a number of contextual values have to be made explicit. One of the values we considered was that the stakeholder (i.e. TU/e) uses one of latest versions of Microsoft Windows (version Windows 2000 and later) as its operating system. A constraint was that a total time of four years was available to develop the tool. The first year was spent developing the model and the defining the functional requirements. The last year was spent testing the tool and writing this thesis. Therefore, the actual development time was less than two years. In this period several working prototypes of the tool were delivered. We decided to have the first working prototype ready within the first six months.

3.3 ACCEL document: Object oriented data model

We developed the data model by considering types of data, syntax and the basic operations that would be performed with the data.

3.3.1 Syntax

We will refer to anything that the user inputs into the tool as ‘*an expression*’.

- D. 67 (Syntactical form of an expression) This is the form of an expression as the user inputs it into ACCEL. We will denote such an expression by pointy brackets, i.e. <expression>.

According to the functional requirements, an expression may consist of a mixture of the following elements: c_i , a_j , v_{ij} , $c_i.a_j$. The tool’s syntax enables the analysis and translation of the expressions typed in by the user into the tool. Thus, using the syntax of ACCEL transforms expression → <expression>.

The syntax should be familiar and natural to the user, while at the same time being formal and straightforward to translate into an efficient representation. A natural language such as English is an extreme example of user friendliness, but it would be extremely difficult to formally translate expressions written in English. An opposite but extreme example is machine code, which enables us to represent knowledge in automatically processable terms, but which is extremely unnatural for non-experts and is time consuming to learn.

We therefore based the syntax on the notion of a function. We assume that the user is familiar with the general syntax of functions and otherwise this skill can be learned from section 2.4. Given these assumptions, we treat any expression as a ‘*functional expression*’. The resulting syntax is shown in Table 8.

Table 8. The syntax of ACCEL

<pre> expression::=function function::=<fname>(<arguments>) arguments::=<argument>; <arguments> argument::=function reference concept value operator fname::= IFE ALT ASKUSER IS, ... /* for a complete set of functions see Appendix B*/ concept::= an element from the set of concepts C attribute::= an element from the set of attributes A. reference::=concept[.attribute]* ²⁴ operator::=argument <oname> arguments. oname::= + - / ^ , ... /* for a complete set of operators see Appendix B*/ value::= one of the basic types of values. </pre>

²⁴ Here, [P]* means one or more instances of P. a[b]* → ab, abb, abbb,...

3.3.2 Everything is a function

We use the inheritance principle to derive a software representation for the defined syntax. Every syntactically distinguished term is a class, which is inherited from the class **function**. Therefore, any user input is treated as a function, i.e. every chunk of knowledge is represented as a function in ACCEL. The purpose of the class **function** is to define a common interface for all inherited classes, such that the variety of functions can be managed in a uniform way. The class **function** is an abstract class, which determines the following common interface:

- an array of arguments
- a set of public methods to access and modify the arguments
- polymorphic methods to enable error checking,
- a method Evaluate(), which is specific for every function and is therefore polymorphic
- a number of polymorphic methods to get the result of an evaluation

Note that inheritance and polymorphism will allow us to deal with the evaluation of functions in a distributed way, as shown in section 3.4.3 Five classes representing ‘*basic functions*’ are inherited from the class **function**, namely **concept**, **attribute**, **nfunction**, **bfunction** and **cfunction**. These classes share the interface of **function**, but unlike **function** they are not abstract; they are allowed to instantiate new objects. We refer to these classes as basic because they represent basic ingredients of the model, namely concepts, attributes, values and functional dependencies. ‘*Nominal values*’ are represented by the class **cfunction**; ‘*numeric values*’ are represented by the class **nfunction**; ‘*boolean values*’ are represented by the class **bfunction**; concepts by the class **concept** and attributes by the class **attribute**. Since objects of these classes represent constants, basic functions have no arguments. In order to represent functional dependencies we introduce more specific functional classes that are inherited from the basic classes, e.g. **plus** is inherited from **nfunction** and enables the operation of summation, **equal** is inherited from **bfunction** and enables the operation of comparison etc. The resulting hierarchy of classes is shown in Appendix B. This appendix also provides the list of functions and operators that are included in the syntax of ACCEL.

3.3.3 Data structure

Using the developed hierarchy of classes, all data is organized in the object-oriented data model, which is aggregated in the ACCEL document. ACCEL document is embodied in a class **CAccelDoc** which is inherited from a standard class **CDocument**. The ACCEL document stores and controls the program’s data. Each kind of data, for instance, objects of the class **attribute**, is aggregated in a collection object of the class “**attributes**”. The purpose of the class **attributes** is to collect all attribute objects in one array and to enable useful operations with attributes, such as adding a new attribute, removing an attribute, renaming an attribute. A similar aggregation structure is used to aggregate other objects.

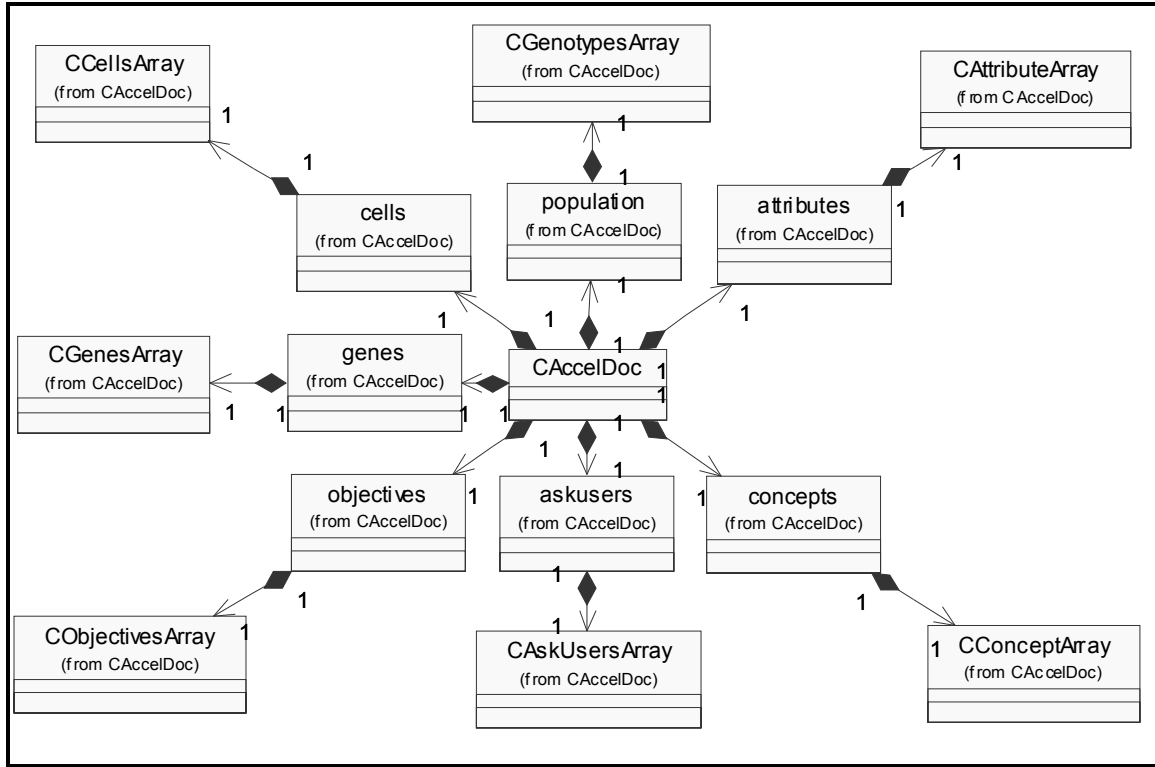


Figure 22. Data structure of ACCEL document

The following classes are contained in the ACCEL document:

- class **cells** aggregates all values v_{ij} and functional dependencies
- class **population** aggregates solutions s_i
- class **concepts** aggregates concepts c_i
- class **attributes** aggregates attributes a_j
- class **askuser** aggregates references to all functions ASKUSER() (see section 3.5.2)
- class **objectives** aggregates references to the objectives and the constraints
- class **genes** aggregates references to all decision variables s_i^d .

3.3.4 XML format of data storage

Any knowledge entered by the user into the tool must be stored in a suitable external format. The choice of the external format for data storage was therefore dictated by the following objectives:

- flexibility, i.e. easy to update and extend the format
- easy to implement
- compatibility with other applications and external programs.

We considered the following alternative methods of external data storage: databases, a mechanism of serialization built in VC++ and the XML language. We chose the XML

language because we think it is the optimum solution with respect to the chosen objectives. A mechanism of serialization built in VC++ is not compatible with other external programs. Data storages based on databases are less easy to implement and they require additional support.

XML, otherwise known as eXtensible Markup Language, resembles other markup languages like HTML. XML's purpose is to describe both the structure and the content of a document. XML allows us to semantically markup the contents of any document, describing the contents in terms of its relevance as data. According to its specifications [Bray et al 2000] the design goals of XML are:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human legible and reasonably clear.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.

XML Constructs

An XML document is created using the following basic constructs: elements, attributes and entities.

- D. 68** (XML Elements) Elements are labels used to describe the content. They are described by element declarations and are invoked in the document element as tags. By default, element declarations define tag pairs. Tag pairs contain text as well as other elements and their content. An element declaration may also define an empty element, i.e. one that is not designed to contain any text or other elements.
- D. 69** (XML Attributes) Attributes provide extra information about the element. XML elements can have attributes in name/value pairs just like in HTML.

XML example

Table 9 shows an example of a fragment of an ACCEL document represented in XML format. This fragment corresponds to a box example, which we present in Ex. 20. The first symbol in each line of an XML document represents an operation of folding and unfolding²⁵. The root element includes all other elements in the document. Each element is distinguished by its name, which is given first in the pointy brackets. In our case the names of the elements correspond to the names of the classes in our data model. The name of the element is followed by a number of attributes. For instance, the element ATTRIBUTE is followed by the name of a concept, a description and coordinates of a concept in a GUI, etc.

²⁵ The fold – unfold option allows the sub-elements for every element to be hidden or shown.

Table 9. Example of an XML document

XML document	Model
<pre> _<RootElm> _ <CONCEPTS> ± <CONCEPT Type="3" Name="wire" Description="" Row="4" Column="0"> _ <ATTRIBUTES> ± <ATTRIBUTE Type="0" Name="1" Description="" Row="0" Column="2" Kind="0"> _ <CELLS> _ <CELL Type="0" Concept="wire" Attribute="1" Description=""> _ <Arguments> _ <NFUNCTION Type="0" Value="400"> </pre>	<p>$c_1 = \underline{\text{wire}}$ represents the concept 'wire'</p> <p>$a_1 = \underline{1}$ represents the attribute 'length'</p> <p>value $c_1.a_1 = 400$</p>

3.4 ACCEL manager: evaluation semantics

In section 3.3.1 we described the syntax that allows the user to formally express knowledge in ACCEL according to the formalism of our model. The next step is to discuss how the support tool enables operations with the expressed knowledge with respect to the eight functional requirements. We therefore introduced semantics into ACCEL. The purpose of the semantics is to enable the evaluation of expressions by the computer; our semantics are therefore of the operational kind. According to Winskel [1993], operational semantics describe the meaning of a programming language (which we refer to as a formalism). It describes how a language executes on an abstract machine.

3.4.1 Parser

The parser is part of the ACCEL application manager which implements the ACCEL syntax. Thus, the parser transforms the syntactical form of expressions into the semantic form.

- D. 70 (Semantic form of an expression) The semantic form is a form of an expression that has been interpreted and evaluated by the tool. We will denote such expressions as <expression>.

The function of the parser therefore corresponds to the transition <expression> \rightarrow <expression>. Two types of rules underlie the transition, namely syntactical rules and priority rules, which are defined for all the functions. The class of an expression is fully determined by the syntax of the expression. These rules are such that a function with the lowest priority, i.e. the one that has to be evaluated last one of all, will determine the class of the expression. Every expression is parsed into a functional tree from top to bottom (see Figure 23). Parsing is done recursively, such that the class of a function is

determined first, then the arguments are distinguished, and finally the arguments are parsed by a recursive call of the same parsing procedure.

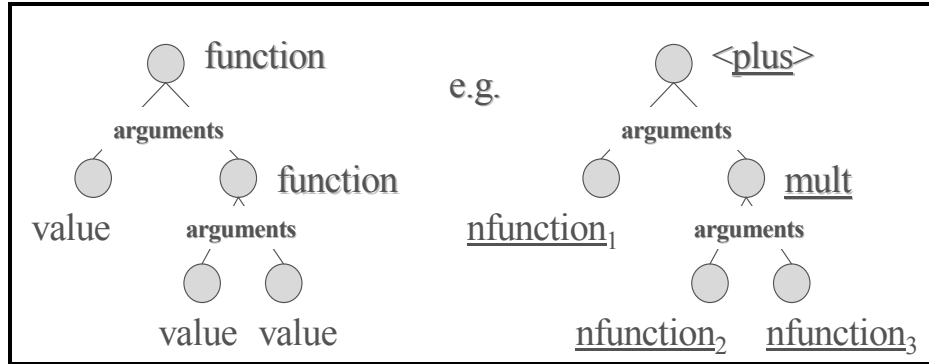


Figure 23. Functional tree

Ex. 16 An example of an expression that would correspond to the structure in Figure 23 would be $4+7*6$. Here, function object plus is at the top level. plus has two arguments, namely nfunction₁ with the value 4 and the function of multiplication mult; mult has two arguments called nfunction₂ with the value 7 and nfunction₃ with the value 6. The list of functions with some of the attributes that correspond to this tree is shown in Table 10. These attributes are: name (name of a function), value (value of a function), arg₁ (first argument) and arg₂ (second argument). In the next subsection we will consider the dynamics of how functional trees are built.

Table 10. The list of functions

	name	value	arg ₁	arg ₂
1	<u>plus</u>	46	<u>nfunction₁</u>	<u>mult</u>
2	<u>nfunction₁</u>	4		
3	<u>mult</u>	42	<u>nfunction₂</u>	<u>nfunction₃</u>
4	<u>nfunction₂</u>	7		
5	<u>nfunction₃</u>	6		

3.4.2 Factory of functions

After the parser has detected a function, the question is how to create an object of the right class? This task could be delegated to the parser itself, but this would result in delegating two complex tasks to the class **Parser**. A more elegant approach is to employ the software design principle (also known as the ‘*Design Template*’) called Abstract factory [Gamma 1995]. According to this principle, the class **Factory** provides an interface for a variety of different factories. Each factory is dedicated to a special class of functions or a specific software platform. Every time a new element in an expression is identified, the parser passes the name to one of the factory objects, which tries to create an object. The task of creating the semantic form of an expression is therefore divided between the classes **Parser** and **Factory**. **Parser** detects the names of the functions and builds functional trees, and **Factory**(s) creates proper objects. Now if the syntax has to be extended with a new function or a group of functions, this

problem can be simply solved by deriving a new class from the class **Factory**, instead of rewriting the code of **Parser**.

- D. 71 Consider the previous example. First, the function <plus> is detected by **Parser**. At this point parsing stops and this name is passed to **Factory**, which according to the syntax creates an object plus of class **plus**. This corresponds to row number 1 in Table 10. The pointer to the object plus is passed back to **Parser**, and the process of parsing continues. The function <4> is detected. The parsing procedure is recursively called. Since <4> cannot be parsed any further, the recursive procedure passes the name **Factory**, which according to the syntax creates an object nfunction1 and returns a pointer to this object to the recursive procedure. The recursive procedure returns to the original parsing procedure. Since objects of class **nfunction** have higher priority than objects of class **plus** according to the priority rules, object nfunction1 is aggregated in plus as the first argument and not the other way around. The parsing procedure continues in a similar way.

3.4.3 Evaluation of expressions

In this section we describe how semantic form of expressions is evaluated.

- D. 72 (Evaluated expression) An evaluated expression is a semantic expression that has been evaluated to a basic value. We will denote it as <expression> → <value>.

The chosen data structure enables the distributed evaluation of expressions, i.e. every function object evaluates itself according to a polymorphic method Evaluate(). Unlike the process of parsing, there is no single object that is responsible for the whole evaluation process.

Every function class derived from **function** inherits the public method Evaluate(), which is class specific. There are characteristic steps that underlie the evaluation of all functions. We present these steps as an evaluation algorithm. The input to the algorithm is the array of arguments of a function. The output of the evaluation algorithm is a pointer to the resulting function object and a Boolean value that indicates the successfulness of the evaluation process. If the evaluation was successful, the resulting pointer will always point to an object of a basic function class. Note that when evaluating basic functions that represent constants, such as **nfunction** or **concept**, the resulting pointer will point to the basic function itself. The evaluation algorithm is presented below.

- D. 73 (Evaluation algorithm)

Input: CFunctionArray arg; (array of function arguments)
Output: function* result; (a pointer to the result)
 bool e; (the result of an evaluation)

Steps	Code	Comments	Function plus from Ex. 16
Step 1	e=true;	Initialize evaluation	
Step 2	e=e&EvaluateArguments();	Evaluate all arguments and return a Boolean result.	Evaluate <u>nfunction1</u> and <u>mult</u> . e= true.

Step 3	<code>e=e&CheckType()</code>	Checks the type of the arguments and return a Boolean result	<code>e=true.</code>
Step 3	Perform specific operations with the arguments and assign the result to the value	Examples for the defined functions are shown in APPENDIX	Value=46
Step 4	Return <code>e;</code>	Returns the result of evaluation	<code>e=true</code>

3.4.4 Error handling

The final topic related to the evaluation of knowledge is error handling. As we discussed in the introduction, design knowledge is incomplete and can be inconsistent. The detection and feedback of such knowledge pit-falls would help to correct them. We will first describe what kinds of errors are handled by ACCEL and then indicate how the feedback about errors can be given in a convenient way.

Our model enables the automated detection of the following pit-falls that prevent the evaluation of expressions:

- syntax errors, such as non-parsing parenthesis, unknown names of functions, repeating names of concepts or attributes
- semantic errors, such as wrong types of arguments for the functions, inconsistency between the value type and the range of the corresponding attribute
- hanging references, such as the use of unknown names of concepts or attributes in reference functions
- cyclic dependencies among values

The detection of errors in ACCEL is done in an object oriented way, i.e. every function has a polymorphic method that enables specific error checking.

The method of providing feedback about the detected errors is an important design decision; too obtrusive or unobtrusive forms of giving feedback should be avoided. The feedback about an error should ideally be given in a way that will help the error to be fixed. Decisions have to be taken about accuracy, and about the place and the time of giving feedback to the user.

We based the decision about the place of giving feedback on logical considerations. The errors come from the expressions and therefore unevaluated expressions should indicate detected errors. However, detailed feedback inserted in an unevaluated expression would disturb the perception of the expression. This problem is solved in ACCEL using the partial evaluation of expressions.

Expressions that are not fully evaluated may still contain some evaluated arguments. The essence of partial evaluation is to compose unevaluated semantic expressions from the partially evaluated arguments. In the composed expression the user can see arguments which were not evaluated. Additional information about the reasons for the

error is shown in a separate field on the GUI. See section 3.6.5 for more details on this and for an example of giving feedback.

The decision about when to give the feedback follows from the necessity of the feedback and the frequency of the errors. Since we assume that knowledge in the early phase is often incomplete, the feedback should ideally be given as soon as possible. ACCEL therefore provides continuous error checking and user notification.

3.5 Special functions

We were able to satisfy most, but not all, of the functional requirements using standard functions, or functions which are identical to Microsoft Excel functions. Some functional requirements, e.g. giving multi-valued answers, searching, asking, parameterized answers and postponing answers, could not be mapped onto existing mathematical functions. These requirements therefore required dedicated approaches and dedicated user interfaces.

3.5.1 Function ALT()

The function $\langle \text{ALT}() \rangle$ is abbreviated from the word “alternative”. This function allows the designer to express a multi-valued answer, i.e. a decision. Decision options are represented by the arguments of $\langle \text{ALT}() \rangle$ according to the syntax: $\langle \text{ALT}(\text{function}_1, \text{function}_2, \dots) \rangle$. Note that the function $\langle \text{ALT}() \rangle$ does not allow us to work with continuous ranges. The user therefore needs to select a number of values from the range if the range of the variable is continuous. We plan to remove this limitation in the future.

Ex. 17 (Energy supply decision) Consider the designer of an electric lamp. One might ask the question: what could the energy supply for the lamp be? A new concept Electric lamp and a new attribute .Energy supply with range $R_{\text{energy supply}} = \{\text{battery}, 220\text{V}\}$ would formally represent the question. The answer aims to produce the value for the variable $\langle \text{electric lamp.energy supply} \rangle$. The answer could be a battery or a 220V circuit. Such a multi-valued answer can be formally expressed in ACCEL as $\langle \text{ALT}(\text{battery}, 220\text{V}) \rangle$.

The function $\langle \text{ALT}() \rangle$ can be evaluated in two mutually exclusive ways: 1) explicitly represented decision values are allowed to be varied during the optimization procedure 2) a decision value is manually selected by the user. In the second case, the corresponding variable is treated as a fixed value and is not varied during optimization. The function $\langle \text{ALT}() \rangle$ will contain all the alternatives even if the decision is manually selected. The decisions taken can be easily backtracked. For usability purposes, the evaluation of undecided $\langle \text{ALT}() \rangle$ functions result in the first argument except when generating solutions with the generic algorithm. However, this does not mean that the decision is made.

3.5.2 Function ASKUSER()

The function $\langle \text{ASKUSER}() \rangle$ allows the designer to express interpretational functional dependencies, i.e. dependencies which cannot be expressed in mathematical functions due to the implicitness of knowledge about them. The missing functional dependencies

can often be replaced by a subjective opinion from the user, which in this case performs the role of a function: the user interprets the input arguments and produces an answer. However, the technical problem is to ask questions in an appropriate way, so that the questions are accurate and that the same question is not asked twice. Indeed, it is important to minimize the number of questions, i.e. the number of necessary interactions with the user, as these might be disturbing.

<ASKUSER()> allows us to acquire a value from the user whenever necessary. Not all values contained in a solution are necessary for the user to form an opinion about a certain objective value. A few decision values of a solution will often be sufficient to form an opinion. This means that, even if millions of different solutions have to be evaluated, far fewer different combinations of the values need to be considered. ASKUSER> has the following syntax: <ASKUSER(arg₁,arg₂,arg₃...)>. The answer obtained from the user is expressed according to the syntax of the expression. If the number of arguments is sufficiently limited there will be,

<ASKUSER()> can be used most effectively during optimization procedures. Automatically varied decision values might create new combinations of the arguments. A new answer is required for every new combinations. For this reason all answers are kept in the history object, which is aggregated in the ASKUSER() function. As a result, the user input is only needed for newly-encountered combinations of the arguments (new questions). Each time the same set of values for the arguments re-occurs, the information can be taken from the history, so that the same question never has to be asked twice. This history allows the number of interactions with the user to be reduced significantly.

Consider an example from the previous chapter, where we evaluated the attribute Safety for various means of transportation. Each transportation means Tm might contain a large number of values, which we can neglect during the evaluation of the safety. Let us assume that we are able to form a rough opinion about tm.safety if we are given information about the energy source and the media of an instance of Tm. In this case values for <tm.safety> will be obtained by interpretation, which is expressed in ACCEL as <Askuser(tm.energy_source; tm.media)>. Even if millions of solutions are generated, the evaluation of their safety will require only 3*3=9 interactions with the user, since decision values are taken from R_{energy_source} and R_{media} , which have three values each.

3.5.3 Reference function

The reference function is designed to enable 1) referencing to other values from expressions 2) postponing the answers and 3) flexibility in decision making. It has the following syntax: <concept₁.attribute₁.attribute₂...>. This section addresses the mechanism of the reference function. At the end of this section we will consider how the function <THIS> enhances the flexibility of this mechanism.

In the simplest case, namely when the syntax <concept₁.attribute₁> is used, the meaning of the expression is clear; it means the value v_{11} . Now suppose that v_{11} is a concept in its own right, say <concept₂>. In this case syntax <concept₁.attribute₁.attribute₂> would mean <concept₂.attribute₂>. Note that $c_1.a_1.a_2.a_3...a_{n-1}.a_n$ assumes that $a_n \in \phi(c_1.a_1.a_2.a_3...a_{n-1})$, otherwise such an expression cannot be evaluated, i.e. $attribute_2 \in \phi(\text{concept}_2)$.

In this way, the designer can refer to information about the concept $\langle \text{concept}_2 \rangle$, even if concept_2 has not yet been made specific. This enables us to postpone answers without losing the possibility of referring to such postponed answers. It is obvious that the expression $\langle \text{concept}_1.\text{attribute}_1.\text{attribute}_2 \rangle$ cannot be evaluated until the concept $\langle \text{concept}_2 \rangle$ is known. Also note that if $\langle \text{concept}_2 \rangle$ changes in the future, the expression $\langle \text{concept}_1.\text{attribute}_1.\text{attribute}_2 \rangle$ does not have to be changed, while the expression $\langle \text{concept}_2.\text{attribute}_2 \rangle$ would need to be updated manually. In such a way the reference function provides extra flexibility to the developed product model. We will explain the evaluation of the reference function in a simple example. More interesting examples are provided with the installation of ACCEL²⁶.

- Ex. 18** Consider a conceptual solution $\langle \text{car} \rangle$. A question such as “what engine should the car have?” results in a new variable $\langle \text{car.engine} \rangle$. Suppose there can be a diesel engine or an electric engine, but we are not yet certain. We can postpone giving an answer to this question. For the spatial arrangement of the car it might still be necessary to talk about the volume of the engine. In this case we can refer to the volume of the engine even without knowing which specific engine is going to be used. In all expressions that require a reference to the volume we can write: $\langle \text{car.engine.volume} \rangle$. This expression is transformed into the semantic form $\langle \underline{\text{car.engine.volume}} \rangle$ as explained in Table 11. Now, suppose the variable $\langle \text{car.engine} \rangle = \langle \text{ALT}(\text{diesel engine}, \text{petrol engine}) \rangle$ represents a decision, where $\langle \text{diesel engine} \rangle$ is a concept in its own right, such that $\langle \text{diesel engine.volume} \rangle = 1.8$. and $\langle \text{petrol engine} \rangle$ is a concept in its own right, such that $\langle \text{petrol engine.volume} \rangle = 2$. Consider the evaluation of $\langle \underline{\text{car.engine.volume}} \rangle$ according to evaluation algorithm D. 73. All the arguments of the reference function are basic functions, therefore their evaluation does not yield new functions and results in a true value for the result. The evaluation of the reference function begins with the first two arguments. If the selected pair of arguments is evaluated into a concept, the resulting concepts and the next unconsidered argument will be considered as the next pair of arguments. And so it continues until all the arguments have been considered. $\langle \underline{\text{car.engine}} \rangle$ is evaluated by default into the first argument, which results in the concept $\langle \underline{\text{diesel engine}} \rangle$. The next argument is then considered. This results in the expression $\langle \underline{\text{diesel engine.volume}} \rangle$, which is evaluated to 1.8 according to the definition. Now, suppose we decide to use a petrol engine, i.e. $\langle \text{car.engine} \rangle = \langle \underline{\text{petrol engine}} \rangle$. In this case the evaluation of $\langle \underline{\text{car.engine.volume}} \rangle$ will yield value 2. We can also let the optimization algorithm vary values of the decision variable as will be explained in section 3.7.

Table 11. Semantic form of the expression $\langle \underline{\text{car.engine.volume}} \rangle$

	name	value	arg ₁	arg ₂	arg ₃
1	<u>reference</u>		<u>concept₁</u>	<u>attribute₂</u>	<u>attribute₃</u>
2	<u>concept₁</u>	car			
3	<u>attribute₂</u>	engine			
4	<u>attribute₃</u>	volume			

²⁶ The installation is available at the web site <http://sts.bwk.tue.nl/Ivashkov>

In the considered example the arguments of the reference function always refer to a specific concept and specific attributes. The syntax of the function is therefore context independent, i.e. if the reference function is assigned to another variable, the meaning of it stays the same. In many cases it is convenient for the user to make the reference function context dependent, so that if the function is assigned to a new variable, its arguments will automatically refer to a concept or an attribute of this new variable. For this reason we introduce the function `<THIS>`. The `<THIS>` function has no arguments but its outcome depends on the context. It can be used in a reference function of a functional dependency, which aims to obtain a value of a dependent variable $c_i.a_j$. In this case, `THIS` will return the concept c_i or the attribute a_j of the variable. `THIS` simplifies modelling, because the same expression can be reused easier.

Ex. 19 (Function THIS) Suppose the price of a car can be computed using the following expression: `<car.price> = <car.production.price+car.marketing.price>`. Suppose we have a `<diesel car>` and a `<petrol car>`, and their prices can be computed according to the expression for `<car>`, but the reference functions in this expression would yield different results for the two cars. For instance, `<diesel car.marketing.price>` is computed differently than `<petrol car.marketing.price>`. In order to make the expression context dependent we use the `<THIS>` function, i.e. `<car.price> = <THIS.production.price+ THIS.marketing.price>`. Now this expression can be copied to both cars. In this case the reference functions `<THIS.production.price>` and `<THIS.marketing.price>` will automatically refer to the values of the right cars. In a similar way we gain context independence for the attribute of a variable, i.e. `<car.price> = <THIS.production.THIS+ THIS.marketing.THIS>`. If the same expression is to be used to compute `<car.investment>`, the user can copy the expression for `<car.price>`, i.e. `<diesel car.investment> = <THIS.production.THIS+ THIS.marketing.THIS>` and `<petrol car.investment> = <THIS.production.THIS+ THIS.marketing.THIS>`.

3.6 ACCEL GUI: Graphical user interface

Microsoft Visual Studio provides a programming model that separates a program's data from the display of that data and from most user interaction with the data. In this model, a document object reads and writes data to persistent storage. The document may also provide an interface to the data wherever it resides (as we described in section 3.3.4, all data is stored externally in an XML document). A separate view object manages the data display, from the rendering of the data in a window to user selection and the editing of data. The view's responsibilities are to display the document's data graphically to the user and to accept and interpret user input as operations acting on the document.

The document/view separation introduces compelling reasons to follow the Visual Studio programming model. One of the main advantages is that multiple views of the same document can be provided; for instance, ACCEL supports multiple spreadsheet views of the same document, an optimization view and a graph view, which are synchronized with the document data, while the code for operations with the data resides outside of the objects responsible for visual components. The document takes on the task of updating all views whenever the data changes as described in sections 3.3-3.4.

At the heart of view are three key classes:

- **Cview** instantiates **CACcelView**, an object used to display a document's data and manage user interaction with the data. ACCEL's view aggregates a spreadsheet object, and implements message-handler member functions for user operations with the spreadsheet.
- **Cframewnd** instantiates an object that provides the frame around one or more views of a document. ACCEL's main frame stores several graphical controls as member variables, and implements message-handler member functions and a message map to specify what happens when messages are directed to the window.
- **CMultiDocTemplate**, an object that coordinates one or more existing documents created in ACCEL, and manages the creation of the correct document, view and frame window objects for that type (the so called multiple document interface (MDI)). An MDI application uses the main frame window as a workspace in which the user can open zero or more document frame windows, each of which displays a document.

The structure of the GUI is shown in Figure 24. Each class in this structure has a corresponding visual component that enables interaction with the user.

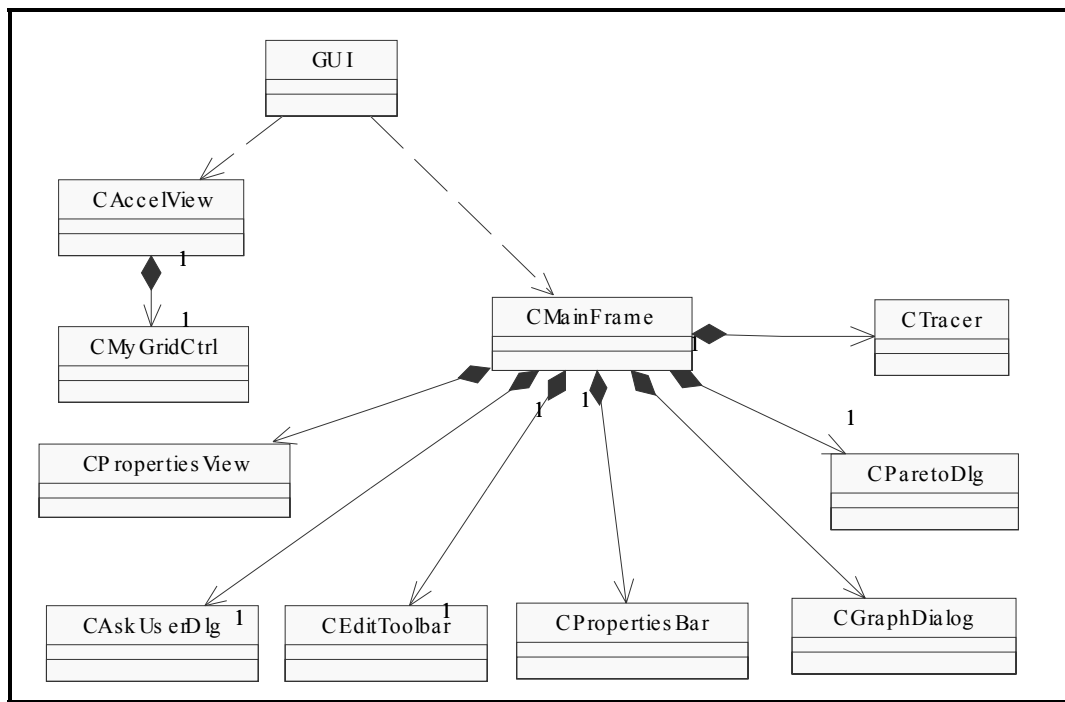


Figure 24. The structure of GUI

We will describe ACCEL's GUI classes in the following order:

- **CMyGridCtrl** is a class inherited from a publicly available class **CGridCtrl**²⁷. This class implements a spreadsheet-like user interface, which is discussed in section 3.6.1. It allows the user to add and remove concepts, attributes, values and functional dependencies.

²⁷ See www.codeguru.com for more details.

- **CPropertiesView** is inherited from the **CGridCtrl** class and provides control of properties of the functional objects. In particular, this class allows the user to differentiate between four categories of knowledge. This class is described in section 3.6.2.
- **CAskUserDlg** provides the visual interface for access to all <ASKUSER()> functions, as described in section 3.6.3.
- **CGraphDialog** enables the sensitivity analysis, as described in section 3.6.4.
- **CTracer** provides feedback to the user and is related to the error handling mechanism, as described in section 3.6.5.
- **CEditToolBar** provides enhanced functionality for the user input of values and functional dependencies, for instance enabling names of variables to be added into an expression by a mouse click on the variables. A further description of this class can be found in ACCEL's help file.
- **CPropertiesBar** is an auxiliary class of **CPropertiesView** that enables the floating functionality of visual controls.
- **CparetoDlg** provides control over the optimization problem. In our opinion this class implements the most interesting functionality of ACCEL, as is described in section 3.7.

3.6.1 Spreadsheet

Our initial intuition about the representation of product knowledge was that it should be in a form that the designers feel comfortable with. Another requirement was that the representation should resemble and be compatible with our formalism. We also took into account the time spent by the user to add a new concept, a new attribute, a new value or a new functional dependency; this time had to be minimized. Finally the interface has to be standard and therefore simpler to implement. The logical choice was therefore to use a spreadsheet representation.

Spreadsheet representation is a familiar device for incremental 'what-if' analysis and data analysis in many disciplines [Ronen, 1989] [Hicks, 1991]. Spreadsheets have an easy user interface, are sufficiently flexible, and allow us to represent even large models. The user interface of conventional spreadsheet systems (such as Microsoft Excel) provides convenient functionality for the user with regards to the editing of expressions, operations of copying and pasting, etc. However, standard spreadsheet systems are chiefly data-analysis tools concentrating on computation. Some fundamental functional requirements could therefore not easily be satisfied using the conventional spreadsheet systems, i.e. the support of knowledge spaces, concepts and attributes, multi-valued answers, postponing answers, using references, multi-objective optimization, partial evaluation of functional expressions. We have therefore implemented a spreadsheet user interface with an extended functionality²⁸. Our interface enhances the traditional weak aspects of the spreadsheets, i.e. insight into calculations and the structure of the model.

²⁸ The implemented spreadsheet is based on the publicly available spreadsheet "GridCtrl", available at www.codeguru.com.

The spreadsheet interface provides a straightforward and efficient link to the formalism of the model from chapter 2. It means that the optimization algorithm can be triggered without any extra steps from the user²⁹. In the spreadsheet environment, the state-transitions model of the design process becomes a sequence of updates of a table. The table contains zero or more rows; every row can represent one or more concepts; every column in the table represents one or more attributes. A cell is characterized by a row (concept c_i) and by a column (attribute a_j). Each cell corresponds to a variable $c_i.a_j$, which can be assigned a value v_{ij} or a functional dependency to obtain v_{ij} ³⁰. Thus, the cell is used as the container for a value or a functional expression, which explicitly represents a functional dependency³¹. If a functional expression can be evaluated, it also represents *the value* of this expression, in a similar way as occurs in a spreadsheet. ACCEL supports immediate evaluation, i.e. a change of any cell immediately results in an update of the whole product model.

Spreadsheet representation prescribes the following format for the state transitions: adding a concept (i.e. adding a row); adding an attribute (i.e. adding a column); adding or modifying the value or the functional dependency (i.e. modifying the content of a cell). Note that a variable $c_i.a_j$ is defined automatically if c_i and a_j are defined. This saves the user a tremendous amount of time, since the same attribute can be applied to several concepts. If we consider the case when five concepts are defined, the definition of an attribute would result in five new variables. This efficiency is paid for by the less efficient use of space; not all five variables will be required, in fact, the introduced attribute may not become part of signature for some of the concepts. Many cells will therefore stay empty. We partially solve this problem by allowing the user to introduce concepts and attributes outside the first column or the first row. Each concept and attribute is characterized by its coordinates in the spreadsheet. Additional functionality of the spreadsheet allows the user to drag concepts and attributes within the spreadsheet. Note that the operation of dragging causes the automatic replacement of the variables, i.e. cells. Dragging of a cell is not permitted, since this operation would cause the automatic deletion of the value. Instead, the user can cut and paste a value to the desired position.

In a similar way, the efficiency of the user can be improved if a knowledge space is expressed for the attributes rather than for every variable. The '*Category of an attribute*' allows the user to express a knowledge space for all variables related to this attribute. The disadvantage here is that the same attribute might be related to variables from all four knowledge spaces. Four attributes with an identical name are needed in this case. Since the names of the attributes are required to be unique, the user would need to invent several modifications of the name. We use the following notation for an attribute that occurs in several knowledge spaces: e.g. <a_area> means that the attribute is related to auxiliary knowledge space. <d_area> means that the attribute is related to decision knowledge space, etc. In the next section we will explain how the user can choose a category for the attributes.

We will demonstrate an application of the spreadsheet in an example, which gives us an idea of how knowledge expressed in terms of our model can be represented in a spreadsheet.

²⁹ Strictly speaking, one button still has to be pushed.

³⁰ This is the main distinction; in traditional spreadsheets a cell is an address in memory that can contain anything and has no meaning unless extra steps are taken.

³¹ In the traditional spreadsheets the calculation can be hidden in macros; this complicates the analyses.

Ex. 20 Consider the design of a box (new concept $\langle \text{box} \rangle$) from a given piece of wire (new concept $\langle \text{wire} \rangle$) of length 400m (new contextual attribute $\langle \text{length} \rangle$), i.e. the value $\langle \text{wire.length}=400 \rangle$. Let us suppose that we would like to use this box as a container and cover it with some expensive material. The box should therefore have the maximum possible volume (new attribute $\langle \text{volume} \rangle$) and the minimum possible area (new attribute $\langle \text{area} \rangle$), which are the two objective attributes. The design decisions we can make are the following: we can choose a different width for the container (new design attribute $\langle w \rangle$) and we can choose a different shape for the cross section (new design attribute $\langle \text{shape} \rangle$). Let us suppose that $\langle w \rangle$ has the range $\langle R_w=\{10;20;30;40;50\} \rangle$ and the attribute $\langle \text{shape} \rangle$ has the range $\langle R_{\text{shape}}=\{\text{square};\text{hexagon}\} \rangle$. The Cartesian product $R_w \times R_{\text{shape}}$ is therefore the decision space V_D . We can depict this example as shown in Figure 25.

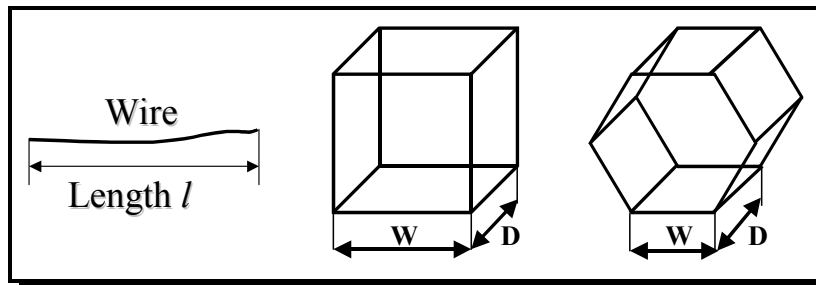


Figure 25. Two alternative shapes of the wire frame

In order to select optimal solutions from V_D we need to connect the values in V_D with the values in the objective space V_O , which is structured by the two objective attributes $\langle \text{area} \rangle$ and $\langle \text{volume} \rangle$. We will demonstrate how ACCEL enables modelling in small incremental steps.

The two alternative shapes of the box cross-section initially are basic values of the nominal type. These values are promoted to concepts; we have a new concept $\langle \text{square} \rangle$ and a new concept $\langle \text{hexagon} \rangle$. It is important that the names of these new concepts are identical to the names of the values derived from them. In this case $\langle \text{box.shape.area} \rangle$ can be evaluated to $\langle \text{square.area} \rangle$. For these concepts we introduce an auxiliary attribute $\langle a_area \rangle$, which differs from the objective attribute $\langle \text{area} \rangle$ in its meaning, and therefore needs to be identified by a different name. For the attribute $\langle a_area \rangle$ we do not wish to maximize or minimize the value. We will use it to express values from the auxiliary space V_A . Another new auxiliary attribute is the depth of the box $\langle d \rangle$. We also introduce a new contextual attribute $\langle \text{sides_nr} \rangle$, which accesses the number of sides for any concept. Using these simplifying steps, the value for the objective variable $\langle \text{box.area} \rangle$ can be obtained from the functional expression $\langle 2 * \text{box.shape.a_area} + \text{side.a_area} * \text{box.shape.sides_nr} \rangle$, and the value for the objective variable $\langle \text{box.volume} \rangle$ can be obtained from the functional expression $\langle \text{box.shape.a_area} * \text{box.d} \rangle$. The resulting representation is shown in Figure 26. The box example is provided as one of a list of examples that are built into ACCEL.

	∞ shape	∞ w	∞ a_area	∞ d	∞ area	∞ volume
C box	square	10	3400	80	3400	8000
C side			800			
	∞ sides_nr					
C square	4		100			
C hexagon	6		259.8			
	∞ length					
C wire	400					

Figure 26. The representation of the box example in ACCEL

3.6.2 Properties view

Each basic function has different properties. For instance, a concept has a row and a column; an attribute has a name and a category; a cell has a corresponding attribute and a concept etc. The control and visualization of these properties is important for the user and great care should therefore be taken with it. The design of the properties view should take three objectives into account, namely the transparency of the properties of the basic functions, uniform representation and extendibility.

A logical solution would be to use a spreadsheet to control the basic functions, since the user has already got used to the spreadsheet way of representing knowledge. From an implementation point of view, a spreadsheet is a flexible control because it can represent an arbitrary number of properties in a structured way. Every function can therefore be made responsible for the representation of its own properties. However, an interface between the basic functions and the properties spreadsheet does need to be established.

We therefore gave the class function a polymorphic method OnView(). When a basic function object (a cell, a concept, or an attribute) is accessed (for instance by a mouse click), the object executes its OnView() method. This method results in different behavior for each basic class, which allows the right set of properties to be displayed and controlled. Although this approach has a limitation, i.e. the user can only see one set of properties at a time, it is a flexible and simple approach. Figure 27 gives an example of a screenshot of some properties.

Concept		Attribute		Cell	
Concept	box	Attribute	shape	Cell	box.shape
Description	design a box from a pe...	Description	Shape of the crosssection	Full Expression	ALT(square;hexagon)
		Category	Design	Evaluated Expres...	square
		Type	Concept	Dependent Cells	{box.d;box.volume;box...
		Range	{square;hexagon}	Depends on Cells	{}
				Description	

Figure 27. Properties view for the concept, an attribute, and a cell

Most of the presented properties are fixed, i.e. they cannot be changed by the user. The user does have access to, and can express knowledge about, some of the properties, as defined in the model of chapter 2.

- D. 74 (Attributes' types) The type of the attribute allows the user to express the type of the attribute as it was defined in the model of chapter 2: type '*real*' corresponds to numerical values of Ordinal, Interval and Ration types; type '*bool*' corresponds to the Boolean type; type '*string*' corresponds to the Nominal type; type '*concept*' corresponds to the Nominal type of values, which are represented as concepts; type '*attribute*' corresponds to the Nominal type of values, which are represented as attributes.
- D. 75 The range of the attribute allows the user to limit the range of acceptable values. The ranges for attributes of '*real*' type are expressed according to the following syntax: <[min value; step; max value]>, where '*step*' corresponds to a numerical value that defines the step between the min and the max values for purposes of sensitivity analyses. The range for attributes of other types is expressed according to the following syntax: <{value₁;value₂;...}>.

Note that if a value of a variable violates the type of the corresponding attribute, a type error occurs; this is reported to the user as will be explained in section 3.6.5.

3.6.3 AskUser dialog

The AskUser dialog provides a visual interface for access to all <ASKUSER()> functions. At all necessary moments, i.e. moments when the <ASKUSER()> function encounters a combination of values which is not in its history, a new question is composed and the AskUser dialog is invoked to present a new question and to obtain an answer from the user (as described in section 3.5.2).

The main idea behind the design of the dialog is to help the user to express his or her early intuitions about a dependency, and eventually to arrive at a functional dependency, which is supposed to replace <ASKUSER()> functions by mathematically-specific dependencies. The use of the AskUser dialog might produce the following positive results:

- by answering many similar (but not identical) questions, the user recognizes the dependencies which underlie the answers given
- questions can be facilitated by group discussions
- newly-created questions can inspire new, creative ideas, because questions have been interpreted
- eventually, advanced data analysis procedures could be applied to derive a functional dependency from the already gathered answers and to suggest it to the user

The dialog design is based on the history of answers. The history of previous answers can be helpful when forming an opinion. We will demonstrate an example of the application of the AskUser dialog as a continuation of an example from section 3.5.2.

- Ex. 21 In the example a functional dependency for the objective value *tm.safety* is unknown. The value is therefore obtained using the <Askuser()> function <Askuser(*tm.energy_source*;*tm.media*)>. The AskUser dialog contains the combinations of values, which represent different questions for the user. The

arguments in this example are design decisions. New combinations of values will be generated during an optimization procedure; the necessary user input is therefore obtained either before or during the optimization.

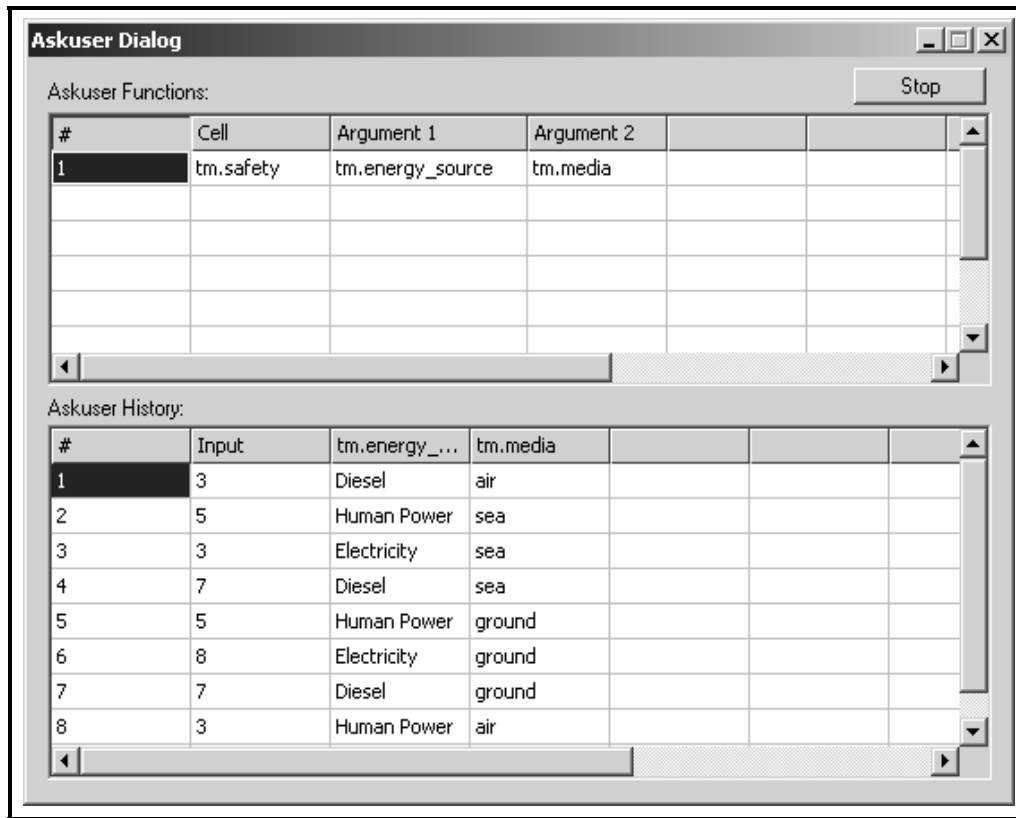


Figure 28. Application of the AskUser dialog for evaluation of the safety of transportation means

3.6.4 Sensitivity dialog

The development of the product model is a time consuming process. Time should therefore be invested in critical dependencies and values for the product. Visualization provides one of the quickest ways to analyze the complex chains of functional dependencies and to detect any of the following situations that are interesting for the designers:

- (strong sensitivity) a small change of an independent value causes a big change of an objective value
- (trade-offs) a change of a decision value causes one objective value to improve and another one to worsen
- (extreme detection) an objective or objectives has extreme value(s)
- (independency) variables are independent

A sensitivity analysis is complicated in traditional spreadsheets. They can visualize data but cannot visualize the dependencies directly³². ACCEL's sensitivity dialog enables the

³² They can visualize them indirectly; tuples of values need to be produced first.

visual analysis of functional dependencies between multiple cells. The open structure of the spreadsheet gives the user easy access to the product model and enables quick analysis. The main idea behind the design of the sensitivity dialog is to minimize the time needed to visualize dependencies. This enables the systematic analysis of the product model. The required steps are as follows:

- The user selects cells that need to be analyzed and opens the sensitivity dialog box.
- In the sensitivity dialog the user needs to select a cell, which should be treated as being independent. The other cells are automatically considered to be dependent.
- The user presses the ‘DRAW’ button.

The whole procedure, including the time for calculations, takes a few seconds. There are some restrictions on the type of ranges. An independent cell (variable) can be of any type (including Nominal). Dependent cells (variables) are required to have a numerical type. The range of the cells is determined by the ranges of attributes corresponding to them. The ranges of the attributes can be adjusted as described in section 3.6.2. Note that the values of the decision variables are likely to influence the analysis. The decision values can be operated using the optimization dialog.

Ex. 22 (Sensitivity analysis) Let us consider an example of a sensitivity analysis for the box example from Ex. 20. We will consider the variable <box.w> as being independent and two variables <box.area> and <box.volume> as being dependent. We will consider two graphs, where one graph corresponds to <box.shape>=<square> (left hand graph of Figure 29) and the other graph corresponds to <box.shape>=<hexagon> (right hand graph of Figure 29).

The functionality of the sensitivity dialog is provided by “Flipper Graph Control”, which was purchased from “ProWorks Corp”³³. This control provides additional functionality, e.g. scaling of graphs, adding properties and graphical editing such as font, color etc.

3.6.5 Trace View

The Trace View is a graphical component of ACCEL that provides feedback to the user. This feedback can be a consequence of syntactic errors, which were described in section 3.4.4. We will consider an example of user feedback in combination with a partial evaluation mechanism.

Ex. 23 Consider the box example as it was described in Ex. 20, in the case where the value of <wire.length> is not defined. This value is used in several other functional dependencies. The missing value therefore makes the product model incomplete and causes an error. The feedback about this error is shown in Figure 30. Feedback should be interpreted from the top to the bottom. This is because feedback is given in the order in which errors are detected. As a consequence of an error, expressions in the spreadsheet may only partially be evaluated. The situation in the spreadsheet that corresponds to the error is shown in Figure 31. If the user positions the mouse pointer above the cell of the expression in question in the spreadsheet, he or she will see a partially evaluated version of an expression. In this case, the functional dependency in the cell <box.d> reduces to <wire.length-80/4>. The original

³³ The web site of “ProWorks Corp” can be found at: <http://www.proworks.com/>

expression is shown at the top of Figure 31. Unevaluated cells are additionally marked in red and are tagged with the error symbol.

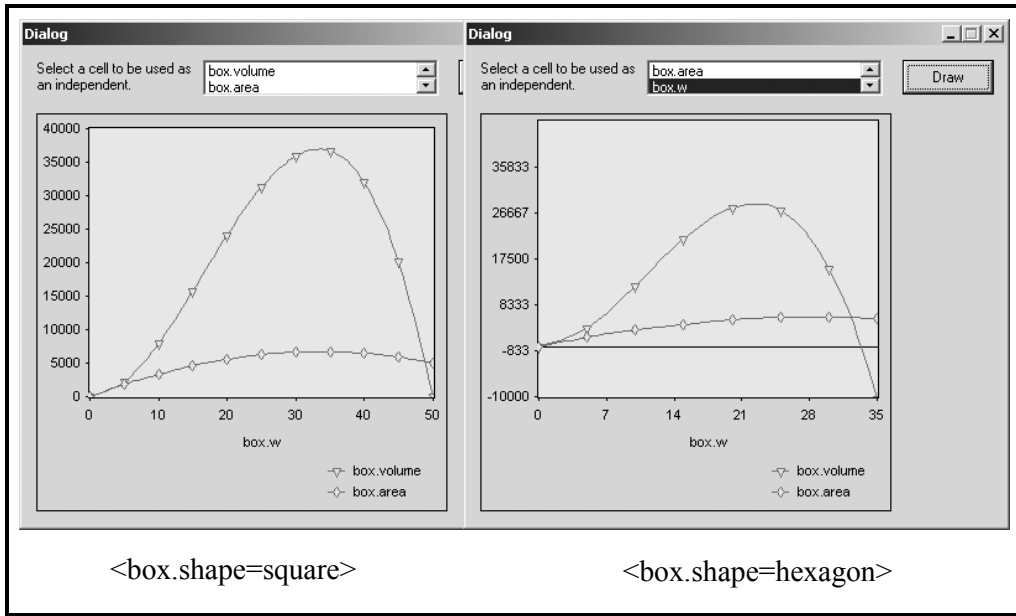


Figure 29. Sensitivity analysis for the box example

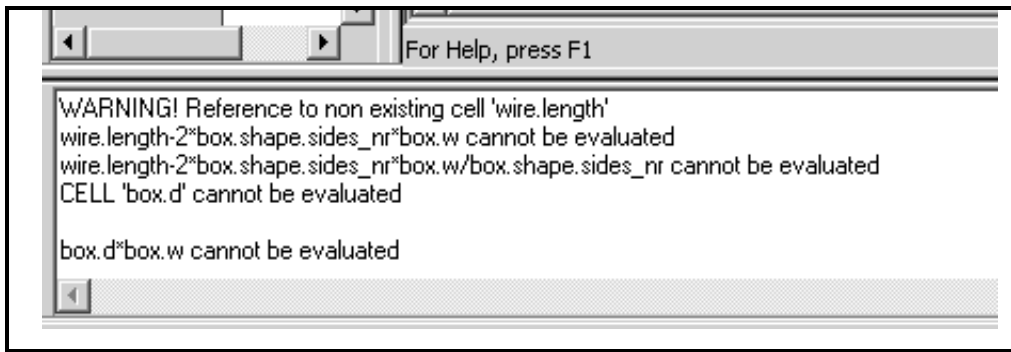


Figure 30. Example of feedback to the user

Figure 31 shows a table with columns for 'Attributes', 'Cell', 'shape', 'w', 'a_area', 'd', 'area', and 'volume'. The table contains data for 'square' and 'hexa...' shapes, and 'side' and 'wire' attributes. The 'wire' cell is highlighted in red, indicating it is unevaluated.

Attributes	Cell	shape	w	a_area	d	area	volume
c box	square	10		wire.length-80/4...	100*...		
c side			wire.l...				
	sides_nr						
c square	4		100				
c hexa...	6		259,8				
	length						
c wire							

Figure 31. Example of partial evaluation

3.7 Optimization with ACCEL

In this section we describe an optimization dialog, we introduce our method for decision making, and we give an example of optimization.

3.7.1 Optimization dialog

The optimization dialog enables the operation of the SPEA machinery, which we introduced in section 2.7. The design of a dialog for optimization has the following challenges:

- In the early design phase the formulation of the optimization problem is frequently changed; it should be easy to update and control the formulation.
- During the optimization procedure previously unnoticed errors may occur due to variation of the decision values; we need to decide how to give the feedback about errors.
- Optimization is time consuming; we need to decide how to inform the user about progress.
- The design should be easy to implement and extend in the future.

The above requirements resulted in a number of decisions:

- Base the optimization dialog on several spreadsheets. Arrange the spreadsheets in a way resembling the representation of the knowledge spaces, i.e. design decision in the top-left corner, objective on the top-right. The optimized solutions were positioned in the center and the control at the bottom (see a detailed description after this list).
- Use the optimization dialog for both manual decision-making and optimization.
- Display objectives and constraints in a uniform way, so they could be easily interchanged.
- In case of an error, freeze the product state so that the user can see which combination of decision values caused an error.

Top part

In the top part of the dialog box (see Figure 32) the user can see the inputs and the outputs of the optimization problem; on the left, the user can see design decisions. These are all occurrences of the function `<ALT()>` expressed by the user in ACCEL. Conversely, every occurrence of the function `<ALT()>` gives rise to the definition of attribute corresponding to a design category. For every design decision, the choice can be made manually from the provided alternatives plus the option “all”. This option means that the optimization procedure will vary the values of the design decision. On the right, the user can see the objective functions, i.e. the functional expressions that compute objective variables. For each objective function the user can choose the following options:

- The objective kind can be *‘minimize’*, *‘maximize’* or *‘free’*. The last option means that the user does not desire a minimum or a maximum value. The user can turn an objective into a constraint with a single mouse click using the kind.

- The left constraint and the right constraint need to be numerical values or functional expressions that evaluate to numerical values. These values define the range of acceptable outcomes. The optimization procedure will remove solutions with objective values from consideration.

Central part

The dialog will position all Pareto optimal solutions in the central part of the dialog box. Each solution is represented by the decision vector and the objective vector. A special procedure hides all irrelevant values from the user, so that the user only sees values of an optimal solution from the solution signature. Note that in Chapter 2 we explained how the notion of the extended signature is used during the optimization.

Bottom part

In the bottom part of the dialog box, the user can control one of the two optimization algorithms. One algorithm is referred to as ‘exhaustive’, because it allows the user to generate all possible solutions in the space V_D . Optionally, the user can choose to generate fewer solutions. In this case, an intelligent random generation takes place such that any two generated solution vectors are different. Indeed, simple random generation is likely to produce many identical solutions. Another algorithm is SPEA. We have described this algorithm in section 2.7. The optimization dialog is shown in section 3.7.3.

3.7.2 Tightening constraints method

In this section we present one of our results, which is the ‘tightening constraints method’. Our method eliminates problems of two classical decision-making methods, which we considered in section 2.7.2. The main advantages of our method are listed below:

- It avoids the process of prioritization of objectives. All personal preferences of the participants with respect to objectives can be expressed as left and right constraints on the objectives, i.e. l_f and r_f .
- No a priori domain knowledge is necessary to assign the constraints. The use of the method follows after a round of optimization. The constraints can therefore be approximated from the analysis of the available Pareto optimal solutions.

We will demonstrate our method in the following example. Some criticisms on the weighting method (see section 2.7.2) are: a) it is still broadly practiced in the technological settings, although people find it confusing to compare objectives that are incompatible with each other, or to assign weights to complex technical objectives b) assigning weights does not simplify the evaluation procedures, but makes the results less certain.

We support the above claims with a small literature survey:

- The coefficients need to be assessed very carefully to ensure that the results of the evaluation match with the preferences of the decision maker. A great deal of behavioral research had focused on the correct procedure to assess the coefficients [Webber, 1993]. Unfortunately, experimental studies have revealed

numerous sources of inconsistencies rather than a single, superior assessment technique [Schoemaker, 1982].

- Conducting a sensitivity analysis of the weights is often gives insights, but current techniques typically vary a single weight and observe the effect on the result of the model. Special techniques need to be developed in order to vary all, or at least a large subset, of the weights [Butler, 1996].

3.7.3 Optimization example

We now demonstrate the functionality of the optimization dialog in a follow up of the box example.

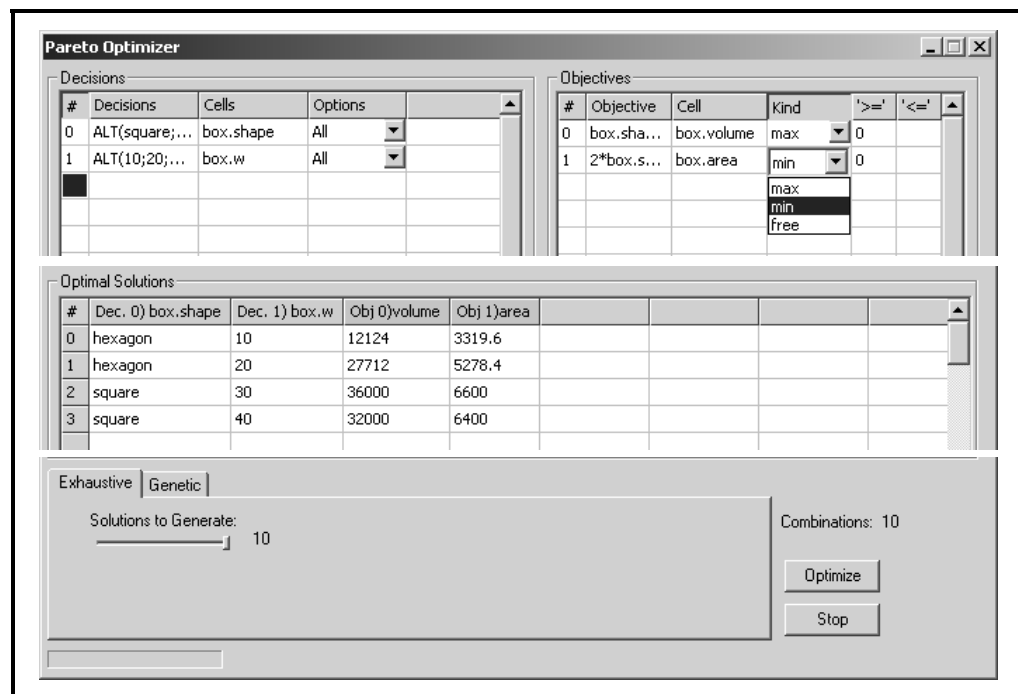


Figure 32. A representation of the box example in the optimization dialog

In this example, the decision space V_D contains 10 solutions. From these 10 solutions, four solutions were selected in accordance with the Pareto optimality principle, as is shown in Figure 33. For any Pareto optimal solution we can see that there is no other solution that has both a bigger volume and a smaller area. We will demonstrate the follow up decision-making process by applying our tightening constraints method. Consider the following constraints on the objective functions:

$$\text{box.area} < 6000[\text{m}^2], \text{box.volume} > 25000[\text{m}^3].$$

Given these constraints, the optimization algorithm produces a single solution, which corresponds to $(\text{hexagon}, 20)$, presented in line one in Figure 32.

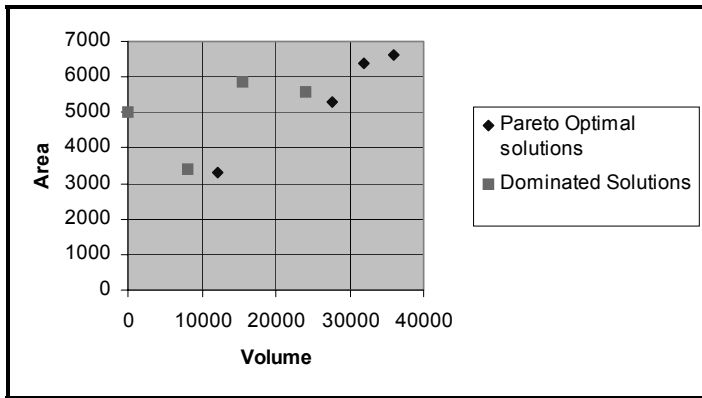


Figure 33. Pareto optimal solutions for the box example

CHAPTER 4. CASE STUDIES

4.1 Introduction

In the previous chapter we gave a limited demonstration of the abilities built into ACCEL. This chapter provides case studies, which allow us to describe the methodology of designing with ACCEL and to present the results of our evaluation of the tool. Section 4.2 describes the design of an air-conditioning unit for a swimming pool. This example aims to demonstrate how ACCEL can support a re-engineering project. In this section we will focus on the following aspects of ACCEL:

- how to extend the sequential decision making process with a number of simultaneously-considered design decisions and objectives
- how less-technical objectives can be modeled in ACCEL
- how to translate a design problem into a product model and produce optimized solutions
- how to provide a quick analysis of the results and gain more certainty

Section 4.3 describes how three groups of students approach the problem of a vertical transportation system for a new World Trade Center (WTC). By comparing the approaches we demonstrate how the groups differed in their approach to the formalization of the design problem and the generation of optimized solutions. The gathered feedback allowed us to make some preliminary conclusions. In section 4.4 we describe a series of early experiments. Throughout this chapter we present several Assessment Methods (AM) which we practiced in our experimental studies. In a similar way to Atman [2000], we use several well-known assessment methods to achieve more credibility for our results. These methods are presented in Table 12.

Table 12. Assessment Methods (AM) suitable for early validation experiments

Method	Advantages	Disadvantages	Outcome
1. Closed-ended surveys	Easy to administrate and analyze. Allows statistical analysis within the known space of options.	Narrows down the number of possible responses. Not efficient with a small number of participants.	Structured and concrete opinions about the proposed issues.
2. Open-ended questionnaire	Stimulates the appearance of new ideas. Can work with any number of participants. Gives more evidence of the experience	Difficult to draw conclusions, useful information can be hidden	Some new ideas but in an unstructured fashion.
3. Observations and interviews	Direct access to attitudes and interactions	Requires an extra model in order to interpret the observations. Time consuming.	Implicit information on how students experienced the method.

4.2 Air-conditioning system for a swimming pool

4.2.1 Introduction

This example demonstrates the application of ACCEL to the re-engineering of an air-conditioning system for a swimming pool, which is shown in Figure 34. This project was performed in collaboration with an expert from the Technical University of Prague [Lain, 2003]. The product we considered belongs to the HVAC domain (Heating Ventilation Air-Conditioning). We started with an existing design and the intention to evaluate ACCEL and compare the re-engineered solution with the original one. The topic of this section is therefore the confrontation between the conventional engineering approach and our approach. The ‘*re-engineering*’ means that we can go back to previously-made decisions and reconsider or extend them while taking into account new objectives that were not previously considered. It means that we reformulate the initial design problem and therefore we shift the original solutions back to the early design phase. Knowledge about the original solution and of the original design process allows us to explain some advanced methodological aspects related to designing with ACCEL.

Consider the design of an air-conditioning system for small and medium-sized swimming pools. From a technical point of view this design is of moderate complexity. In practice, a designer does not necessarily use advanced tools, even if they are available; a limited number of objectives are therefore considered, possibly only one. The system has to maintain a certain relative humidity inside the swimming pool at low costs. There are two standard approaches to reduce the relative humidity in swimming pools: by supplying more air with humidity lower than swimming pool air, or by increasing the dehumidification capacity of the dehumidification unit. The first approach is initially cheaper but increases the energy costs, because the outside air has to be preheated in the winter season. The second approach relies on an expensive dehumidification unit, whose costs are proportional to its dehumidification capacity.

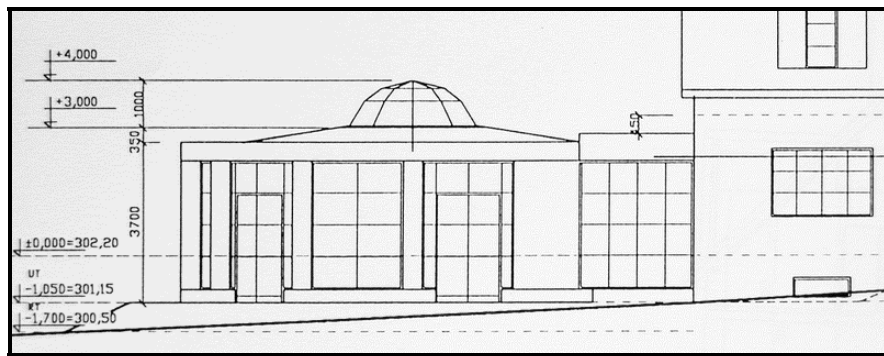


Figure 34. The swimming pool

This technical problem becomes a very complex problem if the designer is trying to consider multiple design decisions, multiple objectives and their relationships. The following points are needed to be taken into account:

- there are alternative or complementary ways to deal with the humidity, such as using a plastic sheet to cover the pool when it is not in use
- there is specific context in which a product will be installed and used

- the end product will be used for many years, so there are multiple aspects related to the usage of the product, such as costs due to energy consumption and costs of the maintenance
- there are less-technical objectives that are related to the end user, e.g. noise level, quality of the air, usability of the control.

Our aim was to use ACCEL to manage the increase in complexity and generate optimized solutions, which we would then compare with the original solution. We describe this case study in a number of steps according to the ACCEL design methodology, as presented in Appendix C

4.2.2 Design solutions

In this section, we present the decision variables of a re-engineered solution concept <New S>. We introduce several standard decision variables, which express decisions that we had to make, as shown in Table 13. The novel approach here is that we were aiming to optimize these decisions simultaneously. In a conventional approach, these decisions would be handled sequentially, for instance: <AHU.FR>, <FAN.AIR FLOW> and then <DU.DC>.

Using these decision variables, we were able to introduce the original solution <S>. The original solution was based on the first approach to remove the humidity, i.e. there was no dehumidification unit. The decision values of this solution are as follows: the air supply flow <FAN.AIR FLOW>= $\langle 315 \rangle$ [m³/h], the total flow rate <AHU.FR>= $\langle 600 \rangle$ [m³/h], the dehumidification capacity <DU.DC>= $\langle 0 \rangle$ and the air distribution system <ADS.TYPE>= <normal ADS>. Note that the decision variables do not belong to a single concept; we describe them briefly in the following paragraphs.

Table 13. Decision variables of the new solution concept

Concept	Description (see also below the table)	Attribute	Decision variable	Range
<AHU>	The air handling unit	Flow rate	< AHU.FR >	600-5000 m ³ /h
<DU>	The dehumidification unit	Dehumidification capacity	< DU.DC >	0-3.5 Kg/h
<ADS>	The air distribution system	Type of the air distribution system	<ADS.TYPE >	{poor ADS, normal ADS, good ADS}
<FAN>	The fan	The fresh air flow rate from outside	<FAN.AIR FLOW >	50-350 m ³ /h

The air-handling unit <AHU> supplies the air to the swimming pool via the air distribution system. The total amount of air supplied can vary between 600 and 5000 m³/h. The amount of the supplied airflow, due to fan <FAN>, may vary between 50 and 350 m³/h. The supplied air is preheated, but the maximum temperature of the supply should not exceed 50 °C.

The dehumidification unit <DU> can have a dehumidification capacity of between 0 and 3.5 Kg/h. Such a unit dehumidifies the air using a refrigerant-vapor-compression cycle. The air inside the swimming pool circulates via the dehumidification unit, which decreases the humidity ratio of the air and increases its temperature.

The air distribution system <ADS> consists of ducts, a noise damper and distribution elements (diffusers). When choosing the air distribution system we needed to consider numerous variations of pipes, ducts, etc. It was therefore not feasible to consider all possible configurations. We avoided this problem by introducing three classes of air distribution system. From experience, we were able to define general characteristics of each class. Since we did not know the sensitivity of the objectives with respect to the air distribution system, this was a valuable first approximation. We therefore considered the following three alternatives: poor ADS (short duct with few big grill diffusers); normal ADS (with a more extensive duct network and more advanced air distribution); and good ADS (with a very extensive duct system and special diffusers, e.g. linear slot diffusers, which are placed under the windows).

Initially we considered the variable for the power of the radiator <RADIATOR.POWER> to be a decision variable. We later moved this variable into the auxiliary category. The radiator only needed to provide additional heating if the joint heating output of <AHU> and <DU> did not compensate for the winter losses. We were therefore able to derive the necessary power for the radiator.

Ex. 24 (Methodology example) The transformation of a decision variable into an auxiliary variable is a typical design mistake. There may be multiple objectives that depend on the power of the radiator, i.e. maintenance costs, air quality, etc. By assigning the value to a decision variable using a functional dependency, the decision value cannot be optimized with respect to other objectives. ACCEL allows multiple decision variables to be optimized, so we did not need to reduce their numbers as engineers intuitively do. In this example, the optimization algorithm and the functional dependency produced the same optimal value, as only one objective depended on the power of the radiator.

4.2.3 Objectives

We extended the initial set of objectives to consider the prices of the system over five years and the quality of the air distribution:

- Objective functions
 - <Price Over 5 Years> [euro] to be minimized
 - <AD quality>, air distribution quality [dimensionless] to be minimized³⁴
- Constraint functions
 - <air per person> [$m^3/(h*people)$] has to exceed 50
 - <relative humidity> [%] should be between 40% and 65%

To evaluate these functions we needed to know the functional dependencies between the decision variables and the objective variables. These dependencies were only available

³⁴ We measured the air distribution quality on a scale of 1 to 3. Our expert intuitively assigned 1 to the best air distribution quality.

for physical causalities such as humidity, as described in section 4.2.5. We needed to develop functional dependencies for the prices, and the quality.

Prices

The prices and the technical parameters of the components can be supplied via a database or using equations. Databases have to be available electronically for the optimization; we had no such databases and therefore used approximated equations derived from our research of commercial catalogs. We assumed that the price of each unit depends linearly on the single most important technical parameter. In our case we considered decision values to be parameters. We assumed that the total price of the solution included the following ingredients: the price of the air-handling unit, the price of the dehumidification unit, the price of the radiators, and the price of the air distribution system.

The price of the air-handling unit is represented in Figure 35. This price includes the unit with a filter, a mixing box, a heater, a fan and the basic control system. Figure 35 clearly shows that the price of the air-handling unit is proportional to the total flow rate, but that the constants differ for the manufacturers 'A' and 'B'. This is a nice methodological example.

Ex. 25 Apparently systems produced by 'A' and 'B' have different values, i.e. service and quality. This inspired us to consider 1) a new decision variable, which would reflect the difference between 'A' and 'B', 2) a new objective variable, which would explain and quantify the reason for 1), and 3) new functional dependencies to relate 1) and 2). For the sake of simplicity, we chose the cheaper air-handling unit, i.e. manufacturer 'A'.

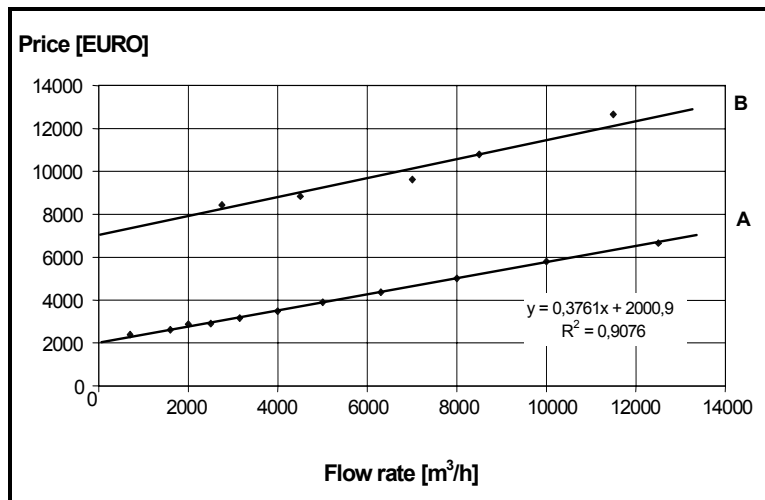


Figure 35. Linear interpolation of the air-handling unit's price

The price of the dehumidification unit is roughly proportional to the dehumidification capacity, as is shown Figure 36.

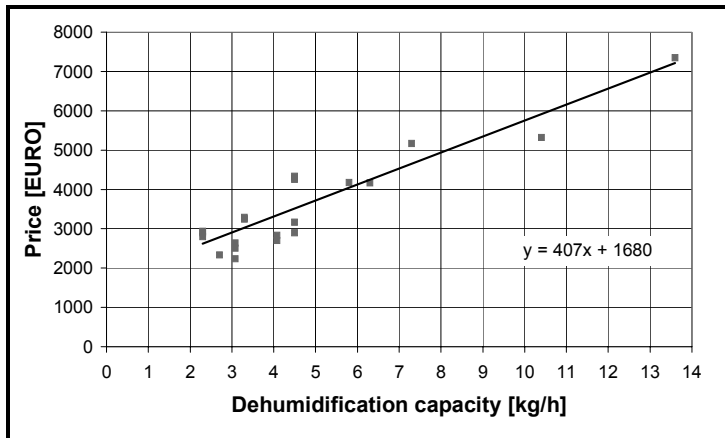


Figure 36. Linear interpolation of the price of the dehumidification unit

The heating capacity, electricity and airflow rate, and the input of the dehumidification unit were approximated as linear functions of the dehumidification capacity. The linear coefficient for the heat output is 1.136 kW per kg/h. The linear coefficient for electric power input is 0.41 kW per kg/h. The linear coefficient of the dehumidification airflow rate is 222 m³/h per kg/h. This coefficient is strongly dependent on the manufacturers of the unit, since there is no direct physical relationship between the dehumidification capacity and the unit flow rate. The actual airflow can therefore vary significantly.

- Ex. 26** (Decisions versus auxiliary variables) As a follow up to the previous example, we detected a new technical variable that explains the variation in price of the dehumidification units with the same capacity: the airflow rate of the dehumidification unit. This variable should be positioned in the decision space. For simplicity's sake we positioned it in the auxiliary space and obtained its value using a functional dependency.

Price over 5 years

To approach the next objective we needed to calculate the energy consumption costs over a five-year period. The five-year time span was arbitrarily chosen as a reference point, so that we could evaluate the costs of the solution over time. Later we performed a sensitivity analysis to see how the chosen period affected the set of optimal solutions.

A basic method to calculate the energy consumption in regions with cold winters is the so called Day-Degree method [Ashrae Fundamentals, 1989], [Martinaitis, 1998]. This method allows the 'day-degrees' to be calculated given the starting temperature of the heating season. The day-degree method is based on the dimensionless temperature distribution curve; Figure 37 shows such a curve for Prague. The day-degree is found by integrating the curve from t_{\min} to the system starting temperature. In the graph T_{\min} corresponds to 1 on the dimensionless temperature axis. Strictly speaking, the graph curve should therefore be inverted. We assumed that the same amount of vapor was produced all year long. The dehumidification unit was therefore operated during the entire year, resulting in the continuous production of an amount of heat due to the dehumidification effect. The additional heat production shortens the heating season since the air-handling unit and the radiator need to start supplying heat at a lower outside temperature. This meant that we needed to modify the day-degree method to incorporate the heat that was produced by the dehumidification unit.

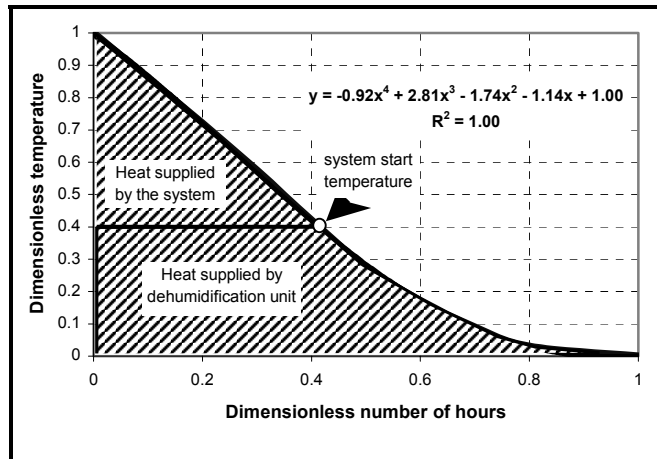


Figure 37. The dimensionless temperature distribution curve for Prague

We provided an additional flexibility in our model by introducing two coefficients, namely the time reduction coefficient, which is equal to 0.7, and the power reduction coefficient, which is equal to 0.9. These coefficients allowed us to take into account the fact that the solution may not be operated at full power for the whole year. We computed the additional energy consumption due to the electricity consumption of the dehumidification unit. The resulting energy consumption was converted to the usage costs for the five-year period based on prices of natural gas, the efficiency of the heating system and the electricity costs. The total price over the five-year period was then calculated as the sum of the price of all units of the system and the energy consumption over five years.

Air distribution quality

The air distribution quality represents our less-technical objective. To be able to argue about the quality of the air distribution system, we split this objective into four criteria that, in our opinion, constitute this objective. We defined that each criterion could vary between one and three, where one represents excellent quality, two represents standard quality and three represents poor quality. The resulting air distribution quality was calculated as the average value of these four criteria. We assumed that an expert could judge the quality criteria based on the information about the quality of the air distribution system and the air exchange rate in the pool. This is an example of interpretational dependency. The resulting limits that were used to translate the qualities into numbers represent personal preferences.

Ex. 27 (Methodological example) Note that we combined four objective functions (criteria) into one objective function. In general this is an incorrect approach; we should have considered all four of them separately. In this case objective values did not have to be normalized and could be expressed in physical units (instead of 1,2,3).

The first criterion addressed the quality due to the condensation of vapor on the windows inside the swimming pool, as depicted in Figure 38. The second quality criterion described the quality due to the air currents in the pool, as shown in Figure 38. It seems that a poor system achieves low condensation on the windows only with a very high window flow rate, whereas a good system achieves excellent quality with a lower window flow rate.

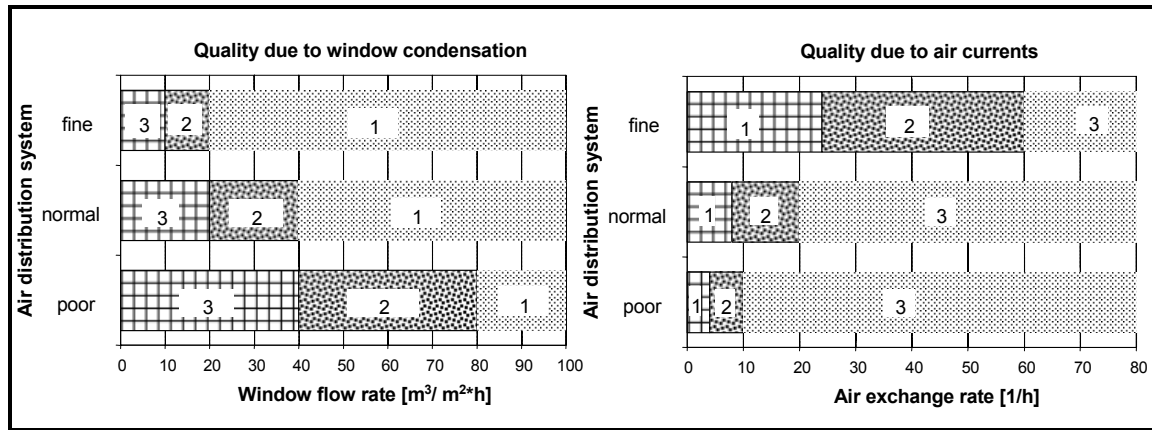


Figure 38. Quality of the air distribution

The third quality criterion described the quality of the exchange rate. The fourth criterion describes the amount of air supply per person. The resulting air distribution quality was therefore computed as follows: $\langle \text{air.quality} = \text{air.condensation on the windows} + \text{air.draft risk} + \text{air.Q Ex rate} + \text{air.Q Air pers} \rangle / 4$.

4.2.4 Contextual information

We represented observations about the context of the swimming pool by introducing several concepts and attributes of the contextual type such as a new concept $\langle \text{pool} \rangle$, a new attribute water area $\langle S \rangle$, a new attribute $\langle \text{winter losses} \rangle$, a new attribute $\langle \text{summer gains} \rangle$, a new attribute $\langle \text{win area} \rangle$ (i.e. the area of the windows) and a new attribute $\langle \text{people} \rangle$ (i.e. the number of people). Winter losses and summer gains were typically calculated as having a maximum value and were assumed to be contextual. The climatic data was provided according to the extreme weather conditions in the Czech Republic. We described important information about the climate by introducing a new concept $\langle \text{climate} \rangle$ and a number of contextual attributes such as the temperature inside $\langle t_{in} \rangle$, the maximum relative humidity $\langle fi \rangle$, the water temperature $\langle \text{water } t \rangle$ and so on.

4.2.5 Auxiliary (intermediate) information

According to the ACCEL design methodology, after a number of design and objective variables have been introduced we need to relate these values using functional dependencies. We therefore introduced several variables in the auxiliary knowledge space, whose values were obtained using functional dependencies. We expressed the humidity production (evaporation) inside the swimming pool as a new attribute $\langle Mw \rangle$. The functional dependency to compute $\langle \text{POOL.Mw} \rangle$ was taken from the German standard VDI:

$$Mw = \beta \cdot (p_{aw}'' - p_d) \cdot A$$

Mw - humidity production [g/s]

β - mass transfer coefficient [g/Pa s m²], which depends on the type of activities in the pool. For swimming pools that are used just to swim in, this coefficient is lower than for "all day use" swimming pools

- p_{dw} - saturation partial pressure of water vapor at the water temperature
- p_d - partial pressure of water vapor for the internal air
- A - water area [m²]

A minimum flow rate <flow rate winter> of outside air, needed to keep the inside relative humidity under than the maximum acceptable value, was calculated for the winter and for the summer according to the humidity production of the pool and the difference between the outside and the inside humidity. Note that the actual flow rate of the outside air will be less due to the dehumidification effect, which reduces the amount of air supply needed.

4.2.6 Representation in ACCEL and optimization

We used ACCEL continuously from the first day of the project until the last. The resulting representation in ACCEL is shown in Figure 39. The product model was completed by connecting variables from the objective space with the variables in decision space and contextual space so that there were no non-evaluated cells. We then used a Pareto optimization dialog to generate the optimized solutions. We managed to add some additional constraints in this dialog. All the design decisions were left for variation by the optimization algorithms.

	5	winte...	summ...	Win Area	People	kg_h	V	pdi	pd*w	Mw	x inside	flow rate ...	pd'i	flow r...	
pool	36	7	13	72	2	4.76896	329.8	2759.01	3781.16	1.32471	17.6479	238.716	4244.63	597.8	
t in	fi	water t	t out su...	t out ...	x out sum...	x out w...	hours...	t start	curve						
climate	30	0.65	28	32	-15	11	1	7604.56	25	0.8681					
b	air de...	air ca...	time re...	powe...	max dt	air pres...	Lat heat	req ex	Air pers						
const	3.6e-005	1.2	1.01	0.7	0.9	20	100000	2500	2	70					
FR	inside...	dt	air heater	power	price	Q	t dim	h dim	dist int	t heat	tadim	tav	en co...	fan g	
unit	600	47.1493	20	7.07023	6.12095	4177.66	10.0302	0.324675	0.528586	0.112655	12.013	0.537801	3.48797	5.82922	0.2222
dc	Type	airflow	heat out												
DU	2		444	2.27273	0.927644	2493.24									
AD5		ADpoor				822									
Rad				0.687273		50.6182									
fan	50	400	0.6	0.7575											
Air	1044	3.16555	14.5	2	3	1	3	2.25	25						
Coef1				3	1.5	0.4	2								
A1	B1	comm...	flow1	IN				Heat ...	Powe...						
U_AIR	0.3761	3952													
DH	406.74	1679.76		222	0			2.2	0.408163						
Radiator	30														
ADpoor	1.37		Poor air ...			10	40								
ADnorm	2.46		Normal ...			20	20								
ADfine	3.55		Exelent ...			60	10								
El	Gas	Eff	Heat												
Energy	0.1	0.3	0.85	34											

Figure 39. Air-conditioning problem represented in ACCEL

An example of an additional constraint is the relative inside humidity, which has to be between 40% and 65%. For this constraint we were not interested in maximizing or minimizing the values; it therefore had the type “free”. In a similar way we expressed the fact that the amount of air supply per person had to exceed 50 [m³/h].

The total number of possible solutions in the decision space was 1176. This number of solutions was easily manageable by an exhaustive optimization algorithm. It took 45sec

on a pc with a 600MHz processor to generate and evaluate the decision space. The algorithm selected four optimized solutions, as shown in Table 14.

Table 14. Pareto front optimal solutions

	Decisions				Objectives			
Solu- tion	Supply air flow rate	Air handling unit flow rate	Dehumid- ification capacity	Air distr. system	Inside humid.	Air supply per person	Air distr. quality	Price over 5 years
1	150	600	2.5	ADpoor	50	75	1.75	14 455
2	150	600	2.5	ADfine	50	75	1.5	15 763
3	150	900	2.5	ADfine	50	75	1.25	17 203
4	150	1500	2.5	ADfine	50	75	1	20 150

Remember that the decision vector of the original solution was $\langle S^D \rangle = (315, 600, 0, ADnorm)$. The corresponding objective vector was $\langle S^O \rangle = (50, 157, 2, 18355)$. By comparison, we can see that the original solution is dominated by three of the four produced solutions (since $\langle \text{Air supply per person} \rangle$ is a constraint and not an objective).

We needed to select a single solution for the product from the optimized solutions. We will demonstrate our method of tightening of the constraints, which was introduced in section 3.7.2. By analyzing the solutions, we defined that the price after 5 years should not exceed 16,000 Euro and that the air distribution quality should be less than 1.5. This meant that only the second solution was left. This resulted in a cost saving of more than 2500 Euro over a five-year period.

4.2.7 Analysis of the model

The model can be further validated using some quick analysis methods that are supported by ACCEL.

Range analysis

The quickest way to analyze product models is to consider the chosen decision values. Remember that each decision value is varied within the range which is specified by the function $\langle ALT() \rangle$. For decision values of a nominal and higher order (those that can be compared with each other), the following consideration is applicable: the chosen decision values should not be the minimum or the maximum values from the range. If this is the case, the range should be extended unless there are reasons for the range to be as it is. In our case all the decision values for all optimized solutions were within the ranges.

Analytical sensitivity analysis

We define analytical sensitivity analysis to be the sensitivity of the resulting set of optimized solutions on contextual or decision values, i.e. how much the resulting set of

optimized solutions changes when a contextual value changes. We will give two examples.

Ex. 28 (Number of reference years) For instance, how would the set of optimized solutions change if we considered costs after one or ten years instead of five years? Given the fact that it is easy to generate optimized solutions we will check this for both one and ten years. The result for a ten-year time span is not surprising. The set number of optimized solutions is the same but the optimal dehumidification capacity of the solutions changes from 2.5 to 3 Kg/h with respect to the five-year time span. This is easy to explain, since the higher dehumidification capacity saves more energy and has paid for itself after ten years. The set of optimized solutions for the one-year time span is more surprising. The solutions are divided into two categories: solutions that rely on the dehumidification effect and solutions that rely on the air supply (see Table 15). We can explain this result if we consider that after one year the approach of dehumidifying with the air supply is still reasonable because of the low investment costs. We can see that this is no longer the case in the set of optimal solutions for five years.

Ex. 29 (Starting temperature of the heating season) Consider the dependency of the result on the starting temperature of the heating season. This temperature is known for residential and office building but not for swimming pools, since it is very sensitive to the type of building. We assume that the temperature can vary from 15 °C for swimming pools with many windows and large summer gains, up to 30 °C for underground swimming pools. The sensitivity analyses for the starting temperature show that the decision values of optimal solutions stays nearly the same. For instance, the optimal dehumidification capacity increases up to 3 kg/h if the starting temperature is 15 °C. The total price after five years grows, mainly due to the increased energy consumption costs.

Table 15. Pareto front optimal solutions for one-year time span

Solution	Decisions				Objectives			
	Supply air flow rate	Air handling unit flow rate	Dehydration capacity	Air distr. system	Inside humid.	Air supply per person	Air distr. quality	Price in 5 years
1	150	900	2.5	ADfine	50.4	75	1.25	11586.3
2	150	1500	2.5	ADfine	50.4	75	1	14060.3
3	300	900	0	ADpoor	52.7	150	1.75	8111.23
4	300	900	0	ADfine	52.7	150	1.5	10073.2

Graphical sensitivity analysis

ACCEL allows the designer to quickly visualize dependencies between cells, as was explained in section 3.6.4.

Ex. 30 (Number of years) For instance, consider the dependency of the solution price on the dehumidification capacity after five years. Two examples of these dependencies for different numbers of years are shown in Figure 40. From these graphs we can see that the dehumidification capacity has an optimal value after five years. After one year there is no such optimal value.

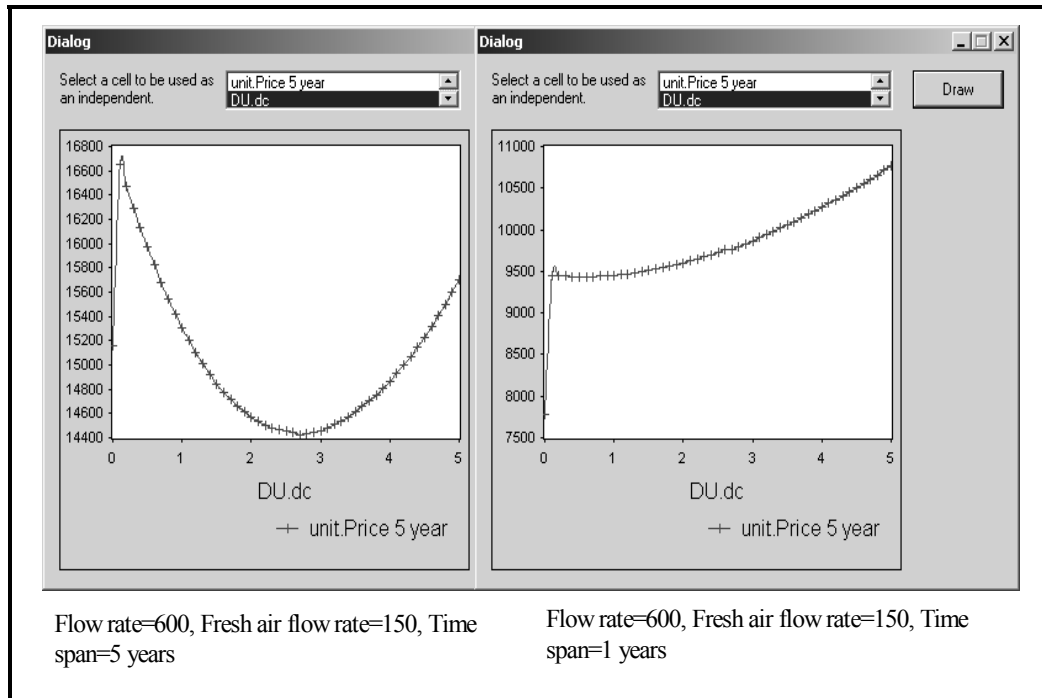


Figure 40. Sensitivity graphs created in ACCEL

4.2.8 Conclusions

We now present some conclusions based on our own opinions, the observation of the results and the opinion of our expert.

Our conclusions

- We achieved the goal of this case study, i.e. we developed a new solution that we were able to compare with the original solution.
- It turned out that the new solution was more economical than the original one. The cost difference over five years compared to the original solution would be about 2,500 Euro; for a ten-year time span, the difference would be about 8,000 Euro.
- By considering multiple objectives, such as the costs after five years, we were able to design a solution that would be much cheaper. However, the initial costs of the new solution would be higher than the costs of the original one.
- This exercise has estimated the potential costs saving and has hopefully provided the motivation for others to develop models using multiple decisions and objectives.
- The problem of re-engineering of well-known engineering systems appear to be similar to design problems. Broaden considerations require unconventional approaches to development of functional dependencies. It appears to be a time consuming process that might require research efforts, communication and unconventional decision-making. Therefore ACCEL appears to be useful in re-engineering types of design projects.

- Our re-engineering case study was carried out in a domain that is well supported by computer tools. ACCEL cannot compete with these tools in many respects. The question is whether the standard tools can be easily broadened to consider new objectives and decisions simultaneously.
- Our expert appreciated the main features of ACCEL i.e. multi-objective optimization, knowledge representation and sensitivity analysis. He was able to learn ACCEL easily and work independently with the tool. From the feedback that our expert gave, we can conclude that ACCEL was taken seriously and was considered to be an acceptable tool.

Conclusions of our expert

After the project, we asked our expert to give some feedback about his experience with the tool. The results are summarized below, with minor editing of the original text.

Our expert was initially skeptical about the tool, but he found that the tool can be very useful in the design process. The definition of the problem and the data inputs are pretty easy but more copy-paste or automatic generation possibilities would be useful. The inbuilt optimization and analysis tools are implemented well. You only need a limited amount of information to be able work with ACCEL. During our work we had no problems with unexpected program errors. The main problem in using this tool more extensively in the HVAC system design process is the absence of predefined databases of solutions. All the solutions have to be defined in the tool. In a standard HVAC system design it takes less time to define a problem. Some functional dependencies were not available and could not easily be derived, especially the dependencies used to evaluate air quality and energy consumption. During this case study more than half of the time was spent preparing functional dependencies for the price of equipment versus the technical parameters. The tool would be easier to apply in research and education.

4.3 Transportation system for WTC

4.3.1 Introduction

This section describes an application of ACCEL in education of the post-graduate designers. The presented here case study is mainly intended for didactic purposes, as we demonstrate how several groups of student designers made use of ACCEL. We will consider more innovative designs than in the previous example; therefore the meaningfulness of the developed models is a secondary issue. The correctness and detailed analysis of the developed models will be left out of the discussion.

Several of the post-graduate design programs at the Stan Ackermans Institute include the course “Methods and Techniques of Design” in the curriculum. In this course, dr Kees van Overveld presents design methods and techniques in a systematic manner. This course and the methodology used were specially developed for a technological environment. Part of the course is a design project. During the project, the students are divided into teams, which allow them to apply their newly learned methods in a collaborative setting. The students use ACCEL during the conceptual design phase, when the selected conceptual solutions needed to be optimized. Before using the tool

the students were given a short (one or two hours) introduction to the basic principles and functionality of the tool.

In the described here case study the students were asked to design a vertical transportation system for the World Trade Centre in New York. This problem was chosen for this course for the following reasons:

- It is multidisciplinary and requires a team approach.
- The students are partially familiar with the problem, and can grasp it after a short introduction.
- Even without sophisticated domain-dependent knowledge, good solutions can be distinguished from bad ones.
- The problem's decision space is huge. This gives the students the freedom to apply various design methods. Design problems that are dedicated to a specific discipline would usually require a great deal of additional effort from the students in order to avoid standard, textbook-like solutions.
- Although a thorough mathematical analysis of the capacity of elevators is quite complicated, simple models can give reasonable outcomes.
- The problem contains all the characteristics from the early design phase: identification of the stakeholders, formulation of assumptions, idea generation, generation of solution concepts, stating objectives, optimization of the solution concept(s) and selection of the best solution(s).

The teams were allowed to make their own assumptions and re-define the design problem. We will present the results of the three groups as follows:

- Some of the manually-generated solutions are described in section 4.3.2.
- The choice of objectives is addressed in section 4.3.3.
- The optimization of the chosen solutions using ACCEL is presented in section 4.3.4.
- Our conclusions are presented in section 4.3.5.

Our aim is to present the results as close to the original form as possible; some additional editing of the results has been applied, however.

4.3.2 Manual generation of solutions

At the start of the project, each group generated a number of ideas using design methods for the generation and structuring of ideas, such as brainstorming and some additional techniques [Overveld, 2003]. The ideas were further evaluated and the selected ideas were combined into solution concepts. Each group produced a number of concepts, which are shown in Table 16. The concepts were evaluated further and a single conceptual solution was selected for the conceptual design phase. The first two groups made the selection using weighting techniques. The third group made use of ACCEL. The selected concept for any group is underlined and is listed in the first position of each column in the table.

Table 16. Concept solutions of the groups

Group 1	Group 2	Group 3
<p>Concept 1. Elevator with ranges and intervals</p> <p>Concept 2. Auto Garage around the building</p> <p>Concept 3. Escalators</p>	<p>Concept 1. Elevators, stairs and supplementary - helicopters, catapults and parachutes</p> <p>Concept 2. External moving wheels, stairs, supplementary-balloons, rockets</p> <p>Concept 3. Carriages, moving surfaces, supplementary stairs</p>	<p>Concept 1. Elevator</p> <p>Concept 2. Escalator</p> <p>Concept 3. Train: railroad track around building in a spiral</p>

From these results we can see that all the groups chose an elevator-like solution concept, although they did consider conceptually different solutions as well. The chosen concepts are not identical, which will become clearer in the following pages where we present the selected concepts.

Group 1

This group decided to proceed with a concept which they called “range-interval elevators”. The idea behind this concept was to use two kinds of elevators. The range elevator moves within its limited range as follows: start at a middle floor, ascend to an upper floor, descend to a lower floor, and ascend to the middle floor again. During this path the elevator can stop at designated by the users floors, as shown in Figure 41. Several range elevators would operate in a long shaft.

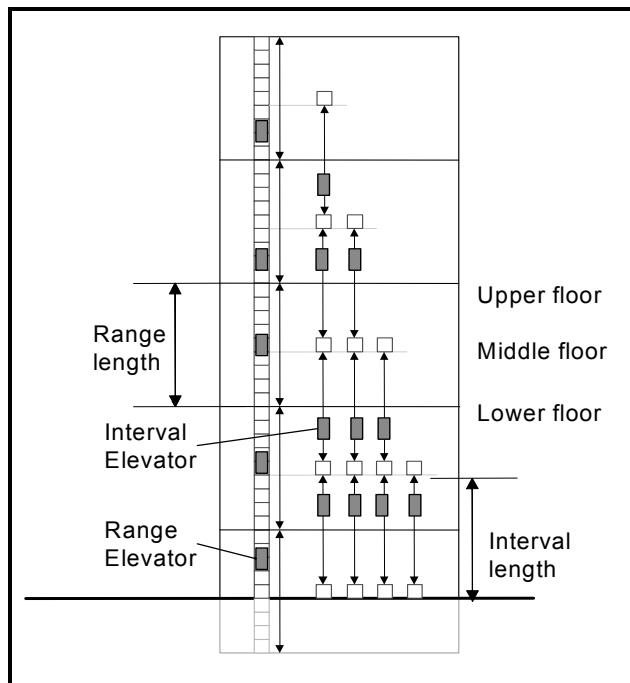


Figure 41. Concept for the product of Group 1

The interval elevators move within the intervals, which are defined by the middle floors of the range type of the elevators. The interval elevator stops only at the ends of the interval. This allows to transport the users quickly to a certain floor. At each interval there can be a different number of the interval elevators, which move in synchronization

with their vertical and horizontal neighbors. This means that all parallel interval elevators would make the same movement, i.e. they would depart and arrive together. When the set of interval elevators depart from the top of their interval, the interval elevator in the interval above would arrive at the lowest floor of its interval. Since intervals have an overlap of one floor, the change to the next interval would therefore not cause any delay.

There are three different types of elevator propulsion: traction, magneto-propulsion and hydraulics. These types determine the volume occupied by the elevators, the maintenance costs, the energy consumption, the construction costs and the speed of the elevators. The group assumed that the range elevators would use the same propulsion method, which could differ from the propulsion system of the interval elevators. In addition to the choice of the propulsion system, several other solution elements that required decisions were also identified. The results for this group are shown in Table 17.

Table 17. Decisions values considered by the first group

Decision variable	Description	Expression
Building.#ranges	Number of ranges per building	ALT(2;3;4;5;6;7;8;9;10;15;20;25;30;45;50)
Elevator.capacity	Capacity of an elevator	ALT(10;15;20)
Range.propulsion	Type of propulsion system for range elevators	ALT(Traction;Water;Magneto)
Interval.propulsion	Type of propulsion system for interval elevators	ALT(Traction;Water;Magneto)

Group 2

Group 2 took a broad approach to the problem by including the following elements in the solution concept: stairs, standard elevators and auxiliary transportation means such as helicopters. The resulting design decisions are shown in Table 18.

Table 18. Decision values considered by the second group

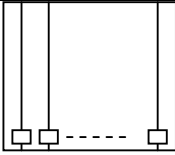
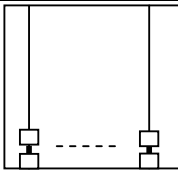
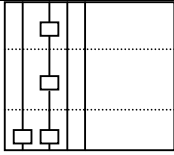
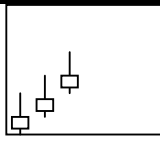
Decision variable	Description	Expression
VT.stairs	Position of the stairs	ALT(internal stairs; external stairs)
VT.elevators	Type of elevators	ALT(traction elevator; hydraulic elevator)
Traction elevator.capacity	Capacity in number of people	ALT(30;20;10)
hydraulic elevator.capacity	Capacity in number of people	ALT(30;20;10)
Number of units	Number of units (stairs, elevators)	Different sets
VT. Coeff4	Percentage of people evacuated by the stairs	ALT(0.1;0.3;0.5;0.7;0.9)

Group 3

This group approached the choice of the solution concept in two steps. In the first step the group considered three conceptually-different alternative solutions, namely an elevator, escalator and train. The group built a model in ACCEL in order to select an

optimized solution. The concept of an elevator was selected. In the second step the group therefore considered six alternative elevators as shown in Table 19.

Table 19. Six different concepts of the elevator system

 <p>The standard elevator (ST elevator) All elevators are bottom-top type.</p>	 <p>Double-Decker elevator (DD elevator). Two elevators are used in one shaft.</p>	 <p>Sky-lobbies. Extra sky lobby elevators, which stop on the ground floor, the 40th floor and the 70th floor.</p>
<p>Even/Odd floors elevator systems (EOF_elevator). Half of standard elevators are used for even floors, and half are used for odd floors.</p>	<p>Traffic controlled elevators (TD_Elevator). This concept is based on control of the traffic. More elevators are allocated for “busy” floors.</p>	 <p>Stair-shaped elevator system (SS_Elevator). Elevators are structured in the form of a staircase.</p>

The resulting design decisions that represent the solution concept are shown in Table 20.

Table 20. Decision values of the elevator concept solution

Decision value	Description	Expression
# of elevators of each type	Each elevator type has a different capacity for people, therefore various alternative numbers of elevators are considered	Different numerical ranges, e.g. ALT(20;25)
Solution.elevator type	Type of the elevator	ALT(ST_Elevator;DD_Elevator;Skylobbies;EOF_Elevator;TD_Elevator;SS_Elevator)

4.3.3 Objective values

Each group had to extend the initial formulation of the problem and define the stakeholders and the objectives. The results of this analysis are shown in Table 21 and Table 22.

Table 21. Stakeholders identified by each group

Group 1	Group 2 assumptions	Group 3 assumptions
Companies that rent space Employees who work in the building Investors who finance the building	Employees Residents Tourists Owners Municipal authorities Specialists (builders)	Owners: The owners of the tower Future clients: Future tenants, users Staff: Employees Visitors: Tourists, citizens, businessmen, constructors Municipalities: The authorities, and fire departments etc.

Table 22. Objective attributes considered by the groups

Group 1	Group 2	Group 3
Travel time [s], Rentable space [m ³], Energy costs [\$/year], Maintenance costs [\$/year], Realization costs [\$.	Latency (max waiting time for elevators, sec) Energy (average energy consumption, MJ) Aoc (area occupied by construction, m ²) EvacTime (evacuation time, sec)	Total Cost [\$] Waiting time [s] Evacuation Time [s] Comparative Construction Time [dimensionless]

Contextual values

The designers were stimulated to explore the context of the product and hence to add to the minimal information that was initially given about the context. The designers had to identify important contextual information and make their own assumptions or ask questions. The resulting assumptions made by each group are shown in Table 23.

Table 23. Different contextual values and attributes considered by the groups

Group 1	Group 2	Group 3
The people are equally distributed over the height of the building People are considered to travel from floor 0 (ground level) to floor x , where x can be any floor. The elevator sequence then becomes the following: a person first takes the interval elevator to the desired section of the building, then that person takes the range elevator to get to the precise floor. Floor.height=5 Building. arrival rate=1 person/sec	Number of floors: 100 Height of the building: 450 m Area that one person needs (5 sq meters) Area of a floor 14000 m ²	Building.Height= 400m Building. Number of Floors=100 Building.number of people per floor=1000

4.3.4 Analysis of the results

We briefly analyzed the models in order to see whether the designers were able to use ACCEL and the built-in optimization facilities correctly. The resulting models and the solutions of the groups are presented in Appendix D. From the analysis of these results we came to the following conclusions:

- The groups made proper use of the auxiliary attributes to decompose complex expressions.

- The groups managed to translate the design problems into an optimization problem and run the optimization facilities provided by ACCEL.
- The resulting decision space had significant differences compared to each other in the number of solutions and types of solutions.
- Our analysis of the developed models showed that the ranges of the decision variables in some models were not adjusted properly (i.e. the optimal solution took the min or max values from the ranges).

For the first two groups the number of solutions was relatively small and the exhaustive algorithm could be used to generate all possible solutions. For the third group the large set of decision values and alternatives led to a huge decision space, which could not be managed by the exhaustive algorithm. We checked the results using the genetic algorithm with the following settings: Generations=50; Population size=50; Mutation rate=10%. The tests were done on a computer with the following properties: Memory - 130MB, System - Windows 2000 and Processor - Intel 662MHz. The test results of this optimization are shown in Table 24.

Table 24. Results of optimization

	Decision space	Optimization Type	Time (min:sec)	Pareto solutions
Group 1	1536	Exhaustive	0:10	3
Group 2	4320	Exhaustive	1:03	10
Group 3	1181000	Genetic	0:08	1

4.3.5 Conclusions

All groups delivered a report, a presentation and personal feedback. In particular, they were asked to give feedback on ACCEL. This information was used to carry out further analysis, and led us to draw the following conclusions:

- The negative comments were constructive and suggested 1) improving the user interface 2) improving the introduction to ACCEL and support of the complementary examples and 3) extending the functionality of ACCEL in order to deal with continuous ranges and vectors.
- A number of bugs (program errors) were discovered in ACCEL.
- None of the negative comments suggested that ACCEL was useless or that the designers were incapable of achieving the results using the tool. The feedback suggested that ACCEL was adequate for generating concepts.

4.4 Early experiments

The formalism of the model was developed early on in this PhD project, but we initially had no software support for it. Our experiments revealed that a) the formalism helps designers to work systematically and think more clearly b) a simplified version of the

formalism can be practiced without software and c) software support is essential for the efficient use of the formalism. In the early experiments we took a re-engineering approach because many design processes are forms of *re-engineering*, where the first stage consists of making an inventory of properties of an *existing* artifact. In such cases the postponing of conditional decisions and multiple-valued decisions are not yet required, and a simplified form of the formalism can already be implemented using a standard database and/or spreadsheet tools. We presented results of some of the early experiments in Ivashkov [2002].

The early assessment of the design model allowed the validation of underlying ideas even without the formal underpinning of the model and without the dedicated software support being fully available. The use of design assignments based on re-engineering provides practical help in the choice of assignments and the validation of results. It also allows us to assess the consistency and usefulness of the model independently of each other.

4.4.1 The second year

First experiment

In the first experiment two groups of two students from the IPPS program (Intelligent Product and Production Systems, one of the SAI programs mentioned earlier) had to re-engineer a coffee maker and a desk-lamp. The goal of the course was to concentrate on one of the issues: environmental aspects, lifecycle, energy, or material consumption. The students used the Quality Function Deployment (QFD) techniques to work with the requirements, and the Life Cycle Assessment (LCA) method to deal with the complex interaction between a product and the environment.

Second experiment

In the second experiment, five groups of between two and three students from the USI program (User System Interaction) had to design various communication systems, including a communication system for grandparents and grandchildren and an Electronic Interactive System for Supporting Brainstorm Sessions.

In both experiments the students were asked to describe an existing artifact (in the second experiment it was a system that they designed themselves) using the terms of the model, thereby representing the knowledge in the formalism of our operational model (the model). Both the descriptions, provided in a spreadsheet format, and the students' opinions about the model were attached to their final reports.

Third experiment

In the third experiment ten students from different programs (USI, IPPS, Mathematics for Industry, Process and Product design, Software Technology) volunteered to participate in an interdisciplinary workshop organized at SAI. The system to be re-engineered was supposed to be well-known, interesting and challenging enough for the students. We chose a lighting system for a bicycle based on a mechanical dynamo. The purpose of our experiment was not to design a realistic and competitive substitute for

the existing solution, but rather to monitor the process of idea generation and reasoning in terms of concepts, attributes and values.

The review of the descriptions given by the students showed that the students were capable of conveying an architectural description of their designs in terms of the formalism. We gathered more evidence that the functional view of attributes invited students to think about alternative values, which produced meaningful results they might otherwise have overlooked. The descriptions were also useful for reviewers, helping them to understand the made decisions behind the designed systems. The use of the model to represent the design decisions invited the students to include alternative designs that might have been skipped in conventional, less-structured reports.

Several times during the workshop, the mechanism of attributes gave rise to quite unorthodox solutions that nevertheless needed to be taken seriously. For instance, the students did not know that the value “discrete” for the attribute “dynamics” had already led to the US patent 5,804,927, “Light emitting apparatus for a bicycle” in September 1998. The authors proposed to emit light using impulses instead of a conventional continuous emission, which enables energy saving. A value “piezoelectric” for attribute “Energy Generator” (EG) is the main idea in US patent 5,624,175, “Bicycle safety light” from April 1997. The power source includes piezo-electric elements that generate electricity when struck by a moving weight.

The usability and the usefulness of the model were assessed using the assessment methods (AM) 1 to 3. From the collected data we could see that the students did not have problems in understanding and application of the model to the design process. In the first experiment we used more closed questions, which allowed us to assess the students’ opinions about the model from specific viewpoints. In addition to questions directly related to the model, we also assessed the students’ opinions about the model as a tool for decision-making, knowledge management, idea generation and communication. The broadness of the initial survey helped us to spot the most useful points of the model which appeared to be (1) the stimulation of creativity and (2) the facilitation of communication within the group. In the second and the third experiments, we used the AM2 more extensively to assess the extent of (1) and (2). Again it turned out that (1) and (2) were recognized by the majority of the students, while some students commented on the difficulties of managing the knowledge due to the somewhat formal appearance of the information format.

To summarize, our experience indicated that:

- The students’ creativity was stimulated by formulating knowledge in terms of concepts, attributes and values, according to the gathered feedbacks.
- Most of the students appreciated the administration of the concept, attribute, and value according to the model after if they had get used to the more formal way of formulating and expressing their thoughts. The time of adaptation varies in dependence on the students background and skills. For post-graduate students we estimate the necessary time is about two to three days.
- Above all, the model seemed to be appreciated as a tool to improve the clarity of communication among the members of a design team.

4.4.2 The third year

The first software prototype with a Graphical User Interface appeared during the third year of this PhD project, but it was still too early to give it to the designers. The use of the model therefore still relied on standard spreadsheet software and standard methods of decision making.

In one of the experiments that we conducted in this year, students from one of the design programs at SAI had to design a system for the supply and distribution of coffee during a large pop concert. Students were asked to deliver the following results:

- Develop three different concepts for a coffee distribution problem.
- Describe the main assumptions, decisions and their rationale in terms of concepts, attributes and values. Students were also asked to explicitly distinguish the dependent variables from independent variables.
- Provide quantitative and qualitative modeling using the techniques that have been studied.
- Compare the advantages and disadvantages of the concepts and choose the best one.

Some results of the groups are shown in Appendix E. On the basis of the students' reports we came to the following conclusions:

- Students experienced difficulties in managing the complexity of models using the standard spreadsheet software.
- Students appreciated and used sensitivity analysis to support decisions about improving the precision of their product models.
- Students experienced problems when using the weighting method. Once the objectives were assigned weights and were normalized, their application to complex solution concepts became difficult.
- Students found it interesting and unusual to describe product knowledge in terms of concepts, attributes and values. They expressed the need to acquire additional design techniques in order to become fluent with the formalism.
- In the developed models we observed that students had difficulties in deciding which variables were dependent and which were independent. While practicing the model, some variables move from one category to another. In this way, our model made the students aware of knowledge categories and helped them to define the product model.

CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Results

In this thesis we presented a tool to support generating concepts. Our tool aims to help designers in this early design phase, which has traditionally been poorly supported by computational tools. In the presentation of the tool we considered the following chain of transformations:

Design problem →

The product model →

Mathematical programming →
problem

Genetic algorithms →

Optimal solutions

In the following list we highlight the major results that made our support tool possible.

1. The four knowledge spaces, i.e. the decision space V_D , the objective space V_O , the contextual space V_C , and the auxiliary space V_A . We showed that these spaces represent orthogonal dimensions of the four ways of operating product knowledge.
2. A formalism that facilitates the structured and systematic translation of the designer's implicit knowledge into the product model.
3. A formalized mechanism of questions and answers for reflections; this mechanism aims to support the designers during the reflections on the design activities. We derived eight categories of questions and eight corresponding formats in which to give the answers.
4. Interpretational dependencies which allow less-technical objectives to be taken into account.
5. The application of an existing genetic algorithm, which was implemented and applied to the early design phase.

6. A complete set of functional requirements for a tool aiming to support generating of concepts in the early design phase.
7. A graphical spreadsheet interface that supports multi-value answers, the partial evaluation of knowledge, and the use of a multi-objective optimization procedure without extra steps.
8. A method to tighten constraints, which enables convergence to few optimal solutions without the prioritization of the objectives.
9. The evaluation of our support tool on several occasions.

5.2 Compliance with initial objectives

Our conclusions about the compliance to the initial set of objectives are derived from the feedback gathered from student designers, publications and reviews.

- **Systematic.** We introduced a state-transition model and formalized it. We explained how transitions can result from a) reflections and a question and answer mechanism and b) operations performed by the computer.
- **Multidisciplinary.** The basic ingredients of our formalism are not attributed to a single engineering discipline. We made no further assumptions about the domain of the product or type of the design process. The developed support was evaluated in multidisciplinary teams and applied to the design of different products.
- **Acceptable.** From the conducted experiments we conclude that the developed support tool is acceptable. This opinion is based on students' opinions and the results obtained using our tool. During this PhD project several publications were positively reviewed by experts in the field.
- **Relevant.** The developed support tool allowed the students to experience the complexity of realistic design problems and be better prepared for working in industry.
- **Teachable.** The developed support tool has been used in design courses for the last few years. From the feedback gathered, we conclude that prospective designers appreciate the systematic approaches, and are willing to learn them. We have developed a design methodology that enables the effective teaching of our approach.

5.3 Recommendations for design courses

From the gathered feedback, we conclude that the operability of the tool does not limit creativity; in fact, it facilitates thinking about multiple design decisions, multiple objectives, exploration of the context, and making more considered decisions using product models, analysis and optimization. The students are able to practice the formalism after a short introduction. Group discussions benefit from a more formal way of individual thinking. Using this formalism, creativity techniques can be introduced into the technological context more easily. The conducted experiments and courses allow us to conclude that the developed approach can be used to effectively teach and

support systematic approaches to the early design phase. We can therefore recommend our tool for design courses. More tests and some improvements are necessary for application of the tool in industry. Below we present some recommendations for that.

5.4 Recommendations for software development

- Allow the matrix form of expression. The matrix form is an alternative way of representing structural product models. From the gathered feedback we conclude that the support of the matrix form of expressions is required in order to deal with more complex product models.
- Support of continuous ranges. In the current version of the support tool the user is restricted to discrete ranges of the variables. We recommend that continuous ranges also be supported.
- The support of the ‘is-a’ and ‘has-a’ relations needs to be build in.
- The “Undo” function needs to be build in.
- Assist the analysis of the developed product models. We have described several forms of analysis, which we recommend to build in.
- Improve the user interface. The graphical user interface had relatively low priority. Our experiments revealed a high demand for improved user-friendliness of the interface.
- Facilitate project management. The designers expressed the need to have basic functionality of project management tools.
- Help the user in the detection of conflicts. We recommend building in an advanced analysis feature such as the detection of conflicts between objective functions. Such an analysis can be done by analyzing the generated set of optimal solutions. The techniques for resolution of conflicts can then be linked.
- Pay more attention to resolution of conflicts. We believe that there are possibilities of benefiting from existing design knowledge, which is multidisciplinary and formal. We therefore recommend the enabling of a link with TRIZ theory [Altshuller, 1990] for the systematic resolution of the detected conflicts.
- Support interface to other data formats. A flexible interface with the existing detailed design tools can improve productivity and enable the reuse of the available product models.

5.5 Epilog

Although a software prototype was developed and applied in teaching and practice, we still have little recorded evidence of usability of our approach. Our experiments make the future research and development only less undirected. More rudimentary testing needs to be done. This can be done only through practice. A demonstrate of this point requires some action on the part of the reader.

Ex. 31 We ask you to turn your head and focus on an object on your left hand side, and as quickly as possible try to turn your head and focus on an object on the right hand side. These actions might require a few iterations before you notice that focusing is not easy and takes time. The same procedure can be done with the aid of a pointer, e.g. a finger. Point to the same object on the left hand side so that you see the object in the direction of the finger. Turn your finger and the head synchronously to the same object on the right hand side. Does it help you to focus?

The presented approach can be seen as just such a pointer. It can be applied to any design problem. After some time our pointer helps us to model the view and manage the complexity of this view. A little later it suggests the best solutions from the model of the view. However, in the end, it is completely dependent on the eyes and the ideas.

Appendix A. SPEA algorithm

(Strength Pareto Evolutionary Algorithm)

Input: P_t (population at the moment t)
 \underline{P}_t (archive population at the moment t)
 N (population size)
 \underline{N} (maximum size of archive set)
 T (maximum number of generations)
 pc (crossover probability)
 pm (mutation rate)

Output: A (non-dominated set)

Step 1: **Initialization:** Generate an initial population P_0 and create the empty archive set $\underline{P}=\emptyset$,
 Set $t=0$.

Step 2: **Update of external set:** Set the temporary archive set $\underline{P}' = \underline{P}_t$.

- a) Copy non-dominated individuals regarding P_t to \underline{P}'
- b) Remove dominated individuals from \underline{P}'
- c) Reduce the number of individuals in \underline{P}' if the size $> \underline{N}$ by applying an elitism strategy, and assign the resulting reduced set to \underline{P}_{t+1} .

Step 3: **Fitness assignment:** Calculate fitness values of individuals in \underline{P}_t and P_t

Step 4: **Selection:** Set $P' = \emptyset$, For $i=1 \dots N$ do

- a) Select two individuals $i, j \in \underline{P}_t \cup P_t$ at random.
- b) If $F(i) < F(j)$ then $P' = P' + \{i\}$ else $P' = P' + \{j\}$ Note that fitness is to be minimized here.

Step 5: **Recombination:** $\dots P''$, see 2.7.3

Step 6: **Mutation:** $\dots P'''$, see 2.7.3

Step 7: **Termination:** Set $\underline{P}_{t+1} = P'''$ and $t=t+1$. If $t \geq T$ or another stopping criterion is satisfied then set A to non-dominated individual in \underline{P}_t else go to Step 2.

The main loop of the algorithm is outlined in Figure 42.

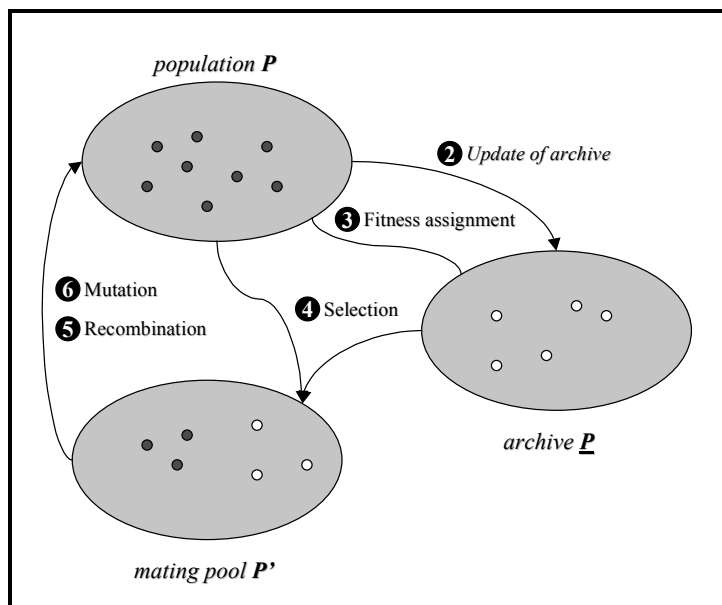


Figure 42. Basic steps in the strength Pareto evolutionary algorithm

Appendix B. Syntax of ACCEL

Operators

Operators specify the type of calculation that can be performed on the elements of an expression. ACCEL includes four different types of calculation operators: arithmetic, comparison, Boolean, and reference.

Arithmetic operators

To perform basic mathematical operations such as addition, subtraction or multiplication, to combine numbers and to produce numeric results, ACCEL supports the following arithmetic operators.

Arithmetic operator	Meaning	Example	Result
+ (plus sign)	Addition	3+3	6
- (minus sign)	Subtraction Negation	3-1 -1	2 -1
* (asterisk)	Multiplication	3*3	9
/ (forward slash)	Division	3/3	1
^ (caret)	Power	3^2	9

Comparison operators

To compare the two values with the following operators. When two values are compared using these operators, the result is a logical value, i.e. either TRUE or FALSE.

Comparison operator	Meaning	Example	Result
= (equal sign)	Equal to	Product.Price=40	TRUE or FALSE
> (greater than sign)	Greater than	ProductA.Price > ProductB.Price TRUE or FALSE	TRUE or FALSE
< (less than sign)	Less than	ProductA.Price < ProductB.Price	TRUE or FALSE

Boolean operators

Boolean operators enable logical operations with Boolean expressions.

Text operator	Meaning	Example	Result
& (ampersand)	Logical AND	TRUE&FALSE	FALSE
	Logical OR	FALSE TRUE	TRUE
!	Logical NOT	!TRUE	FALSE

Reference operators

Reference operators enable us to refer to values v_{ij} in expressions.

Reference operator	Meaning	Example
.	Reference to a cell	Product.Price
this	Reference to the corresponding concept or the corresponding attribute of a cell	This.price Car.this

Table 25. Functions supported by ACCEL

Name	Description	Example
ACOS(numerical expression)	Returns the arccosine of a numerical expression. The arccosine is the angle whose cosine is number. The returned angle is given in radians in the range 0 (zero) to pi.	ACOS(-0.5) equals 2.094395 (2*pi/3 radians)
ALT(f1,f2,...)	Contains a set of decision alternatives from which a single alternative has to be chosen. The choice can be made in the optimization dialog.	ALT(1;2;3;4;5) ALT(red;green;blue) ALT(car1.price;car2.price;car3.price)
ASIN(numerical expression)	Returns the arcsine of a number. The arcsine is the angle whose sine is number. The returned angle is given in radians in the range -pi/2 to pi/2.	ASIN(-0.5) equals -0.5236 (-pi/6 radians)
ASKUSER(f1, f2,...)	Asks the user a question about the combination of f1,f2,... and saves the input in the history.	See example in section 3.6.3.
ATAN(numerical expression)	Returns the arctangent of an expression. The arctangent is the angle whose tangent is number. The returned angle is given in radians in the range -pi/2 to pi/2.	ATAN(1) equals 0.785398 (pi/4 radians)
COS(numerical expression)	Returns the cosine of the given angle.	COS(1.047) equals 0.500171
EXP(numerical expression)	Returns e raised to the power of number. The constant e equals 2.71828182845904, the base of the natural logarithm.	EXP(2) equals e ² , or 7.389056
IFE(logical_test,value_if_true,value_if_false)	Returns one value if a condition you specify evaluates to TRUE and another value if it evaluates to FALSE.	IFE(cap.price<5000;in the budget;IFE(cap.price<6000;a bit over the budget; much over the budget))
LN(numerical expression)	Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).	LN(2.7182818) equals 1 LN(EXP(3)) equals 3
LOG(numerical expression)	Returns the base-10 logarithm of an expression	LOG(86) equals 1.934498451
SIN(numerical expression)	Returns the sine of the given angle.	SIN(3.14/2) equals 1
SQRT(numerical expression)	Returns a positive square root of an expression.	SQRT(16) equals 4 SQRT(cap.price)=2 (in case cap.price evaluates to 4)
TAN(numerical expression)	Returns the tangent of the given angle.	TAN(0.785) equals 0.99920 TAN(45*3.14/180) equals 1

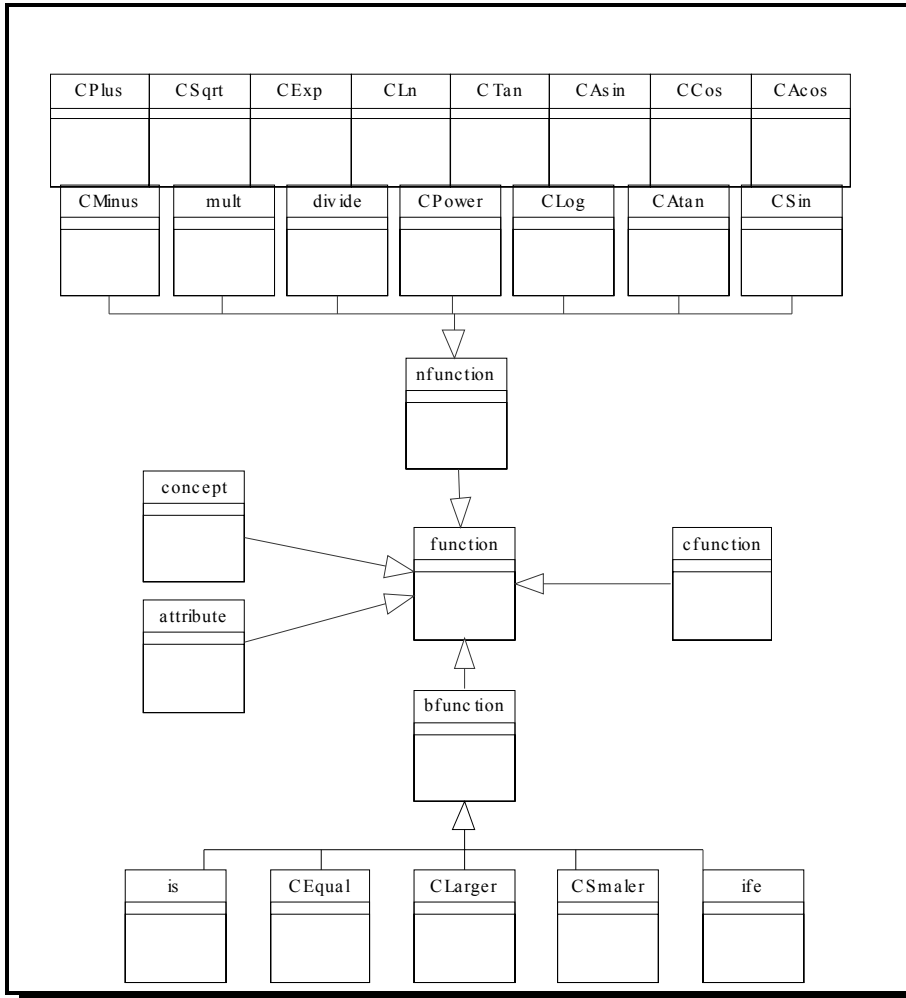


Figure 43. The hierarchy of function classes in ACCEL

Appendix C. Designing with ACCEL

Our suggested methodology of designing with ACCEL consists of seven phases; these phases make up the phases of a single cycle of the design process. Our methodology prescribes that the phases should follow in the given order, but that some phases can be skipped at each cycle of the process. The phases are presented in Table 26.

Table 26. The phases of ACCEL's design methodology

Phases	Description	Input	Output	Supporting techniques
1. Briefing	Initial analysis of the problem. Definition of the design team (chair, secretary, type of designers and advisors). Definition of the design process (time planning, choice of a procedure, etc)	Designers, Stakeholders	Process domain, Organization domain, V_O, V_C	PPO model
2. Brainstorming	Generation of the initial set of alternative solutions and structuring it in the form of a hierarchy	Designers	V_D	Intuition, Imagination, Brainstorming, TRIZ
3. Problem definition	Definition of the stakeholders, objective variables, constraints	Designers, stakeholders	V_O, V_C	ACCEL
4. Observation	Structuring V_D by identifying decision values for available solutions in S . Definition of decision variables	S	V_D	Attribute-seeking technique, ACCEL
5. Generation	Generate new alternatives using structures in V_D	V_D	S	Hierarchical structures, Manual generation, ACCEL
6. Modeling	Relating objective variables with decision variables and contextual variables by introducing auxiliary variables	Existing models, Intuition	V_A	Expertise, design knowledge, ACCEL
7. Optimization	Use the defined objectives and constraints to evaluate solutions and remove dominated solutions from S	V_D	V_O	ACCEL, Decision-making methods
8. Analysis	Analysis of optimal solutions and the product model	V_D, V_A, V_O, V_C	V_D, V_A, V_O, V_C	Sensitivity analysis in ACCEL

We represent the dynamics of the product knowledge domain that correspond to the defined phases as follows:

- During **phases 1-5** concepts, attributes, variables and their values appear in the decision space, the contextual space and in the objective space. The result of this process is shown in Figure 44.
- During **phase 6** the available variables are related by means of functional dependencies possibly via new variables in the auxiliary space. The result of this process is shown in Figure 45.
- During **phase 7** the values of decision variables are varied; the solutions resulting from the variation are evaluated and selected.

- During **phase 8** dependencies and values are analyzed. See examples of range analysis, analytical sensitivity analysis and graphical sensitivity analysis in section 4.2.7.

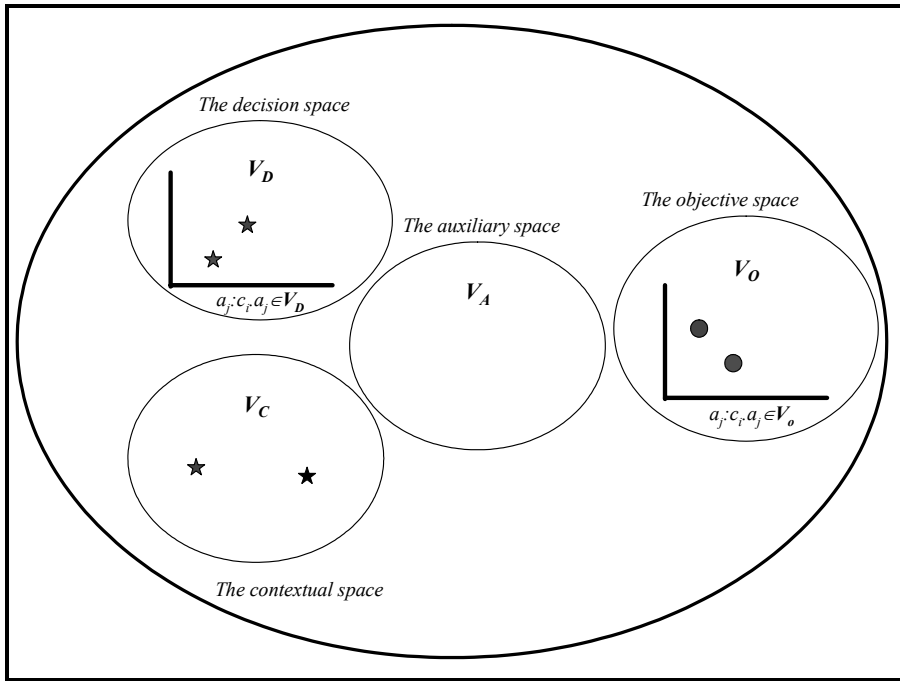


Figure 44. Phases 1-5 of ACCEL's design methodology

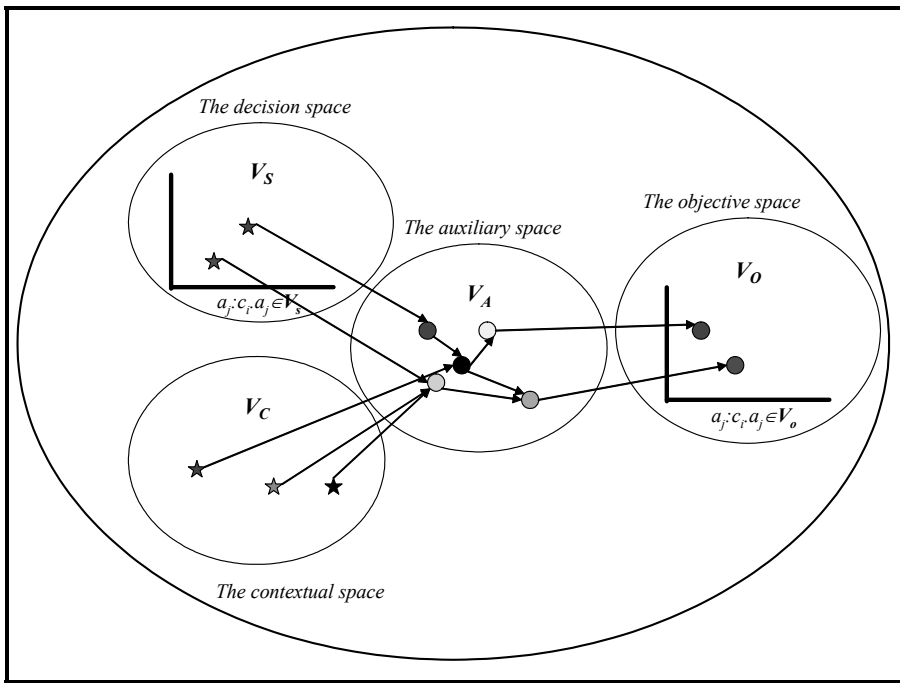


Figure 45. Phase 6 of ACCEL's design methodology

Appendix D. Results of WTC Project

This appendix contains a representation of the vertical transportation product in ACCEL and the generated optimized solutions that were delivered by the groups in the WTC assignment.

Group 1.

	#Lift_R	Arriv...	#floors...	Volum...	Volum...	Total...	Total...	#I_U...	#R_U...	Buili...	Rent...	Total...	Total...	Reals...	Trave...
c Building	2	1	6	1844.18	3775.61	10.6818	4.14344	8.01818	10.2044	30	1.49438...	284.269	198.988	28426.9	14.8253
	∞ Capacity														
c Lift	10														
	∞ Propulsion ∞ Hopti... ∞ StopProb ∞ WaitT... ∞ Avg... ∞ Trave... ∞ tmp ∞ waitin... ∞ Waiti... ∞ Hopti...														
c Range	Traction		0.4	0.8	0	0.227273	0	14.1727	1.81743						
c Interval	Traction	10		0.8		0.454545		11.1364	2.32601	0					
	∞ Height														
c Floor	5														
	∞ Volume ∞ Energy ∞ Build_c...														
c Traction	50	7	1000												
c Water	250	3	3000												
c Magn...	150	10	5000												

Figure 46. The transportation product represented by Group 1

Propulsion Range Lift	Propulsion Interval Lift	Length Range Lift	# Range Lifts	Lift capacity	Rentable Space [m^3] [x 1000]	Maintenance Cost [\$] [x 1000]	Energy Cost [\$] [x 1000]	Realisation Cost [\$] [x 1000]	TravelTime [s]
Water	Traction	5	25	10	29.387	73	33	7.341	40
Traction	Magneto	19	7	15	29.228	53	33	8.155	42
Water	Traction	13	10	15	29.644	81	17	8.144	47
Magneto	Traction	13	10	15	29.994	56	31	11.729	46
Traction	Magneto	5	25	15	29.941	60	31	14.138	31
Water	Magneto	5	25	15	29.799	74	29	15.503	31
Magneto	Magneto	5	25	15	29.888	66	33	16.539	30
Traction	Traction	3	40	20	29.243	33	23	3.331	48
Water	Traction	3	40	20	29.198	38	23	3.775	48
Magneto	Traction	3	40	20	29.226	35	24	4.087	48
Constraints:					Rent. Space Between 29M and 30M				
					Maintenance <= 90,000				
					Energy <= 35,000				
					Realisation <= 20,000,000				
					TravelTime <= 50				

Figure 47. Resulting optimal solutions produced by Group 1

Group 2.

Stairs position	Elevators Type	Capacity of people	Number of Elevators	Stairs Evac. Coeff	Latency	Energy	Aoc	EvacTime	energy
External	Traction	30	100	0.1	11433.3	437341	5.35714	28648.2	153720

Figure 48. Resulting optimal solutions produced by Group 2

	∞ Coeff4	∞ eleva...	∞ suple...	∞ stairs	∞ Velocity	∞ h	∞ A	∞ Coeff1	∞ Coeff2	∞ Delay...	∞ nF	∞ maxPW	∞ tT	∞ pT
C VT	0.1	traction ...	helicopters	internal ...		450	14000	5	0.1		100	28000	1225	1200
	∞ nU	∞ flightL	∞ Coeff3	∞ areaPP		∞ areaP	∞ nFlows	∞ nPPFlow	∞ flightW	∞ areaSt	∞ throu...	∞ tGD		
C Intern...	6	10	150	0.3	0.17	17.7778	10.3704	33.3333	3.11111	62.2222	345.679	26.4706		
C exter...	6	5	80	0.3	0.17	0	19.4444	16.6667	5.83333	0	324.074	26.4706		
		∞ Capa...	∞ areaEl	∞ effci...			∞ grossW	∞ unloa...	∞ nT					
C tracti...	40	30	7.5	0.8	4	2.14286	2900	1740	23.3333	5				
C chain ...														
C pneu...														
C electr...														
C hydra...	40	30	7.5	0.5	1.5	2.14286	2900	2900	23.3333	5				
							∞ tFlight	∞ tEvac						
C ballons														
C catap...														
C helico...	3	30			30		915	25620	84					
C rockets														

Figure 49. The transportation product represented by Group 2

Group 3.

	∞ UnitLen...	∞ StatC...	∞ UnitMain...	∞ Powe...	∞ #Eq	∞ EqCost	∞ Maint...	∞ Oper...	∞ Const...	∞ Speed	∞ StopT...	∞ Length	∞ Max...	∞ ElevA...	∞ PersP...
C Elevator	1	0	2	2	100	200	200	200	400	4	5	400	266.667	90	40
C Escalator	2	0	2	2	800	1600	1600	1600	1395.37	0.5	0.5	697.687	136.281	35	30
C Train	3	3	3	3	20	60	60	60	34641.9	8	20	11467.3	2711.22	2	75
	∞ Height	∞ FloorH	∞ Width	∞ PerFF...	∞ #Floor	∞ PopP...									
C WTC	400	4	80	1000	100	100000									
	∞ Vertran	∞ Total...	∞ Evacuati...	∞ Max...											
C Solution	Elevator	1000	2475	266.667											
	∞ EPerUnit	∞ ESpeed	∞ ESize	∞ EStop...	∞ EUnit...	∞ EMac...	∞ EShaf...	∞ E#Eq	∞ EPow...	∞ EMain...	∞ EOpe...	∞ Econ...	∞ EMax...	∞ EEvacT	∞ E#Ma
C ST_Elevator	35	4	15	9	20	20	100	80	2	160	160	11200	12.5	3535.71	80
C DD_Elevator	20	4	12	5	10	20	100	100	1	100	100	8000	3.5	4950	100
C SkyLobbies	50	5	22.5	12	30	20	100	65	3	195	195	5750	19.6923	2436.92	65
C EOF_Elevator	35	4	15	8	20	20	100	80	2	160	160	11200	22.5	3535.71	80
C TD_Elevator	35	4	15	8	20	20	100	80	2	160	160	11200	7.5375	3535.71	80
C S5_Elevator	35	4	15	8	20	20	100	80	2	160	160	11200	11.25	3535.71	80
	∞ Elevato...	∞ E_Tot...	∞ E_EvacT...	∞ E_Ma...	∞ E_Co...										
- Solution2	ST_Elevator	11520	3535.71	12.5	8480										

Figure 50. The transportation product represented by Group 3

	Elevator Total	Evacuation	Max wait	Construction
	Cost	Time	Time	Time
Skylobbies	6140	1692.31	19.4872	2890
DD_Elevator	8200	3168	3.3	5600
ST_Elevator	11520	2475	11.25	8480

Figure 51. Resulting optimal solutions produced by Group3

Appendix E. Coffee problem.

Examples of hierarchical structures for the coffee problem.

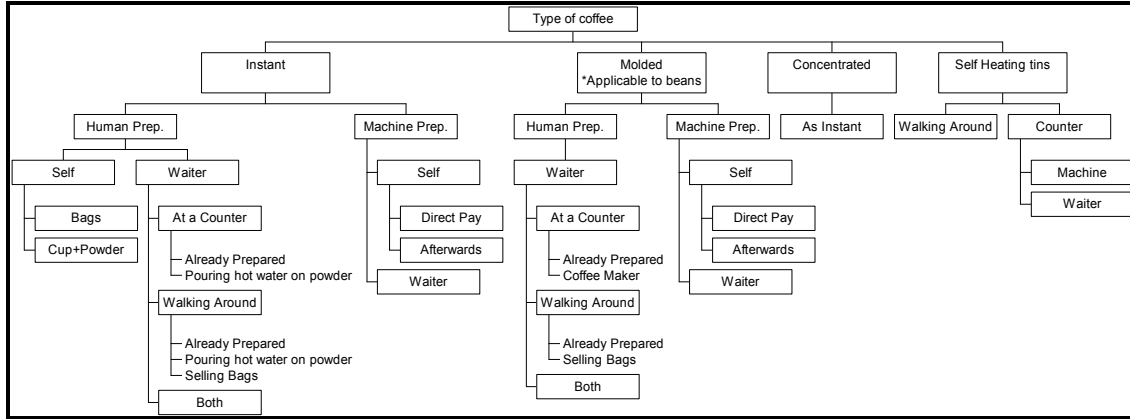


Figure 52. Type of coffee

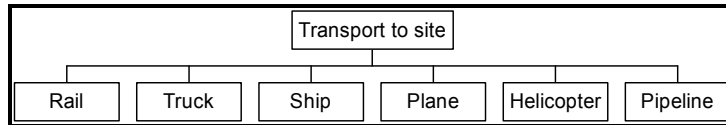


Figure 53. Transport to Site

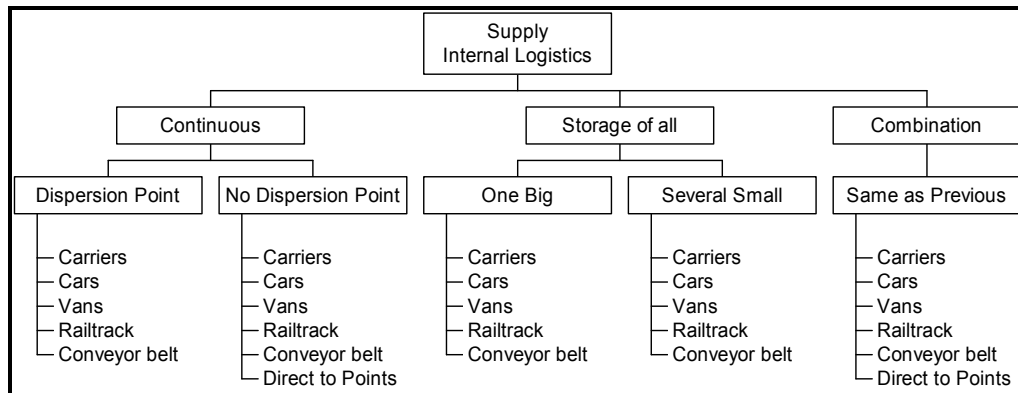


Figure 54. Internal logistics of supply

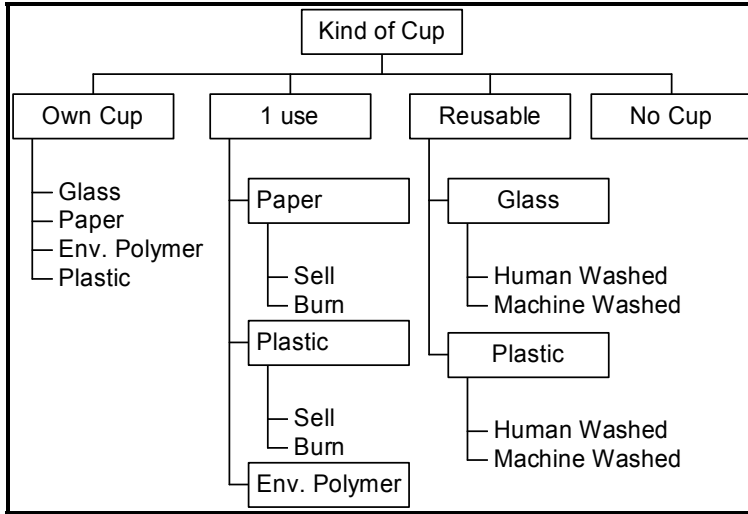


Figure 55. Kind of cup

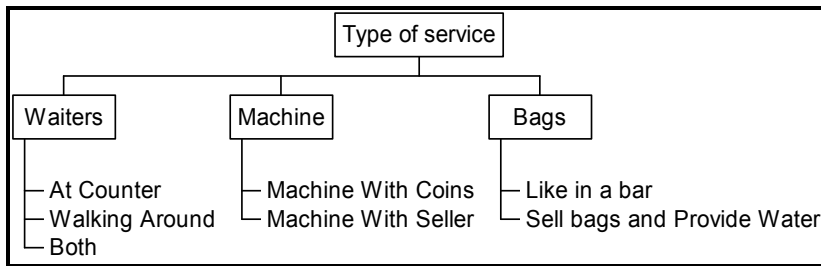


Figure 56. Type of service

Table 27. Example of a product model

COFFEE CUP		
Average number of coffee cups consumed per person per day	=	Normal, average coffee consumption per person per day * (Living standard * Special event amplification factor) / Selling price per cup
Total number of coffee cups sold during concert	=	Number of people in the audience * Number of concert days * Average number of coffee cups consumed per person per day
Number of paper cups of coffee sold during concert	=	Total number of coffee cups sold during the concert* Paper cups / Total cups consumption ratio
Number of reusable cups of coffee sold during concert	=	Total number of coffee cups sold during the concert - Number of paper cups sold during the concert
Cost of the paper cups	=	Total number of paper cups of coffee sold during the concert * Buying price of a paper cup
Cost of the reusable cups	=	Total number of people in the audience* (1 - Paper cups / Total cups consumption ratio) * Buying price of a reusable cup
Cost of the stirring sticks	=	Total number of coffee cups sold during the concert * Buying price of a stirring stick
Total expenses with the cups	=	Cost of paper cups + Cost of reusable cups + Cost of stirring sticks
CONCERT		
Duration in hours	=	Number of days * (24 - recess time)
ELECTRICITY		
Electricity consumption request	=	Number of coffee facilities * Maximum electricity consumption of a coffee facility (kw / h)
Total electricity consumption	=	Total water consumption (liters) * Energy required to heat one liter (kw/h)
Total expenses with electricity	=	Total electricity consumption * Cost of one kw/h
WATER		
Water consumption request (liters / min)	=	Number of coffee facilities * Maximum water consumption of a coffee facility (liters / min)
Total water consumption (liters)	=	Total number of coffee cups sold during the concert / Capacity of one cup
Total expenses with the water (NLG)	=	Total water consumption * Cost of one liter
TRUCKS		
Time needed to transport coffee machines (hours)	=	(Number of coffee machines / Truck capacity for coffee machines) * (Time to load the truck with machines * 2 + Distance * 2 / Average truck speed) * 2 / Number of trucks
Total fuel consumption for coffee machine transportation	=	(Number of coffee machines / Truck capacity for coffee machines) * (Distance to coffee machines suppliers / 100 * Average fuel consumption) * 4

(liters)		
Total expenses with transportation of coffee machines (NLG)	=	Total fuel consumption for coffee machines transportation * Price of fuel + ((Time needed to transport the coffee machines / 24 + 1) * Rent + Time needed to transport the coffee machines * Driver price) * Number of trucks
Time needed to transport the fork loaders (hours)	=	(Number of fork loaders / Truck capacity for fork loaders) * (Time to load the truck with loaders * 2 + Distance * 2 / Average truck speed) * 2 / Number of trucks
Total fuel consumption for fork loaders transportation (liters)	=	(Number of fork loaders / Truck capacity for fork loaders) * (Distance to fork loaders suppliers / 100 * Average fuel consumption) * 4
Total expenses with transportation of fork loaders (NLG)	=	Total fuel consumption for fork loaders transportation * Price of fuel + ((Time needed to transport the fork loaders / 24 + 1) * Rent + Time needed to transport the fork loaders * Driver price) * Number of trucks
Time needed to transport the ingredients (hours)	=	(Total weight of ingredients / Truck capacity for ingredients) / Number of trucks * (Time to load the truck with ingredients * 2 + Distance * 2 / Average truck speed) / Number of trucks
Total fuel consumption for ingredients transportation (liters)	=	(Number of fork loaders / Truck capacity for fork loaders) * (Distance to fork loaders suppliers / 100 * Average fuel consumption) * 4
Total expenses with ingredients transportation (NLG)	=	Total fuel consumption for ingredients transportation * Price of fuel + ((Time needed to transport the ingredients / 24 + 1) * Rent + Time needed to transport the ingredients * Driver price) * Number of trucks
FORK LOADERS		
Number of units required for coffee machine handling	=	(Number of coffee machines / Truck capacity for coffee machines) * Time to unload the coffee machines * 2 / Time needed to transport the coffee machines * Number of trucks / 3
Number of units required for ingredients handling	=	(Total weight of ingredients / Truck capacity in tons) * Time to unload the ingredients * 2 / Time needed to transport the ingredients * Number of trucks / 3
Total expenses with renting the fork loaders	=	Number of units required to handle coffee * (Time needed to transport the coffee machines / 12) * Rent + Time needed to transport the coffee machines * Driver salary + Time needed to transport the coffee machines * Fuel consumption * Fuel cost + Number of units required for ingredients * (Time needed to transport the ingredients / 12) * Rent + Time needed to transport the ingredients * Driver salary + Time needed to transport the ingredients * Fuel consumption * Fuel cost
Total expenses with logistics	=	Total expenses with transportation of coffee machines + Total expenses with transportation of fork loaders + Total expenses with ingredients transportation + Total expenses with renting the fork loaders
MATERIALS		
Length of electrical cables (m)	=	SQRT(Area surface / Number of stages) * 2 * Number of stages
Total expenses with electrical infrastructure (NLG)	=	Length of electrical cables * (Cost of one meter of cable + Cost of installation of one meter)
Length of water pipes (m)	=	(SQRT(Area surface / Number of water facilities) + SQRT(Area surface / Number of stages)) * Number of stages
Total expenses with water infrastructure (NLG)	=	Length of water pipes * (Cost of one meter of pipe + Cost of installation of one meter)
Total expenses with infrastructure (NLG)	=	Total expenses with water infrastructure + Total expenses with electrical infrastructure + Total expenses with coffee facilities
COFFEE MACHINES		
Number of machines	=	Total water consumption * Redundancy factor / (Concert duration [hours] * 60 * Delivery capacity of one machine [liters/min])
Expenses with renting (NLG)	=	Number of machines * (Duration of the concert [days] + Time of transportation [days]) * Rent for one machine
Expenses with facility (NLG)	=	Number of machines * Cost of one facility
Expenses with exploitation (NLG)	=	Number of machines * Duration of the concert [hours] * Salary of the operator
Expenses with installing / uninstalling (NLG)	=	Number of machines * Duration of installation [hours] * Salary of the technician
Expenses with maintenance (NLG)	=	Number of machines * Duration of the concert [hours] * Salary of the technician * Number of technicians
Total expenses with the coffee machines (NLG)	=	Expenses with renting + Expenses with exploitation + Expenses with installing / uninstalling + Expenses with maintenance
DELIVERY		
Number of delivery persons	=	(Store capacity of one machine / Redundancy factor) / ((60 / Time to serve one person) * (60 - Time to refill) * Cup capacity) * Number of machines
Total expenses with delivery (NLG)	=	Number of delivery persons * Salary of a delivery person * Duration of the concert [hours]

MONEY COLLECTION		
Number of money collectors	=	(Number of coffee machines / Number of machines checked per hour) / Number of recess hours per day
Total expenses with the money collection (NLG)	=	Number of money collectors * Salary of one collector * Duration of the concert [days] * Number of recess hours per day
DELIVERY MANAGEMENT SYSTEM		
Number of pagers	=	Number of delivery persons + Number of technicians+ Number of money collectors+ Number of coffee machine operators
Total expenses with the management system (NLG)	=	Number of pagers * (Rent + Subscription) * Duration of concert [days]+ Transportation costs + Development costs + Number of operators * Salary of one operator * Duration of the concert [hours]+ Cost per message * Pagers per stage * Number of redirection activities per hour * Duration of the concert [hour]
INGREDIENTS		
Quantity of coffee [kg]	=	Number of coffee cups sold during the concert * Coffee concentration of one cup
Total expenses for coffee [NLG]	=	Quantity of coffee * Coffee price per kilo
Quantity of sugar [kg]	=	Number of coffee cups sold during the concert * Sugar concentration of one cup
Total expenses for sugar (NLG)	=	Quantity of sugar * Sugar price per kilo
Quantity of milk (kg)	=	Number of coffee cups sold during the concert * Milk concentration of one cup
Total expenses for milk (NLG)	=	Quantity of milk * Milk price per kilo
Total expenses with ingredients (NLG)	=	Total expenses with coffee + Total expenses with sugar + Total expenses with milk
GARBAGE		
Total garbage weight (kg)	=	Number of paper cups sold during the concert * Weight of one cup + Number of coffee cups sold during the concert * Weight of one stirring stick
Number of small garbage store units	=	(Total garbage weight / Capacity of one small garbage unit) / (Concert duration [days] * Number of disposal actions per day) * Garbage redundancy
Number of large garbage store units	=	Total garbage weight / Capacity of one large garbage unit
Total expenses for garbage storages (NLG)	=	Concert duration [days] * (Number of small garbage units * Rent of small units + Number of large garbage units * Rent of large units)
Number of garbage collectors during recess	=	Concert area / (Coverage area * Number of recess hours per day)
Number of garbage collectors during performance	=	(Number of small garbage units / Number of stages) / (24 - Number of recess hours) / Time to dispose one garbage unit)
Total expenses for garbage collectors (NLG)	=	(Number of garbage collectors during recess * Number of recess hours + Number of garbage collectors during performance * (24 - Number of recess hours per day)) * Concert duration [days] * Salary of one garbage collector
Total expenses for garbage disposal (NLG)	=	Total garbage weight * Price of disposal of one ton
Total expenses with garbage (NLG)	=	Total expenses with garbage collectors+ Total expenses with garbage disposal+ Total expenses for garbage storages
PERSONNEL		
Traveling expenses (NLG)	=	Number of employees * Traveling expenses for one person
Total expenses with recruiting personnel (NLG)	=	Number of low qualification personnel * Low qualification personnel fee + Number of high qualification personnel * High qualification personnel fee

Solution: Instant Coffee		Solution: Percolator	
Independent variables context	standard value	Independent variables context	
Number of Visitors	6,00E+06	Number of visitors	6,00E+06
Staff members	5,00E+04	Number of Staff members	5,00E+04
Price per kg waste (all in) €/kg	0,5	Price per kg waste (all in) €/kg	0,5
Independent variables decision		Independent variables decision	
consumption per visitor	4,32	avarage consumption per visitor	4,32
price/cup of coffee (€)	1,5	price/cup of coffee (€)	1,5
Taste coffee (excellent, good,reasonable,yuk!,disgusting)	reasonable	Taste coffee (excellent, good,reasonable,yuk!,disgusting)	good
Temperature coffee	85	Coffee Temperature	80
opening hours from 7am till...(h)	15	openinghours from 7am till...(h)	15
number of peak hours per hour opened	0,25	number of peakhours per hour opened	0,25
preparation time (days)	30	Preparation time (days)	40
staff works in a team of .. People	12	staff works in a team of .. People	12
guests per staff member	120	guests per staff member	120
coffee whole sale price €/kg	36	filter coffee waste weight per cup served	0,005
sponsor money cents per cup	0,01	Coffee price €/kg	4,36
		sponsor money cents per cup	0,01
Independent variables temporary		Independent variables temporary	
costs of 1 SP (€)	500	Cost of 1 SP (€)	275
serving time (sec.)	15	serving time (sec.)	15
number of sales points	6000	number of sellingpoints	6000
Dependent variables		Dependent variables	
Number of cups of coffee per concert	25920000	Number of cups of coffee per concert	25920000
kg coffee per concert	46656	Cup cost € per concert	1048205
man-hours(money)	10550000	kg coffee per concert	155520
waiting time	0,2	available stafftime 5%	25000
garbage cost	55728	minimal number of guests per team per hour	1440
sponsor income €	259200	man-hours(money)	10550000
Total costs	13589353,28	Waiting time (h)	0,2
		garbage cost €	101088
Profit	22549846,72	sponsor income €	259200
Quality	5	Total costs	14450089,8
		Profit	24689110,2
		Quality	6,6

Figure 57. Example of solutions to the coffee problem

The weights of objectives are calculated in the top part of the matrix in Figure 58; in the lower part of the matrix the solutions are evaluated. In this example, the preferred solution is the first solution using percolator coffee.

Priority Matrix							
Equal=1 Slightly different=3 Very different=10							
		Q1	Q2	Q3	Q4	Q5	Σ Wi
Quality	Q1		3.00	1.00	0.33	10.00	14.33
Profit	Q2	0.33		0.33	0.33	1.00	2.00
Garbage Cost	Q3	1.00	3.00		0.33	3.00	7.33
Waiting Time	Q4	3.00	3.00	3.00		3.00	12.00
Man Hours	Q5	0.10	1.00	0.33	0.33		1.77
Steps: 0; 2,5; 5; 7,5; 10							
		Q1	Q2	Q3	Q4	Q5	
Percolator	C1	7.5	7.5	2.5	5	5	
Instant	C2	5	5	5	5	5	
Iced	C3	2.5	2.5	7.5	2.5	10	
		Q1*W1	Q2*W2	Q3*W3	Q4*W4	Q5*W5	Total
Percolator	C1	107.50	15.00	18.33	60.00	8.83	209.67
Instant	C2	71.67	10.00	36.67	60.00	8.83	187.17
Iced	C3	35.83	5.00	55.00	30.00	17.67	143.50

Figure 58. Example of priority matrix for the prioritization of objectives

Bibliography

- Akman V, P J W ten Haagen and Tomiyama (1990), "A fundamental and theoretical framework for an intelligent CAD system", *Computer-Aided Design* 22-6, Butterworth-Heinemann Ltd, pp 352-368.
- Alexander C., (1971), "The State of the Art in Design Methods", in: Cross N. (Ed).
- Altshuller G (1990) "And Suddenly the Inventor Appeared", ISBN # 0964074028.
- Altshuller G "The Innovation Algorithm", ISBN # 0964074044.
- Asimow M. (1962), "Introduction to Design", New York: Prentice-Hall, pp. 3,24,64.
- Atman C.J. et al. (2000) "Using Multiple Methods to Evaluate a Freshmen Design Course", 30th ASEE/IEEE Frontiers in Education Conference S1A-6, Kansas City, MO.
- Andrews P. Snowden D., (2002) "Next generation Knowledge Management: The complexity of humans", Executive Tek Report, IBM Global Services.
- Austina S, Newtonb A, Steeleb J, Waskett P, (2002) "Modelling and managing project complexity" *International Journal of Project Management* 20, pp. 191-198.
- Baarda D.B., (1995) "Methoden en technieken", de Goede MPM, Stenfert Kroese, Houten.
- Back T., Hammel U. and Schwefel H.-P. (1997), "Evolutionary computation: Comments on the history and current state". *IEEE Transactions on Evolutionary Computation* 1(1), pp. 3-17.
- Boer de Thomas (1999) "Kritisch Denken", Nieuwezijds Publishing Company, ISBN 9057120577.
- Berg van den N. W., et al. (1995) "Beginning LCA: a guide into environmental life cycle assessment", Leiden: Centre of Environmental Science.
- Bono de E., (1970). "Lateral Thinking". Penguin Books, London, UK.
- Booch, Grady, (1990), "Object-Oriented Design with applications." Benjamin/Cummings, ISBN:0-8053-0091-0.
- Booch G. (1993), "Object-Oriented Analysis and Design with Applications" Addison-Wesley Pub Co; 2nd edition ISBN: 0805353402.
- Booch G., Rumbaugh J., Jacobson I. (1998), "The Unified Modeling Language User Guide" Addison-Wesley Pub Co; 1st edition (September 30, 1998) ISBN: 0201571684.
- Bray T., (2000) "Extensible Markup Language (XML) 1.0" XML standard, 2nd edition.
- Brown K.N., et al, (1994), "Constraint unified grammars: specifying languages of parametric design.", In: Gero J.S., Sudweeks F, et al., editors. *Artificial intelligence in design*. Dordrecht: Kluwer Academic Publishers, pp. 239-256.
- Bruijn A. (2003) "Inbreng TNO is Voor Ons Onmisbaar Geweest", TNO magazine, Februari, pp.12-13.
- Butler. J Jia J. Dyer J. (1996) "Simulation techniques for the sensitivity analysis of multi-criteria decision models". Report MSIS, CBA 5.202 The Graduate School of Business, University of Texas at Austin.

- Cross, N, (1994), *Engineering Design Methods: Strategies for Product Design*, John Wiley & Sons, Chichester, UK. (2nd ed.).
- Davenport T.H., DeLong D.W., Beers M.C. (1998), "Successful knowledge management projects", *Sloan Management Review* 40: 43-57.
- Denn M.M. (1969) "Optimization by Variational Methods", McGraw-Hill.
- Dorst K. (1997), "Describing Design. A comparison of paradigms", ISBN 90-9010822-X.
- Evbuomwan N., Sivaloganathan, S., Jebb, A, (1996), "A Survey of Design Philosophies, Models, Methods and Systems", in *Proceedings of the Institution of Mechanical Engineers, London, Part B: Journal of Engineering Manufacture*, Vol. 210, pp. 301-319.
- French, M., (1971), "Conceptual Design For Engineers", 1st edition, The Design Council, London.
- Fricke, G.(1996) "Successful individual approaches in engineering design", *Research in Engineering Design*, 8, pp. 151-165.
- Friedl G., Rutten P., Trum H., "Innovative process modeling of Building Design; Design as a crystallization process", W096 - Architectural Management, Value Through Design, CIB Publication 280 - ISBN 90-6363-032-8
- Gamma E., Helm R., Johnson R., Vlissides J. (1995), "Design Patterns", Addison-Wesley Pub Co; 1st edition ISBN: 0201633612.
- Geoffrion A.M., (1988), "The Formal Aspects of Structured Modeling", *Operations Research*, 41, pp.33-45.
- Gilb T., (1988) "Principles of Software Engineering Management", Addison-Wesley, ISBN 0-201-19246-2.
- Glover J., Ronning R. and Reynolds C. (eds.), (1989). "Handbook of Creativity". Plenum, London, UK.
- Glynn Winskel.(1993) "the formal semantics of programming languages", an introduction, MIT press.
- Hicks, J. O., Hicks, S. A. & Sen, T. K. (1991) " Learning Spreadsheets: Human Instruction vs. Computer-Based Instruction Behaviour & Information Technology, Vol. 10, No. 6, 491-500.
- Hubka, V., (1992), "Design For Quality and Design Methodology", *J. Engng Des.*, 3(1), pp.5-15.
- Hurlimann T. (1999) "Mathematical Modeling and Optimization. An Essay for the Design of Computer-Based Modelling Tools ", Kluwer Academic Publishers, ISBN 0-7923-5927-5.
- Jones A. (1979), "The Object Model: A Conceptual Tool for Structuring Software", *Operating Systems*, New York, N.Y.: Springer-Verlag, p.8.
- Ivashkov M. and van Overveld K., (2001) "An Operational Model for Design Processes", *Proceedings of ICED '01*, Vol.2, Glasgow, pp.139-146.

- Ivashkov M., van Overveld K., (2002) "Early Validation of a Design Method Based On Structured Reflection", Proceedings of International Design Conference Design 2002, Dubrovnik, May 14-17, Vol 1, pp. 343-348.
- Ivashkov M, Souchkov V, (2004) "Establishing Priority of TRIZ Inventive Principles in Early Design", Design 2004, Dubrovnik, Accepted.
- Jones J.C. (1992), "Design methods: seeds of human futures", ISBN: 0471284963.
- Hatchuel A, Weil B. (1992), "L'expert et le système", Paris: Economica.
- Killander A.& Sushkov V. (1995), "Conflict-Oriented Model of Creative Design", in Proceedings of the 3rd Int. Roundtable Conference on Computational Models of Creative Design, J.S. Gero, M.L. Maher & F. Sudweeks, 3-7 December 1995, Heron Island, Queensland, Australia, 1995, 369-397.
- Klimoski, R and Mohammed S. (1994), "Team Mental Model: Construct or Metaphor" Journal of Management Vol 20 pp. 403-437.
- Knowles J. D. and Corne D. W.. (1999), "The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation". In Congress on Evolutionary Computation (CEC99), volume 1, Piscataway, NJ. IEEE Press, pp. 98-105.
- Kramer, J. (1991) "CASE Support for the Software Process: A Research Viewpoint", Proceedings of Third European Software Engineering Conference, Milan, Italy, October 1991, LNCS 550, Springer-Verlag. pp. 499-503.
- Lain, M., (2003) Personal Communications.
- Lawson B, (1980), "How Designers Think", The Architectural Press, London, UK.
- Lee Jongsoo, Hajela Prabhat, (2001), "Application of classifier systems in improving response surface based approximations for design optimization", Computers and Structures 79 pp. 333-344.
- Levesque H J, (1984), "The logic of incomplete knowledge bases" in Brodie M L , Mylopoulos J and Schmidt J W (eds) On conceptual modelling Springer-Verlag, pp 165-186.
- Martinaitis V., (1998), "Analytic calculation of degree-days for the regulated heating season", Energy and Buildings, Volume 28, Issue 2, October, pp. 185-189.
- Meyer D.E. and Schvaneveld R.W., (1976), "The Structure of Human Memory", Cofer (Ed.), San Francisco:Freeman.
- Mili F. et al. (2001), "Knowledge modelling for design decisions", Artificial Intelligence in Engineering 15, pp. 153-164.
- Mohammed S. and Dumville B C , (2001), "Team mental models in a team knowledge framework: expanding theory and measurement across disciplinary boundaries" Journal of Organizational Behavior Vol 22 pp. 89-106.
- Neville, G.E. (1989), "Computational models of design processes", in proceedings of the 1988 NSF Grantee Workshop on design theory and methodology, Design theory '88 (eds S.L. Newsome, W.R. Spillers and S. Finger), (springer-Verlag, New York), pp 82-96.
- Nooteboom B. (1996), "Towards a cognitive theory of the firm. Issues and a logic of change", Proceedings of the AFEE conference, San Francisco.

- Overveld K, Ahn R., Reymen I, Ivashkov M, (2003) "Teaching Creativity in a Technological Design Context", *IJEE*, Vol 19-2, pp. 260-271.
- Overveld van K, Ivashkov M , (2003) "From creative ideas to optimised concepts and back: A method for collaborative creation of solution alternatives in decision support systems." *Proceedings of ICED03*, Stockholm.
- Pahl, G. and Beitz W., (1984) "Engineering Design", The Design Council, ed. K. M. Wallace
- Parmee, I C & Denham, M J, (1994), ."The Integration of Adaptive Search Techniques with Current Engineering Design Practice". In *Adaptive Computing in Engineering Design and Control -'94*, Plymouth, pp.1-13.
- Polanyi M. (1966), "The Tacit Dimension", Garden City, NY: Doubleday.
- Prasad B. (1998) "Review of QFD and Related Deployment Techniques" *Journal of Manufacturing Systems* 1998, Vol. 17;3 p.221-235.
- Radcliffe D. and Lee T.Y., (1989), "Design methods used by undergraduate engineering students", *Design Studies*, 10-4.
- Reymen, I.M.M.J. (2001) "Improving Design Processes through Structured Reflection: A Domain-independent Approach", Ph.D. thesis, ISBN 90-386-0831-4, Technische Universiteit Eindhoven, Eindhoven, The Netherlands.
- Ronen, B., Palley, M. A. and Lucas, H. C. (Jan 1989) "Spreadsheet analysis and Design" *CACM*, Vol. 32, No. 1, 84-93.
- Rosenberg, M.J. (2001) "E-learning. Strategies for delivering knowledge in the digital age", ISBN 0-07-136268-1, McGraw-Hill.
- Rosenman M.A., Gero J.S. (1999), " Purpose and function in a collaborative CAD environment ", *Reliability Engineering and System Safety* 64, pp. 167–179.
- Rosenman M.A., Gero J.S. (1998) "Purpose and function in design: from the socio-cultural to the techno-physical", *Design Studies*, Volume 19, Issue 2, April, Pages 161-186.
- Rudolph G. and Agapie A.(2000), "Convergence properties of some multi-objective evolutionary algorithms". In *Congress on Evolutionary Computation (CEC 2000)*, volume 2, Piscataway, NJ. IEEE Press pp.1010-1016.
- Rutten, P.G.S. and Trum, H.M.G.J. (1998), "Meer Ontwerpen dan Rekenen; een Meta-Ontwerpomgeving voor Gebouw en Installatie", *TVVL magazine* 4, pp. 14-22.
- Schön D.A. (1983), "The reflective Practitioner", Basic Books, New York.
- Schön, D. A. (1991) "The Reflective Practitioner—How Professionals Think in Action", Avebury, Aldershot, UK.
- Schoemaker, P.J., Waid, C.C. (1982) "An experimental comparison of different approaches to determining weights in additive utility models," *Management Science*, 28, 182-196.
- Silverman B. W. (1986), "Density estimation for statistics and data analysis". Chapman and Hall, London.
- Simon, H. A. (1984) "The structure of ill-structured problems" in N. Cross (ed) *Developments in Design Methodology*, Wiley, Chichester, UK.

- Simon H. A. (1992), "Science of the Artificial", The MIT Press, Cambridge MA.
- Smith, E.E. and Medin, D.L., (1981), "Categories and Concepts", Cambridge, MA: Harvard University press.
- Soslo, R.L., (1998), "Cognitive Psychology", 5th edition, Allyn&Bacon, ISBN 0-205-27418-8
- Stempfle J, (2002), "Thinking in design teams - an analysis of team communication", Design Studies, Volume 23, Issue 5, September 2002, Petra Badke-Schaub, pp. 473-496.
- Teminko. J., Zusman A. and Zlotin B., (1998), "Systematic Innovation", Ideation International Inc. Southfield, Mich.
- Valkenburg R. and Dorst K., (1998), "The reflective practice of design teams", Design Studies, Volume 19, Issue 3, July 1998, pp. 249-271.
- Veth, B, (1987), "An intergrated data description language for coding design knowledge", in ten Hagen, P J W and Tomiyama, T (eds) Intelligent CAD systems 1: theoretical and Methodloogical Aspects Springer-Verlag, pp 295-313.
- Vinck D. (1997), "La connaissance : ses objets et ses institutions. In Intégration des savoir-faire, capitalisation des connaissances", ed. Fouet JM. Paris: Editions Hermès.
- Weber. M. and Borchering K. (1993), "Behavioral influences on weight judgments in multiattribute decision making", European Journal of Operational Research, 26, 35-41.
- Watts, R,D,, (1966), "The elements of Design", In the Design Methods (Edited by S. Gregory) London: Butterworths.
- Wickens D.D., Engle R.W. (1970) "Imaginary and abstractness in short-term memory", Journal of Experimental Psychology, 84, pp. 268-272.
- Wolfram Stephen, (1996), "The Mathematica Book", third edition, Cambridge University Press.
- Zitzler E., Deb K., and Thiele L.. (2000), "Comparison of multiobjective evolutionary algorithms: Empirical results". Evolutionary Computation, 8(2), pp.173-195.
- Zitzler E., Laumanns M., and Thiele L.(2001) "SPEA2: Improving the Strength Pareto Evolutionary Algorithm". Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.
- Zitzler E.(2002), "Evolutionary methods for design, optimisation and control" K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou and T. Fogarty (Eds.) c CIMNE, Barcelona, Spain.
- Zwicky, F., (1969) Discovery, invention, research through the morphological approach. New York: MacMillan,

Summary

In recent years, industry has stressed the need for graduates who are capable of solving design problems. Design problems are typically ill-defined, they have no single best way to proceed, no single best formulation of the problem, and have several possible but unknown solutions. The designers therefore have to make many decisions. In the early design phase the decisions cover conceptual design issues, which are often directive while, at the same time, being restrictive and irreversible. Early design decisions therefore need to simultaneously take into account a wide-range of considerations.

Skilled designers rely on their past experience to manage the complexity. New designers do not yet have this type of experience. Several research studies have shown that a systematic approach can be helpful to students, and that systematic procedures correlate positively with both the quantity and the quality of the students' design results. The lack of computational tools capable of efficiently dealing with the complexity caused by these systematic approaches makes it difficult to demonstrate their effectiveness.

In this thesis we investigate a computer-supported systematic approach to the early design phase. The design problem addressed in this thesis is defined as follows: Develop a systematic procedure and a tool to support the early design phase in a domain-independent way:

- To evaluate the application of systematic procedures in the early design phase;
- To increase the complexity of the design problems that can be considered by the students during the design courses, and to approach the complexity of realistic design problems in industry;
- To improve the clarity of the decision-making process and the process of communication.

Development of a support requires formalization and modeling of the design process. We developed an operational model of the design process that turns concept generation process into a computer supported procedure. We model the design process in a step-by-step manner, such that after each step the participants reflect on their design activities by means of our formalism. Regular and formalized reflections aim to stabilize progress in the design process and enable computational usage of the available knowledge. Our model is based on existing theories but introduces several novelties, which we consider as contributions of our work:

- A formalism that allows to describe the design process formally. The formalism resembles natural language, but at the same time it relates the expressed knowledge to mathematical objects, such as functions, variables and values;
- Four knowledge spaces. The formalism relates the knowledge expressed in these spaces to mathematical objects, such as functions, variables and values. The four knowledge spaces distinguish between four orthogonal categories of operations with knowledge. This distinction makes application of the formalism closely related to different ways in which people ask and answer questions;
- A formalized mechanism of questions and answers. This mechanism aims to enhance the clarity of reflections and make application of the formalism more

natural. Using the knowledge spaces we were able to formalize eight categories of questions and eight corresponding formats in which to give the answers;

- The computational usage of expressed knowledge. Knowledge is linked to computational procedures, which are capable of automatically evaluating knowledge and generating optimal concepts without requiring extra steps from the designer;
- An adapted genetic algorithm that supports the process of generating concepts and enables consideration of complex design problems with multiple objectives and multiple design decisions;
- A decision-making method that allows the convergence to a single optimal solution without prioritizing objectives;
- The last but not the least, our approach is embodied in a support tool that provides a graphical user interface and implements the derived from the model functional requirements. The tool turns modelling of the design process from a theoretical into a practically useful activity.

In the past few years we have validated the approach in design education. From classroom experiments we can conclude that this approach provides computational and modeling support for various design activities, ranging from the systematic application of creativity techniques to modeling, sensitivity analysis and optimization. This enables students to learn various design approaches quickly, to consider more alternative solutions, and eventually to be better prepared for realistic design problems. In an experiment in a more industrial setting we were able to show that our method gives a significant improvement over traditional design methods.

Samenvatting

Sinds de laatste jaren wordt vanuit de industrie een behoefte gesignaleerd aan afgestudeerden die in staat zijn om ontwerproblemen op te lossen. Ontwerproblemen zijn typisch slecht gedefinieerd, er is geen standaardmethode om ontwerproblemen te formuleren of aan te pakken, en ontwerproblemen hebben doorgaans meerdere, onbekende oplossingen. Daarom moeten ontwerpers een veelheid aan beslissingen nemen. In de beginfase van het ontwerpproces betreffen deze beslissingen conceptuele ontwerpaangelegenheden die vaak een grote stempel drukken op het vervolg van het proces. Ze zijn doorgaans in hoge mate beperkend en onomkeerbaar. Bij ontwerpbeslissingen in de beginfase van het ontwerpproces moet daarom een grote hoeveelheid aan aspecten in ogenschouw genomen worden.

Geoefende ontwerpers vertrouwen op hun ervaring om met deze moeilijkheden om te gaan. Aankomende ontwerpers missen deze ervaring. Verscheidene studies hebben aangetoond dat een systematische aanpak behulpzaam kan zijn voor studenten, en dat het gebruik van systematische methoden positief correleert zowel met de kwaliteit als de kwantiteit van de ontwerpresultaten van ontwerpers in opleiding. Omdat echter voor beginfasen van ontwerpprocessen nog geen software gereedschappen beschikbaar waren om met de complexiteit om te gaan die door zo'n systematische aanpak veroorzaakt wordt, was het tot dusverre moeilijk om de effectiviteit van een systematische aanpak in die beginfasen aan te tonen.

In dit proefontwerp bestuderen we een door software ondersteunde systematische aanpak van de beginfase van ontwerpprocessen. Het ontwerpprobleem dat aan dit proefontwerp ten grondslag ligt wordt gedefinieerd als volgt: 'Ontwikkel een systematische procedure en een software gereedschap om beginfasen van ontwerpprocessen te ondersteunen waarbij geen aannamen gemaakt worden over het domein waarbinnen het ontwerpproces plaatsvindt.' Met name was onze doelstelling:

- na te gaan hoe systematische aanpakmethode voor beginfasen in ontwerpprocessen toegepast kunnen worden;
- het verhogen van de complexiteit van ontwerproblemen, zoals die door studenten tijdens het ontwerponderwijs aangepakt kunnen worden, opdat die ontwerproblemen de complexiteit van de industriële praktijk benaderen;
- het verbeteren van de helderheid van beslissings- en communicatieprocessen tijdens het ontwerpen.

De ontwikkeling van ondersteunend ontwerpgereedschap vergt formalisering en modellering van het ontwerpproces. We ontwikkelden daartoe een operationeel model van ontwerpprocessen waardoor het genereren van ontwerpconcepten door een computer ondersteund kan worden. We modelleren het ontwerpproces als een discrete reeks stappen waarbij na iedere stap door de ontwerpers op hun ontwerpactiviteit gereflecteerd wordt middels het hier voorgestelde formalisme. Regelmatige en formele reflectie is bedoeld om een zekere voortgang in het ontwerpproces te garanderen en computationeel gebruik te kunnen maken van kennis die tijdens dat ontwerpproces beschikbaar komt. Ons model is gebaseerd op bestaande inzichten en theorieën, maar het introduceert verscheidene nieuwe aspecten. Als voornaamste bijdragen van dit werk beschouwen we:

- een formalisme dat toestaat een ontwerpproces formeel te beschrijven. Het formalisme vertoont gelijkenis met natuurlijk taal, maar tegelijkertijd relateert het de gerepresenteerde kennis aan wiskundige objecten zoals functies, variabelen en waarden;
- vier specifieke soorten ontwerp-kennis. Deze onderscheiden vier onderling onafhankelijke categorieën van bewerkingen op (ontwerp-)kennis. Het onderscheid tussen de vier soorten kennis is nauw verbonden met verschillende wijzen waarop in natuurlijke taal vragen gesteld en beantwoord worden;
- een geformaliseerd mechanisme voor het stellen en beantwoorden van vragen. Dit mechanisme beoogt de helderheid van reflecties te stimuleren, en de toepassing van het formalisme meer natuurlijk te maken. Door gebruik te maken van het onderscheid van de vier soorten (ontwerp-)kennis werden acht categorieën vragen geïdentificeerd, met corresponderende formaten waarin de antwoorden gegeven kunnen worden;
- het computationeel gebruik van gerepresenteerde kennis. De kennis in het voorgestelde systeem is geassocieerd met computationele procedures, die de ontwerper in staat stellen om automatisch bepaalde kennisinhouden te verifiëren en in zekere zin ‘optimale’ concepten te genereren zonder dat daarvoor afzonderlijke inspanning van de gebruiker benodigd is;
- een aangepast genetisch algoritme dat het proces van het genereren van ontwerpconcepten ondersteunt, en waardoor complexe ontwerpproblemen met meerdere doelfuncties en meerdere ontwerpbeslissingen beschouwd kunnen worden;
- een beslissingsmethode die toestaat om naar een enkele ‘optimale’ oplossing te convergeren zonder dat in de doelfuncties een voorkeursvolgorde aangegeven hoeft te worden;
- tenslotte, een implementatie van de voorgestelde aanpak in een software systeem dat een grafische user interface aanbiedt en een realisatie vormt van de functionele vereisten zoals die uit het formele model afgeleid werden.

In de afgelopen jaren hebben we de voorgestelde aanpak in het ontwerponderwijs gevalideerd. Uit experimenten in onderwijssituaties concluderen we dat deze aanpak computationele ondersteuning biedt voor diverse ontwerpactiviteiten, die uiteenlopen van de systematische toepassing van creativiteitsmethoden, via modellering, en gevoeligheidsanalyse, tot optimalisatie. Dit stelt studenten in staat om snel met verschillende aanpakken voor ontwerpproblemen kennis te maken, om meer alternatieve oplossingen te beschouwen, en uiteindelijk beter voorbereid te worden op realistische ontwerpproblemen. Ten slotte, in een experiment in meer industriële omstandigheden waren we in staat om aan te tonen dat onze methode een significante verbetering oplevert in vergelijking met traditionele ontwerpmethoden.

Acknowledgement

I would also like to thank all the members of the committee who examined my thesis. This thesis involves several disciplines, with each of its five chapters. Because so many people helped me in any way, by using 'we' in this thesis I refer to myself and those people.

The first chapter introduces the early phase of the design process and belongs in the design science domain. I would like to thank prof. Paul Rutten and dr. Henk Trum for the useful discussions concerning this chapter.

The second chapter focuses on development of an operational model of the design process and belongs to the domain of mathematics and artificial intelligence. I would like to thank dr. Kees van Overveld and dr. Rene Ahn for the valuable dialogues we shared about this PhD project and about the presented in this chapter formalism in particular.

The third chapter focuses on the development of a support tool and belongs to the computer science domain. I would like to thank dr. Kees van Overveld for his involvement in the development process and his positive attitude during the first experiments.

The fourth chapter focuses on case studies and demonstrates application of our approach to various engineering domains. I would like to thank post-graduate students who dared to use the developed tool and provided valued feedback. I am grateful to ir. Milosh Lain for a case study and an excellent opportunity to apply the developed tool to design of an air-conditioning system.

The fifth chapter focuses on the results and recommendations. This chapter summarizes my personal involvement in the results of the presented work. It would be too odd to thank myself, but I am glad that I started this project and especially that I finished it in a way.

My particular thanks go to dr. Marloes van Lierop for recognizing the problems in teaching of design and for initiating this project. My thanks also go to dr. Kees Dorst and dr. Isabelle Reymen for the valuable discussions we shared on the methodology of research in the field of design which made my start up easier.

I would like to show appreciation to prof. Ton van Kemenade for his numerous initiatives that have strengthened the links between the Netherlands and Belarus.

Last but not least, I express thanks to my friends and my family, especially my mother - Ivashkova Lucia and my wife - Molchanova Galina.

Biographical note

Maxim Ivashkov was born on February 19, 1975 in Minsk, Belarus. He gained his first research experience as a research fellow in the Scientific Research Institute of Applied Physical Problems in Minsk. He was involved in the development of a spectral method for quickly identifying complex phenol compounds, which was the topic of his Master's thesis. After his graduation in 1997, Maxim worked as a scientific officer at Invention Machine Corp, where he participated in the process of knowledge development for the scientific database 'IM-Phenomenon'. Between 1998 and 2000 he participated in modeling projects for Océ, the faculty of Precision Engineering at the Technical University of Eindhoven (TU/e) and "Fontijne Holland". In 2000 Maxim started his PhD design project with the goal of developing a tool for supporting concept generation in the early design phase.

Maxim's academic background ranges from an M.Sc in physics obtained from Belorussian State University in 1997 to a 'Mathematician-Economist' degree, obtained from the same university. Between 1998 and 2000 he followed a Master's program in 'Mathematics for industry' at TU/e, where he improved his personal skills, learned mathematical modeling and participated in few projects. During the program he learned and practiced software development processes and followed a course entitled 'Methods and Techniques for Design'. These two ingredients prompted him to start a PhD in design project. His current academic interests include research into the application of systematic procedures in the early design phase, such as TRIZ (Theory of Inventive Problem Solving), and computational support of decision-making in the early design phase. During his PhD project Maxim enjoyed teaching TRIZ and giving training sessions about the developed support tool.

Stellingen (propositions)

-ACCEL: a Tool for Supporting Concept Generation in the Early Design Phase¹ –

Maxim Ivashkov

1. People use knowledge in four different ways. The same question can therefore be answered in eight different ways (this thesis, page 113).
2. Similar to conceptual solutions, the stakeholder may be regarded as a concept occurring in a formal model of a design process (this thesis, page 29).
3. Formalisms (languages) become natural if they appear to be useful (this thesis, page 115).
4. Tools for the early design phase are still in the early phase of design (this thesis, page 6).
5. We should disconnect the requirements from the new solutions, but ... this cannot be done within a single brain. A computer can play the role of solution generator, provided we can represent the space of all solutions (Kees van Overveld, a presentation for the day of design, TU/e, November 2001).
6. Design problems do not easily fit predefined categories (Kleban, 2001).
7. By combining two successful solutions one can produce a solution with a better chance of market success than would be achieved using random mutations.
8. Users tend to make critical remarks about the usability of a product only if the product has potential functionality.
9. Just as words are necessary to understand a picture, likewise a product model is necessary to understand a design problem.
10. Anyone who has ever written a text knows what a design problem is.
11. Guess if you can, choose if you dare (Pierre Corneille, *Heraclius*, IV, 4).
12. Stating problems is just as important as solving problems. (Translated from L. Apostel).
13. It is not the sea that makes ships sink; it is the wind (Russian proverb).

¹ The thesis and the tool are available at www.acceling.com