Activity / Constraint - Object Modeller

A development environment for
Linear Programming
based
Decision Support Systems
Activity / Constraint - Object Modeller

A development environment for Linear Programming based Decision Support Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Delft, op gezag van de Rector Magnificus Prof. ir K.F. Wakker, in het openbaar te verdedigen ten overstaan van een commissie, door het College van Dekanen aangewezen, op dinsdag 24 september 1996 te 13.30 uur

door

Eric Adrianus Fransiscus KRAAN,

wiskundig ingenieur, geboren te Gouda.
Dit proefschrift is goedgekeurd door de promotor:
Prof. dr ir J.J.M. Evers

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof. dr ir J.J.M. Evers
Prof. dr ir R. Dekker
Prof. dr ir R.E.C.M. van der Heijden
Prof. dr ir J. van Katwijk
Prof. drs C.J. Ruijgrok
Prof. ir H. Bikker

Technische Universiteit Delft, promotor
Erasmus Universiteit Rotterdam
Technische Universiteit Delft
Technische Universiteit Delft
Katholieke Universiteit Brabant
Technische Universiteit Delft

CIP-DATA Koninklijke Bibliotheek Den Haag

Kraan, Eric A.F.

Design of the Activity / Constraint - Object Modeller: A development environment for (M1)LP-based Decision Support Systems /
Eric A.F. Kraan. - Delft : Delft University of Technology, Faculty of Mechanical Engineering and Marine Technology, - Ill.
Thesis Technische Universiteit Delft. - With summary in Dutch
ISBN 90-370-0144-0
Subject headings: modelling / mathematical programming / optimisation / graph-based modelling / decision support systems.

Copyright © Eric A.F. Kraan, 1996

All rights reserved. No part of this thesis may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

The author makes no warranty that the methods, calculations, and data in this book are free from errors. The application of the methods and the results is at the user's own risk and the author disclaims all liability for damages, whether direct, incidental, or consequential, arising from such applications or from any other use of this book.
Nihil est in intellectu quod non prius fruit in sensu.

Comenius (J.A. Komenský)

Spicilegium didacticum, Mathematica 11, Amsterdam 1680
PREFACE

This research project was carried out at the Transportation Technology group (Logistics Engineering) in the Department of Mechanical Engineering and Marine Technology of the Delft University of Technology during the period September 1991 till March 1996 under supervision of Professor Joseph J.M. Evers.

Many thanks are due to all the members of the scientific staff, the post graduate students and other employees of the Transportation Technology group who have supported me during the project and the preparation of my thesis.

I would like to address a special thanks to Marcel Mourits who has been a close colleague and an ideal sparring partner for nearly three years and who has remained a dear friend since our professional careers parted. Marcel, thank you for sharing your thoughts and for lending your ear.

Above all, I wish to thank Katleen, my parents, Susanne and Gerard for everything they have done for me. Their endless enthusiasm, patience, support and encouragement throughout the entire period without which I would never have been able to finish this thesis.

Eric Kraan

Waddinxveen, The Netherlands

July 1996
SUMMARY

Manufacturing systems face major challenges: buyer's markets and intensified competition are forcing the improvement of their performance and efficiency and more frequent and rapid adaptation to changes in their environment. The arrangement of the manufacturing system is a complex task, the more so because the gradually decreasing cost of transportation, the rise of telematics, and other phenomena have stimulated companies to view the entire world as the playing field on which to establish their production systems.

Many companies recognise the importance of using quantitative analysis to substantiate their strategic manufacturing decisions. (Mixed Integer) Linear Programming — (MI)LP — models have proven well suited to provide quantitative support in this area. The literature contains a vast amount of (MI)LP models for strategic manufacturing planning: facility location models, (facility-product-customer) allocation models, models that focus on the timing and sizing of capacity investments, and models for evaluating manufacturing technologies.

In the last decade, facility location and allocation models have been applied successfully in Decision Support Systems (DSS). Despite the many advantages of these (MI)LP-based DSSs, the full potential of (MI)LP models does not become available to the user of these systems due to their limited flexibility (the user cannot change the structure of the underlying models) and focus on facility location and allocation decisions. The strength of (MI)LP-modelling systems is that these systems offer a great deal of flexibility. Algebraic-oriented modelling languages, however, are non intuitive. (MI)LP-modelling systems which offer an intuitive model representation, on the other hand, either support only a limited range of models or are not capable of representing all relationships between the components of a model. Some (MI)LP-modelling systems do not contain a model-independent database subsystem which is generally considered an indispensable part of any DSS. None of the (MI)LP-modelling systems we encountered contains a model-independent data dictionary which enables users to model, maintain, and view the structural relationships between the elements of the database in agreement with the logical structure of the data. Finally, current (MI)LP-modelling systems typically offer the user an interface to the data that is in keeping with the relational data model which major modern database systems use as an external data model. The relational data model, however, does not support the notion of conceptual modularity: logically related data are scattered across the database in seemingly unrelated tables. Yet, the support for conceptual modularity is an essential
property of (MI)LP-modelling systems which focus primarily on inexperienced model builders for it reduces cognitive complexity.

The objective of this research project is to develop a prototype DSS which serves as an environment for conducting experiments with tailored models for strategic and tactical production planning. Users must be able to configure the system to fit their own production system by selecting model components from a library of 'building blocks'. These building blocks—termed 'objects'—must show enough resemblance with real-world objects to encourage inexperienced model builders, including the majority of decision makers, to use the system themselves for decision making and analytical purposes.

This thesis describes the design and implementation of the Activity/Constraint-Object Modeller (ACOM). The design may be viewed as an attempt to obtain a synergy between the flexibility offered by general-purpose (MI)LP-modelling systems and the user-friendliness and decision-support functionality offered by (MI)LP-based DSSs. For that reason, ACOM integrates an intuitive, graphical, and generally applicable (MI)LP-modelling facility, a data-modelling facility, and an intuitive model-data linking facility that integrates both. Although the applicability of ACOM is not limited to strategic and tactical production planning, these fields of application are used as a frame of reference throughout the thesis. Conceptual modularity played a key role in the design of both the (MI)LP-modelling scheme and the data-modelling scheme. Due to this approach, the logical structure of the problem domain is made explicit in both the organisation of the models and the database as a result of which the modular arrangement of the models and the database may be broadly parallel. This similarity makes it possible to integrate the representation of a modelled entity in terms of logically related model activities and an accompanying data abstraction in an intuitive way — an AC(Activity/Constraint)-object. The integration of a functional abstraction ('behaviour') and a data abstraction ('state-space') corresponds to the notion of objects which is becoming increasingly popular as a building block for the design of complex systems.

Two variants of the algebraic notation that is customarily used to denote (MI)LP models are presented. The referential and ordered-referential notation constitute the formal basis of the (MI)LP-modelling scheme that is contained in ACOM. Both variants, unlike the algebraic notation, comply with the gradual development of models and model instances when using ACOM.

Three case studies are presented. The models deal with strategic logistics decisions and have been used as the basis of prototype planning support systems. It was found that
activities and constraints which act upon flows of goods are represented most naturally: the formulation of such constraints requires very little effort. A simple procedure facilitates the formulation and interpretation of logical constraints to which the analogy with physical flows does not apply. We learned when using ACOOM, it is intuitively clear whether to use a top-down or a bottom-up formulation strategy, or a mixture of both: this greatly depends on the structure and the complexity of the intended model. It appeared that the modular arrangement of models that deal with logistics chains which consist of different types of actors coincides with the connections between the actors that exist in the problem domain.

In all three case studies, it required little effort to create a data model. Primarily, this is due to the fact that, for aggregate levels, the (MI)LP-modelling scheme supports an ‘actor-oriented’ representation of the problem. The same high-level representation may be used to structure the data model the result of which is that the modular arrangement of the (MI)LP model and the data model broadly parallel. The clustering of related data makes it intuitively clear where to locate the data.

First responses to brief demonstrations with earlier prototypes of ACOM supported the view that ACOM provides an intuitively appealing representation of many models which may be used directly as a basic means of communication between experienced model builders (analysts) and decision makers. ACOM may be used to short-cut this dialogue, possibly using rapid prototyping, thus contributing to these decision makers’ understanding and confidence. ACOM may encourage inexperienced model builders to attempt to use (MI)LP models for decision making and analytical purposes. Finally, we expect that ACOM may prove a valuable tool for instruction in linear programming.
SAMENVATTING

Produktiesystemen worden geconfronteerd met een omgeving die in snel tempo verandert en voortdurend hogere eisen stelt. Door intensieve concurrentie en de noodzaak om in te spelen op de wensen van klanten worden produktiesystemen gedwongen om hun prestaties en efficiency te verbeteren en zich meer frequent en in kortere tijd aan te passen aan veranderingen in de omgeving. Het vinden van een optimale inrichting van een produktiesysteem is een complex vraagstuk, te meer omdat door bijvoorbeeld de geleidelijke daling van de kosten voor transport en de opkomst van telematica, bedrijven worden gestimuleerd de gehele wereld als speelveld te beschouwen voor de inrichting van hun produktiesystemen.

Veel bedrijven erkennen het belang van kwantitatieve analyses in de voorbereiding van strategische en tactische produktieplanningsbeslissingen. Mixed-Integer (gemengdgeheeltallige) en Lineaire Programmerings — (MI)LP — modellen hebben bewezen zeer geschikt te zijn als basis voor de kwantitatieve ondersteuning van dit type beslissingen. In de literatuur is een groot aantal publicaties verschenen met (MI)LP modellen voor strategische produktieplanning: locatiemodellen, allocatiemodellen, modellen die zich richten op de timing en omvang van investeringsbeslissingen en modellen voor het evalueren van de inzet van verschillende technologieën.

In de afgelopen 10 jaar zijn locatie- en allocatiemodellen met succes toegepast in beslissings-ondersteunende systemen (Decisions Support Systems — DSS). Tegenover de vele voordelen van deze systemen staat de beperkte flexibiliteit (de gebruiker kan de structuur van de onderliggende modellen niet wijzigen) en de beperking van de ondersteuning tot locatie- en allocatiebeslissingen, waardoor het grote potentieel van (MI)LP modellen slechts in beperkte mate ter beschikking komt van de gebruiker. De kracht van modelleersystemen voor (MI)LP modellen ligt juist in de grote mate van flexibiliteit. De algebraisch-georiënteerde representatie van modellen die in veel modelleersystemen wordt gehanteerd is echter niet intuïtief en vraagt om een behoorlijke achtergrond op het gebied van het formuleren van (MI)LP modellen. Systemen die gebruik maken van een intuitieve model-representatie, ondersteunen daarentegen slechts een beperkte verzameling van modellen of zijn niet in staat alle relaties tussen de elementen van een model expliciet te tonen. Sommige modelleersystemen bevatten geen database subsysteem dat algemeen wordt gezien als een essentieel onderdeel van een DSS. Geen van de modelleersystemen die we zijn tegengekomen bevat een data-dictionary subsysteem. Een data-dictionary subsysteem biedt gebruikers de mogelijkheid om logische relaties tussen de gegevens binnen de
database te modelleren en te onderhouden. Tenslotte maken de huidige modelleersystemen gebruik van het relationele datamodel om gebruikers toegang te verschaffen tot de database. Ofschoon het relationele model is uitgegroeid tot de standaard voor externe data-modellen van database systemen, is een belangrijk nadeel van dit model dat het geen ondersteuning biedt voor conceptuele modulariteit: logisch samenhangende gegevens liggen verspreid opgeslagen in de database in ogenzienlijk ongerelateerde tabellen. Conceptuele modulariteit is een essentiële eigenschap van modelleersystemen die zich richten op de onervaren modelbouwer, omdat het bijdraagt aan het verminderen van de cognitieve complexiteit van het formuleren van modellen.

De doelstelling van dit onderzoeksproject is de ontwikkeling van een prototype DSS dat gebruikt kan worden als experimenteeromgeving voor op maat ontworpen strategische en tactische produktieplanningsmodellen. De toekomstige gebruikers van dit systeem moeten in staat zijn het systeem te configureren door gebruik te maken van een bibliotheek van ‘bouwstenen’. Deze bouwstenen, ‘objecten’ genoemd, moeten voldoende tastbaar zijn om onervaren modelbouwers, waaronder de meerderheid van beslissers, te stimuleren het systeem te gebruiken ter ondersteuning van beslissingen en voor analytische doeleinden.

Dit proefschrift beschrijft het ontwerp en de implementatie van ACOM: Activity / Constraint-Object Modeller. Het ontwerp kan gezien worden als een poging een synergie te bereiken tussen de flexibiliteit van modelleersystemen voor (MI)LP modellen enerzijds en de gebruikersvriendelijkheid en de op beslissingsondersteuning gerichte functionaliteit van op (MI)LP modellen gebaseerde DSSs anderzijds. ACOM integreert voor dit doel een intuitieve en algemeen toepasbare grafische modelleertaal voor (MI)LP modellen, een data-modelleerfacilité, en een intuitieve koppelingsmechanisme dat beide modelleerfacilitelen integreert. Het begrip modulariteit heeft een sleutelrol gespeeld in het ontwerp van zowel het modelleerformalisme voor (MI)LP modellen en het data-modelleerformalisme. Door deze ontwerp-aanpak kan de logisch-modulaire structuur van het probleem domein expliciet zichtbaar worden gemaakt in de organisatie van de modellen en de database. Als gevolg hiervan komt de modulaire organisatie van de modellen en de database grotendeels overeen waardoor het mogelijk wordt de representatie van een entiteit in termen van logisch-gerelateerde modelactiviteiten op een intuitieve wijze te integreren met een bijbehorende data abstractie in een AC(Activity/Constraint)-object. De integratie van een functioneleabstractie (‘gedrag’) en een data-abstractie (‘toestandruimte’), correspondeert met de notie van een object dat in toenemende mate aan populariteit wint als een bouwsteen voor het ontwerp van complexe systemen. Ofschoon de genericiteit van het ontwerp van ACOM de toepasbaarheid van het systeem niet beperkt tot modellen voor
strategische en tactische produktieplanning zijn beide toepassingsgebieden binnen dit proefschrift gebruikt als referentiekader.

In dit proefschrift worden twee varianten geïntroduceerd op de algebraïsche notatie die gewoonlijk wordt gebruikt voor het weergeven van (MI)LP modellen. De referentiële en de geordende-referentiële notatie vormen de formele basis van het binnen ACOM gehanteerde modelleringsformalisme. In afwijking van de algebraïsche notatie sluiten beide varianten nauw aan bij de geleidelijke ontwikkeling van modellen en model-instanties bij het gebruik van ACOM.

In dit proefschrift worden drie case-studies gepresenteerd. De gebruikte modellen hebben betrekking op logistieke beslissingen van strategische aard en zijn ieder afzonderlijk gebruikt als basis van een prototype DSS. Uit deze studies is gebleken dat activiteiten en beperkingen die samenhangen met goederenstromen op een natuurlijke wijze gemodelleerd kunnen worden en dat bovendien het formuleren ervan, gebruik makend van de intuïtieve representatie, zeer efficiënt verloopt. Voor het formuleren van logische 'als-dan' condities waarop de analogie met fysieke stromen niet van toepassing is blijkt een eenvoudige procedure te bestaan die het formuleren en interpreteren ervan aanzienlijk vereenvoudigt. De grafische model-representatie blijkt het aanleren van mechanismen die behulpzaam zijn bij het formuleren van modellen aanzienlijk te verkorten. Uit onze ervaringen blijkt verder dat in het gebruik van ACOM het intuïtief duidelijk is wanneer er gekozen moet worden voor een top-down, een bottom-up, of een mix van beide formuleringstrategieën. Dit blijkt in hoofdzaak af te hangen van de structuur en de complexiteit van het model dat de bouwer voor ogen heeft. Met name in het geval van modellen die zich richten op activiteiten binnen een logistieke keten van verschillende typen actoren, bleek de modulaire opbouw van de modellen samen te vallen met de structuur van het probleemdomein.

In alle drie case-studies bleek het opzetten van een data model zeer eenvoudig te zijn. Dit is primair het gevolg van het feit dat het modellerformalisme op geaggregeerd niveau bijdraagt aan een actork-oriënteerde decompositie van het probleemveld. Hierdoor kan de modulaire structuur van de modellen voor een belangrijk deel worden gebruikt voor de inrichting van de database, met als gevolg dat de organisatie van de modellen en de database grotendeels overeenkomt. De clustering van logisch samenhangende gegevens maakt het intuïtief duidelijk waar bepaalde gegevens binnen het datamodel te vinden moeten zijn. Uit onze ervaringen blijkt dat de gekozen opzet de 'span of control' van de gebruiker over het systeem vergroot.

De eerste reacties op demonstraties met eerdere en het huidige prototype van ACOM ondersteunen de hypothese dat het systeem een intuïtief aantrekkelijke representatie
biedt van veel modellen en dat deze model-representatie gebruikt kan worden als interactie-medium met onervaren modelbouwers en op deze manier kan bijdragen aan het verbeteren van de dialoog tussen ervaren modelbouwers (analisten) en beslissers. ACOM kan gebruikt worden om een brugfunctie te vervullen, mogelijk door het gebruik van ‘rapid prototyping’. Het is de verwachting dat een dergelijk gebruik van het systeem bijdraagt aan het vertrouwen en het begrip van de beslisser. ACOM kan onervaren modelbouwers stimuleren gebruik te maken van (MI)LP modellen voor de ondersteuning van beslissingen en voor analytische doeleinden. Tenslotte, verwachten we dat ACOM een bruikbaar instrument kan zijn voor het onderwijs op het gebied van gemengd-geheeltallige en lineaire programmering.
CONTENTS

Part I - Introduction

1 Introduction 1

2 Strategic and tactical manufacturing planning 27

3 Survey of existing tools for (MI)LP-based decision support in production planning 61

4 Outline of the Activity/Constraint - Object Modeller 115

Part II - Conceptual Design

5 Design of the (MI)LP-modelling scheme 129

6 Design of the data model 149

7 Design of the data-model link 189

Part III - Implementation & Validation

8 Design and implementation of the prototype 209

9 Case studies and evaluation 255

Part IV - Conclusions

10 Conclusions and recommendations 293

References 309

Appendices 315

Curriculum Vitae 343
Part I - Introduction

1 Introduction ................................................................................................................... 1

1.1 Problem statement and objective ............................................................................. 2

1.2 Introduction to manufacturing planning .................................................................... 3

  1.2.1 Production ........................................................................................................... 3

  1.2.2 Production and the logistics chain ..................................................................... 5

  1.2.3 Integral management of the logistics chain ......................................................... 6

  1.2.4 Manufacturing planning and control .................................................................. 9

1.3 Manufacturing's potential to obtain competitive advantage ............................. 13

1.4 Managerial approaches to deal with increased manufacturing complexity .......... 15

1.5 Application of optimisation technology to manufacturing planning .................. 19

1.6 (MI)LP-based decision support for manufacturing planning ......................... 20

  1.6.1 (MI)LP-based decision support systems .......................................................... 21

  1.6.2 (MI)LP-modelling systems ............................................................................ 21

  1.6.3 Customised intuitive (MI)LP-based decision support systems ................. 23

1.7 Outline of the proposed system ............................................................................. 24

1.8 Structure of this thesis ............................................................................................ 25

2 Strategic and tactical manufacturing planning ...................................................... 27

2.1 Strategic manufacturing planning .......................................................................... 27

  2.1.1 Corporate, business, and manufacturing strategy ................................................. 29

    2.1.1.1 The components of a corporate strategy ...................................................... 29

    2.1.1.2 The components of a business strategy ....................................................... 30

  2.1.2 The components of a business strategy ............................................................. 33

  2.1.3 The components of a manufacturing strategy ................................................... 34

    2.1.3.1 Examples of multi-plant manufacturing facilities location strategies .......... 36

    2.1.4 (MI)LP-models supporting the development of a manufacturing
strategy ........................................................................................................ 42
2.1.4.1 Facility location models ......................................................... 42
2.1.4.2 Allocation models ................................................................. 46
2.1.4.3 Models for capacity acquisition ........................................... 47
2.1.4.4 Technology selection models ............................................. 50
2.2 Tactical manufacturing planning ..................................................... 51
    2.2.1 Structuring the aggregate manufacturing planning process .... 52
    2.2.2 Aggregate planning decisions ............................................. 53
    2.2.3 Disaggregation .................................................................... 54
    2.2.4 (MI)LP-models for aggregate manufacturing planning ........ 55
        2.2.4.1 Monolithic models ....................................................... 56
        2.2.4.2 Hierarchical production planning ............................... 56
2.3 Using computationally intractable models for decision support .. 58

3 Survey of existing tools for (MI)LP-based decision
   support in production planning ......................................................... 61
3.1 Introduction .................................................................................... 62
3.2 A reference architecture ............................................................. 63
    3.2.1 The user interface ............................................................... 66
    3.2.2 The model processor and the (MI)LP-solver ..................... 66
    3.2.3 The solution analyser .......................................................... 68
    3.2.4 The database management subsystem ................................ 69
    3.2.5 The model management subsystem .................................... 74
3.3 (MI)LP-based decision support systems ...................................... 74
    3.3.1 Justification ....................................................................... 75
    3.3.2 Functional analysis of using (MI)LP-based DSSs ................. 76
    3.3.3 Criteria for comparison ....................................................... 77
    3.3.4 DP-PSS: Distributed Production Planning Support System .... 81
    3.3.5 PHYDIAS ........................................................................ 83
    3.3.6 SLAM ............................................................................. 86
3.4 (MI)LP-modelling systems ......................................................... 88
3.4.1 Justification ................................................................. 88
3.4.2 The process of developing an (MI)LP model ....................... 91
3.4.3 A typical model structure ............................................. 92
3.4.4 Functional analysis of using (MI)LP-modelling systems ........... 93
3.4.5 Criteria for comparison ................................................ 98
3.4.6 Introduction of a reference model .................................. 100
3.4.7 AMPL ........................................................................ 102
3.4.8 Activity-Constraint graphs .......................................... 104
3.4.9 gLPS ........................................................................ 106
3.4.10 LOGS ...................................................................... 108
3.4.11 LPFORM ................................................................. 111

4 Outline of the Activity/Constraint-Object Modeller ____ 115

4.1 Introduction ..................................................................... 116
4.2 Outline of the functional architecture ................................. 116
  4.2.1 Outline of the (MI)LP-modelling scheme ......................... 119
  4.2.2 Outline of the data-modelling scheme ............................ 120
  4.2.3 Outline of the model-data link: AC-objects ...................... 126
  4.2.4 Overview ................................................................... 128

Part II - Conceptual design

5 Design of the (MI)LP-modelling scheme ................................. 129

5.1 Introduction ..................................................................... 130
5.2 Formal notation .................................................................. 130
  5.2.1 A non-structured generalised algebraic representation of an
         LP model .................................................................... 130
  5.2.1.1 Index-symbol expressions ....................................... 133
  5.2.2 The structured form ................................................... 136

5.3 Compacted Activity/Constraint graphs ................................. 137
  5.3.1 Syntax for representing constraints and activity bounds ...... 137
Part III - Implementation & Validation

8 Design and implementation of the prototype ........................................ 209
  8.1 Introduction .................................................................................. 210
  8.2 Tools for implementation .............................................................. 212
  8.3 The syntax of class diagrams and object diagrams ....................... 213
  8.4 General design topics .................................................................... 216
    8.4.1 A base class for run-time type checking .................................. 216
    8.4.2 Container classes ..................................................................... 217
  8.5 The model management subsystem ................................................. 218
    8.5.1 Object-oriented decomposition of the modelling scheme .......... 219
    8.5.2 Object-oriented decomposition of the implementation scheme 221
    8.5.3 The interface between the MS and the IS: Model instances ....... 229
    8.5.4 MPS files ............................................................................... 230
  8.6 The database management subsystem ............................................. 232
  8.7 The interface between the models and the data ................................ 236
  8.8 The model processor subsystem ..................................................... 237
    8.8.1 Outline of the model construction algorithm ......................... 237
    8.8.2 The interface between the model processor and external (MI)LPS solvers ......................................................... 237
  8.9 The user interface .......................................................................... 239
    8.9.1 User interfaces and object orientation ..................................... 240
    8.9.2 Design of the object-action mechanism .................................... 241
    8.9.3 The working environment ....................................................... 243
    8.9.4 Container objects .................................................................... 245
    8.9.5 User-interface design for compacted AC-graphs ...................... 247
    8.9.6 User-interface design for the data-modelling scheme ............... 252

9 Case studies and evaluation ................................................................. 255
  9.1 Introduction .................................................................................. 256
  9.2 Case 1: The reference model .......................................................... 259
<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3 Case 2: The coal-chain model .................. 264</td>
</tr>
<tr>
<td>9.3.1 Introduction .................................. 264</td>
</tr>
<tr>
<td>9.3.2 Block-schematic representation ............... 265</td>
</tr>
<tr>
<td>9.3.3 AC-graph representation ....................... 269</td>
</tr>
<tr>
<td>9.3.4 Data Model ................................... 271</td>
</tr>
<tr>
<td>9.4 Case 3: The deployment model ................... 272</td>
</tr>
<tr>
<td>9.4.1 Introduction .................................. 272</td>
</tr>
<tr>
<td>9.4.2 Block-schematic representation ............... 273</td>
</tr>
<tr>
<td>9.4.3 AC-graph representation ....................... 282</td>
</tr>
<tr>
<td>9.4.4 Data Model ................................... 284</td>
</tr>
<tr>
<td>9.5 Evaluation ...................................... 285</td>
</tr>
<tr>
<td>9.5.1 Evaluation of the (MI)LP-modelling scheme .... 285</td>
</tr>
<tr>
<td>9.5.2 Evaluation of the data-modelling scheme ....... 288</td>
</tr>
<tr>
<td>9.5.3 Evaluation of the current implementation of ACOM ... 289</td>
</tr>
</tbody>
</table>

**Part IV - Conclusions**

| 10 Conclusions and recommendations .................. 293 |
| 10.1 Main points and conclusions ..................... 294 |
| 10.2 Topics for further research ..................... 303 |

| References ............................................. 307 |

<table>
<thead>
<tr>
<th>Appendices ............................................. 315</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A: (Mixed Integer) Linear Programming .... 315</td>
</tr>
<tr>
<td>Appendix B: The relational and object-oriented data model ... 329</td>
</tr>
</tbody>
</table>

| Curriculum Vitae ..................................... 343 |

xvii
PART ONE

Introduction
1. INTRODUCTION

OUTLINE
Until the begin eighties, little attention had been paid to strategic manufacturing planning. Strategic management had been focused almost entirely on marketing and finance. Increased competition, however, is now forcing the improvement of the performance of the manufacturing system (shorter lead-times, higher quality) and more frequent and rapid adaptation to changes in its environment. As a result, companies have come to realise that strategic decisions concerning the manufacturing system can have a decisive impact on their competitive strength.

The complexity of manufacturing planning has increased significantly in the last few decades. A number of different approaches have been developed in order to deal with this increased complexity. From the field of operations research, mixed-integer programming has proven to be a particularly suitable technology to provide quantitative support to both strategic and tactical manufacturing planning. During the last decade, MIP-models have been successfully applied in decision support systems. In this chapter, however, it is argued that the applicability of these systems is limited. The limited applicability of existing decision support systems led to the problem statement of this thesis which will be briefly introduced in this chapter, together with the proposed approach. (Chapter 4 will elaborate on these two issues in more detail.) The last section of this chapter is used to outline the structure of the thesis.
1.1 Problem statement and objective

Mixed-Integer Programming and Linear Programming — (MI)LP — constitute two categories of optimisation techniques which are particularly suitable to provide quantitative support to the arrangement of the production system and the deployment of capacity resources, as well as the aggregate planning of materials and products flows. Despite the many advantages of currently available (MI)LP-based Decision Support Systems (DSS) in this area, the functionality that these systems offer does not cover the full potential of the underlying technology.

An important disadvantage of existing DSSs is that the user cannot change the structure of the underlying models. As a result, the applicability of existing DSSs is restricted to those situations in which the decision maker feels that the trade-offs and system characteristics that are fundamental to the problem are sufficiently reflected in the underlying model. There is great diversity in the models described in the literature so it is most likely that a decision to use a readily available DSS is governed by the huge cost and the risk associated with the development of a dedicated DSS.

Another drawback of many of today’s (MI)LP-based DSSs is their partial support of the production planning process, in most cases using a single model. A series of models would enable the decision maker to cover a broader range of production planning issues and to evaluate dependent decisions which cannot be solved by means of a single model owing to computational or cognitive limitations.

The limited flexibility and the partial approach to the broad range of production planning decisions typical of (MI)LP-based DSSs for production planning led to the initiation of this project which focuses on the development of a prototype production planning support system that:

- Enables the user to configure the system with a tailored production system by means of a library of ‘building blocks’;
- Which serves as an environment for conducting experiments with tailored (MI)LP-models for manufacturing planning.

The building blocks—termed ‘objects’—must show enough resemblance to real-world objects to allow decision makers to use the system themselves.

Before outlining our research approach (Section 1.7) and the structure of this thesis (Section 1.8) the intermezzo briefly introduces the concepts and terms used in manufacturing planning and control (Section 1.2), two developments of the last decade
that contributed to a growing understanding of manufacturing’s potential to obtain competitive advantage (Section 1.3) and, amongst other approaches, the role of optimisation techniques in supporting a company’s manufacturing planning function (sections 1.4-1.5). Prior to the introduction of our research approach, Section 1.6 discusses the strengths and weaknesses of existing tools for incorporating (MI)LP-technology in the decision-making processes.

1.2 Introduction to manufacturing planning

An investigation of the literature on the subjects of both manufacturing and manufacturing planning indicated that the authors use a large variety of terms and concepts. For this reason, the following sections introduce the area of manufacturing planning and provide an explanation of the terminology.

1.2.1 Production

Usually production is defined as the process of converting materials into finished products. The terms ‘production’ and ‘manufacturing’ are often used as synonyms. Production, however, covers not only manufacturing — the actual making of goods — but also includes preparative activities like engineering, process planning, and manufacturing planning. Production is one of the central business activities that are linked to all main functional areas of the company: marketing/sales, purchasing, distribution, and co-ordinating functions as corporate management and finance. Figure 1-1 depicts some illustrative linkages.

Figure 1-1 Stereotype interactions between production and other central business activities.
PART I - INTRODUCTION

In this thesis, the term production will be used in the context of a company repetitively producing series of quite similar products. This type of production environment is relatively stable over a long period of time. Often, because of this stability, it will be profitable or even necessary to tailor the manufacturing system to the specific characteristics of the suppliers, the resources that it obtains from these suppliers, the products that are being manufactured, and to the customers that are being served. For example:

- **SUPPLIERS** If the transportation of raw materials is more costly than the transportation of intermediate-stage products, a company may decide to locate some of its manufacturing activities close to its suppliers. A company may also negotiate with suppliers on long-term delivery contracts concerning, for example, prices, delivery-time requirements, or changes in raw material inventory capacity. The term resources refers to both non-renewable items such as raw materials and renewable items such as tools, product designs, and machines [Biemans, 1990];

- **PRODUCTS** If the products have similar manufacturing characteristics, a company may decide to invest in dedicated manufacturing equipment. This may be profitable due to economies of scale;

- **CUSTOMERS** A company may decide to locate part of its manufacturing process close to its customers. Possible considerations in this respect are the reduction of transportation costs and delivery lead times, goodwill and service.

It is clear that this stereotyped production environment provides numerous options for the arrangement the production system. Using the series-production environment as a reference system is not intended to constrain the scope of this thesis but rather to serve as a vehicle for a discussion of the entire range of strategic and tactical production planning decisions. In specific situations, even if the system under consideration falls into the category of series production, certain decisions concerning a stereotype series-production environment may be less applicable or even irrelevant. For example, in a series-production company, the decision on where to locate a new manufacturing facility is usually partly based on a trade off between the distance between suppliers, manufacturing facilities, customer regions and cost differences (e.g. labour costs) between optional sites. Other considerations that may govern the decision are, for example:

- For a company building highly specialised highly customised equipment on an engineering to order basis, the distance between the manufacturing plant and the customer is usually irrelevant. The customer’s decision to select a manufacturer will be largely based the manufacturer’s technological ability and image. The cost of transportation comes second to these;
• Technological constraints and considerations that relate to the infrastructure may also dominate the selection of optional sites for the location of new facilities. Aircraft manufacturers must have access to airfields. Electrical power plants require access to large amounts of cooling water;

• The Hewlett Packard Company [Hayes and Wheelwright, 1984] selects sites which have surrounding areas, climatic conditions and other features that are similar to those of its existing plant in California. This plant is considered to have an ideal location and is used to test changes to the manufacturing system. When establishing a new facility, HP copies its Californian plant including the latest approved changes.

1.2.2 Production and the logistics chain

A shortcoming of the above given definition of production is that it suggests that the products that are being produced remain under the control of production management until their completion. This, however, depends largely on the arrangement of the logistics chain: "(...) the total flow of goods, from acquisition of raw materials to delivery of finished goods to the customer." [Gelders et al., 1981]. In some cases it may be profitable to carry out manufacturing steps which require relatively few and easy-to-manage pieces of equipment in distribution centers located closer to the customer. Hewlett Packard, for example, decided to relocate the ‘localisation’ steps of adding dedicated power supply units, power cord and plug, and manuals which differ per country of sale, from its manufacturing facilities to its distribution centres [Davis, 1993]. Such an arrangement of the production and distribution system, sometimes referred to as product postponement, offers a number of advantages which may stimulate a company to act in this way:

• The manufacturing system may be tailored to the production of high volumes of similar products without having to deal with a final assembly schedule of small series of printers with specific power supply unit, plug, and manual;

• The subassemblies may be shipped in larger volumes which is more efficient and reduces costs;

• Owing to product postponement, inventories may be relocated to a different country of sale as the generic product does not contain specific options;

• A company might expect to increase the quality standards of its products by using local knowledge and skills;

• A company might expect to receive goodwill and acceptance as a result of using local labour forces.
PART I - INTRODUCTION

In other cases, manufacturing plants sell their finished products directly to their customers without using intermediate distribution centres. This situation occurs, for example, in the European automotive industry when serving its native market. A declining market forced the companies to reduce their costs. As a result of using both advanced production technology and production-control methods which reduced delivery lead time and permitted a 'customer-order driven' type of production it was no longer necessary to maintain large and expensive inventories. The first example shows that in some cases the entire production process extends over both the production system and the distribution system. The second example shows that in other cases even the distribution to final customers is accounted for by the production system.

1.2.3 Integral management of the logistics chain

Despite the fact that both in the literature and in practice a general consensus exists that an integral approach to the management of the logistics chain offers the best opportunities to optimise its overall performance, the distinction between the production system and the distribution system still remains an important issue. Integral management of the logistics chain focuses on the interactions between procurement, production, and distribution instead of viewing each subsystem separately. The interdependencies encountered when deciding upon the design of the logistics chain include:

- The locations of new manufacturing plants or distribution facilities. If transportation costs play a dominant role, decisions on the locations of new plants depend on the spatial distribution of vendors of raw materials, existing plants, and distribution centres. Likewise, the location of distribution facilities is partly determined by the spatial distribution of manufacturing plants;

- The timing and sizing of goods flows between vendors of raw materials, manufacturing plants, and distribution centres which affects decisions on inventory and transport capacities in the entire chain depends on mutual supply contracts and on order policies.

Assuming an integral approach towards the management of the logistics chain does not, however, imply that the only way to provide effective quantitative support to decisions affecting the logistics chain is to make exclusive use of models that deal with each subsystem simultaneously. The use of models that contain an integral description of the logistics chain closely parallels this managerial attitude, yet two pragmatic considerations which relate directly to the inherent complexity of the underlying decisions may preclude such an approach.
The first consideration is that the use of models for practical purposes is restricted by the available computer and algorithmic power. In order to perform iterative analysis using these models, turnaround times less than several hours — but preferably shorter — are essential. Not surprisingly, optimisation models which focus on the complex interactions of procurement, production, and distribution activities depart from existing spatial distributions of both manufacturing and distribution facilities. Mirchandani et al. (1990) have categorised facility location problems into four prototype formulations containing a basic description of the decision governing the establishment of facilities. The time-complexity function of an optimisation problem expresses the greatest amount of time required to solve the problem for an arbitrary problem instance of a given size. Mirchandani and Francis show that there is no polynomial function of the size of the problem which provides an upper bound of the time-complexity function of each of the four prototype formulations. The complexity of the algorithms used to solve these problems increases exponentially as the problem size increases. Such optimisation problems are referred to as Non-Polynomial (NP) hard. Large facility location models containing an adequate description of the underlying problem are therefore very difficult to solve in a reasonable time and usually require dedicated solution algorithms like Erlenkotter’s (1978) algorithm to solve plant location problems assuming unlimited capacity and Van Roy’s (1986) algorithm to solve the capacitated plant location model. Models like that of Cohen, Fisher, and Jaikumar (1989), which consider, amongst other things, the monetary aspects related to cross-border product flows, therefore usually assume that the location and capacity of the manufacturing facilities are fixed. Separate treatment of production and distribution both reduces the complexity and offers an opportunity to use the computational capacity thus gained for other purposes:

- The decision not to use an integral model reduces demands on computational power which might thus be available to cover a broader range of related decisions on a distinct subsystem into a single model. Although such an approach is suboptimal by definition, in practice it is common to adopt a solution procedure in which separate models are used to support a shared set of interrelated decisions. For example, production planning is typically carried out sequentially. First an attempt is made to obtain a feasible production plan, stated in terms of product families which consist of products sharing certain manufacturing characteristics. In subsequent stages, this aggregate plan is broken down until — finally — feasible production plans for components are obtained. If, during this process, an unfeasibility is encountered, a feedback mechanism, a co-ordinating scheme, is put into work which requires that this unfeasibility is resolved in previous higher-level stages.

Several researchers, including Biemans (1990), argue that, in practice, an optimal solution is not necessarily the only solution that will satisfy a company. Biemans
argues that optimal solutions frequently require excessive computational efforts which result in high costs. If the costs associated with the efforts to obtain an optimal solution play a part in the decision on which solution is optimal, a non-optimal solution may suit the firm’s objectives equally well;

- A second alternative which may be considered, is to apply the available computational capacity to try to solve less ambitious optimisation problems with general-purpose solution algorithms. The use of general-purpose solution algorithms has a number of advantages. General-purpose solvers are generally and readily available. Often companies already have a general-purpose solver in operation which facilitates the integration of a newly developed DSS. Implementing a dedicated solution procedure requires expert knowledge and will thus be more expensive. Finally, a general-purpose solver is more flexible as these systems are not restricted to a specific range of optimisation models.

The second consideration which may lead to the decision not to use an integral model of the logistics chain is that, from a cognitive standpoint, large models make it difficult to track cause and effect so they are less suited for decision making purposes.

The problem of how to separate the production system from the distribution system then arises. This problem is particularly important because, to some extent, the two subsystems have conflicting objectives. The task of manufacturing is to meet agreed-upon production plans with a maximum degree of efficiency. In general, long production runs provide an excellent way to maximise economies of scale, which clearly contributes to the efficiency of the manufacturing system. From the standpoint of a distribution manager, however, on-time delivery of the right quantities to the right places in the most efficient ways is the main target. Small inventories, and thus small replenishment batches, generally reduce the overall costs of the distribution system. Therefore, the demarcation line between the production system and the distribution system determines the main objective that will guide the arrangement, the planning and the control of the activities.

1.2.4 Manufacturing planning and control

The term Manufacturing Planning and Control System (MPCS) is frequently used to refer to the part of a production system that is responsible for both the planning and control of manufacturing processes in order to meet agreed-upon production targets. Within the context of an MPCS equally important managerial responsibilities with respect to, for example, personnel, finance, and safety, are ignored.
Vollmann, Berry and Whybark (1988) provide a functional framework of an MPCS (cf. Figure 1-2). This framework depicts the tasks a stereotype MPCS must perform in order to obtain feasible plans for the entire goods flow. The framework consists of three different planning layers which link aggregate planning levels with detailed manufacturing schedules. The front end is directed at the establishment of the company objectives for the manufacturing planning and control system. The production plan determines manufacturing’s role in the company’s strategy. Hayes and Wheelwright (1984) distinguish three strategic planning levels which they have found in most

Figure 1-2 Manufacturing planning control system. [Source: Vollmann et al., 1992]

---

1 Although Vollmann et al. explicitly state that their framework only contains a set of functions and that it does not prescribe how these functions are to be performed, their framework is often referred to as a blueprint of Manufacturing Resource Planning, also known as MRP II.
modern companies. At the highest level, the so-called corporate strategy, the company specifies the definition of the business and the acquisition and allocation of key corporate resources to each of those businesses. The definition of the business may be stated in terms of either a specific material, a specific production process, or a market segment. This highest level strategic decision making, which Vollmann et al (1988) refer to as game planning, is outside the scope of an MPCS.

The next stage of strategy formulation divides the specification of the business of the firm into distinct Strategic Business Units (SBU). Each SBU is allocated either an organisational unit or a product range. The second strategy level in the hierarchy of Hayes links these SBUs to the corporate strategy. A business strategy specifies the scope of the business and the basis on which the SBU is supposed to achieve and maintain a competitive advantage. The third and lowest level strategy consists of strategies associated with each of the functional areas that is involved in the business e.g. research and development, marketing, and manufacturing. The production planning activity of Vollmann's framework is located at the functional strategic level. Consequently, the interface with marketing and finance is crucial to this part of the framework so the production plan is stated in terms that are meaningful to both marketing and finance. The production plan is not an anticipated-build schedule. Demand planning co-ordinates all business activities that place demands on manufacturing capacity: forecasting end product demand, order entry, order promising, interplant and inter company demand, and spare parts requirements. In order to enable the production system to meet long-term production targets, it should plan its resources. Resource planning encompasses the translation of extended production plans into long-term resource requirements (e.g. facilities, equipment, technologies, labour).

Unlike the production plan, the so-called Master Production Schedule (MPS) is an anticipated-build schedule. In order to postpone manufacturing commitment to a specific product mix an MPS may be stated in terms of end products, product options (high-level subassemblies), or groups of related end items. The information that is required for determining the MPS includes the production plan, the customer orders received, the backlog of orders, the ending inventory of MPS-items, and the Available-To-Promise (ATP) quantities. At this level, a rough-cut capacity check is carried out to determine whether the MPS is feasible. This capacity check considers both the planned aggregate capacity and the workload that is expected to result from the MPS. The capacity requirements plan is driven solely by exploding the MPS against the bill of resources. Unlike the MPS, the Final Assembly Schedule (FAS) is the actual build schedule. The FAS disaggregates the production plan into exact end-item requirements. Fixing of the FAS should therefore take place as late as possible. The final commitment to end items is dictated by the final assembly lead time. A feasible MPS is
the key link with the middle-third part of the framework, the engine. The engine, which receives the most of attention, is built around a Material Requirements Planning (MRP) system which disaggregates the MPS into a time-phased collection of component and raw material requirements. In subsequent stages, MRP-records are used to compute the required labour and machine centre capacity. Unfeasibilities that are encountered provide feed-back to the MRP system. Finally, the back end part of the framework encloses the dispatching and scheduling of shop-floor orders and purchase-order management systems.

**PRODUCTION PLANNING AND PRODUCTION CONTROL**

The IFIP (International Federation of Information Processing) glossary of terms used in production planning [Burbidge, 1987] defines production planning as the function of management concerned with planning, directing and controlling the provision and arrangement of production facilities, and the design of the processing methods to be used to produce the products or services provided by a company. This definition restricts the use of the term production planning to the decision making process and to the implementation of the decisions of both the arrangement of the production system and the layout of production processes. Production control is then defined as the function of management which plans, directs and controls the material supply and processing activities of a firm. Activities ranging from the planning of finished products down to the planning of shop orders, purchase orders for materials, bought parts, and supplies are considered tasks of production control.

A more common approach to distinguishing between manufacturing planning and manufacturing control is based on the concept of a control loop. In this view, manufacturing control guides the execution of a process in order to meet agreed-upon performance standards by using feed-forward and feed-back mechanisms whereas manufacturing planning is considered to be a separate activity. The criteria that are used to measure performance serve as input to the control activity. Usually, manufacturing is considered to be responsible for meeting the master plan, so master production scheduling cannot be part of the task description of manufacturing control. The process of defining production targets from aggregate plans primarily stated in terms of product/market options (cf. Chapter 2) and financial budgets down to the scheduling of production plans that serve as a contract with manufacturing is therefore beyond the scope of manufacturing control. The scope of the process of disaggregating the MPS into a time-phased collection of component/raw material requirements and the subsequent scheduling and dispatching of these jobs to distinct processing units (e.g. cells, lines, workstations) depends on the units that are used to define the master plan and the number of control layers on the shop floor.
PART 1 - INTRODUCTION

ANTHONY’S CLASSIFICATION OF PRODUCTION PLANNING DECISIONS
A commonly applied framework that classifies production planning decisions is that of Anthony (1965). Anthony distinguished three categories of planning decisions: strategic planning, tactical planning, and operations control. These three categories constitute a hierarchical decomposition of production planning decisions in which higher level decisions are linked with lower level decisions, and in which decisions that are made at higher levels impose constraints on lower level decision making. These detailed decisions provide the feedback to evaluate the quality of aggregate decision making [Hax and Candea, 1984]. Anthony’s hierarchical classification of production planning decisions recognises the distinct characteristics of the type of management that is involved, the scope of the decision, the risk associated with the decision, the capital investment, the ease of reversing the decision, the level of aggregation of the information that is needed, and the frequency with which a decision has to be made. The following enumeration provides a brief description of Anthony’s classification scheme with respect to production planning decisions:

1) STRATEGIC PRODUCTION PLANNING (SPP) decisions anticipate long-term market share targets by deciding upon the arrangement of both the manufacturing and the planning and control system. SPP decisions which affect a multi-plant manufacturing system as a whole include decisions on the spatial distribution of plants, the specialisation of the plants in terms of the manufacturing processes that are being executed and both the products that are being manufactured and those being assembled, the capacity that is allocated to each facility and inter-plant co-ordination. Inter-plant co-ordination decisions include the allocation of planning authority and the establishment of supply and delivery contracts. SPP decisions that relate to individual facilities include the selection of manufacturing technologies, the deployment of capacity resources, the establishment of stock-keeping facilities and the arrangement of planning and control processes. Depending on the industry and the nature of the company, a typical company decides upon at least one major strategic issue every year. The implementation of strategic decisions requires a relatively long period of time and substantial capital investments. Moreover, the impact of strategic decisions on the manufacturing system is difficult to change. Strategic planning decisions largely determine a company’s operational performance and thus its competitive strength. Chapter 2 discusses SPP in more detail;

2) TACTICAL PRODUCTION PLANNING (TPP) decisions determine time-phased aggregate levels of production in order to satisfy both (forecast) demand and the accumulation of inventory according to an agreed-upon service level. Given the service level objective and the existing manufacturing environment, the prime objective of TPP is to strive for the most efficient utilisation of the company’s
resources. TPP decisions, contrary to strategic decisions, are usually integrated in a company’s routine planning process. In literature, tactical production planning is also referred to as aggregate production planning. Chapter 3 elaborates on TPP in more detail;

3) OPERATIONS CONTROL includes the complete disaggregation of aggregate production plans and the subsequent scheduling and dispatching of shop-floor orders.

From a theoretical standpoint, each category of decisions in Anthony’s framework corresponds to a distinct managerial responsibility. In practice, however, it is difficult to draw the demarcation line between strategic, tactical, and operational management for companies in general [Sijbrands, 1993, p.84]. What is considered a significant risk and a substantial capital investment by one company, and thus a clear strategic decision requiring top-management’s involvement, may be a tactical decision taken by intermediate managerial levels to another company.

1.3 Growing understanding of manufacturing’s potential to obtain competitive advantage

For a long time little attention was paid to the arrangement of the production system. Manufacturing was viewed as an instrument to implement the company’s marketing and financial strategy, not as a means to gain competitive advantage. However, owing to intensified competition, management has become aware that small differences in the performance and efficiency of the entire logistics chain can have a large and decisive impact on a company’s competitive strength.

GLOBALISATION

The tremendous decrease in transportation costs over time has had an enormous impact on the arrangement of production systems. As a result of the rise in large-scale container transportation, for example, the cost of transporting a TV set from Hong Kong to Mönchengladbach (Germany) had decreased to approximately 35 Dutch guilders ($1.0 = f1,9) in 1994 (cf. Figure 1-3). This is only a small portion of its commercial value. In particular Japanese, Taiwanese, and South Korean companies have succeeded in offering their locally manufactured products at competitive prices throughout the world. Important factors that have contributed to this success include large investments in the automation of production processes, large investments in research and development, high priority given to the education of the working population and, in some cases, significant differences in wage levels compared to those of western countries. Conversely, cheap global transportation offers western companies
the opportunity to take advantage of the huge differences in labour costs by moving some of their manufacturing processes to countries with low wage levels. Cheap global transportation is one of the main reasons for the fifteenfold increase in the volume of world trade in manufactured goods over the last three decades. This long-term trend is not slowing down [Ferdows, 1989].

A development similar to the one just sketched in goods manufacturing may be observed in service industries. Global telecommunications and computer networks and rapid integration of information systems offers companies the opportunity to move or even contract out engineering and basic financial accounting activities on a global basis. For example, Ireland, with one of the lowest wage-levels of Western Europe, an educated English-speaking labour force, and a good telecommunications infrastructure, has a rapidly growing industry comprising companies providing financial services to other companies world-wide. In short, the decline of transportation costs has stimulated companies to view the entire world as the playing field on which to establish their production systems. The complexity of arranging the production system has grown accordingly.

Figure 1-3 Transportation costs of a TV-set from Hong Kong to Mönchen-Gladbach. Terminal handling charge at Hong Kong harbour: $1.72 • Transportation costs from Hong Kong to the port of Rotterdam: $25 • Terminal handling charge at the port of Rotterdam: $1.72 • Clearance: $0.45 • Transportation costs to Mönchen-Gladbach: $4.10 • Total transportation costs per TV-set: $34.34. [Source: Rabovisie, 1994]
In addition to the development of global sourcing policies, companies face the challenge of emerging global markets. According to Verter and Dincer (1992), the emergence of global markets is caused by the rapid improvements in communication technology yielding a standardisation in the demands of people with different cultures who live in different geographical regions. Often manufacturers have no choice but to enter these global markets. A growing number of competing firms operating in the same market reduces the average profitability, while larger capital investments are necessary to enable competition in terms of higher quality standards, smaller product life-cycles, and shorter delivery lead-times. Global markets offer manufacturers the opportunity to maximise their economies of scale and scope [Fawcett, 1992, p1091]. Unfortunately, entering larger markets requires companies to expand and, with the increase of both the production and distribution system, the complexity of the production planning increases at an even greater pace.

**BUYER’S MARKETS**

When there is a large number of suppliers, customers are encouraged to raise their standards with regard to quality, service, and price. They require that products meet quality standards, be delivered within the agreed delivery lead time, in the right quantities, and at the right place. In such cases their need to stay in business forces companies to adjust more frequently and more rapidly to the requirements imposed by customer-dominated markets and, in order to ensure the survival of the company, the internal structure of the production system must be continuously adapted.

### 1.4 Managerial approaches dealing with increased manufacturing planning complexity

The immense complexity of manufacturing systems is determined by the number of elements (i.e. the number of human/machine combinations), the number of executive and control relationships among elements and the number of changes of the elements and relationships in time (i.e. the degree of stability of the system) [GST, 1987]. Important factors that determine manufacturing complexity include [Bertrand, 1990]: the variety of products manufactured, the number, location and requirements of clients, the nature of operations, the average number of manufacturing steps that make up an average routing, the variety of resources and the average number of resources that is involved in a manufacturing operation. Many problems in manufacturing planning and control are not analytically complex; their complexity derives from the enormity of the underlying database required to properly support routine decision-making systems [Vollmann et al., 1988].
PART I - INTRODUCTION

Production management has adopted a wide range of technologies and organisational paradigms that aim to reduce the complexity of the planning process. An example of such an organisational paradigm is cellular manufacturing which comes down to the manufacturing of a family of parts by a dedicated group of machines [Rolstadås, 1988]. Cellular manufacturing rationalises the material flow and reduces the number of bill-of-material levels that have planned and controlled inventory levels because a number of product conversions is included in one routing. This paragraph briefly discusses three lines of approach to deal with the growing complexity of manufacturing: relocating the production planning decisions within the organisational structure, the application of information technology, and the application of quantitative analysis.

ALLOCATING PLANNING AUTHORITY THROUGHOUT AN ORGANISATION

The allocation of the production planning processes to distinct parts within the organisational structure has a large impact on the effectiveness, the efficiency, and the complexity of the resulting production planning and control problems. Meal (1984) distinguished three approaches to arranging the planning process, each of which has been applied in practice. Historically, these approaches can be viewed as subsequent stages in a maturation process in production management.

First, the conventional approach which is based on a decentralised control architecture within which each individual plant is responsible for its own production management. Corporate management provides no interplant co-ordination of production plans. The disadvantages of the conventional approach are obvious. Planning on a plant-by-plant basis yields high inventories and poor supply chain performance.

The second approach to production planning is based on a centralised production control architecture. The centralised approach is a maturation of the conventional approach in the sense that it improves overall co-ordination by defining detailed schedules for each individual plant at the corporate level. A serious disadvantage of this approach, however, is that the planning of detailed production schedules is concentrated into a single planning stage. Although the computational complexity of this immense task may not prove insurmountable, huge forecast errors on market demand at item level will remain. Moreover, in order to be able to determine detailed production schedules at the corporate level, enormous amounts of data have to be communicated in an upward direction. Finally, a centralised approach is also undesirable from a managerial point of view. Large and complex models dealing with huge amounts of data reduce managerial insight. Additionally, effective corporate decision making on detailed production plans is hampered due to the lack of up-to-date information on the current state of production processes. Centralised decision making takes away much of the authority and independence of lower managerial echelons.
INTRODUCTION

The third approach is to tackle the production planning problem by decomposing the planning process in agreement with the organisational structure. From a managerial point of view, the underlying philosophy is that decisions have to be matched with appropriate levels of management [Meal, 1984]. Clearly, partitioning the overall planning problem and the subsequent delegation of subproblems to distinct managerial layers yields a decentralised planning structure. To balance the advantages of the conventional approach with the improved co-ordination of centralised planning, a framework is provided in which higher level decisions impose constraints on lower level decision making.

Hierarchical planning methods are most commonly used in current production planning practice. Two reasons are largely responsible for this approach. The first, is the inherent hierarchy of personnel-ranking that may be observed in organisational structures. The second is the fact that a hierarchy is no more than a way of ordering the problem field and thus reducing its complexity to cognitively attractive properties. In this field of research much attention has recently been focused on co-ordination schemes with respect to the control part of the business activities. The impact of decomposition decisions, which, in essence, involves the allocation of decision variables to managerial units, on the complexity and effectiveness of the organisation structure, has been extensively studied by Baligh and Burton (1984) in their search for design methods of organisation structures.

APPLICATION OF DATABASE AND INFORMATION SYSTEMS TECHNOLOGY

The rise of computer-based information systems which depend heavily on large corporate databases provided a means to deal with the increase in process complexity. If maintained and properly updated, databases can quickly provide high volumes of up-to-date operational data. The availability of up-to-date and reliable data is crucial to effective production management, or as Gunn (1992) put it: “What good is it to be able to manufacture a product in three days if it takes a week to a month to create the company’s MPS for the next six months”.

MRP has been the most widely implemented large-scale production management system since the early 1970s, with several thousand MRP type systems implemented [Harhen, 1988]. Several surveys [Anderson et al., 1982; LaForge et al., 1986] report significant effectiveness and efficiency improvements from MRP II systems. The data-processing capabilities of MRP and MRP-II systems are considered the greatest contributions to production planning. However, the use of MRP systems generated an even larger amount of data which made these systems become a source of complexity by their own. The application of computerised MPC systems has changed the way in which production management operates: “Firms (...) typically find that the day-to-day
PART I - INTRODUCTION

job becomes management by the database and management of the database.” [Vollmann et al., 1988, p14].

APPLICATION OF QUANTITATIVE ANALYSIS

Logistics management shows a growing need for simplification and model-based analysis to support its decision-making processes [Bolwijn et al., 1990]. The range of analytical techniques that have the potential to provide support to a planning problem depends on factors such as the degree to which the problem is structured, the level of detail that is required to depict the problem and the time-slot in which a decision must be reached. Practical considerations such as data requirements, integration with existing infrastructures of information systems and database systems usually turn the scale when assessing whether a technology should actually be put into use.

Spreadsheets support repeated calculation of compact financial and logistics metrics and “what-if” calculations based on relatively small-scale mathematical models. In many logistics departments spreadsheets are being used to help generate production and distribution plans. The key to the success of spreadsheet-based tools and models lies in the fact that, from the user’s perspective, there is still a spreadsheet model, i.e., a familiar and relatively easy to understand modelling environment [Bendiner, 1993]. Most of today’s spreadsheet packages can easily download data from major database systems through routine queries.

In simulation, each event that occurs in the system being modelled is mimicked in a computer model [Law and Kelton, 1982]. Simulation models depict complex systems with a relatively high level of detail, incorporating both stochastic and deterministic system behaviour. Simulation is becoming increasingly popular as a tool that supports decision making at all planning levels. Strategic management may use simulation to evaluate the impact of manufacturing strategies at the operational level or to decide upon priority dispatching rules that guide operations management by simulating the flow of production. Simulation can also be applied in shop-floor control to determine, for example, detailed schedules of work orders in complex job-shop environments. Today’s full-graphics simulation software packages, which contain libraries of almost ready-to-use machines, conveyors and other equipment, have certainly contributed to the popularity of simulation in practice.

The applicability of optimisation techniques in MPC is limited to aggregated levels of production planning. Although models that solve detailed scheduling problems exist, the computational intractability of these models limits their applicability to small-size theoretical studies. Aggregated production planning problems, as opposed to detailed
scheduling problems, are depicted by less-detailed aggregates and thus less data. Section 1.5 elaborates on the use of optimisation techniques.

1.5 Applicability of optimisation technology to manufacturing planning

Optimisation techniques guarantee the provision of an optimal, not necessarily unique, solution to the formal representation of a feasible decision problem. Optimal solutions result from a search of the complete set of feasible alternatives. Alternative approaches like, e.g., simulation and heuristics, only search a subset of feasible alternatives which is believed to contain an acceptable solution.

Of equal importance is the fact that (MI)LP models still depict the modelled system in relatively little detail. A detailed problem representation would place too much reliance on data that is either difficult or impossible to obtain with a precision that matches the level of detail used to represent the problem. Moreover, the outcome of experiments with strategic optimisation models is used to assist the decision maker to arrive at a decision which is largely based on qualitative judgements. A detailed model would wrongly suggest that the outcome of the model represents a prescription, i.e. a guide to action, for the decision maker.

Until recently, only the efficiency provided by LP-solution algorithms was acceptable for practical applications. Now, owing to substantial improvements in computer power, LP algorithms (which serve as the engines of the more complex MIP-solution algorithms), modelling techniques, pre-processing and branch-and-cut, [Johnson and Nemhauser, 1992] MIP models have become increasingly effective as a basis for DSSs. During the last two decades, solution times have been improved by more than two orders of magnitude [Nemhauser,1994], thus clearing the road for time-critical applications and more realistic models.

The literature shows large numbers of MIP-models that focus on SPP (cf. Chapter 2). During the last decade, MIP-models have been used successfully in DSSs for strategic planning (cf. Chapter 3). These observations justify the conclusion that MIP-technology provides a useful means to model strategic trade-offs to an extent that enables the decision maker to better substantiate and, hopefully, to improve the decisions that have to be made.

The popularity of hierarchical arrangements of the production planning process (cf. Section 1.4) is also reflected in the design of methods that aim to support these
decision making processes. Starting with a paper of A.C. Hax and H.C. Meal "Hierarchical Integration of Production Planning and Scheduling" (1975) increasing attention has been paid to HPP-schemes and the aggregation-disaggregation issue, which formalise the interface between adjacent layers (cf. Chapter 2). In a hierarchical planning system, decisions are made in a sequence instead of simultaneously. Aggregate decisions are made first and these impose constraints within which more detailed decisions are made. In turn, detailed decisions provide the feedback used to evaluate the quality of aggregate decision making [Bitran and Hax, 1977; Hax and Candea, 1984]. Hierarchically arranged mathematical procedures that support HPP include a partitioning of the overall planning problem into subproblems, which are described by mixed integer (linear) programming models, and which are linked through formal aggregation-disaggregation schemes. These mathematical HPP-schemes are suboptimal by definition. Nevertheless, the optimal solution to a hierarchical model may not provide an optimal solution for the single-level model of the entire system, frequently the error is more than justified by the ease of the solution procedure [Dempster et al., 1981].

HPP-frameworks that have been published need to be tailored to fit specific problems. Important factors that guide the partitioning of the overall planning problem are the categorisation of decisions along the lines of the organisational structure, the computational difficulty of the resulting subproblems, the client-order decoupling point, the product structure and the planning horizon. Despite the need for customisation, mathematical frameworks that support HPP seem to provide a complete coverage of the tactical production planning problem. Gelders and Van Wassenhove (1981) even state that: "(...) production planning problems in manufacturing may always be formulated as mixed integer (linear) programming models." Zäpfel and Missbauer (1993, p.307), however, state that: "The production problems of HPP, which are described in literature, usually seem to be simpler than the overall planning problem of MRP II." The literature also provides indications that MRP II systems are more commonly applied than (MI)LP-based hierarchical planning systems. Still, the suitability of (MI)LP-based hierarchical production planning frameworks for application within integrated DSSs for the evaluation of strategic scenarios in terms of aggregate flows of production is evident.

1.6 (MI)LP-based decision support

Basically, today's decision makers, who want to incorporate (MI)LP techniques into their decision making processes, have three options.
1.6.1 (MI)LP-based decision support systems

The first option is to use a readily-available —possibly commercial— (MI)LP-based DSS tailored to the needs of a specific application area: e.g., DP-PSS [Evers,1992], Linx [Numetrix,1994], PHYDIAS [Bender,1992], and SLAM [Nunen,Benders,1980]. Much of the strength of these systems stems from the design of the user interface. These systems use terms and concepts that are familiar to the decision maker. In this way the level of knowledge regarding (MI)LP-technology that is necessary to operate the system is kept to a minimum. Ideally, although this is not common working practice, the decision maker is enabled to use the DSS personally. Once the database subsystem contains images of the relevant parts of corporate databases, the evaluation of different decision strategies under different circumstances (often referred to as scenarios) only requires the decision maker to define a relatively small collection of data that is primarily used to parameterise the underlying models. It is precisely the quantification of such a collection of parameters that requires the specific practical knowledge and skills of the decision maker. Another advantage of these DSSs is that the design of the user-interface may be based upon a model of the decision-making process (see, for example, the Distribution Planning Support System [Mourits,1995] of which the design is based on an Integrated Planning Support Framework for the design of distribution networks). This model may be reflected in the design of the (MI)LP models underlying the DSS, in the data interdependencies that exist among these models and in the typical order in which the models may be applied. Consequently, the system not only supports the ‘choice’ phase [Simon,1960] of the decision making process which is generally considered the focus of Management Science and Operations Research. The (MI)LP models, the data model, and the user-interface, also support —although more passively— the ‘design’ phase identified by Simon since they serve as a reference framework of the decision-making process and as a checklist of the decisions being involved and the data that are needed.

An important disadvantage of these DSSs is, however, that the user cannot change the structure of the underlying models in order to accommodate the inevitable changes in user demands.

1.6.2 (MI)LP-modelling systems

The second option is to use an advanced (MI)LP-modelling system which is designed to express a problem in a way in which human modellers understand it rather than the form in which algorithms solve it [Fouyer et al.,1990; Fouyer,1983]. The translation of the modeller’s form into a format that is accepted as input by general-purpose (MI)LP
PART I - INTRODUCTION

solvers\(^1\) like CPLEX [CPLEX Optimization, 1990] and OSL [IBM Corporation, 1991], can be performed entirely by the computer. The majority of today’s (MI)LP-modelling systems has evolved into integrated software tools for construction, modification, documentation, and analysis of models.

Algebraic modelling languages, e.g., AMPL [Fourer et al., 1990], and GAMS [Brooke et al., 1988] are non-intuitive: the physical structure of the underlying problem is not made explicit [Murphy et al., 1992b]. Moreover, the algebraic notation makes it difficult to provide conceptual modularity through hierarchical abstraction or bottom-up composition [Murphy et al., 1992b]. Experienced modellers often prefer algebraic modelling languages, owing to their similarity to the algebraic notation customarily used to represent (MI)LP models. Non-expert model builders, including the majority of decision makers, favour an intuitive, preferably graphical, model representation. An intuitive modelling language makes it possible for the non-expert to develop a model with relative ease. The advantage of an intuitive modelling language to both the expert and the non-expert model builder is that it makes it easier for them to communicate the structure and mechanisms underlying the models to others.

The intuitive (MI)LP-modelling languages that we have encountered, i.e., LOGS [Brown, Northup, Shapiro, 1986] and LPFORM [Ma, Murphy, Stohr, 1989], either support only a limited range of models (LOGS) or are not capable of representing all relationships that exist among the components of a model (LPFORM).

Some (MI)LP-modelling systems that we encountered did not contain a model-independent database subsystem. For example, gLPS [Collaud et al., 1994] uses the representation of the structure of a model to store the data that are to be used in it. MPL\(^2\) only provides linkages with external databases. However, as a DSS is used to anticipate future states of the problem domain, a database subsystem that contains a representation of the state space of the system under consideration constitutes the core of any DSS. The benefits of an internal database that contains images of the relevant parts of remote databases have long been recognised in the area of DSSs [Sprague, Carlson, 1982]: (1) The structure of the contained database may be adapted to meet the requirements of the DSS and of its users; (2) Most aggregation and subsetting of the necessary data can be done prior to downloading the data which improves

\(^1\) (MI)LP solvers provide limited support to the model formulation process. The Mathematical Programming Systems (MPS) format [IBM Corporation, 1992] which most solvers accept as input, is generally considered unsuitable for modelling and debugging [Meeraus, 1984]. The inherent complexity of the modelling process, which involves minute attention to detail, and the necessity to improve both its efficiency and its effectiveness has initiated the rise of (MI)LP-modelling systems.

\(^2\) According to Sharda et al. (1995), Maximal Software, Inc., is currently developing an internal database.
performance; and (3) The database subsystem can be manipulated without endangering vital information that is shared by different users.

None of the (MI)LP-modelling systems we encountered contains a model-independent data dictionary. A data dictionary is a software subsystem which provides facilities to model and maintain structural relationships among the elements of the database. For example, in a conventional relational database management system, these relationships may consider referential integrity constraints between relations or semantic rules at the record level. A data dictionary is not only a powerful tool to maintain database integrity, it may also provide the user with a comprehensive model-independent view of the data model in which the logical structure of the data is made explicit.

Current modelling systems typically offer the user an interface to the data that is in keeping with the relational data model which today’s major database systems customarily use as an external data model. Although the relational approach to data modelling is widespread and known for both its strong support for database consistency and the simplicity and uniformity of its operations, data associated with logically related model activities is scattered across the database in seemingly unrelated tables. From the user’s point of view, however, it is more natural to group logically related data elements as is customary in object-oriented database design. Some algebraic modelling languages provide syntactical constructions that allow the modeller to group related data by using, for example, indexed sets (sets of sets). However, these languages do not enforce a modular and unequivocal approach to data modelling which is used consistently throughout the design of the system and the way in which it is operated by the intended group of users.

1.6.3 Tailor-made (MI)LP-based decision support systems

The third option is to develop a customised (MI)LP-based DSS. Companies, however, are often reluctant to commence such a project. A feasible ‘low’-cost approach would be to implement a customised application on top of an existing algebraic modelling system which can be accessed through a DLL (Dynamic Link Library). The algebraic-oriented model representation is, however, not suited for interaction with untrained modellers. A Graphical User Interface (GUI) builder will therefore be necessary to implement a dedicated and easy-to-use DSS. The resulting system, however, still does not exhibit the advantages of an (MI)LP-modelling system based on an intuitive model representation which is tightly connected with a data-modelling language and which both serve as the primary means for interaction with the user. The models remain ‘black boxes’ while the database facility consists of seemingly unrelated data tables without a data-dictionary which governs the interaction with the database by using the
logical structure of the underlying data. Therefore, the implementation of a dedicated, intuitive and powerful DSS is still a significant and costly undertaking.

1.7 Outline of the proposed system

The previous section indicated that the applicability of currently available (MI)LP-based Decision Support Systems (DSS) in the field of strategic production planning is limited. Moreover, the development of a tailor-made DSS is a complex, time-consuming, risky, and costly task. This thesis presents a different approach to the design of an (MI)LP-based DSS.

To implement a library of 'building blocks' that may be used to model a tailored production system for conducting experiments with (MI)LP models for strategic and tactical production planning, one approach could be to base the design of the building blocks on the functions that must be fulfilled within an MPCS. For that purpose, the components of Biemans' (1990) reference architecture provide an excellent point of departure for they embody the necessary degree of generality and the required level of concreteness. If the building blocks come with predefined interfacing relationships, then the composition of a tailored production system is a relatively easy task. It is easy to imagine a system that subsequently uses this user-defined coherent set of building blocks to automatically extract the necessary data from a database subsystem in order to construct an (MI)LP model, provided that the structure of this model is known in advance. But then a building block must have a predefined 'behaviour', i.e., a predefined (MI)LP-model substructure which is in line with its role within the MPCS framework. This requirement, however, limits the intrinsic flexibility of the system and is therefore incompatible with the objective of this research to design an environment that supports the development of tailor-made DSSs.

The use of a general-purpose (MI)LP-modelling facility to define the MPCS-behaviour of the building blocks requires a flexible mechanism to connect the parameters of the models to the data that is associated with the building blocks. Moreover, the possibility to configure the system by using tailored models also requires a flexible way of defining the data structures associated with the building blocks, since the data requirements cannot be known in advance. Thus, a natural way of implementing the data structures associated with a building block, is to use a data-modelling scheme.

The system that is presented in this thesis may be used to implement a library of MPCS-inspired building blocks. For that purpose, it integrates a general purpose (MI)LP-modelling facility, a data modelling facility, which also serves as a mechanism
to exchange data among different models, and basic support for data representations and multi-model experiments. To obtain the desired degree of flexibility, the system makes use of concepts which closely relate to the technology that is being used rather than to a specific application area. Therefore, it is not until after the system has been configured that it interacts with the user by using application-specific terms. Although the applicability of the proposed system is not limited to strategic and tactical production planning, this area will be used as a reference field of application throughout this thesis.

The graphical (MI)LP-modelling system elaborates on today's graphical modelling systems. The graphical modelling language provides an intuitive modular representation of a large range of models. It is capable of representing any (MI)LP model, and it depicts all relationships that exist between the components of a model. Unlike existing modelling languages, many of which only support the modelling of real-world entities via groupings of logically related (MI)LP-activities, the modelling system described in this thesis also includes a data-modelling facility which actively supports and even enforces the grouping of logically related data elements. This way, as a result of a consistent and unequivocal carrying through of the notion of conceptual modularity throughout the design of the system, the structure of the underlying problem domain may be reflected in the organisation of both the models and the database. This similarity may then be used to integrate both modelling schemes, i.e. to integrate logically related model activities with an accompanying data abstraction, to improve the modelling of real-world entities and thus to improve the manageability of the system. This corresponds to the notion of objects which is becoming increasingly popular as a building block for the design of complex systems. In addition, the graphical modelling language supports the construction of model templates. Model templates are expected to facilitate the creation or assembly of models by inexperienced users from predefined components.

The proposed system may be viewed as an attempt to obtain a synergy between the flexibility offered by general-purpose (MI)LP-modelling systems and the user-friendliness and decision-support functionality offered by (MI)LP-based decision support systems. The system is aimed at shortening the process of implementing flexible, intuitive, and tailor-made DSSs based on (MI)LP models.

### 1.8 Structure of this thesis

This thesis consists of four parts. Part I, *Introduction*, comprises four chapters. This introductory chapter introduced the area of strategic and tactical manufacturing
planning, the complexity of the planning processes, the increased awareness of the potential of the manufacturing system to obtain competitive advantage and the applicability of (MI)LP-based DSSs in support of the planning processes. We discussed the strengths and weaknesses of the options available to the manufacturing system in order to incorporate (MI)LP-technology into the planning processes and the approach that we advocate. We also outlined the design of the system that is presented in this thesis. Chapter 2, *Strategic and tactical manufacturing planning*, elaborates on strategic and tactical manufacturing planning and will provide a brief survey of the (MI)LP models that have been described in literature on this subject. Chapter 3, *Survey of existing tools for (MI)LP-based decision support in production planning*, provides an outline of the functional design that is typical to many (MI)LP-based DSSs and (MI)LP-modelling system and presents a survey of typical packages in each of these two categories. Chapter 4, *Added value*, uses the contents and terminology of the previous chapters to provide a more detailed picture of the added value of the system that we propose in this thesis.

Part II, *Conceptual design*, deals with the conceptual design of the proposed system. Part II consists of three chapters. The subdivision of part II into chapters is based on the common parts of the functional architecture of (MI)LP-modelling systems presented in Chapter 3. Chapter 5, *Design of the (MI)LP-modelling scheme*, deals with the conceptual design of the graphical (MI)LP-modelling scheme. Chapter 6, *Design of the data model*, deals with the conceptual design of the data model. Chapter 6 discusses the data-modelling scheme together with the operations for manipulating the data model and the data, and the integrity constraints. Chapter 7, *Design of the model-data link*, presents the connection between the (MI)LP-modelling scheme and the data-modelling scheme, and the conceptual design of AC-objects which integrate both modelling schemes in so far as they relate to the representation of a real-world entity that is being modelled. Chapter 7 also outlines how the support for multi-model experiments may take shape.

Part III, *Implementation & Validation*, consists of two chapters. Chapter 8, *Design and implementation of the prototype*, and Chapter 9, *Case studies and Evaluation*. Chapter 9 reports on the application of the prototype on three case studies and discusses our findings.

Part IV, *Conclusions*, contains only a single chapter: *Conclusions & Recommendations*. In this final chapter, the main conclusions that follow from this research are summarised. Chapter 10 also contains a number of issues that should merit attention as a continuation of the present research.
2. STRATEGIC AND TACTICAL MANUFACTURING PLANNING

OUTLINE
This chapter elaborates on the process of strategy development, provides a survey of strategic and tactical manufacturing planning decision categories, and discusses the (MI)LP models that may be used to support making these decisions.

Whereas the literature shows examples of (MI)LP-based planning systems which have automated most of the routine tactical (or aggregate) planning, models play an important, yet secondary, role in strategic planning. The decisions involve trade-offs that are often difficult to quantify and sometimes not even part of the formal decision making process. Moreover, the great number of issues involved easily exceeds today's modelling and computational capability.

To reflect the relative importance of models for strategic manufacturing planning, this chapter discusses these models in the context of developing a manufacturing strategy.
2.1 Strategic manufacturing planning

Traditionally, the process of strategy development is based on management's perception of environmental variables, and the resulting collection of both emergent and intended strategies [Melnyk, Handfield, Carter, 1994]. Fawcett (1992), for example, mentions the contingency theory which states that companies adapt to changes in their environment in order to maintain or enhance their competitive performance. A company showing this reactive attitude towards strategy development perceives its competitive environment as highly complex and uncertain. The strategic planning process is aimed at outguessing the future development of the product markets in which the company is involved.

In today's markets, the traditional reactive attitude towards strategy development may be a dangerous one, for it easily persuades a company to follow a wait-and-see policy. In some markets, however, the products and production processes change at such fast rates that playing the waiting game may result in continuously lagging behind and rapidly decreasing sales. A company that has the capability to quickly switch to new product markets by using the appropriate technology [Hayes and Pisano, 1994] will be more responsive to changes in the market place. Firms able to take the lead in clearly defining their position in the light of the existing environment and to take appropriate action will have a greater likelihood of predicting the future development of the marketplace. Apart from the fact that these firms seem to have developed both accurate mechanisms to analyse the market and effective procedures to implement their strategy, to some extent they also have an opportunity to create their own environment. In this way a company creates complexity in product markets, thereby thus catching competitors unaware, in order to obtain a time-based competitive advantage [Melnyk et al., 1994]. Tight integration of strategy formation (intent) and implementation (action) is therefore vital to this proactive approach to strategy development.

---

1 Mintzberg (1978) and Mintzberg and Waters (1985), distinguish intended strategies and realised strategies as two ends of a continuum along which real-world strategies lie. Their empirical research on strategy formation is based on a reconstruction of strategies—which they defined as 'a pattern in a stream of decisions or actions'—by looking primarily at the actions taken. Mintzberg observed that consistencies in a range of actions may result from a clear intent (deliberate strategies) and from spontaneous convergence in the behaviour of a variety of actors, from causes both internal and external to the organisation without the existence of a plan, or an intention, worked out beforehand (emergent strategies). Hence, Mintzberg opposes the view which equates strategy making with planning [Mintzberg and McHugh, 1985].
2.1.1 Corporate, business, and manufacturing strategy

In most multi-business companies three levels of strategic planning can be identified [Hayes and Wheelwright, 1984]: A corporate level of strategic planning, strategies that deal with specific product-market combinations or Strategic Business Units (SBU, cf. Chapter 1) that the company is engaged in, and strategies that deal with the contribution of each of the business functions (e.g., procurement, production, distribution, marketing, finance) that is involved in a business unit.

Hence, companies develop and pursue multiple manufacturing strategies, each of which is tailored to the needs of a specific business unit or product-market segment. The term ‘corporate’ manufacturing strategy that is sometimes used in the literature then relates to parts of the manufacturing strategy that are shared by all strategic business units. In these cases, the policy can be adopted company-wide. This, of course, improves the ease of control for the company as a whole, but may also force a SBU to adopt the company policy which does not seem to fit. The term ‘corporate’ manufacturing strategy may also be used to refer to those areas where a corporate-wide perspective and focused efforts would be preferable to letting each business unit develop its own manufacturing policy. Research and development functions, for example, are often separated from other business units and grouped into corporate business units. Some authors still only refer to a corporate manufacturing strategy if each business strategy adopts a very similar manufacturing strategy.

2.1.1.1 The components of a corporate strategy

A corporate strategy specifies two areas of overall interest to a company: the definition of the businesses —e.g., a particular material and its associated production processes, a market segment, or a group of consumers— in which the company will participate, and the acquisition and allocation of key corporate resources\(^2\) to each of those businesses [Hayes et al., 1984]. The development of a corporate strategy also includes the consolidation of the various business strategies, for example, by preventing competition between SBUs and by identifying business policies which corporate

---

1 The length of the planning cycle and the number of managerial echelons that is involved in a company’s strategic planning process greatly depends on the nature of the business the firm is in, its managerial attitude towards strategy development, the maturity of its strategic planning process, and the company’s organisational structure.

2 E.J. Flynn (1994, Detroit:MB.18.1) mentions four properties that identify key resources. Key resources must be either: (1) valuable (a company must be willing to pay for it); (2) rare; (3) hard to copy; or (4) provide special opportunities to attain competitive advantage.
PART I - INTRODUCTION

management considers crucial to the long-term continuity of the company and which require the focused efforts of the company as a whole.

2.1.1.2 The components of a business strategy

A business strategy specifies the scope of the business (the product-market segments), in a way that links the strategy of the business to that of the firm as a whole, and the basis on which the business unit intends to achieve a sustainable competitive advantage [Hayes et al., 1984]. Another major concern of business strategy development is to define strategic objectives for each of the participating business functions so that each will contribute fully to attaining a sustainable competitive advantage for the business as a whole.

DEFINITION OF A PRODUCT-MARKET SEGMENT

The definition of an SBU necessitates the identification of existing and new product-market-service segments to be included in the strategic planning process for exploring their potential profitability. Criteria that may be used to identify product-market segments include the geographical location of markets, the type of distribution channels to be used and the potential use of a product. "Some degree of ambiguity in the definition of markets for products that are potential substitutes will arise inevitably, but an effort has to be made to define these segments as products in mutually exclusive competitive markets." [Hax and Majluf, 1978].

ANALYSIS OF PAST PERFORMANCE

A minimum set of 'bench marks' that is needed to measure a company's competitive strength in a market vis-à-vis its main competitors includes [Hax et al., 1978]: the size of the total market, the company sales and the sales of the most important competitors, the market growth rate, the relative market share, and the price levels when the company does not act as a price taker. Competitors usually keep their profitability figures secret so usually these figures cannot be used for comparison. Other important figures relate, for example, to the quality, dependability and costs of the production processes, to the supply chain as a whole and, ultimately, to the level of customer service.

Hax et al. stress the power of using graphs to support the analysis of the performance of companies in a specific market by means of a matrix categorisation\(^1\) that may be used to position a product with respect to its life-cycle (introduction, growth, maturity, and decline) and its relative market share. The matrix is a visual tool that can be used

\(^1\) Hax et al. adopted the 'relative market share-growth matrix' from the Boston Consultancy Group.
complementary to detailed financial analyses. Figure 2-1 displays an example of such a graph. Each circle corresponds to a product-market segment. The area of the circle is proportional to the total net sales of the product. The centre of the circle is determined by the relative market share (X-axis) and the market growth (Y-axis) respectively. The vertical line corresponds with a value of 1 along the X-axis; an average market share. Values greater than one imply that the company's sales are above those of its major competitors. The horizontal line indicates an average level of growth. This level is commonly chosen as the growth of the Gross National Product (GNP) or the growth of the particular industry. Low-growth markets usually correspond to products in their maturing or declining stages of their life-cycle. These markets have developed beyond the point of saturation and sales are mostly limited to replacement of existing products. Newer products may have taken their place. High-growth markets relate to products that have been introduced only recently or that have a market that has not yet reached the point of saturation. To illustrate past performance, a series of circles for the same product-market segment can be plotted.

![Graph showing relative market growth and share](image)

**Figure 2-1** Graphic assessment of product-market segments with respect to their life-cycle and the portfolio of the firm. [Source: Hax and Majluf, 1978, p59 (modified)]

**Assessment of future development of the market**

The next step is to try to assess the future development of the market on the basis of detailed analyses of the past performance of a product-market segment. This step involves three major categories of issues that should be focused upon [Hax et al., 1978]:

1. **The definition of scenarios**, including determining trends and expectations on the general economic environment, the industry, product markets, the expected behaviour of competitors, the firm's situation;
PART I - INTRODUCTION

2. Assessment of the strengths and weaknesses of the product markets, including their vulnerability to new technologies, inflation, raw material supply, competitor’s actions, consumer preferences, cyclical fluctuations, strikes, worker’s union actions and government regulations;

3. Analysis of the evolution to be expected in the total market for the product and in the firm’s market share under different scenarios. Issues that must be considered include the total projected market, its position in the life-cycle, the firm’s absolute and relative market share.

IDENTIFICATION OF THE OPPORTUNITIES OPEN TO THE COMPANY

In short: what can the company do to improve its competitiveness. Options may include the introduction of new products, the penetration of markets, the adoption of new technologies, alternative ways of arranging the company’s distribution network and outsourcing business functions which are not considered as core competencies.

Hayes and Wheelwright (1984) distinguished four principal performance dimensions capable of providing a firm with a competitive advantage and to which a company must attach clear priorities: price, quality, dependability, and flexibility. Hayes, Wheelwright, and Clark (1988) added a fifth competitive dimension: innovation¹. If a company decides to compete mainly on price by producing high volumes of similar products, then manufacturing’s role is to maximise efficiency, for example, by using highly automated equipment or a combination of standard equipment and cheap labour. If a company, however, is to compete in terms of quality, then it may be necessary to buy expensive high-precision equipment.

DEFINING AND CO-ORDINATING FUNCTIONAL STRATEGIES

The co-ordination of the major business functions that are involved in a specific SBU is of crucial importance to the success of a business strategy. Inevitably, this requires an implicit or explicit assessment by management of the relative importance of each of the participating business functions to achieve competitive advantage. This judgement determines the role each of the functional areas is allowed to play in establishing the business strategy.

At this stage, management decides, amongst other things, upon the strategic targets for the manufacturing function. These objectives constrain the options that are available to the management of the participating business functions when developing their functional strategy.

¹ For example, the fact that Intel Corporation has been capable of maintaining and gradually expanding their large market share on the CPU market for PCs is largely due to their high rate of innovation.
Since the early 1960s, in many companies marketing considerations have dominated the strategic planning process\(^1\). Only recently intensified competition, higher quality standards, and globally dispersed markets brought about a change in management’s perception of the development of a business strategy. In many manufacturing-based business areas, companies recognised the increasing importance of the logistics function to the attainment of a sustainable competitive advantage (cf. Chapter 1). Manufacturing is no longer viewed as a means to implement the strategic goals formulated by marketing but as a mature part of business strategy development in which marketing options are considered vis-à-vis the implications to the manufacturing system, the trade-offs involved and the costs associated.

### 2.1.2 Objectives and performance measures of the manufacturing function

At the business unit level of strategy formation, a decision is taken on the nature of the contribution of the manufacturing function to the pursuit of the strategic targets set for the entire SBU. In the literature there appears to be a common agreement on four competitive priorities for a firm’s manufacturing function: cost, quality, delivery performance and flexibility with respect to the type and the quantities of end items\(^2\).

To measure the actual performance of the manufacturing system, these priorities can be used as stepping stones for the definition of so-called ‘performance indicators’ or evaluation criteria. Which criteria are used and how these criteria are defined depends on the relative importance assigned to each of these priorities and the actual

---

1 North-American authors often mention the portfolio approach which views a company’s product range as a portfolio of opportunities and options. In this view, the purpose of strategic management is to continuously monitor the allocation of cash generated by the most mature product lines (‘cash cows’) and to move into promising markets while disposing deteriorating ones when considered appropriate. This attitude towards strategy development has been much criticised later because it ignores the potential contribution of the logistics function to obtain competitive advantage [Hayes et al.,1984; Skinner,1978]. Hayes et al. observed that (p35): “(...) most manufacturing-based companies seek to minimise the negative impact that manufacturing can have (...) reflecting the view that marketing, sales, and R&D provide better bases for achieving competitive advantage”. This view may, however, turn out to be counterproductive for a strong focus on financial performance measures rather causes the performance of the logistics function to deteriorate. Some authors state that the portfolio approach is typical of American companies and cannot be observed in Europe.

2 In an empirical study conducted by J.E. Ettlie and J.D. Penner-Hahn (1994) thirty-nine Northern-American companies were requested to allocate a total of 100 points to 4 strategic foci: quality, cost, delivery performance, and flexibility. An average of 36.65 points was allocated to quality. Cost was next with 27.18 points. Delivery performance ranked third with 21.77 points and flexibility ended with only 14.55 points.
manufacturing system. For example, the cost level is frequently used as a common denominator to measure the level of investments in tangible assets, direct machine and labour costs, the use of cash to finance pipeline inventories, the level of scrap and overhead costs. Delivery-performance indicators often concern order lead time reliability. Many researchers currently focus on accurate performance indicators that deal with various types of flexibility. Gupta(1994) has defined necessary conditions for flexibility measures.

2.1.3 The components of a manufacturing strategy

Given the statement of manufacturing’s contribution to the business strategy and the set of criteria for judging its performance, management must decide how to design or reshape the manufacturing system in order to achieve optimal performance. The policies that result must adhere to the key objectives assigned to manufacturing. For example, a business strategy statement on the quality standards to be pursued not only affects the choice of the technology and equipment and the work force policy to adopt, it may also even affect the location of plants.

A manufacturing audit provides an approach to the designing of a production system and for analysing an existing production system to reveal inconsistencies and opportunities for improvement [Skinner,1978]. Skinner (1978,p111-115) provides a checklist that can be used to make the audit. Hayes and Wheelwright (1984,p31) also describe a framework that groups the diversity of manufacturing decisions that must be made over time and which collectively make up a manufacturing strategy. Skinner’s checklist and the framework of Hayes et al. largely overlap. Both separate “structural” manufacturing policies, those that deal with the “hardware” or “bricks, mortar, and machinery”, from policies that are considered as “infrastructural”. According to Hayes et al., decision categories are considered structural due to the long-term impact of the decisions, the difficulty of undoing a decision, the considerable investment that is required to implement a decision. Because infrastructural decisions are closely linked with specific operating aspects of the business, these decisions are made more frequently and generally require less capital investment. The reason to include these decision categories in the framework is that: “(...)their cumulative impact can be just as difficult and costly to change (if not more so)(...)”.

Tables 2-1 and 2-2 contain checklists of structural and infrastructural manufacturing strategy decision categories respectively. The contents of both tables is largely based on a compilation of the work of Hayes et al. and Skinner. A few explanatory comments:
• Not every decision that fits a particular category of decisions is the kind of decision that is typically considered part of a manufacturing strategy. The question of whether a specific decision is considered a strategic decision is highly situation-dependent. One of the criteria for judging the strategic content of a decision is the amount of money involved which clearly depends on the ‘size’ of the company;

## Examples

<table>
<thead>
<tr>
<th>Process span</th>
<th>What to make and what to buy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
<td>The extent to which distinct facilities should cover a range of process steps with respect to single product type (vertical integration) and the extent to which facilities should perform their processing steps on a range of product types (horizontal integration).</td>
</tr>
<tr>
<td>Facilities</td>
<td>The number of plants, their size¹, their location, and their focus or specialisation.</td>
</tr>
<tr>
<td></td>
<td>An important issue in this respect is the notion of a minimum and a maximum economic size for a manufacturing plant [Hayes et al., 1984, p.107].</td>
</tr>
<tr>
<td>Capacity</td>
<td>The capacity to be acquired, replaced, or disposed of, the type of capacity, the time, and the location. Hayes et al. (1984) distinguish four generic policies observed in practice:</td>
</tr>
<tr>
<td></td>
<td>1. Don’t built capacity until the need for it develops;</td>
</tr>
<tr>
<td></td>
<td>2. Outguess the market: if you expect a shortage of capacity, built additional capacity while you still can;</td>
</tr>
<tr>
<td></td>
<td>3. Buy for the long haul;</td>
</tr>
<tr>
<td></td>
<td>4. Follow the market leader(s).</td>
</tr>
<tr>
<td>Technology</td>
<td>At any plant, the designated amount of capacity can be acquired in terms of alternative technologies. Trade-offs include those between labour-intensive technologies and capital intensive technologies (automation), trade-offs that involve the integration of manufacturing steps (capital investments, reduction of inventories), and trade-offs that concern the specifications of different pieces of equipment (price, performance).</td>
</tr>
</tbody>
</table>

Table 2-1 Structural manufacturing strategy decision categories.
- The difference between a ‘Capacity’ decision and a ‘Technology’ decision is that capacity decisions deal with the acquisition, replacement, or disposal of capacity with respect to aggregate parts of the manufacturing system (e.g., plants, entire product lines), whereas technology decisions primarily focus on the choice between different types of equipment.

**Examples**

| Logistics: | The arrangement of the manufacturing planning and control structure, including the definition of planning levels (usually along the paths of the organisational hierarchy), the delegation of control authority, the location of the client-order decoupling point, the choice of MPS (Master Production Schedule) and FAS (Final Assembly Schedule) end-items, size and location of inventories, vendor selection, inter-plant co-ordination and purchasing policies. |
| Production | |
| Planning | |
| Materials | |
| Management | |
| Inventory Control | |

1 Niman(1992) and Baligh and Burton(1984,1981) focus their research on the trade-off between the degree to which a co-ordinating structure should incorporate the characteristics of a market or a hierarchy. This is considered one of the greatest challenges in organisational design.

| Workforce | Skill levels, selection, training, job content, wage systems. |
| Quality | Quality standards, process design and control, monitoring. |

Table 2-2 Infrastructural manufacturing strategy decision categories.

### 2.1.3.1 Examples of multi-plant manufacturing facility location strategies

This section describes a number of multiplant facility-location strategies that have been observed in practice. When designing a facility network, two issues are of particular importance: configuration and co-ordination [Porter,1986]. Configuration deals with the location of facilities and the allocation of production activities amongst the facilities. Co-ordination involves the integration of activities into a unified system. Despite this decision being only a small part of the issues involved in establishing a manufacturing strategy, this section is intended to improve the reader’s conception of the complexity and the diversity of the trade-offs regarding a manufacturing strategy.
Section 2.1.3 provides an extensive checklist of issues that strategic manufacturing management must take into consideration when designing a complete new manufacturing system. Only a brief description of the components of a manufacturing strategy will be given.

**Capacity Expansion**

Prior to taking a decision concerning the establishment of a new facility, management has identified the need for additional capacity by establishing new company-owned plants, probably within two or three years. Capacity expansion decisions typically deal with four dimensions: the *additional* capacity to be added, the *types* of capacity to add, *where* to allocate the additional capacity, and *when* to install the new capacity. A company’s capacity strategy determines decisions on both when to add capacity and how much capacity to add. According to Hayes et al. (1984), "Capacity should be added in chunks. The question is, how big should these chunks be, and when should they be added?" When a company decides to increase its manufacturing capacity substantially, it reduces the cost per unit and provides excess capacity. This reduces the risk of lost sales, but it increases risk of under utilisation (interest and maintenance costs) of the added capacity until demand reaches the point where the additional capacity is needed.

A firm’s facility strategy typically deals with where to add new capacity and what kind of capacity to add. The facility strategy is therefore an integral part of a firm’s capacity strategy. Both dimensions relate to the *location* of new facilities, the *size* of new facilities, and the *specialisation* of specific facilities.

**Domestic Multiplant Facilities Planning**

Multiplant facility planning deals with the development of a set of facilities that will continuously provide the specific manufacturing capabilities required by the firm’s products, markets and production processes. Multi-plant location strategies can be divided into two major categories: strategies dealing with domestic operations only and strategies that deal with globally dispersed operations. First, we discuss five basic facility location strategies that may be pursued by firms supplying domestic markets. [Hayes et al., 1984; Schmenner, 1983]:

1. The physical approach to multiplant facility planning simply concerns providing sufficient floor space, so that a firm’s employees can meet the requirements imposed by its long-term aggregate sales plan [Hayes et al., 1984]. It is based on ratios that can be used to forecast the physical space. The physical approach is often observed in high-growth companies. The growth rate of the market yields very short lead times for adding capacity. The company usually lacks accurate projections of its future need for specific types of capacity. Hence, additional rules
of thumb are necessary to enable the break down of the aggregate estimate capacity to that of individual plants and to determine the location of new facilities. The size of new facilities may be based on that of other leading companies or by comparison with those on previously established sites. The same reasoning may be used to determine the location of new facilities. Frequently, however, other considerations guide the decision like, for example, special demands on the infrastructure close to an optional location, the degree of traffic congestion, the attractiveness of the region surrounding a particular site and climatic conditions. Although this approach seems rather unsophisticated, the Hewlett-Packard Company, for example, uses a quite similar planning method. HP has established clear guidelines for all of its facilities and their surroundings which look exactly like HP’s plants in California;

2. The second approach seeks to determine the best set (number and size) of facilities and the customer regions to be served using the spatial distribution of demand, production economies of scale, overhead costs, and transportation economics. This approach is often observed in cases where transportation costs make up a substantial part of the total delivery costs of a product (basic material, bulky or heavy commodity-type products) or proximity to customers and markets is a key source of competitive advantage. A small company that is competing in a market that is dominated by a large competitor may decide to copy part of the facilities location strategy of the larger player by establishing sites in the neighbourhood of the facilities of the market-leader.

Short-term trade-offs include those between transportation costs and production costs. As production scale economies increase, the optimal plant size increases, thus reducing the cost per product so that the trading area of the facility expands as the cost benefits incurred trade off against higher transportation costs. In the long-term, however, the company must consider the pattern of market development. For example, in the early phases of market development, a single facility at the core of the market might be capable of servicing total demand. This approach promises greater economies of scale than using a number of smaller facilities located more closely to their specific market regions. However, when the market is developing towards or beyond the point of saturation, a number of equal-sized plants might become more appropriate, as a single plant would get too big. The company may decide to establish small plants close to specially selected market areas anticipating the market’s development and thus avoiding the complicated relocation of production that would be necessary if a centralised production facility is used initially. A distributed network of manufacturing plants is more difficult to manage, has higher initial costs and depends on the availability of efficient production technologies at the lower rates of production in the earlier stages of the market;
3. The third multiplant location strategy, also known as the process-plant strategy, is mainly concerned with the separation of the constituent stages of the production process on a plant-by-plant basis. The spatial distribution of facilities often receives less attention. Stages are dispersed by function to those locations where the greatest production efficiencies can be achieved (typically low-cost labour). Different facilities may very well be located at the same geographical location to compose a so-called campus facility. The size of campus facility may exceed that of a single facility if each facility is operated as a distinct organisational entity;

4. Strategy four is based on the concept of a focused facility: a facility with a limited task assignment in terms of the markets it serves, the volumes of its products, the degree of product customisation and the width of the process span that is assigned to it. A major difficulty of most traditional approaches is that after a firm puts together a set of facilities, it still faces fundamental decisions regarding which products and markets should be served by each facility and what adaptations should be made to facilitate those assignments. Contrary to the process-plant strategy, this approach considers both the product range that a facility must be capable of producing and the part of the process that it is held responsible for.

INTERNATIONAL MULTIPLANT FACILITIES PLANNING

A company can decide to start a manufacturing plant abroad for a variety of reasons. Ferdows (1989) has suggested five categories of reasons, of which the first three were found to be more prevalent than the latter two:

- **Access to low cost production input factors**: labour, materials, energy, and capital. Among the four factors cheaper labour and, to a lesser extent, cheaper raw materials and energy induced firms to establish a manufacturing plant outside their home country. Firms that seek to internationalise their manufacturing base consider the availability of cheaper capital only when choosing among alternative sites;

- **Proximity to market**. The presence of a firm with one or more manufacturing plants in a particular market may improve goodwill, product customisation and delivery;

- **Use of local technological resources**. A manufacturing base can provide an effective way to tap into local technological resources. Increased interaction between the company and its environment may result from using facilities close to universities, research centres, sophisticated suppliers;

- **Control and amortisation of technological assets**. A company that expands internationally may take advantage of manufacturing scale economies. If the company owns the facility abroad it is in control of its technological resources and may earn better returns than are yielded by partnerships, license agreements, and
PART I - INTRODUCTION

joint ventures;

- **Pre-emption of competition.** In a developing country that is showing increasing sales, a company may decide to establish a manufacturing base in order to pre-empt competition.

In contrast with domestic companies, firms that produce and source globally face a variety of additional issues. Besides cultural differences between employees—which to some degree also exist within domestic firms in multiracial societies like North America and Western Europe—these issues include [Cohen et al, 1989]:

- Duties and tariffs based on material flows, the impact of which must be considered when decisions have to be made concerning the choice of vendors and the location of manufacturing plants;
- Currency exchange rates fluctuate randomly and affect profits in each country of operation;
- The effects of significant differences in corporate tax rates among countries;
- Global sourcing which must be managed to account for trade-offs between longer lead times, lower costs, and access to new technologies;
- Market penetration strategies, local content rules, counter trade and quotas, all of which act to constraint the flow of products throughout the international supply chain;
- Product designs that may vary according to national markets;
- Centralised control of operations which is difficult to attain and therefore demands that appropriate incentives and transfer price mechanisms must be devised.

According to Fawcett (1992), two generic global manufacturing strategies have evolved and been widely implemented in recent years: the factor-input manufacturing strategy and the market access manufacturing strategy. The factor-input global manufacturing strategy is intended to improve the competitive position of a firm in its home market by identifying countries (regional economies) that have a comparative advantage in one or more stages of the production process and setting-up operations in these countries. The second type of global manufacturing strategy is the market-access strategy which is intended to increase the firm’s access to foreign markets. Viable market-entry strategies include exporting, licensing (third-party management and production), management contracts (management, 3rd party production), joint ventures (transfer of ownership), and local production. In many cases, firms implement a mixture of both strategies. A firm that is employing a factor-input strategy demonstrates a global orientation which leads it to move into global markets. The reverse happens when a firm that is participating in a large number of foreign markets
encounters cost pressures that force the company to take advantage of local economies. Ferdows (1992) states that looking at the primary reason for establishing a manufacturing site, together with an examination of the manufacturing process currently planned or actually being performed for that site, provides significant clues for understanding the strategic role that is assigned to a factory. Ferdows suggests six generic strategic roles (cf. Figure 2-2):

1. **Off-shore factories** whose only reason of existence is the local availability of cheap production factors. Their only purpose is to supply a small range of products at minimal costs;

2. **Source factories**, unlike off-shore factories, are responsible for a specific range of products or even a specific part of the company’s conversion process. Unlike off-shore facilities, source factories carry the responsibility for the development and/or the production of their products;

3. **Server factories** are designed to serve specific national or regional markets. Server factories have no responsibility for product or process changes. Unlike to off-shore factories, however, a server factory has greater autonomy in the management of information and material flows between the site and its customers;

4. **Contributor factories** also serve a specific national or regional market but they are assigned specific tasks to develop know-how for the company;

5. **Outpost factories** are intended to collect information. Companies which want to be present in a particular technologically advanced area usually prefer the acquisition of an existing factory or a joint venture instead of establishing a new factory;

6. **Lead factories** not only have to collect crucial information regarding new technologies but they also have to develop capabilities in that particular area.

![Diagram showing strategic roles of international factories](image)

**Figure 2-2** Generic roles of international factories. Source: Ferdows, 1992.
PART I - INTRODUCTION

2.1.4 (MI)LP-models supporting the development of a manufacturing strategy

(MI)LP-models for strategic manufacturing planning focus on the structural decision categories of the framework outlined in the previous sections — the decisions that deal with the “bricks, mortar, and machinery” [Skinner, 1978].

This section provides a brief survey of the vast amount of literature on facility location allocation models (Section 2.1.4.1), allocation models (Section 2.1.4.2), capacity acquisition, replacement, and/or disposal models (Section 2.1.4.3), and technology selection models (Section 2.1.4.4). This division into subsections corresponds to the fragmented development in the literature caused by the computational intractability of each of the models [Verter and Dincer, 1992].

For practical reasons, we only consider deterministic models. Despite the fact that capacity expansion and technology selection models can be modelled more accurately by using probabilistic models, deterministic models are easier for decision makers to understand [Mirchandani and Francis, 1990]. Moreover, stochastic models are difficult to solve.

The focus of this section is to describe the components of the various types of models in order to identify broad categories of models rather than to review the literature or to discuss the formulation of basic models. A detailed discussion of the literature is beyond the scope of this thesis.

2.1.4.1 Facility location models

Facility location models for production systems focus on how manufacturing facilities should be placed in relation to the spatially dispersed set of suppliers of raw materials, in relation to each other (supply of intermediate-stage products), and in relation to the spatially distributed set of customers for end items (e.g., distribution centres, markets) that is to be served by them. Facility location problems occupy a prominent place in the literature. Current, Min, and Schilling (1990) identify four possible causes for this: (1) facility location decisions are ubiquitous; (2) location decisions are often considered of strategic importance; (3) it is difficult to solve even the most basic models to optimality (cf. Section 2.3); and (4) location problems are often uniquely defined by their particular problem setting — both the constraints and the objective function of the models will vary for every application.\(^1\)

\(^1\) Brandaeu and Chiu (1989) found fifty-four underlying problem types that have appeared in literature and ten different objective-function types.
CATEGORIES OF OBJECTIVE FUNCTIONS
In a review of 45 papers on the subject of facility siting decisions, Current et al. (1990) uncovered 4 broad categories of objectives:

1. The largest category was cost minimisation which has been a traditional objective in facility location modelling. Cost reducing measures include transportation costs and fixed facility costs. Distance is the most commonly used surrogate for cost. Another surrogate for cost minimisation is the minimisation of the total number of facilities located;

2. The second most common category of objective functions consists of demand-oriented objectives. Typically, these objectives are based upon some measure of proximity or accessibility to a facility. Most facility location models ignore the possibility that demands may be influenced by pricing and the distance between facilities and customers [Hakimi and Kuo, 1991];

3. The third category is formed by profit maximisation objectives which focus primarily on product profitability differentials and market share;

4. The fourth category of objective-function types is made up of environmental objectives which may address issues like pollution and quality of life.

STATIC VERSUS DYNAMIC MODELS
Facility location and allocation problems are usually formulated as deterministic static—single-period—(MI)LP models. Static models span a single, typically large, period of time which is not subdivided into smaller periods as is the case in dynamic or multiperiod models. The set of indices on the parameters of dynamic models is augmented by special indices to distinguish the states that correspond to each of the periods.

Much work on locational analysis emphasises static rather than dynamic formulations and deterministic rather than probabilistic formulations [Mirchandani and Francis, 1990]. Attempting to introduce stochastic and dynamic elements directly may lead to a degree of refinement which far exceeds the other levels of abstraction employed, and which cannot be supported by the available data.

SPATIAL TOPOLOGY
The underlying spatial topology has great impact on the model structure and hence provides a well-accepted feature for categorising facility location models [Verter and Dincer, 1992]. Francis, McGinnis, and White (1983) distinguish: Planar location models that presume the spatial topology to be a plane. Consequently, the number of possible locations is infinite and planar distances represent the distances travelled. Network location models which utilise the underlying network structure. The nodes and arcs of the network constitute the set of possible locations and travel distances are
calculated from shortest-path network distances. *Discrete location models* that support the selection of sites to establish facilities from a finite set of possible locations selected via some prior analyses. The spatial topology underlying discrete location model is a network. A subset of the nodes of the network constitutes the set of possible locations. Fixed costs (or net profits) associated with opening (or closing down) facilities at the selected sites are also incorporated in the model. Discrete location models are often preferred in practice because in most cases decision makers consider a discrete representation a more accurate portrayal of the problem to hand [Mirchandani et al., 1990]. Moreover, it is often appropriate to use a discrete formulation because decision makers tend to consider a relatively small set of sites for locating new facilities.

**Prototype Discrete Facility Location Models**

Mirchandani et al. (1990) provide a reference for the state of the art in discrete location theory. They distinguish three prototype facility-location models:

1. **$p$-Median Problems ($p$-MP):** For given $1 \leq p \leq n$, the $p$-median problem is to establish $p$ facilities in $\rho$ of the $n$ potential locations and to supply each customer from a subset of the established facilities, such that the demands of all clients are met and such that the total costs thereby incurred are minimised;

2. **$p$-Centre Problems ($p$-CP):** For a given value of $p$, $1 \leq p \leq n$, the $p$-centre problem is to open $p$ facilities in $\rho$ of the $n$ potential locations and to assign each customer to only one of them, such that the maximum distance from any open facility to any of the clients assigned to it is minimised.

$p$-CP differs from $p$-MP in several respects. Whereas $p$-MP minimises the total costs of transportation, $p$-CP minimises the maximum distance between a customer and the nearest facility. Moreover, $p$-MP satisfies the node optimality property — if the finite set of potential sites is extended by regarding any point on any arc of the network as a potential site then, if an optimal solution exists, all established facilities are located at the nodes. The node optimality property does not apply to $p$-CP.

3. **Uncapacitated Facility Location Problems (UFLP)** deal with the supply of a single commodity from a subset of the potential facility locations to a set of clients with known demands for the commodity. UFLP seeks a minimum cost production transportation plan (in terms of the number of facilities established, their locations, and the amount shipped from each facility to each client) satisfying all demands. Facilities are assumed to have unlimited capacity. Unlike $p$-MP, UFLP associates a non-negative fixed cost with each potential facility site which is
incurred only if a facility is actually located at that site. Moreover, the number of facilities to be opened is not prescribed.

When the cost associated with the supply to a customer from an optional facility location is defined as either zero or one, indicating whether or not the facility located at that site can or cannot serve the customer, and only fixed costs are used to determine the quality of a feasible solution, the problem is referred to as a set covering problem. (A set of open facilities that is capable of serving all customers is referred to as a ‘cover’.)

EXTENSIONS TO THE PROTOTYPE MODELS
The prototype models have been used as optimisation models in their own right or have been employed as subroutines in more integrated models. However, the computational intractability of the prototype models is such that additional formulations should be added with utmost restraint. Several major characteristics of realistic facility location models include:

• TRANSPORTATION LINKS. Upper bounds on the product quantities that can be transported along certain routes; different modes of transportation: e.g., ship, vehicle, plane; duties and tariffs on material flows across national boundaries; restrictions on the commodity flow along a supply link, e.g., South Korea does not allow inbound shipments of Japanese manufactured products;

• PRODUCTS. Single-commodity models represent different products by a surrogate, aggregated product. Multi-commodity models distinguish several products;

• FACILITIES. Upper bounds and lower bounds on the product quantities that a facility can produce and ship per period of time; capacity that may result from policies that relate to the optimal size of facilities or from government regulations that force companies to maintain certain production levels in exchange for a license to establish a manufacturing facility (local content rules); upper bounds on the distances between facilities; restrictions on the range of manufactured products, for example, to take advantage of economies of scale;

• SUPPLIERS. The selection of suppliers who will negotiate long-term contracts in cases where the location of the suppliers affects the long-term layout of the materials flow and thus the arrangement of the production system. Sourcing policies;

• CUSTOMERS. Upper and/or lower bounds on the number of products supplied to customers to accommodate, for example, rules regulating interstate commerce (quotas). Policies that guide the allocation of customers to facilities. For example,
the single assignment policy requires that all demand for a customer is supplied by a single facility.

2.1.4.2 Allocation models

Allocation models depart from a given network of facilities mutually connected by transport links. Allocation models for production systems typically consider raw-material vendors, manufacturing plants and customers for end-items (e.g., distribution centres, transhipment centres or ‘landing points’, markets). Allocation models focus on the costs associated with the layout of commodity flows that emerges from an allocation of customers to facilities an allocation of manufacturing processes and product ranges to facilities, and/or an allocation of raw-material vendors to facilities. Models for production systems typically consider decisions like: the choice of which raw-material vendors to use, the allocation of vendors to plants, the type of contracts to use, the flows of raw materials from vendors to plants, the flows of intermediate-stage products between plants, the quantities of each product *type* to be produced at each plant, the allocation of customers to facilities, and the quantities of each product to be supplied to each customer.

Unlike facility location models, which consider the location of facilities and the establishment of supply relationships simultaneously, allocation models only deal with the latter. Therefore, many of the trade-offs underlying facility location models hold for allocation models as well. It also follows that, when compared to facility location models, ‘basic’ allocation models of equal ‘size’ are easier to solve. The relative simplicity of allocation models may be used to:

- **Obtain shorter problem-solving cycles** which improves the responsiveness of decision support systems that use the models for interactive problem solving;
- **Increase problem size**, e.g. by extending the number of vendors and/or customers, to obtain a more accurate portrayal of the underlying problem domain;
- **Incorporate issues usually unaddressed by facility location models** owing to performance considerations. For example:
  - Cohen et al. (1989) describe a comprehensive dynamic allocation model that incorporates many issues that relate to cross-border commodity flows\(^1\);
  - Mourits(1995,p107-127) describes a static ‘deployment’ model for the design of a distribution network which departs from a given geographical arrangement of

---

\(^1\) The objective-function of the deterministic allocation model described by Cohen and Fisher is non-linear. However, the solution procedure which they propose solves two related (MI)LP models.
manufacturing plants, warehouses, and customers. Mourits proposes a design framework for distribution networks in which each stage is supported by a model. In short, the first stage of the framework deals with the location of warehouses; the deployment model supports the second stage. The model focuses on a minimum cost (inventory and transportation costs) assignment of the tasks of each warehouse (keeping inventory, doing final assembly, or serving as transhipment center) with respect to each of the supply chains involved. The model balances requested upstream order lead times against calculated downstream order lead times. Supply frequencies are used to split the flow of commodities into supply batches. These batches are used to determine the cost of cycle stocks. In addition, using unit transportation capacities the model converts the supply batches into shipment quantities that serve as a basis for the calculation of the cost of transportation.

It is easy to envision a modified version of the deployment model that addresses the co-ordination of multiple plants. Mourits’ mechanism for calculating the cost of transportation can be of interest to a model for strategic manufacturing planning that focuses on a thorough evaluation of the costs of a distributed production system configuration using aggregate figures.

2.1.4.3 Models for capacity acquisition, replacement, and disposal

Facility location models do not consider the timing aspect of a decision to establish a new facility, as it is common practice to formulate facility location models (cf. Section 2.1.4.1) as static (single-period) models. Moreover, facility location models do not address the issue of the incremental pattern of the size of facilities. The notion of ‘size’ of facilities in capacitated plant location problems is usually derived from policies regarding the ‘optimal’ size of plants or from rough estimates on demand.

Capacity acquisition, replacement, and/or disposal models depart from a given spatial distribution of facilities. The literature on capacity decisions is mainly focused on capacity acquisition (expansion) problems [Verter et al.,1992; Rajagopalan and Soteriou,1994]. The capacity expansion problem aims to find an optimal set of expansion decisions which enables the firm to satisfy demand over a prespecified time horizon. Capacity expansion problems dwell on the trade-off between the economies of scale in capacity acquisition and the cost of having excess capacity. The objective is to
minimise the present value\(^1\) (discounted value) of a sequence of planned capital outlays associated with the investment decisions taken over the planning horizon. Instead of adding capacity, capacity may also be disposed of due to declining demand or replaced because the older equipment may have higher operating and maintenance costs than newer equipment or it may have deteriorated or become obsolete. When facing capacity shortage for a certain product, a firm can either purchase the necessary amount of capacity or convert some of the excess capacity for other products (if any) to satisfy demand.

Next follows a list of features that may be used to categorise capacity acquisition, disposal, and replacement models:

- **The number of facilities** for which transportation costs must also be considered when deciding upon the best location to put new capacity in place;

- **The support for multiple product types** with demand differentials between product types and where not every type of equipment is suited for use for each type of product;

- **Demand.** Demand may be conceived as deterministic or probabilistic. In deterministic models, the pattern of demand over time may be linear (using a constant growth rate), exponential, or decreasing with a saturation level. Finite horizon models generally support arbitrary demand fluctuations;

- **The feasible expansion, disposal, and replacement sizes:** The primary decision variables in a capacity model are the equipment sizes. It makes a big difference in terms of computational complexity whether equipment sizes may take any value or whether they must be selected from a set of discrete alternatives. Continuous models are much easier to solve.

\(^1\) If capital outlays are spread over several years, the time value of money must be taken into account. Evaluating a sequence of investment decisions using the present value of money takes into account that, if you have the option, it is always a better idea to postpone spending money. Not spending the money yet gives you the option to, for example, put the money in a deposit at a given interest rate. The Future Value (after \(n\) periods) \(FV_n\) of the amount of money put into deposit \(PV_0\) at an interest rate \(r\) can be calculated by the equation of compound interest:

\[
FV_n = PV_0(1 + r)^n.
\]

Finding present values \(PV\) (discounting, as it is commonly called) is simply the reverse of compounding (refer to: Weston and Copeland (1989), chapters 5 and 6):

\[
PV_0 = FV_n(1 + r)^{-n}.
\]

The discount rate can have a significant impact on the model outcomes since capacity acquisition, disposal, and replacement models cover decisions that span relative large periods of time.
The number of equipment sizes available to expand or replace capacity may be small, which may give rise to non-continuous problem formulations [Rajagopalan et al., 1994];

- **The structure of the cost function and the cost factors included.** Capacity acquisition cost functions are usually concave to represent economies of scale. Piecewise concave cost functions are particularly useful for modelling the availability of different technologies for different ranges of expansion sizes.

![Figure 2-3](image_url) (Piecewise) concave costs representing economies of scale.

Rajagopalan et al. (1994) describe two rudimentary (e.g., single facility, single product) IP models for capacity expansion and capacity expansion with disposal and replacement respectively. Both models consider discrete capacity increments only. The capacity expansion model uses integer variables denoting the number of units of a capacity type of a particular size purchased in some period \( t \). The model with disposal and replacement uses integer variables that denote both the purchase of a number of pieces of equipment of a particular size in period \( t \) and the disposal of that same equipment in a later period \( j \).

The capacity expansion model contains a cost function that includes initial purchase costs, operating costs, and maintenance costs. The operating and maintenance costs are assumed to be independent of the degree of capacity utilisation. In cases where these costs vary with the amount of capacity being used, it is assumed that the variable costs depend only on the total amount of demand met and, since the demand is known, the variable costs can be omitted from the model. By using the effective capacity in period \( j \geq t \) of a piece of equipment bought in period \( t \), the purchase cost, and the fixed operating / maintenance cost, Rajagopalan et al. can model:

- **Time-dependent purchase cost differentials**, by letting the purchase cost vary per period;
- **Economies of scale**, by letting the initial purchase costs increase less than proportionally with the size of the capacity;
PART I - INTRODUCTION

- Increasing maintenance costs or loss of effective capacity with age, by letting the operating costs and size parameters increase over time.

- The functional constraints. All capacity acquisition, disposal, and/or replacement models include constraints that balance the available capacity with demand. Other constraints may relate to capital budgeting constraints or capacity conversion (product range, location).

2.1.4.4 Technology selection models

Firms pursue their capacity acquisition strategies by choosing among a set of alternative technologies. Capacity types in the capacity expansion context constitute technologies in the more general technology selection problem [Verter et al., 1992]. Fine (1993) provides a comprehensive survey on the subject of new manufacturing technologies, the tendency to move towards more automation and integration and the range of models for technology evaluation and adoption.

Performance evaluation models for detailed operational analysis of the operational implications consider physical measures such as flow times, queuing times, lead times, inventory levels, and production rates. The level of detail employed by these models makes them unsuited to the strategic problem.

Economic evaluation models examine technologies on the basis of their financial impacts. These models provide a valuable tool for strategic management [Fine, 1993; Verter et al., 1992]. Fine distinguishes two major categories of economic evaluation models:

1. Models that address the issue of optimal timing of investment in technology. This literature assumes that the best available technology is known and available to all firms. New technologies are usually characterised as having lower operating costs than previous technologies;

2. Models that address the question of what technology should be adopted. These models assume that there are multiple technological options available. Alternative technologies may have different cost structures (e.g., costs of purchase, operating cost, production cost), quality, and flexibility implications.

Some of the benefits of the economies of scale provided by dedicated machinery, the economies of scope provided by Flexible Manufacturing Systems (FMSs), and the benefits of new technologies in general (e.g., lower direct costs or improved product quality) to be traded off against the capital expenditures of these investments are not easy to quantify:
• Verter et al. (1992), for example, mention the differences in how different authors treat the variable costs associated with FMSs. Some treat variable costs as linear functions of volume, some assume that the variable costs are linear and technology independent due to the dominance of material costs to other variable cost factors, others state that FMSs have higher variable costs than dedicated transfer lines because their structural complexity makes them more error prone;

• Much research currently focuses on operational measures for all types of flexibility e.g., Benjaafar et al. (1993), Gupta(1994).

Hence, many models on the subject of technology selection are presented as contributions to a conceptual understanding of some aspects of the trade-offs involved in adopting new technologies [Fine, 1993].

The demand in (MI)LP models that focus on technology selection is deterministic, as is the demand for (MI)LP capacity expansion models. Models that focus on the question what technology to adopt are formulated as both static and as dynamic models.

2.2 Tactical manufacturing planning

Production planning is concerned with the determination of production, inventory, and resource levels (e.g., the work force) to meet fluctuating demand requirements. The production planning problem is how to determine a low-cost feasible cumulative production plan that satisfies cumulative forecast demand, given the current state of the production system with regard to the available resources, current inventory levels, and the work-in-process levels. Usually the available resources are assumed to be fixed during the planning horizon and the production planning effort is directed towards the best utilisation of those resources for the given demand requirements [Hax, 1978].

Performing the production planning activity on a regular basis, or when conditions indicate that it is necessary, has several benefits [Vollmann et al., 1992]. It institutionalises the planning process and forces consideration of changed conditions and trade-offs. Moreover, the routine keeps the information channels open for forecast revisions, significant machine failures, different conditions, new opportunities.

Whenever —due to variations in the demand, available capacity, or cost structures—the conditions affecting the production process are not stable in time production should be planned in an aggregate way. The length of the planning horizon is dictated by the nature of the dynamic variations. If seasonality plays a significant role, a planning horizon covering at least a year should be used. Usually it will be impossible to
PART I - INTRODUCTION

consider every fine detail associated with the production process for such a long planning period; therefore, production planning is done by using an aggregate view of the production facilities, the products, the demand for products, and even the time unit. Moreover, aggregate forecasts of demand have reduced variance, aggregation of items can reduce computational cost, and aggregated figures may improve the manager's overall insight [Gelders and Van Wassenhove, 1981].

Aggregation forces the use of a consistent set of measurement units. Furthermore, at the highest level, the production plan (cf. figure 1-2) provides key communication links from management at the business unit level to manufacturing, as well as between manufacturing and the other functions that participate in the business [Vollmann et al., 1992]. The plan must therefore be stated in units that are meaningful to other functional areas, as well as to 'lower' manufacturing planning echelons, so that detailed planning and control decisions may be kept in concert with the strategic objectives of the business. Suitable units might be the monetary value of total monthly or quarterly output, or of aggregate units of output per month, while the MPS is stated in output quantities per week.

Industrial companies often have several plans at different levels of aggregation [Thomas and McClain, 1993]. Less aggregated plans involve smaller time units and end conditions consistent with longer-term plans.

2.2.1 Structuring the aggregate manufacturing planning process

In order to structure the production planning activity, several decisions have to be made. Thomas et al. (1993) identified six such decisions:

1. The number of production plans, e.g., the highest-level aggregate production plan spanning one-year, a quarterly plan, and a one-week plan that is used to release work-orders;

And for each planning level:

2. The time horizon. The most immediate portion of the production plan is usually protected from changes. The order lead time for raw materials and component parts usually requires that orders have been issued, while the stability on the shop floor requires that nearby commitments to manpower and equipment that have been made are maintained. The production plan is made to take effect after a shorter period of time. Decisions within that period are firm and were determined by the previous plan.
Usually, explicit time fences exist to hedge against different types of uncertainty — the frozen horizon. The amount of increase or decrease of production volumes may be constrained;

3. **The time unit** (months, weeks, shifts);

4. **The level of aggregation of the products** (one aggregate product, several product categories, families, or types, or full detail);

5. **The level of aggregation of the production resources**;

6. **The frequency of replanning**. The frequency replanning depends on the stability of the business, the cost of planning (e.g., data gathering, what-if analysis), and the ability to monitor performance.

### 2.2.2 Aggregate planning decisions

The most important input to the production planning process are update sales forecasts which cover the entire planning horizon. These forecasts must include all possible sources of demand for manufacturing capacity, including interplant transfers and service parts.

**UNCERTAINTY: SAFETY STOCK AND SAFETY LEAD TIME**

Uncertainty with respect to demand quantities may be overcome by building up safety stock. Safety stock is subtracted (i.e., not accounted for) from planned or current inventory levels. Using safety times created by planning orders earlier than necessary is a means to hedge against substantial uncertainty with respect to the timing of demand (or the availability of additional capacity).

**RESPONDING TO CHANGES IN DEMAND**

Assuming that the firm's fixed production resources (i.e., plant, equipment) cannot be changed during the planning horizon, there are several ways to respond to changes in demand. These ways can also be applied in some mixed form to create a larger number of alternative production plans [Thomas et al., 1993; Vollmann et al., 1992; Hax, 1978]:

1. A production smoothing strategy involving holding a relatively constant production rate and using the accumulation of inventory and possibly even planned backlogs to satisfy demand peaks;

2. A chase strategy, where production levels follow demand by subcontracting, by taking on temporary personnel, or using the regular work force to work overtime. Low demand periods may be used to plan staff vacations or training, and for experimentation with new production methods which, although useful, do reduce output rate;
3. Turning away demand permanently whenever the marginal cost of production exceeds the marginal benefits (including future losses).

**COST FACTORS AND TRADE-OFFS**

Thomas et al. (1993) mention five types of cost factors that may be used to compare alternative production plans (The first four types of cost factors are more or less similar to the cost elements mentioned by Hax in 1978):

1. Regular production costs. These are the fixed and variable costs incurred in producing a given product type in a given period of time;

2. Costs of changing the production level, perhaps by subcontracting or hiring additional personnel;

3. Costs of oversupply, including inventory carrying costs and the risk of obsolescence;

4. Costs of not satisfying demand or not satisfying demand on time (backlogging costs);

5. Costs of alternative methods of production, possibly by means of subcontracting.

In practice, it is often difficult to obtain accurate cost estimates; backlogging costs are particularly difficult to measure. However, it is important to notice that for the purpose of using these cost estimates as input to the models, it is their relative proportions that matter. The cost estimates that are used as input to models should represent true economic opportunity costs, in so far as possible, in the light of the models that are used and their outcomes [Thomas et al., 1993].

**FEASIBILITY**

To determine the feasibility of a given production plan, the production quantities are translated into capacity-resource requirements, usually on an aggregate basis: e.g., man-hours, machine-hours, key-facility-hours, or some output measure. The feasibility then follows from the estimates on the availability of capacity resources. Vollmann et al. (1992) refer to this activity at the highest level as ‘Resource Planning’. This level of planning involves capacity acquisition, replacement, and disposal decisions (cf. Section 2.1.4.3).

**2.2.3 Disaggregation**

Aggregation would be meaningless if we were not able to disaggregate back to the level of detail that is necessary to issue work orders. The aggregate decisions constrain the disaggregated actions. Clearly, consistency with higher-level plans and feasibility
are the main objectives and constraints of the disaggregation process [Gelders et al.,1981]. Hierarchical production planning (cf. Section 2.2.4.2) is one approach to aggregate capacity analysis that is based upon disaggregation concepts.

2.2.4 (MI)LP-models for aggregate manufacturing planning

Hax (1978) describes a classification of models for aggregate production planning that is based upon the structure of the cost components. Hax distinguishes two types of models that can be formulated as (MI)LP models:

1. **LINEAR COST MODELS.** Many models for aggregate planning assume linear costs. Linear cost models have the advantage that it is easier to grasp the trade-offs underlying these models and that they can be solved efficiently by using standard LP solvers. The disadvantage of these models (and that of fixed-cost models) is that the linearity assumption cannot deal with non-linear and discontinuous cost structures. Convex cost structures (i.e. non-decreasing marginal costs) may result from special measures taken to increase production beyond regular levels, e.g., due to working overtime, hiring additional temporary labour, and irregular use of subcontracting. Concave costs structures can arise from set-up charges, volume discounting, and production efficiencies of scale. Hax (1978) argues that the linearity assumption is less restrictive than it appears. Cost structures might behave linearly within the range of interest and convex separable functions can be approximated to any desired degree of accuracy by means of piecewise linear functions and solved as an LP. Concave cost patterns occur frequently in practice. A specially important class of concave-cost production planning problems is represented by the lot-size problem, where the objective is characterised by fixed set-up costs and linear variable production and inventory costs;

- **LOT SIZE MODELS (FIXED COST MODELS).** In batch-type production operations a cost is incurred when setting up the production facilities for a given run. Integer variables must ensure that each production increment (batch) results in set-up costs. The set-up costs constitute a fixed component in the objective function of the model.

Both types of mathematical programming models have been used in many practical situations and researched extensively [Thomas et al.,1993]. Owing to recent advances in information technology which allow managers to acquire and organise production data on a timely basis, a renewed interest in the application of mathematical programming to production planning may be observed [Shapiro,1993]. As a result, many managers are seeking DSSs based on models to help them unravel the complex interactions and ripple effects inherent in these data.
2.2.4.1 Monolithic models

Several attempts have been made to formulate models that integrate decisions in production planning and scheduling. (Shapiro (1993), provides a survey of single/multiple-item, single/multiple-stage dynamic lot-size models.) In this approach, a single detailed model (monolithic formulation) is formulated to determine optimal planning and scheduling decisions to be solved on a rolling horizon basis. The problems addressed by these (M)ILP models deal with intermittently produced items with associated set-up costs, which are stored in inventory until they are needed. Aggregate production planning models typically assume that the fixed capacity cannot be changed and instead deal with ways of temporarily adding capacity: overtime, extra shifts, subcontracting. Some models assume that labour is the only constraining resource. The objective is typically to find the lowest-cost plan that meets the sales forecast (with respect to both timing and sizing) considering regular and irregular production costs, inventory costs, and constraints on shared capacity resources. Multi-stage models even consider the co-ordination of the timing (by using lead times) and sizing of production runs of items produced at each of several stages allowing general product structures. Typically, the monolithic mathematical program is solved approximately for each planning period with only the immediate period’s decisions being implemented [Graves,1982].

Their computational intractability is not the only disadvantage of monolithic models. Practical problems can become very large; the number of variables makes it almost impossible to review the plan [Meal,1984;Hax and Golovin,1978]. These models also require data (such as the forecast demand for every item over a complete seasonal cycle) that is difficult to obtain, while the resulting planning process becomes unrealistic due to the magnitude of the forecast errors inherent in such detailed long-term forecasts [Meal,1984;Gelders et al.,1981;Hax et al.,1978]. Furthermore, the centralised approach of solving the production planning and scheduling problem with a single model does not conform to industrial practice which requires that decisions should be made where they belong [Meal,1984;Gelders et al.,1981].

2.2.4.2 Hierarchical production planning

An alternative to the use of a single large model to tackle the production planning is a hierarchical decision procedure. Hax and Meal (1975) introduced the concept of hierarchical production planning which uses a series of models which are typically solved in a sequential fashion. As one goes down the hierarchy, the length of the planning horizon decreases while the level of detail increases. In the hierarchical
approach, higher level decisions impose constraints on lower level actions, whereas lower level decisions provide feedback to evaluate the higher level actions.

The advantage of the hierarchical approach to production planning is that the partitioning (disaggregation) of the overall planning problem can (and should) be tailored to follow organisational lines. Decisions which must be made at the corporate level are centralised; those which can be made locally are delegated to the appropriate levels. As a result, managerial input is possible at each stage. Moreover, it is only necessary to provide information at a level of aggregation that is appropriate to the decision. Finally, the resulting subproblems are less difficult to solve while their outcomes are easier to review.

The disadvantage of hierarchical planning procedures is that the submodels are solved independently, and that, regardless of the co-ordination scheme that is being used, there is no guarantee of the optimality of the resulting decisions. At best, hierarchical planning systems can only optimise each of the subproblems.

**THE HAX-MEAL HIERARCHICAL FRAMEWORK**

The hierarchical structure initially proposed by Hax and Meal (1975) and used by Bitran and Hax (1977), amongst others, is described many times in literature [Bitran and Tirupati, 1993; Hax and Golovin, 1978]. The Hax-Meal framework is based on three levels of product aggregation:

1. **Items** are the final products to be delivered to the customer;

2. **Families** are groups of items which share a common manufacturing set-up cost. Aggregating items belonging to the same family provides a means to accomplish economies of scale;

3. **Types** are groups of items belonging to the same family, whose production quantities are to be determined by the aggregate production plan. Items that belong to a type normally have similar unit-costs and similar demand patterns.

Obviously, the number of levels in the product structure needs to be tailored to the characteristics of a particular setting. For example, to reduce the number of distinct items in an assembly-to-order production environment, it is probably better to use MPS items instead of FAS units.

The planning process essentially consists of three steps. First, given the forecast demand and the available capacity resources (in this case, only one: labour), the aggregate plan for the product types is determined by minimising overtime and holding costs. The aggregate plan specifies the inventory and production levels for each type.
In the second step, the aggregate plan is disaggregated to obtain production quantities for each family. Family set-up costs are minimised for the inventory levels for each type obtained from the aggregate plan. Further disaggregation of the family production lots to determine item quantities is performed in the third step. In all cases, only the results for the first period are implemented. Hax and Meal solve the aggregate planning subproblem as a linear program. The family disaggregation problem for the immediate time period is solved by a heuristic.

Graves (1993) observes that the Hax-Meal hierarchical framework seems to work well provided that the cost factors that are accounted for at each stage are indeed the primary cost factors. That is, overtime costs and inventory holding cost associated with the product types are dominant, while the family set-up costs are secondary in importance and magnitude. Graves (1993) proposes an alternative iterative solution method that produces a feasible solution to the overall planning problem. Such a solution is based on a co-ordination scheme that treats the aggregate planning problem and the family disaggregation problem on terms of equality. Both subproblems are derived by separating the Lagrangean relaxation of the overall (monolithic) planning problem that is obtained by bringing the constraints that link the family inventories to the inventory of the type into the objective function. The Lagrangean multipliers may be conceived as the marginal cost of having a mismatch between the total inventory for a type and the sum of the inventory levels of the families contained in the type. The Lagrangean multipliers that appear in the objective function of both subproblems determine the degree of inconsistency between both subproblems and thus form the basis of the co-ordination scheme. By solving both subproblems for fixed values of the Lagrangean multipliers at each iteration, a lower bound is obtained for the optimal value of the overall problem. This particular solution to the Lagrangean is then used to construct a feasible solution to the overall model in order to obtain an upper bound for the optimal solution. In this manner, the solution procedure brackets the optimal solution value. It is possible to stop the solution procedure when the best feasible solution is sufficiently close to the best lower bound generated so far, or after a pre-set number of iterations, depending on whichever occurs first.

2.3 Using computational intractable models for decision support

Algorithms are commonly characterised by a measure related to their so-called time complexity function which expresses the largest amount of time required for solving an arbitrary model instance of a specific size [Mirchandani et al., 1990]. If there is a polynomial function of that size that serves as a bound for the time complexity
function, the algorithm is called polynomial (time-bounded) or efficient. If the time-complexity function cannot be bounded, the algorithm is called exponential.

The theory on computational complexity is defined with reference to decision problems which, unlike optimisation problems, have only two possible answers: 'yes' or 'no'. The most direct way of deriving a decision problem from a given optimisation problem is to introduce a threshold $k$ and to ask whether or not the optimisation model with a given data set has a feasible solution of maximum value $k$. Class $P$ consists of decision problems for which this answer can be provided in polynomial time. Class $NP$ consists of decision problems for which, in all likelihood, such a polynomial bounded solution algorithm does not exist. The feasibility of a given instance of a decision problem in $NP$ can only be affirmed in polynomial time provided that a structure containing information that is related to the feasible solution exists. For example, decision problems associated with the prototype facility location allocation problems discussed in Section 2.1.4.1 are in $NP$. To confirm the existence of a feasible allocation of facilities, it suffices to provide a vector of zero-one values indicating which facilities have actually been established and to solve the resulting transportation problem to check the remaining constraints in polynomial time.

A problem is $NP$-complete if it is in $NP$ and if showing that it is in $P$ would imply $P=NP$. More specifically, a polynomial bounded algorithm for solving the problem could be used once as a subroutine to obtain a polynomial bounded algorithm for every problem in $NP$. A problem is $NP$-hard if a polynomial bounded algorithm for it would result in a polynomially bounded algorithm for every problem in $NP$. The problem itself need not lie in $NP$ and the algorithm may be used as a subroutine more than once. Each of the prototype facility location allocation problems discussed in Section 2.1.4.1 is $NP$-hard [Mirchandani et al., 1990].

There are a number of ways to deal with the computational intractability of, for example, facility location models:

- Use dedicated exact solution algorithms that take advantage of the special structure of a specific optimisation problem. For example, standard solution algorithms can be used to solve capacitated facility location models of the order of 100 locations and 100 customers with a reasonable computational effort [Jacobsen, 1982]. Beasley (1988) presented a solution procedure based upon a Lagrangean relaxation solving problems up to 500 potential locations and 1000 customers;
- Use approximate solution procedures (heuristics). Many heuristics use the fact that for a given set of open facilities the problem is reduced to a transportation problem
PART I - INTRODUCTION

which can be solved efficiently. Heuristics can handle very large problems up to and exceeding the order of 1000 potential sites and 1000 customers;

- Reduce the dimensions of the problem by aggregation and pre-screening options in order to eliminate dominated ones.

The disadvantage of using dedicated solution algorithms, as opposed to using general-purpose Solvers, is that if these algorithms are used the structure of the models cannot be changed. Moreover, systems that are intended to use dedicated solution algorithms must often be tailored to meet non-standard interfacing requirements: e.g. data structures and control parameters. Note that both the use of heuristics and the reduction of the size of the problem by aggregating data and pre-screening alternatives embodies an element of suboptimality.
3. SURVEY OF EXISTING TOOLS FOR (MI)LP-BASED DECISION SUPPORT IN PRODUCTION PLANNING

OUTLINE
This chapter provides a survey of existing tools that support the use of (MI)LP-models for strategic and/or tactical production planning. Basically two types of tools exist: (1) Decision Support Systems (DSS) which are typically based on built-in (MI)LP-models; and (2) (MI)LP-modelling systems. To improve the reader's image of these systems and to facilitate comparison of the packages that are reviewed, a functional architecture for a stereotype modelling system is presented.

This chapter outlines three (MI)LP-based decision support systems: DP-PSS, PHYDIAS, and SLAM. DP-PSS is intended to support both strategic and tactical planning within the context of a spatially distributed production system. SLAM and PHYDIAS focus exclusively on strategic location-allocation issues that relate to integrated production and distribution networks. The particular reason for outlining these three systems is the fundamental differences between the design of the underlying optimisation model(s).

Outlines of four (MI)LP-modelling systems are presented: AMPL, gLPS, and LPFORM, which have an unrestricted application domain, and LOGS which focuses only on logistics planning problems. In addition to these complete modelling packages, this chapter also describes the Activity/Constraint-graph formalism which constitutes the basis for the (MI)LP-modelling scheme presented in the second part of this thesis.
3.1 Introduction

This chapter provides a survey of existing tools for incorporating (MI)LP-technology into strategic and/or tactical production planning. Today's decision makers basically have two alternatives to the option of developing a customised (MI)LP-based DSS:

1. The first option is to use a DSS based on built-in (MI)LP-models tailored to the needs of a specific application area. Quite a few of these systems have been developed for strategic and/or tactical production planning purposes including, for example, Lynx [Numetrix, 1994], Phydias [Bender, 1992], and SLAM [Nunen et al., 1980; Boumans, 1991]. Section 3.3 provides an extensive discussion of (MI)LP-based DSSs for production planning;

2. The second option is to use an (MI)LP-modelling system. The primary reason to consider using a modelling system is that these systems offer great flexibility. The majority of today's (MI)LP-modelling systems are general purpose modelling systems including, for example, AMPL [Fourer, Gay, Kernighan, 1990], gLPS [Collaud et al., 1994] and LPFORM [Murphy et al., 1992a]. Only few systems focus on production planning alone: e.g., LOGS [Brown, 1986] the application domain of which is restricted to production planning problems. Section 3.4 provides an extensive discussion of (MI)LP-modelling systems.

The criteria for comparing both types of systems are qualitative in nature. Qualitative criteria, as opposed to quantitative criteria, cannot be related to a measurable quantity. In this survey, we consider two types of qualitative criteria: qualitative-objective criteria and qualitative-subjective criteria. Evaluating qualitative-objective criteria is easy; the system either satisfies a criterion or not. The evaluation of a qualitative-subjective criterion is more complicated, because a system usually partially satisfies the criterion. In this survey, when dealing with subjective criteria, we have decided not to place values on the score of a system with respect to a criterion using a common non-numerical scale, e.g.: ++, +, +/−, −/−, −. Using such a scale would mistakenly suggest that the purpose of this survey is to make a mutual comparison between the systems being discussed. Therefore, the discussion of subjective criteria is restricted to an enumeration of the features that apply to the criteria.

The criteria for comparing (MI)LP-based DSSs (cf. Section 3.3.3) and the criteria for comparing (MI)LP-modelling systems (cf. Section 3.4.3) are derived from a functional analysis of using a stereotype system from either category and the reference
architecture presented in Section 3.2. A reference architecture\(^1\) represents a (software) system in terms of a structure composed of relatively independent, interacting components and in terms of globally defined tasks of these components [Biemans, 1990, p35]. A component’s globally defined task is a mere characterisation of its task — its not a complete specification of the task. Section 3.2 presents an aggregate reference architecture of an hypothetical (MI)LP-modelling system. The purpose of the reference architecture is to provide the user with an image of the functional architecture of both (MI)LP-based DSSs and (MI)LP-modelling systems, to introduce some of the terminology that is applied in this area, and to discuss some basic design considerations that influence the functionality of the system.

Although (MI)LP-based DSSs and (MI)LP-modelling systems have many differences in terms of flexibility, user-friendliness, and functionality, their functional architecture shows much resemblance. In fact, the two major differences between both type of systems are:

1. A model management subsystem (cf. Section 3.2.5) is a necessary subsystem of any (MI)LP-modelling systems whereas none of the (MI)LP-based DSSs that we encountered contains a model management subsystem;

2. A model-independent database subsystem (cf. Section 3.2.4) is an indispensable part of any (MI)LP-based DSS whereas some (MI)LP-modelling systems only contain a database management system for downloading and uploading data from and to external relational database systems while others use the representation of models to store data. A ‘full featured’ database subsystem also contains a data dictionary subsystem which provides facilities to model and maintain the structural relationships among the elements of the database in agreement with the logical structure of the data.

We have choosen to make a reference architecture of a full-featured (MI)LP-modelling system because the task description of the model-management subsystem is more general than that of a (MI)LP-based DSS.

### 3.2 A reference architecture

Amongst the differences between modelling systems, in terms of the level of modelling skills that is required for their effective use, are the efficiency of the modelling language, their debugging and optimisation capabilities, their support for data handling and model management, their support for analysing solutions, and finally the efficiency

---

\(^1\) Biemans uses the term reference model. To avoid ambiguous terminology (cf. Section 3.4.5), we prefer to speak of a reference architecture.
PART I - INTRODUCTION

of their implementation\textsuperscript{1}. The primary objective of any modelling system, though, is to express a problem in a way in which human modellers understand it rather than the form in which algorithms solve it [Fourer et al., 1990; Fourer, 1983]

An obvious approach to the decomposition of the global task of a modelling system, is to make a functional analysis of the process of using the system. At this stage, it suffices to consider the process of using the system as being composed of three clusters of activities: configuration, experimentation, and reporting. Figure 3-1 depicts this decomposition by means of Structured Analysis (SA) [Ross, 1977]. Boxes represent activities. Arrows entering from the left depict input relationships, whereas outgoing arrows on the right represent output relationships. Arrows entering from above indicate procedures or directives which guide the execution of the activity. Arrows entering from beneath refer to resources capable of implementing the activity. Figure 3-5 provides a more detailed decomposition of the use of (MI)LP-modelling systems. A brief description of the clusters of activities depicted in Figure 3-1 follows:

A1. CONFIGURE SYSTEM which concerns the formulation of models in terms of the system's modelling language and the creation of the necessary data structures that serve to contain the data that is needed to conduct experiments;

![Diagram](image)

\textbf{Figure 3-1} Aggregate functional analysis of using an (MI)LP-modelling system.

\textsuperscript{1} In a limited context, the term (MI)LP-modelling system is used to refer to a system that accepts a complete representation of a model as input from the user and which serves as an intermediate between the user and the solver. Alternatively, in a much broader sense, the term is used to refer to a system that includes additional subsystems like a database subsystem and a model-base subsystem. The latter is also referred to as (MI)LP-modelling environment.
A2. **CONDUCT EXPERIMENTS** which concerns the quantification of models, solving the models by invoking an (MI)LP-solver, uploading the results, preparing the data for the creation of solution reports, and possibly conducting sensitivity analyses;

A3. **CREATE REPORTS** which concerns the creation of reports containing a description of the structure of the model and condensed views of both the input parameters and the results of applying the model, analysing the results, and changing either the structure of the model or the values assigned to the input parameters.

From this decomposition we derived groupings of logically related tasks from which we inferred the components of the reference model. Note that the actual implementation of existing systems does not necessarily have to mimic these components. Some subsidiary components may not exist, for example, some modelling systems lack a data storage facility, whereas others lack a solution analyser subsystem. Moreover, from a software-engineering point of view, it may be beneficial to assign tasks to software modules which are allocated to different modules in the reference model.

Figure 3-2 illustrates the reference model of the (MI)LP-modelling system. The components that make up the reference model, are derived from Murphy, Stohr, and Asthana (1992). The following paragraphs outline the global task of each of the components of the reference model and basic design considerations that influence the functionality of the system.

![Diagram](image)

*Figure 3-2 A reference model of an (MI)LP-modelling system.*
3.2.1 The user interface

The User Interface is responsible for creating output from commands and data generated by the other components of the system and for translating input from the user to commands and data that other components of the system may use to take appropriate action. Sprague and Carlson (1982, pp. 214-217) discuss a reference model for the User Interface component—or ‘dialogue component’ as they call it—of a DSS. Although the User Interface constitutes a significant part of the design and implementation of graphical modelling systems in particular, we will not elaborate on the design of user interfaces.

3.2.2 The model processor and the (MI)LP solver

The Model Processor forms the intermediate between the modelling system and the solver. The description of an (MI)LP model, using the system’s modelling language, an associated set of data, and a sequence of control commands to guide the optimisation process, serves as input to the Model Processor. The model representation need not be explicitly stated (cf. Section 3.4.2). LPPFORM [Murphy et al., 1992a] for example, contains a model processor subsystem that is capable of constructing a complete statement of a model from component parts (Section 3.4.10, discusses this ‘piecemeal’-approach more extensively). The Model Processor merges the model structure with data elements that correspond to the non-structural model coefficients and RHS-constants and constructs a model representation suitable for the solver.

The Solver’s task is to solve the model obtained from the Model Processor. Examples of (MI)LP solvers are CPLEX [CPLEX Optimization, 1990], LINDO [Schrage, 1989], MINOS [Murtagh, Saunders, 1987], and OSL [IBM Corporation, 1991]. (MI)LP-modelling systems are offered both with and without integrated solver(s). A modelling system is said to contain an integrated Solver if the Model Processor communicates with the Solver via a so-called Application Programmers Interface (API). An API provides a collection of functions which the Model Processor can use to feed the model structure and the data into the Solver, to choose and initialise the solution algorithm, and to transfer the output of the Solver back into the Model Processor. From the user’s point of view the advantage of an integrated Solver is primarily the speed that is offered by this type of linkage. At the very least, for faster translation the Model Processor produces the (MI)LP matrix in a format native to the Solver, but both systems may also share the same memory which is obviously extremely fast. Many companies, however, install shared software resources like solvers, databases, and MRP-systems on separate computer systems that run only a small number of applications. These so-called Application Servers are integrated in a network of
computers. Authorised users throughout the organisation use these applications through a physical network connection and a data exchange protocol. In heterogeneous network configurations in which the Application Server usually runs on a Unix platform whereas the local workstations and PCs that run the modelling system have Unix, DOS/Windows or OS/2 installed, the communication between the two systems may be limited to exchanging files and remote execution of commands. Hence, it may not be possible for a modelling system to communicate with the Solver by using function calls. Most (MI)LP-modelling systems therefore are capable of generating an MPS file format representation of the problem. The MPS format (refer to: IBM Corporation, 1991 or almost any user guide of a modelling system) has grown into an industry standard for representing (MI)LP models and is accepted by virtually all solvers. The disadvantage of using MPS files is that the length of a solution cycle increases significantly: i.e. the time that elapses between the moment the user instructs the modelling system to solve the model and the moment the user obtains the results. Long solution cycles make that an iterative way of using the system, while debugging models or conducting experiments, is impractical. If long solution cycles result from large and complex models, there is no practical solution. However, if long solution cycles arise from generating MPS-files, uploading the model to the Solver, and downloading the output, using an integrated Solver improves the practical value of the modelling system — quick response improves the interaction between the modelling system and the user.

Although there is a standard model representation for solvers, there is no similar standard communication protocol to instruct the Solver, for example, to minimise or maximise the problem or to apply a specific type of algorithm, nor is there a standard file format for the output generated by solvers. Therefore the Model Processor subsystem must provide tailored support for each external Solver. Usually, the sequence of control commands is supplied through a batch file (response file).

The majority of today’s general-purpose modelling systems support multiple LP and MIP solvers. Some systems support Non-Linear Programming (NLP) solvers, solvers for Quadratic Programming (QP), and dedicated algorithms for optimising network\(^1\) models. Users like to choose the Solver that best suits their needs. MINOS, for example, is currently known for its very efficient MIP-algorithm. CPLEX is extremely fast at solving large network models.

\(^1\) LP-models which embody a network structure have at most 2 nonzero elements in each column of the coefficient matrix. The conventional network visualisation of a network model represents the constraints of the model as the nodes of the network. The arcs of the network correspond to the activities of the model. The nonzero elements of a pure network model equate to plus or minus 1. The nonzero elements of a generalised network model may take any value.
PART I - INTRODUCTION

3.2.3 The solution analyser

The solution analyser reports the output data obtained from the Model Processor and provides on-line querying facilities to analyse the affect of changing the parameters of the model on the current optimal solution. Appendix A discusses the basics of LP, its vocabulary, and sensitivity analysis in more detail.

STANDARD REPORTS. Standard (MI)LP solution reports contain general information on the optimisation process, e.g., whether or not the solver has reached an optimal solution and, if not, whether the problem is feasible, unfeasible, or unbounded. Most standard reports also include general statistics like the number of iterations of the optimisation algorithm and the time that has elapsed. If the problem has an optimal solution, then most (MI)LP solution reports contain the following information:

- For each decision variable, the report lists the activity level, whether or not the decision variable is a basic variable (which means that the activity is part of the optimal solution to the problem), and the variable's reduced cost\(^1\) (also referred to as the variable's dual activity level);

- For each constraint, the report lists the values of the slack variable (so the user can identify the active functional constraints — cf. Appendix A), the dual price\(^2\) or shadow price associated with the constraint, and the range for the RHS-constant within which the current optimal solution remains optimal;

- The objective-function value associated with the current optimal solution.

Note that in case of an MIP-model being solved, the reduced costs and the dual prices apply to the LP-relaxation of the model in which the integrality constraints have been replaced by continuous equivalents (cf. Appendix A).

GRAPHICAL VISUALISATION. Graphical visualisations are powerful aids to analyse solution data. Bar charts, for example, can be used to display lower bounds and upper bounds on activity levels vis-à-vis their current values in the optimal solution. The graphical user interface of AIMMS [Bisschop, 1992] supports direct manipulation of these bar charts to modify the levels of the lower and upper bounds.

---

\(^1\) A decision variable's reduced cost represents the change of the objective-function value if the variable is relaxed by one unit (made higher if the variable is at its upperbound, made lower if the variable is at its lowerbound). For nonbasic variables in the optimal solution, the reduced cost equals the minimum increase of the associated objective-function coefficient (provided that the problem is to maximise the objective function value) in order to have the objective-function value improved by undertaking a unit of the activity.

\(^2\) The dual price associated with a constraint defines the unit change of the objective-function value per unit change of the RHS-constant associated with the constraint provided that the optimal solution remains feasible. See: Appendix A.
SENSITIVITY ANALYSIS. The values assigned to right-hand side constants, constraint coefficients, and objective-function coefficients may have resulted from statistical analysis, numerical forecasts, judgements, rules of thumb, and managerial policies (e.g., demand levels, delivery-lead-time performance). As a result, the actual value assigned to a parameter may be subject some degree of uncertainty. Usually, there is much more statistically supported confidence that the value assigned to the variable should be located within a particular interval. Whether additional effort is required to estimate sensitive parameters more closely then depends on the sensitivity of the optimal solution to changes in the values assigned to these parameters. A decision maker is probably willing to consider suboptimal solutions that perform well for a large range of likely values: so-called robust solutions.

The purpose of sensitivity analysis is to trace how changes of the values of input parameters affect the current optimal solution to the problem such as the values the parameters may take without changing the set of basic variables in the optimal solution and the what changes occur in the optimal solution if a parameter is assigned a value outside this range. Standard LP solution reports give some indication of how the solution is affected by relatively small changes (that keep the current optimal solution feasible) in each of the objective-function coefficients and right-hand side constants. In addition, most Solvers have special algorithms for evaluating changes of both the objective-function coefficients and the right-hand side constants.

3.2.4 The database management subsystem

A Database Management System (DBMS) is used to create, modify, catalog, access, and protect the contents of one or more databases: collections of data under the control of a DBMS [Sprague,1982;Wortmann,1988]. A database can be considered to represent the state of affairs in a part of the real world [Wortmann,1988,p173]. A database subsystem could be used to store the structure of the models, the data sets that define specific model instances, the relationships between the models and the data, and the output data that are obtained from the Solver1.

DATA MODELS. In order to store data sets and algebraic structures in a database, their logical structure must be represented in terms of the data structures offered by the database. A data model defines a method of representing, organising, storing, and handling data in a computer. A data model consists of three parts [Codd,1980]:

1 In fact, this observation issued the rise of database-oriented modelling languages which view an (MI)LP model as a collection of database tables and extend the conventional query language to express algebraic constructs such as summations over sets [Murphy,Stohr,Asthana,1992,p965].
PART I - INTRODUCTION

1. A collection of data structures including, for example, lists, tables, and networks;

2. A collection of operations which users and application programs can apply on these data structures by invoking the DBMS (it is the DBMS that operates on the database);

3. A collection of integrity rules that define legal sets of values for the data structures and operations to change the values associated with attributes. The integrity of the database is violated, for example, if two contradicting facts are stored (inconsistency), if an attribute is assigned a value which is not a member of the set of values allowed for the attribute (domain integrity violation), or if a referential attribute contains a reference to an entity that used to exist in the database but was deleted without deleting the reference (referential integrity).

The simplest yet most important goal in database design is the concept that each fact should be stored only once. This eliminates redundancy, it simplifies the process of updating the database, it facilitates the maintenance of database integrity, and it reduces storage requirements [Booch, 1991, p. 371].

End users (and application programs) and DBMSs have different views on the data that is stored in a database.

To the user, the data is organised in clusters of entities, so-called entity-types, and their mutual relationships. An entity is anything of interest in a particular field of application or problem domain. An entity-type consists of all entities that share the same set of properties, which are called attributes. Most users use only part of a database. Many of them do not use the data in the form (usually tabular) in which it is made available by the DBMS, but they use derived reports containing visualisations (e.g., bar charts, pie charts, etc.) of condensed data (obtained through subsetting, combination, and aggregation) that is scattered across the database.

Whereas most users and application programs have only partial view of the database, a DBMS concentrates on the entire database. A DBMS views a database in terms of files containing data attributes, cross references, and indexes to speed up queries. A data model must adhere to these different views on the database. That is why the literature on the subject of database design often distinguishes three levels of data modelling [Date, 1977, cf. Figure 3-3]:

1. The external scheme which is closest to the user or the application program and corresponds to the user’s perspective of the data as outlined above;

2. The conceptual scheme which describes the entire structure of the database. Its focus is on neither the internal storage structure and nor the user’s view of the data, but on the meaning — the semantics — of the concepts used in structuring the data.
A so-called mapping defines the transformations needed to obtain the internal data model and the conceptual model and vice versa;

3. The internal scheme which is closest to the physical storage structure of the data. The internal data model corresponds to the DBMS’s perspective on the database as outlined above.

![Diagram: Three levels of data models](image)

Figure 3-3 Three levels of data models. Adapted from: Date, 1977.

There are three dominant approaches to the design of a DBMS [Wortmann, 1988]: the relational model, the hierarchical model, and the network model. The relational model has proven to be very popular for the design of databases and DBMSs. Vendors of non-relational databases have designed relational DBMSs to be used as ‘front-ends’ by application programs and end-users. Lately, a fourth kind of database model has emerged: the object-oriented database model. Appendix B discusses the relational and object-oriented database models and their mutual differences.

**LOCAL VERSUS REMOTE DATABASES.** The presence of an integrated, and almost without exception, relational DBMS does not imply that the system actually ‘contains’ the database subsystem. An integrated DBMS only provides a fast connection with a database system by sharing the same memory. The relational DBMS serves as an API (cf. Section 3.2.2) to the database, thus providing the programmer with the database’s full functionality, including the creation of tables and views, selection using arbitrary SQL\(^1\) statements, insertion, updating, and deletion. If the database is used exclusively to support the application, we speak of an integrated database system or a database subsystem. This definition covers both applications which store their data in a proprietary database (tightly coupled) and applications which use a major stand-alone database offering SQL-server functionality and freely available DBMS (loosely

---

\(^1\) SQL (Structured Query Language) is an industry-standard procedural language for interacting with relational databases based on tuple-relational calculus (cf. Appendix B). SQL may be used both interactively and programmatically.
PART I - INTRODUCTION

coupled). As most companies use dedicated machines to run their centralised databases (database servers) an integrated database is also referred to as a local database.

The benefits of a local database subsystem have long been recognised in the area of decision support systems. Such a database system may be used to store images of the relevant parts of remote database systems. Usually, however, the data are condensed and restructured prior to downloading the data into the local database. The advantages of using a local database are that:

- The modelling system can use data from a variety of external sources using its own internal data model;

- The user can manipulate the data stored in the local database without endangering vital information;

- The local database generally contains less data than the external database and is therefore likely to perform better than the remote system. Moreover, most aggregation and subsetting is done prior to downloading the data. Using an external database requires that every aggregation and subsetting operation is done dynamically.

The disadvantages of using a local database containing extracted data from external sources are that:

- The local database duplicates data that is already contained in other data sources. This may lead to inconsistencies between the original and the copied data. Time stamps provide a mechanism to deal with different versions of database images;

- The data must be downloaded into the local database before the system can use the data;

- The local and remote data management components both offer largely the same functionality thus yielding functional redundancy between the two components.

Some (MI)LP-modelling systems, e.g. MPL, only contain a database management subsystem whose functionality is limited to downloading and uploading data from and to external relational database systems. Other systems, like gLPS, use the model base to store associated data. None of the (MI)LP-modelling systems that we encountered contains a model-independent data dictionary. In many cases, the support for the specification of the values of input parameters is restricted to the use of ASCII files where users are required to enter the data either directly or indirectly through key words that specify a relation that is part of a remote database.

INTERFACING WITH EXTERNAL DATA SOURCES. A modelling system that supports the use of external relational databases enables the user to assign the input parameters of a
model to an attribute of an existing database relation or a newly created view, i.e. a virtual table that holds the results of a ‘JOIN’ of two or more relations to form a single one\(^1\). The set of allowable key values for each of the key fields of the relation must form the members of the index sets associated with the indices of the parameters. The indices then define a record in the table, and thus a numerical value. For that purpose, the data extraction facility should provide four sets of operations [Sprague, 1982]:

1. *Data description* operations are used to describe files in the external database;

2. *Subsetting* operations that should permit any arithmetic and logical criteria to be used to select fields or records from the database;

3. *Aggregation* operations that should permit fields or records to be summed, counted, joined, or combined in any arithmetic manner. “Aggregations are essential in constructing and optimising effective models. It is not possible, necessary, or desirable to perform (...) analyses at the level of transactional detail that exists in the corporate database.” [Shapiro, Singhal, Wagner, 1993, p110];

4. *Presentation* operations that permit viewing the results of aggregation and subsetting operations.

Gradually more packages will provide this functionality as direct linkages to external databases are becoming a key requirement for successful model management [Sharda et al.,1995].

### 3.2.5 The model management subsystem

The *ModelBase Management Subsystem* (MBMS) provides facilities for the composition, modification, storage, and retrieval of (MI)LP-models and model instances: a so-called *modelbase*. The ability to create and modify (MI)LP-models using an Algebraic Modelling Language (AML), a Graphical Modelling Language (GML), or both, is thus part of the task description of the model management subsystem. Section 3.4, elaborates on existing (MI)LP-modelling languages.

Most algebraic modelling systems, and some graphical modelling systems, store models simply as ASCII files. The ASCII files contain a textual description of the model. Algebraic modelling systems use their AML. Graphical modelling systems either use an AML (LPFORM, for example, uses its LPSPEC; gLPS uses LPL), in conjunction with a subsidiary file which contains the data that relates to the graphical representation, or a proprietary model description format that parallels their graphical representation language. The model description can be modified by using a common

---

\(^1\) Appendix B provides a brief outline of relational operations like SELECT, PROJECT, and JOIN.
PART I - INTRODUCTION

ASCII editor. This is also the biggest disadvantage of this approach; the modelling system cannot monitor changes being made to the models. If the model had been solved prior to changing its structure, then the model no longer forms the relationship between the input and output data associated with it. This, in turn, may work confusingly and may lead to mistakes. Another way of storing models is to expand the conceptual design of the database by adding a data abstraction that is capable of representing (MI)LP-models. The advantage of this approach is that the models can only be changed by invoking the DBMS which is most conveniently available in the modelling system containing the database. (MI)LP-based DSSs containing built-in models have a third possibility to store models. As the structure of the models does not have to be changed, these systems can store models as precompiled subroutines which can be executed from the DSS.

3.3 (MI)LP-based Decision Support Systems

The objective of this section is to outline different approaches towards the design of (MI)LP-based DSSs. This section outlines three such systems: DP-PSS [Evers, 1992], PHYDIAS [Bender, 1992], and SLAM [Nunen and Benders, 1980]. PHYDIAS and SLAM focus on integrated production-distribution networks. DP-PSS focuses on the planning of distributed production activities. DP-PSS and SLAM contain an allocation model (cf. Section 2.1.4.2); PHYDIAS contains a facility location model (cf. Section 2.1.4.1). Both PHYDIAS and SLAM contain a single model. The conceptual design of DP-PSS differs from that of SLAM and PHYDIAS in that it makes use of a 2-layer hierarchical modelling approach. DP-PSS uses an optimised strategic Production Allocation (PA) plan to determine optimal flows of goods and materials at the tactical level using a Production Flow (PF) plan.

Section 3.3.2 presents a functional analysis of the use of an (MI)LP-based DSS. From this analysis and the reference model presented in Section 3.2 we have derived a framework for comparing (MI)LP-based DSSs which is presented in Section 3.3.3.

3.3.1 Justification

There are many (MI)LP-based DSSs for strategic and/or tactical production planning. Some manufacturers have reported successful implementations of their tailored DSSs. Owens-Corning Fiberglas [Oliff and Burch, 1985], for example, has developed a three-phase tactical planning model. The first phase, aggregate planning, is concerned with

---

1 Section 8.5.2 (part III) deals with the design of an object-oriented decomposition of algebraic (MI)LP-model structures.
SURVEY OF EXISTING TOOLS

simultaneously establishing overall production, inventory, and employment levels for a given time horizon at minimum costs. Oliff et al. use a heuristic to determine the aggregate production policy because aggregate planning models are both conceptually and computationally complex and have unrealistic assumptions [Oliff et al.,p27]. The second phase, the disaggregate planning, uses the aggregate inventory levels determined in the first phase and the monthly forecasts of individual standard product demands to generate lot size, line assignments, and inventory levels for individual products using an LP model. The third phase uses a heuristic to provide schedules that minimise changeovers. It is difficult to make a comparison between (M)LP-based DSSs which includes these tailored systems. For that purpose, both the problem domain and the approach taken to implement these systems differ too much. We therefore excluded these custom-built DSSs from this comparison.

Some business-consultancy firms have developed more generally applicable decision support systems: e.g.,

- Numetrix S.A.-N.V., a company located in Brussels (Belgium), offers a family of decision support tools, comprising Linx, Planx, and Schedulex 7. These products implement an integrated hierarchical planning approach. Linx focuses on optimal multi-period flow planning over a production-distribution network, Planx deals with master planning, and Schedulex 7 is intended for scheduling.

- Bender Management Consultants, Inc., have developed PHYDIAS;

- J.F. Shapiro Associates, Inc., have developed SLIM [Shapiro,1993].

Most companies offer two types of services: either the customer buys the system or both companies enter into a consultancy agreement that is based on the consultancy firm conducting analyses with the system. In both cases, however, the developers consider the possession of the DSS as a means to obtain competitive advantage. As a result, the provision of information on the underlying model structures is kept to a minimum. The information on PHYDIAS, Section 3.6, is derived from both the information provided by Bender Management Consultants Inc. [Bender,1992]¹ and a comprehensive survey of 13 packages that focus on the design of integrated production and distribution networks conducted by Hagdorn and Warffemius(1993).

This survey includes both DP-PSS and SLAM as the developers of both systems have published both the conceptual design and the Mathematical Program (MP) underlying their system.

¹ This reference is primarily a sales-promoting statement.
3.3.2 Functional analysis of using (MI)LP-based DSSs

Figure 3-4 shows a functional decomposition of the use of an (MI)LP-based DSS. The diagram uses the SA diagram technique (cf. Section 3.2). Figure 3-4 embodies the same basic structure as Figure 3-1 which shows the functional decomposition of the process of using an (MI)LP-modelling system: from the initial configuration, followed by experimentation, and finally reporting. Figure 3-4 contains the following clusters of activities:

A1. **Configure Model(s):** concerns the structural data that usually relates to the geographical structure of the network (sites, road maps, distance tables), existing facilities, and cost structures;

A2. **Specify Scenario:** concerns the specification of scenarios containing typically non-structural data, e.g., demand, sales prices;

A3. **Solve Model:** concerns the generation of the complete statement of the model in a format suitable to the solver, invoking the solver, and uploading the results;

A4. **Analyse Solution:** concerns both the preparation of the output of the solver for the creation of solution reports and sensitivity analysis;

A5. **Create Reports:** concerns the generation of tailored solution reports containing data views (graphics and tables), statistics.

![Figure 3-4 Functional analysis of using an (MI)LP-based DSS.](image-url)
3.3.3 Criteria for comparison

The criteria presented in this section typify the location-allocation models underlying both DP-PSS, PHYDIAS, and SLAM and characterise some of their technical features. Together, these criteria provide a reasonable picture of the merits of these systems. A comprehensive comparison, however, should also consider training needs and support, documentation, hardware requirements, computational performance, and price. For a more comprehensive survey of existing packages that focus on integrated production-distribution networks, the reader is referred to Hagdorn et al. (1993) who examined 12 DSSs that focus on the design of integrated production and distribution networks.

A1: Configure models

Characterisation of the Mathematical Program. Both DP-PSS, PHYDIAS, and SLAM are based upon a fully deterministic static (single-period) location allocation model. All three models contain a discrete space formulation instead of a continuous space formulation. Mirchandani and Francis (1990) have found that in most cases decision makers consider a discrete representation a more accurate portrayal of the problem at hand. Moreover, continuous formulations appear to be more difficult to solve. The reader is referred to Mirchandani et al (1990) for a comprehensive discussion of the discrete location models.

Neither DP-PSS nor PHYDIAS and SLAM consider, for example, the cost associated with cross border shipments, differences in corporate tax rates, or local content rules. Only PHYDIAS supports multiple currencies, most likely by using a mechanism similar to the one used by Cohen and Fisher (1989) where all cash flows are charged in source currencies and where currency exchange rates are used to denominate all cash flows in a single currency unit.

- Problem Span. The range of strategic and tactical production planning decisions supported by the DSS (cf. Chapter 2);

- Type of Mathematical Program. The mathematical program criterion relates to the optimisation model underlying the DSS. The MP is either a Linear Program (LP) or a Mixed-Integer Program (MIP). LP solution algorithms give much better performances than MIP algorithms. Experiments conducted by Heineken Breweries\(^1\) with both SLAM (LP-based) and PHYDIAS (MIP-based showed that

---

\(^1\) Experiments conducted by Heineken Corporate Distribution & Logistics. The SLAM LP contained some 150,000 variables and 10,000 constraints. Solution times using SLAM varied between 2 to 3 hours. It took PHYDIAS between 12 and 24 hours to come up with an optimal solution.
PART 1 - INTRODUCTION

the improvement in the performance of SLAM was 6 to 8 times greater than that of PHYDIAS.

The type of MP contributes to the extent to which the DSS is able to support a particular range of decisions. For example, a common working procedure for operating a DSS which simultaneously decides on the location of new facilities and the layout of goods and materials flows is to feed the system with a number of pre-selected sites. Sites that represent existing facilities may be fixed — these sites automatically become part of the optimal solution (if any). The decision maker is considered to be more-or-less indifferent with respect to the choice between the sites that represent possible locations for the new facilities that is being proposed by the DSS.

Amongst many others, Hayes and Wheelwright (1984) observed that most companies agree on the existence of both optimal maximum and minimum sizes for entire production facilities and for organisational units in particular. Moreover, Hayes et al. also observed that many firms have rather precise measures for these sizes. In support of these observations, virtually all DSSs support the formulation of lower bounds and upper bounds on capacity utilisation.

MIP location-allocation models represent the establishment of a facility at a particular site by a binary variable. If a particular site is not established, both constraints impose zero levels on capacity utilisation. Now consider that the optimisation process consists of two phases [Mirchandani et al, 1990]: the location phase and the allocation phase. During the location phase, using binary variables to model the selection of sites, an MIP location-allocation model comes up with a subset of the pre-selected sites that make up a possibly optimal network configuration. During the allocation phase, this subset of sites is used to evaluate the optimal flow of goods between the appointed sites and also whether the solution satisfies the optimality test. Unlike MIP models, LP models cannot take into account whether a site has been selected. Consequently, if a single new facility has to be located, the capacity constraints hold for all plants simultaneously. Therefore, only upper bounds are modelled for sites. An LP determines the optimal flow of goods while considering all pre selected sites simultaneously. By studying the optimal solution to the LP, the user then decides whether sites playing only a marginal —if any— role in the optimal flow of goods should be considered as superfluous.

Additionally, an MIP can model set-up costs for newly established facilities and costs associated with shutting down existing plants;

- **OBJECTIVE.** Minimisation of cost factors (e.g. variable and fixed production costs, transportation costs), or maximisation of net profits, for example if the model
SURVEY OF EXISTING TOOLS

supports minimum quantities that must be supplied and maximum amounts that cannot be exceeded at every customer location (or demand area). Does the system support both types of objectives?

- **MULTI-PLANT NETWORK REPRESENTATION.** This comprehensive criterion refers to the representation of the transportation infrastructure and the dispersed production system:
  
  - **NODES.** Does the model support different types of sites, e.g., supply sites, manufacturing sites, sites representing customer (regions)? Does the model explicitly\(^1\) support sites containing a mixture of suppliers, manufacturers, and customers?
    
    - Does the system support the configuration of facilities in terms of processing steps each having its own capacity resources;
    
    - Does the system support the modelling of ‘campus’ sites. A campus site is composed of two or more manufacturing facilities. Mutual supply relationships can exist between distinct facilities located on the same site. Campus sites increase the conceptual and mathematical complexity of the MP. In a location model, flow-balance restrictions are necessary to ensure that optimal flow patterns comply with flow-conservation principles. These state that the output of a facility is proportional to the availability of purchased parts and raw materials obtained from external suppliers. These ratios can be obtained from the Bill Of Materials stating how much of a particular component or raw material is needed to produce 1 unit-volume of end-items. For a non-campus production site, the total flow of manufactured products can be balanced with the total inbound flow using single-level BOM relationships. For a campus site, the local utilisation of manufactured goods makes a similar comparison of total inbound and outbound flows impossible. In that case, the flow-balance equations have to be defined at the facility level;
    
    - Does the model contain lower bounds and/or upper bounds to capacity utilisation for each facility as a whole and for specific capacity units in particular;
  
  - **LINKS.** Does the model support different types of linkages between nodes. Does the system support different cost metrics for measuring, e.g., distances, transportation costs, between nodes. Does the system support different modes of

---

\(^1\) Models only capable of representing single-purpose sites can depict multi-purpose sites implicitly by modelling zero distances between clusters of sites.
transportation associated with links, e.g. truck — less-than-full, full, multistop — train, ship, plane;

• NETWORK. Some models limit the flow of goods between suppliers and customers to a fixed number of successive stages. Other model formulations support supply chains of various lengths;

• PRODUCTS. Does the system support multiple product types — aggregating products into equivalent units is only allowed whenever the production and transportation processes are not significantly different [Mirchandani et al., 1990]. Does the system support Bill-Of-Material type of input-output relationships for manufacturing plants and processes;

• CUSTOMER SERVICE CONSTRAINTS. Lead time restrictions, limitations on the number of suppliers per customer, restrictions that constrain the distance between the supplier and the customer. An easy way of implementing the latter is to pre-screen the transport links in the network and to allow only those links that enable timely deliveries [Mourits, 1995];

A2: SPECIFY MODEL DATA

• DATA REQUIREMENTS. What kind of input data is needed to feed the model, what format is required for the data, does the system already contain data like road maps, distance tables, etc.;

• TYPE OF DATA MODEL. The type of data model that is used to provide the user with access to the data: e.g., hierarchical, network, relational;

• DATA-EXCHANGE CAPABILITIES. Decision support systems draw heavily on operational data that is obtained from corporate database systems containing detailed tracking information on shop orders, work-in-progress levels, and up-to-date forecasts of demand. Therefore a prerequisite to effective use of a DSS is that it can be loaded with data from external databases. Data from external sources can be imported either directly using a dedicated import filter or through a common data-exchange file format, e.g. ASCII or ODBC. If a DSS is capable of exporting data from its local database via a common data-exchange format, the data can be used in, for example, solution analysers, word processors, and spread sheets;

• DATA-PROCESSING CAPABILITIES. Usually, additional manipulation of data that is stored in the system’s database will be necessary to obtain data that has a format that suits the needs of the model. The data must fit the purpose of the model;
aggregation decreases the size of the problem and therefore increases speed and managerial insight, and aggregation of data decreases the degree of uncertainty in forecasts of demand. Does the system's database support aggregation of data? Does the system provide efficient support for manipulating large amounts of data?

A3: Solve Model

- **Solver Independence.** Does the system make exclusive use of an integrated solver or does it support external solvers;

- **Maximum Model Dimensions.** Does the system's design impose hardware-independent limitations on the size of the model being solved.

A4: Analyse Solution

- **Solution Reports.** Does the system provide standard (MI)LP-solution reports. Does the system support graphical analysis of the solution to the problem using geographical maps, bar charts, etc.;

- **Scenario Analyses.** Does the system support a side-by-side comparison of the optimal values of parameters obtained from experiments with different scenarios;

- **Sensitivity Analysis.** Does the DSS support the use of the special-purpose sensitivity analysis algorithms that are supplied by most solvers.

A5: Create Reports

- **Presentation.** Is it possible to create customised reports? Is it possible to export report files to standard word processing and presentation software.

3.3.4 DP-PSS: Distributed Production Planning Support System

DP-PSS [Evers, 1992; Wouterse, 1990] emerged from the ESPRIT-II CMSO project on distributed production within the European automotive industry. During the design, the dispersed production system of DAF Trucks was used as a reference case. The design DP-PSS is, however, intended to be more widely applicable.
PART I - INTRODUCTION

A1: CONFIGURE MODELS

- **PROBLEM SPAN.** The conceptual design of DP-PSS is based upon two types of models. Given an optional product assortment, long-term indicative sales forecasts of families of end-items, and a distributed configuration of production facilities, the strategic Production Allocation (PA) model determines the quantities for the most profitable set of products (satisfying the limitations imposed by the available production and transportation capacity), the allocation of products and components to facilities, and the deployment of transportation equipment. The tactical production planning stage departs from a given state of the manufacturing system (parts of which may be derived from the output of a PA model) with specified stock-levels and more accurate forecasts of demand on end-items, supply, prices of products and production factors. The tactical planning stage yields a Product Flow (PF) plan containing optimal monthly production intensities, local purchasing of component parts, resulting stock levels, and transportation quantities;

- **TYPE OF MP.** The PA-model is formulated as an LP. The current implementation of DP-PSS does not yet support the making of a PF-plan;

- **OBJECTIVE.** The objective of the PA-model is to maximise net profits;

- **MULTI-PLANT NETWORK REPRESENTATION.** The participants in the DP-PSS network include suppliers, plants, and customers. At each location both a supplier, a customer, and multiple plants may be present.
  - **NODES.** DP-PSS supports the modelling of campus sites. Manufacturing plants consist of one or more capacity units capable of producing a selected range of products. For each capacity unit, both lower and upper bounds can be defined on capacity utilisation;
  - **LINKS.** Links can be associated with multiple modes of transportation;
  - **NETWORK.** The model supports supply chains of various lengths;

- **PRODUCTS.** DP-PSS supports multiple products and product structures using Bill-Of-Materials;

- **CUSTOMER SERVICE CONSTRAINTS.** The PA-model does not contain customer service restrictions;

A2: SPECIFY MODEL DATA

- **DATA REQUIREMENTS.** Locations (availability of capacity resources, e.g., labour, unit price per capacity resource), transportation links (distance table, modes of
transportation, transportation cost), manufacturing facilities (location, capacity units, capacity limitations, manufacturing capabilities, resource requirements), products (weight, volume, return frequency), product structure, markets (location, lower and upper bounds on demand, unit sales prices), suppliers (location, availability of components, location, unit purchase costs);

- **TYPE OF DATA MODEL.** Relational. DP-PSS contains a db-VISTA database;

- **DATA-EXCHANGE CAPABILITIES.** The prototype of DP-PSS does not provide support for data exchange. Data must be entered manually;

- **DATA-PROCESSING CAPABILITIES.** DP-PSS does not support the aggregation of product and customer data.

**A3: SOLVE MODEL**

- **SOLVER INDEPENDENCE.** DP-PSS generates an MPS statement of the LP. XA is supported as external solver;

- **MAXIMUM MODEL DIMENSIONS.** The conceptual design of the DP-PSS imposes no limitations on the maximum dimensions of the model being solved.

**A4: ANALYSE SOLUTION**

- **SOLUTION REPORTS.** The LP-solution report is based on the standard output of XA;

- **SCENARIO ANALYSES.** DP-PSS labels the data associated with the parameters of the model with the identifier of an integrating strategic and tactical plan. In this way, the system supports a side-by-side comparison of the optimal values of parameters obtained from experiments with different scenarios;

- **SENSITIVITY ANALYSIS.** DP-PSS does not support sensitivity analysis explicitly.

**A5: CREATE REPORTS**

- **PRESENTATION.** DP-PSS provides standard on-screen solution reports.

**3.3.5 PHYDIAS**

PHYDIAS [Bender, 1992] is recommended by Bender Management Consultants Inc., Arlington, USA, as a state-of-the-art tool for optimum logistic and distribution strategy development:
PART I - INTRODUCTION

A1: CONFIGURE MODELS

- **PROBLEM SPAN.** The MP underlying PHYDIAS is a location-allocation model covering an integrated production-distribution network. It supports the selection of suppliers, the assignment of suppliers to production facilities, the location of plants and warehouses, the allocation of manufacturing processes to facilities, the allocation of products to processes, and the establishment of supply relationships between plants and customer centres;

- **TYPE OF MP.** Although the documentation of PHYDIAS mentions neither the structure nor the type of the underlying model explicitly, the model is most likely formulated as an MIP. The model provides the ability to limit the number of open facilities and warehouses. Besides, PHYDIAS is capable of modelling fixed costs associated with the establishment of manufacturing facilities and warehouses. Both features require the use of binary variables;

- **OBJECTIVE.** Cost minimisation, profit maximisation, maximisation of market share;

- **MULTI-PLANT NETWORK REPRESENTATION.** PHYDIAS supports integrated production-distribution networks that consist of suppliers, primary plants, secondary plants, primary warehouses, secondary warehouses, and customer centres:
  - **NODES.** Each facility consists of a collection of processes associated with products. Capacity restrictions can be specified at the facility level, at the process level, and at the product level;
  - **LINKS.** PHYDIAS is capable of handling different modes of transportation;
  - **NETWORK.** PHYDIAS supports integrated production-distribution networks containing up to 6 layers;

- **PRODUCTS.** PHYDIAS supports multiple products and product structures using so-called 'product recipes';

- **CUSTOMER SERVICE CONSTRAINTS.** PHYDIAS supports 'response time' (customer lead time) constraints and various delivery policies:
  - *multiple origins*, where a demand area may receive products from several warehouses;
  - *single origin*, demand areas must receive all products from a single warehouse;
  - *proportional origins*, where a demand area may accept deliveries from any number of warehouses, yet, each warehouse must ship all products demanded in the proportion of total demand;
A2: SPECIFY MODEL DATA

- **DATA REQUIREMENTS.** Geographical locations, transportation links (distances, modes of transportation, transportation cost), suppliers (location, availability of components, purchase costs), plants (location, primary/secondary plant, optional process allocations, capacity limitations, fixed and variable facility costs), warehouses (location, primary/secondary warehouse, capacity limitations, inventory costs, handling costs), customer centres (location, lower and upper bounds on demand, unit sales prices), products, product structure (single-level BOM);

- **TYPE OF DATA MODEL.** Relational. PHYDIAS contains an Oracle database [Hagdorn et al., 1993];

- **DATA-EXCHANGE CAPABILITIES.** PHYDIAS is capable of both importing and exporting data in ASCII and DIF format;

- **DATA-PROCESSING CAPABILITIES.** User-defined selection criteria can be used to aggregate product and client data;

A3: SOLVE MODEL

- **SOLVER INDEPENDENCE.** It is most likely that PHYDIAS contains an integrated Solver. PHYDIAS is able to generate a standard MPS statement of the MIP;

- **MAXIMUM MODEL DIMENSIONS.** PHYDIAS imposes no hardware-independent limitations on the maximum dimensions of the model being solved.

A4: ANALYSE SOLUTION

- **SOLUTION REPORTS.** PHYDIAS provides both numerical and graphical (e.g., maps, tables, bar charts) presentations of the solution;

- **SCENARIO ANALYSES.** Runs can be treated as separate scenarios. PHYDIAS supports comparative reports which contain the results from different scenarios;

- **SENSITIVITY ANALYSIS.** Various types of sensitivity analyses are supported.

A5: CREATE REPORTS

- **PRESENTATION.** PHYDIAS supports the creation of customised reports. Reports can contain costs, good flows, utilisation factors, and delivery times. Report files can be exported to standard presentation and word-processing packages.
3.3.6 **SLAM: Strategic Location Allocation Model**

SLAM [Nunen, Benders, 1980] has been developed as a DSS for location-allocation decisions within a brewery. SLAM is used by the ‘physical distribution’ department of Heineken breweries.

**A1: Configure models**

- **Problem span.** SLAM contains a strategic allocation model covering an integrated production-distribution network. SLAM focuses simultaneously on the allocation of products to plants and warehouses and the assignment of customers to warehouses (each customer is assigned to a single warehouse) while satisfying the capacity constraints for both the production facilities and the warehouses at minimal costs for production, handling, and transportation;

- **Type of MP.** The MP underlying SLAM is formulated as an MIP model (the variables for the assignment of customers to warehouses are binary variables). To shorten solution times, SLAM solves the LP relaxation of the MIP model. A heuristic is used to determine a feasible integer solution. In the author’s experience the heuristic generally yields feasible solutions within 0.1% of the LP solution [Nunen et al., 1980, p.103];

- **Objective.** Minimisation of production costs, handling costs, and transportation costs;

- **Multi-plant network representation.** The integrated production distribution network supported by SLAM comprises plants, warehouses, and customers:
  - **Nodes.** Each production site consists of the following functional tasks: brewing, fermenting, and packaging (e.g. bottling, canning, racking). SLAM supports both lower and upper bounds on capacity utilisation and throughput-time restrictions. Capacity restrictions may differ for different product groups;
  - **Links.** SLAM is capable of handling different modes of transportation;
  - **Network.** Supply chains consist of two layers: the production facility and the warehouse;
  - **Products.** SLAM supports multiple products;
  - **Customer service constraints.** SLAM supports various delivery policies (who is allowed to deliver to whom, single-origin assignments of customers to warehouses);
A2: Specify model data

- **Data requirements.** Production sites consisting of brew houses, tanks, and packaging lines (production costs, capacity restrictions), warehouses (handling costs, minimum and maximum storage capacity, maximum throughput time), customers (location, demand), products, product structure, transportation links (modes of transportation, transportation costs);

- **Type of data model.** Relational;

- **Data-exchange capabilities.** Input: ASCII text file format and Lotus 1-2-3 spreadsheet format. Output: ASCII;

- **Data-processing capabilities.** Products sharing the same semi-finished components may be aggregated. SLAM provides a number of predefined selection criteria for aggregation of client data: e.g., demand levels, commercial region, aggregation code, zone number.

A3: Solve model

- **Solver independence.** SLAM contains an integrated LP solver;

- **Maximum model dimensions.** The conceptual design of SLAM imposes no limitations on maximum model dimensions. Hagdorn et al (1993) state that a typical large model contains 12 brew houses, 15 tanks, 60 packaging lines, 40 warehouses, and 1500 customers. On an IBM RS 6000 system, solution cycles vary from seconds to approximately 20 minutes.

A4: Analyse solution

- **Solution reports.** SLAM provides both textual and graphical (e.g., maps, bar charts) presentations of the solution;

- **Scenario analyses.** SLAM contains a scenario database that stores the non-structural data. Comparative reports are not supported;

- **Sensitivity analysis.** Neither the review of Hagdorn et al. nor the paper of Van Nunen mentions support for sensitivity analysis.

A5: Create reports

- **Presentation.** SLAM supports the creation of customised reports containing costs, goods flows, utilisation factors, and model dimensions. Output formats: Lotus 1-2-3, Word Perfect, and Mapinfo (Geographical Information System).
3.4 (MI)LP-modelling systems

This section outlines different types of declarative (non-procedural) (MI)LP-modelling languages. Declarative modelling languages express what must be computed rather than how to compute it. Declarative MLs focus on the representation of the problem in a form that suits the needs of the model builder instead of using a problem representation that may serve directly as input to an (MI)LP solver. The following procedural modelling languages are thus excluded from our survey:

- **GENERAL-PURPOSE PROGRAMMING LANGUAGES** like C and Fortran which can be used to write code that interfaces with the API or callable library of a Solver or to write MPS matrix generators and solution report writers;

- **MATRIX DESCRIPTION LANGUAGES** like, e.g., DATAFORM* and OMNI* provide language statements which directly correspond to MPS statements. As a result of the column orientation of the *matrix generator*, the user must search through the problem definition to find all activities associated with a given constraint;

- **BLOCK-SCHEMATIC LANGUAGES** like, e.g., MathPro* and MIMI* which recognise the fact that most (MI)LP-matrices may be viewed as being composed of blocks containing non-zero values (often representing incidence matrices) and blocks containing only zeros (cf. Figure 3-8). This way of viewing models closely relates to the way a number of experienced model builders think [Welch, 1987]. Spreadsheet-like user interfaces allow easy data entry or editing.

To broaden our characterisation of the use of (MI)LP-modelling systems, we have extended the survey to the software packages that are used to implement the MLs. The primary focus of the discussion though, is on the modelling languages.

### 3.4.1 Justification

Our survey of (MI)LP-modelling systems includes a few illustrative systems from four broad categories: (1) Algebraic Modelling Languages (AML); (2) Non-algebraic 'intuitive' modelling languages; (3) Graph-based modelling systems; and (4) Icon-based modelling systems.

AMLs. This survey includes one representative from the category of AMLs: AMPL [Fourer et al., 1990]. Sharda et al. (1995) provide an extensive survey of algebraic modelling systems. The differences between these systems primarily concern the degree of similarity between the indexing and set-oriented expressions offered by these

* Refer to Sharda et al. (1995) for the addresses of the makers/distributors which offer these systems.
systems and the customary algebraic notation. Still, these systems should be considered to be more similar than different [Sharda et al., 1995].

Non-algebraic modelling languages. This survey also includes an outline of LOGS [Brown, Northup, Shapiro, 1986]. Unlike general-purpose modelling languages, LOGS contains a Descriptive Modelling Language (DML) which was developed for a particular class of problems that arise in logistics planning. Instead of using metaphors for variables, constraints, and objective functions, the language primitives of the DML contained in LOGS are based on network formalisms that permit the construction of highly intuitive model representations which correspond directly to the features of the problems as the user perceives it.

Structured Modelling (SM) [Geoffrion, 1987, 1989] aims at a comprehensive framework for an unambiguous representation of the essential elements of a variety of management models [Murphy et al., 1992a]. The generic approach underlying SM results in a rather abstract problem representation from which it is difficult to discern the structure of the underlying problem [Murphy et al., 1992a]. For this reason SM is not considered a suitable basis for the modelling subsystem of an (MI)LP-based DSS and is therefore excluded from this comparison.

Graph-based modelling systems. From the category of graph-based representations, this section reviews the Activity-Constraint (AC) graph technique [Schrage, 1987] and gLPS [Collaud and Pasquier-Boltuck, 1994]. The AC-graph technique, in its original form, is not implemented as part of a well-known modelling system. AC-graphs, however, serve as the basis of the AC-object based MS that we present in Chapter 5. Unlike gLPS, AC-graphs provide an intuitively appealing representation of many problems. The reason to include gLPS in this review is that it implements an AC-graph based MS which is in marked contrast to our AC-objects.

The development of Networks [Jones, 1990, 1991] is certainly worth mentioning. Networks is a prototype implementation of a Graph-Based Modelling System (GBMS). A GBMS is a modelling system for management science problems that can be represented by graphs. Examples include simulation models, decision trees, production planning models, minimum cost flow models, and vehicle routing models.

In Networks, models are represented as attributed graphs. An attributed graph is a graph whose nodes and edges are partitioned into types with associated values or attributes. The user of Networks creates a class of graphs that share some properties, e.g. graphs representing vehicle routing problems, by means of a graph schema or graph type. A graph schema specifies the node types and edge types that may be used
to compose a graph, their attributes (e.g., their graphical appearance), structural constraints that must be satisfied by any so-called legal graph, and legal transformations from one graph to another. For example, a graph representing a vehicle routing problem may consist of two types of nodes: warehouses and customers. The edges would represent links connecting a pair of customers or a customer and a warehouse. A typical structural constraint for vehicle routing graphs is that these graphs must be cyclic: vehicles must return to the warehouse from which they started. To ensure that graphs satisfy the structural constraints, Networks uses graph grammars. A graph grammar describes how to construct different types of graph-based models by specifying a set of allowed editing operations called productions that can be performed on a graph that satisfies the structural constraints. By using an editor that complies with a particular graph schema, the user of Networks can create graphs of the specified type. External (optimisation) algorithms can be connected to the graph schema to solve problems that arise within the context of the problem domain that is addressed by the class of graphs. From a scientific point of view, Networks is founded on research that seems to prove that people frequently represent their problems as attributed graphs. Two advantages of using Networks are: (1) attributed graphs make possible a representation of a problem that fits the problem owner's perception of the problem; and (2) Networks enables a flexible use of fast dedicated solution algorithms not accounting some of the practical difficulties that might arise. An important drawback though is that Networks lacks a uniform representation of problems that belong to the same category of management science problems. To be more specific, Networks lacks, for example, a uniform attributed-graph schema for representing (MI)LP-models. To circumvent this weakness, Jones(1990,p145) suggests modelling a graph-based MS to represent (MI)LP-models in terms of attributed graphs using Networks. The absence of an (MI)LP modelling language is, however, the reason not to include Networks in our survey.

Icon-based modelling systems. LPFORM [Murphy et al.,1992a; Ma et al.,1989; Murphy et al., 1986] is the only icon-based MS included in this survey. Icon-based modelling schemes focus on the representation of a model by using icons. The icons represent generalisations of real-world objects or concepts rather than abstract notions. Considering the high level of concreteness (cf. Section 3.4.5) that may be obtained by using this type of MS, non specialists in particular will find iconic modelling schemes the most appealing of the methods evaluated by Murphy at al. (1992a). Moreover the graphical abilities of today's computer systems provide full support to the high-quality type of user-system interaction that is required by icon-oriented MSs.

---

1 Jones refers to: H. Bunke, "Attributed Programmed Graph-Grammars and Their Application to Schematic Diagram Interpretation", in: IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4, 1982, 574-582.
One approach to the implementation of an iconic MS is to design a library of icons representing real-world entities within the context of a particular application domain. The icons serve as building blocks with predefined interfacing relationships which guide the composition of models. The user cannot modify the model structures that underlie each icon — the user is only allowed to quantify its parameters. As a result, the formulation of models is a relatively uncomplicated task which requires little understanding of the mathematical details. The problem, however, is that such a library of icons is inherently incomplete; eventually most users will come up with additional demands. Moreover, maintaining a high level of concreteness would require a large number of icons to cover an acceptable range of model components thus decreasing the compactness of the MS and resulting in loss of legibility.

Another approach to the implementation of an iconic MS is to base its design on a general-purpose graphical MS that provides strong support for conceptual modularity. The creation of a library of icons representing customised model substructures then becomes part of the configuration stage of the system that implements the MS. This approach has been used to design LPFORM.

3.4.2 The process of developing an (MI)LP model

Section 3.4.3 presents a functional analysis of the use of an (MI)LP-modelling system. From this analysis, the reference architecture presented in Section 3.2, and the criteria for evaluation of modelling languages provided by Murphy et al. (1992a), we have derived a framework for comparing (MI)LP-modelling systems which is presented in Section 3.4.5. The (MI)LP-model development process commences with the formulation of models and extents to the point at which an optimisation algorithm can be applied. In the early stages of this process, it is customary to separate the structure of the model from the associated data. To distinguish between a model which Fourer et al. (1990) define as "(...) a general class of problems that share a certain structure and purpose (...)" and a particular problem, the latter — following Mirchandani et al. (1990) — is referred to as a model instance. The distinction between models and model instances has three important advantages: (1) It is essential in dealing with large optimisation problems because the representation of a model is more concise and thus more practically readable than the representation of model instances; (2) The values for the sets and parameters can vary from run to run without affecting the representation of the model; and (3) Formalisms for representing (MI)LP models can support the creation and reuse of submodels thus enabling the user to apply a mixture of top-down and bottom-up formulation strategies.
PART I - INTRODUCTION

In order to illustrate the practical value of the use of submodels, the next section introduces a block-schematic representation of a constraint-coefficient matrix which is typical to many periodic planning models and models that deal with the co-ordination of multiple relatively-independent 'actors': e.g., computers in a LAN, or manufacturing plants and distribution centres in a logistics network. Chapter 5, which presents the AC-object based modelling language that we propose, also draws heavily on the concepts and notation introduced in this section.

3.4.3 A typical model structure

Figure 3-5 depicts a block-schematic representation of the constraint-coefficient matrix of a model (shaded areas represent non-zero elements) which has an almost decomposable structure.

A block-schematic representation depicts the constraint matrix in an aggregate form. Each row corresponds to a subset of similar constraints. The columns correspond to subsets of similar activities. Blocks therefore correspond to submatrices of constraint coefficients. The actual size of the blocks and the values of the coefficients are thus separated from the representation of the structure of the model, as is customary in algebraic notation. It is common practice to denote arbitrary members of the subsets of similar model components, i.e. the activities, the constraint coefficients, the objective-function coefficients, and the right-hand side (RHS) constants, as indexed parameters.

The activities of the model shown in Figure 3-5 have been partitioned into four groups. Each group may be viewed, for example, as a representation of an actor. Activities within groups have a relatively small number of linkages with activities from other groups. Only the top row of the matrix deals with the co-ordination of activities among different groups. Different actors of the same 'type', i.e. showing identical behaviour from the model builder's perspective, are represented by the same model substructure.
Only the dimensions of the index sets and the numerical values of the parameters differ per actor. The four groups of activities depicted in Figure 3-5 can be categorised, on the basis of their structural properties in terms of zero and non-zero blocks, into three types: A, B, and C. The coefficient matrix contains two substructures of type B. Models that have a structure similar to the model shown in Figure 3-5 often contain multiple instances of similar substructures.

Most (MI)LP-modelling systems separate the structure of a model from the numerical specification of a particular problem, using language primitives that may be reduced to the block-schematic representation depicted in Figure 3-5. If a model builder chooses to develop a model by using the familiar algebraic notation prior to using a modelling system, then the use of an algebraic modelling system is the most obvious choice. The conversion of the algebraic formulation to an algebraic modelling language is straightforward. Algebraic modelling languages, however, usually do not exploit the modular structure of models similar to the one depicted in Figure 3-5 to facilitate the model-formulation process by supporting top-down decomposition and reuse of submodels. Modelling systems that provide this functionality are better suited to be used throughout the model-development process including the early stages that deal with the search for an appropriate formulation.

3.4.4 Functional analysis of using (MI)LP-modelling systems

Figure 3-6 shows a decomposition of the model development process into clusters of activities from the perspective of the user an (MI)LP-modelling system which contains a database subsystem. Usually, these systems are operated by OR specialists. The formulation of (MI)LP-models, however, requires different skills than those which are needed to generate appropriate sets of data in order to conduct experiments. In practice therefore, these different requirements may result in both activities being executed by different people. In such cases, experienced model builders configure the system with customised models which enables people who are more deeply involved in the actual decision-making process to use the system to conduct experiments with different sets of data and possibly with different predefined model configurations.

Figure 3-6 provides a more detailed picture of the use of an (MI)LP-modelling system than Figure 3-1. Note that the activities shown in Figure 3-6 need not be executed in a strictly linear order. Figure 3-6 consists of the following clusters of activities:

A1. **FORMULATE (MI)LP-MODEL.** During the model formulation stage, the user specifies the structure of the model. The user of today’s most advanced modelling
PART I - INTRODUCTION

systems is presented with a collection of either textual elements (keywords) or graphical elements (icons) that may be used to compose an (MI)LP-model. These elements together with the syntactical rules that guide the composition constitute a Modelling Scheme (MS). Depending on the MS, its elements serve as metaphors of algebraic structures or higher-level abstractions such as (sub)model templates or even concrete entities.

The quantification of constraint coefficients and RHS-constants which have a structural value is also part of the formulation of models. For example, flow conservation relationships are typically represented by constraint coefficients that structurally equate to plus or minus one. The RHS of flow balance constraints structurally equates to zero.

Algebraic modelling systems provide few—if any—facilities that help in formulating (MI)LP-models. These systems focus on an easy way of inputting the mathematical description of a model. (MI)LP-modelling systems that support hierarchical abstraction and the reuse of previously defined submodels enable the user to apply a mixture of top-down and bottom-up formulation strategies. For example, the 'first principles' approach that is used in LPFORM [Ma et al., 1989] allows the user to first specify the model as a network of real-world objects together with their mutual relationships. In subsequent stages, the user specifies the objects and networks in layers of increasing detail. Alternatively, a model builder may want

Figure 3-6 Functional analysis of using an (MI)LP-modelling system.
to explore several formulations of a submodel before laying out the full model thereby following a bottom-up strategy.

The model formulation activity results in a model description that is either implicitly or, more usually, explicitly stated. An explicit model description contains all the information that is needed to construct the complete algebraic statement of an (MI)LP-model. It includes all coefficients, variables and right-hand side (RHS) constants, together with their mutual relationships, that define the constraints and objective function. An implicit model description, on the other hand, only contains an explicit description of the relationships between the elements of \textit{model components}\footnote{The term model component is derived from Murphy, Stohr, and Ma (1992). In this paper, the term model component is restricted to implicit model schemes while Murphy et al. generalise to explicit model descriptions also. Explicit model descriptions, however, do not require a subsequent assembly of the model components to obtain a complete model description as the term suggests. Murphy et al. also consider, for example, data tables as model components. Their concise description of the model formulation process does not distinguish between models and model instances.} which consist of disjoint subsets of coefficients, variables and RHS-constants. In order to construct a complete model representation, the modelling facility uses a set of composition rules to determine the cross-relationships between the elements of different model components. The graphical modelling formalism that is contained in LPFORM [Murphy et al.,1992a] is an example of an implicit representation scheme. Murphy et al. refer to their approach as the ‘piecemeal’ approach;

\section*{A2. Define Data Model} The data-modelling activity concerns the definition of the data structures which will be used to organise the numerical data that specifies model instances. The data model makes it possible to manage large amounts of data. It also enables the modelling system, through an interface layer connecting the parameters of the model with the database, to locate the appropriate data in order to quantify the endogenous parameters or to store the values of the output variables. A data model therefore defines both the identification of data elements (key), in order to gain unambiguous access, and the type of data elements, for example, integer or real.

Indexed parameters may be represented by compositions of one-dimensional arrays and two-dimensional tables. In this case, the identification of database entries is based on both the name of the parameter and the values of the indices. An explicit data modelling stage, therefore, seems superfluous since the modelling system can automatically generate the data model from an analysis of the structure of the (MI)LP-model. A disadvantage of this approach, however, is that different parameters which may be viewed as attributes of the same logical or physical entity
are represented by seemingly unrelated data structures. It is not the structure of the model but the structure of the underlying problem domain which should determine the organisation of the data — the logical structure of the data. For this reason, we have depicted the data-modelling activity explicitly;

**A3. Specify Model Data:** After the formulation of an (MI)LP-model and the definition of the data structures, the next step, "Specify Model Data", focuses on the specification of a particular model instance. Figure 3-7 disaggregates this cluster of activities into three activities:

**A31. Specify Index Sets.** The first step is to specify the members of the index sets (and thus the dimensions of the problem). Next comes the quantification of the model parameters, i.e. the coefficients associated with the constraints and the objective-function, and the RHS-constants;

**A32. Quantify Constraint Coefficients.** The second step concerns the quantification of the non-structural elements of the constraint coefficient matrix. The structural elements of the constraint-coefficient matrix consist of the submatrices containing only zeros and the incidence submatrices;

**A33. Quantify Objective and RHS Data.** The third step concerns the quantification of the objective-function coefficients and the RHS-constants.

In Figure 3-7, the quantification of the constraint coefficients (A32) and the quantification of the objective-function coefficients and RHS-constants (A33) are depicted separately. To explore the implications of assumptions made in the model.
or when solving series of quite similar planning models, the problem dimensions and the constraint coefficients change less frequently than RHS-constants and objective-function coefficients.

Changing the constraint coefficients usually makes the current optimal solution infeasible and requires the entire problem to be re-optimised (cf. Appendix A). Questions whether an optimal solution remains optimal after changing objective-function coefficients (retaining dual feasibility) or RHS-constants (retaining primal feasibility) can be answered much more efficiently without having to re-optimise the entire problem.

The segmentation depicted in Figure 3-7 reflects the desirability that the modelling system provides differentiated support to each of these activities separately. In practice this would mean that the modelling system could be instructed to treat separate runs as related experiments;

A4. **SOLVE MODEL**: concerns the generation of the complete statement of the model in a format that is suitable to the Solver, invoking the Solver, and making the results available to either the user or the Solution Analyser subsystem.

After specifying a complete set of data, when the dimensions and values of all parameters are known, the modelling system can be used to construct the model instance and to convert it into a format (e.g. MPS) to which an optimisation algorithm can be applied. If there is an explicitly stated model description and the problem dimensions are known, the modelling system can directly identify the variables, coefficients and RHS-constants that appear in the model instance. If certain values are missing, the modelling system has the option to reject the data completely or to proceed, using default values, and to notify the user of the actions being taken. If there is an implicitly stated model description, however, the modelling facility must first construct the complete model. This involves the generation of non-specified relationships between model coefficients, decision variables and RHS-constants from different model components. Errors that occur in this stage generally require the user to reformulate the model.

The representation of an (MI)LP-model that a modelling system uses internally is referred to as the *Implementation Scheme* (IS). The demands that can be imposed on the IS are quite different from those that apply to the MS. The IS usually remains invisible to the user, so all the demands that relate to the reduction of cognitive complexity are irrelevant. Since it is used to implement the MS the most important requirements that apply to the IS are that it supports all features of the MS and that there is an unambiguous one-to-one relationship between the two schemes. After these come requirements that relate to the manageability of the IS.
PART I - INTRODUCTION

from a software engineering point of view. Part III of this thesis elaborates on the
design of the IS that is used to implement a prototype of the proposed system;

A5. ANALYSE SOLUTION: concerns the preparation of the output of the solver for the
creation of solution reports and conducting sensitivity analysis;

A6. CREATE REPORTS: concerns the generation of reports from the solution of the
model.

3.4.5 Criteria for comparison

The criteria presented in this section focus on the support for the formulation of models
and the quantification of model instances. As today’s (MI)LP-modelling systems pay
little attention to data modelling, the data-modelling stage is omitted from this survey.

The criteria can be classified into two basic categories: criteria that apply to both the
formulation of models and the specification of model instances, and criteria that apply
to the activities separately.

A1, A3: FORMULATE MODEL, SPECIFY MODEL INSTANCE

- DIFFERENTIATION concerns the degree of separation of the support for different
  stages of the model development process. The differentiation of the support for the
  specification of model instances highly depends on the capabilities of the solver.
  Presumably, that is why none of the modelling systems that is outlined in this
  section separates the support for the quantification of the coefficient matrix from
  the quantification of objective-function coefficients and RHS-constants. The
  differentiation criterion therefore applies to whether or not the modelling system
  separates the formulation of models from the specification of model instances;

- INTEGRATION concerns the degree to which the modelling system uses the
  representation of a model as an interface to the data (structures);

- MODEL-DATA INDEPENDENCE asserts that the modelling system separates the
  representation of the structure of a model from the data that are to be used in it.
  Thus, the sizes of sets and the values of data items can vary from run to run without
  the representation of the model in terms of the MS having to be changed.

A1: FORMULATE MODEL

Comparing the support for the formulation of models really comes down to the
evaluation of the MSs which are being used. The compactness of an MS, i.e. number of
SURVEY OF EXISTING TOOLS

syntactical elements that is at the user’s disposal, is not used for evaluation purposes as it would only make sense in a comparison of quite similar MSs. The following criteria apply to modelling schemes:

- **CONCEPTUAL MODULARITY**: the extent to which the MS supports problem solving strategies that reduce cognitive complexity such as hierarchical abstraction, top-down decomposition and reusability of previously developed model components;

- **CONCRETENESS**: the extent to which the MS enables the modelling of analogues to real-world objects and relationships;

- **COMPLETELENES**: the extent to which the representation of a model in terms of the MS provides a complete and explicit description of the model structure;

- **GENERALITY**: the range of models the MS is capable of representing. An important issue in this respect is that, in the absence of a powerful modelling system, model builders use mathematical notations to construct compact formulations. The question of whether a MS is able to represent a particular model therefore refers to both its intrinsic capability to provide a possibly different yet equivalent model representation (*theoretical generality*) and to its capability to provide a representation that agrees with the abstraction of the model builder (*practical generality*). To indicate the degree of theoretical generality, we either use the term ‘complete generality’ or we specify the range of models to which the MS applies. Compared to theoretical generality, practical generality is difficult to measure;

- **LABOUR-INTENSIVENESS**: which concerns the amount of detailed bookkeeping work required from the user.

**A3: Specify model instance**

Algebraic (MI)LP-modelling systems support the quantification of parameters through syntactical extensions to the modelling scheme. These extensions focus primarily on the specification of multi-dimensional arrays: the default data structure for indexed input parameters with non-structural values. gLPS uses its graphical user interface to enable the user to specify data sets both manually, at the completely disaggregated level, and through connections with external files basically following the same approach as AMLs. LPFORM, on the other hand, contains a relational DBMS through which the user can access data in remote database systems.

Because of this diversity in the support for inputting the necessary data, we will confine our survey to the **DATA-EXCHANGE CAPABILITIES** of the systems (cf. Section 3.3.3).
PART I - INTRODUCTION

A4: Solve model

Some modelling systems provide capabilities to exploit specific problem structures, others improve performance by making the (MI)LP smaller or otherwise easier to solve. Although they may prove very valuable, only a few of the leading commercial available systems, e.g. AMPL, have implemented these features. For this reason, they have been left out from this comparison. The maximum problem dimensions of virtually all Solvers is limited only by the hardware that runs the Solver, so this criterion is also omitted from this survey.

- **Solver independence.** Does the system make exclusive use of an integrated solver or does it also support external solvers.

A5: Analyse solution

- **Sensitivity Analysis.** The capability to generate standard solution reports and to conduct sensitivity analysis is fundamental to any modelling system.

A6: Create reports

- **Presentation.** Does the system support the creation of customised reports. Can these reports be exported to standard word processing and presentation software.

3.4.6 Introduction of a reference model

This section introduces a production location allocation model. This MIP model is based on an LP model which has been used to implement DP-PSS (cf. Section 3.3.4). The model will be used to illustrate the modelling scheme that is presented in Chapter 5 (part II). The model will also be used to illustrate AMPL [Fourer et al., 1990].

The purpose of the model is to select a number of pre-appointed geographical sites at which manufacturing plants could be established in such a way that demand is met while maximising net profits. Sites differ in terms of the fixed costs associated with the establishment of plants and the availability of parts that can be purchased locally. Plants differ in terms of their unit manufacturing costs and their minimum and maximum aggregate capacity. Manufactured products may be sold to local customers or transported to sites elsewhere. Alternatively, manufactured products may be used as component parts. Bill-Of-Material (BOM) quantities are used to balance the flow of manufactured products with the flow of component parts. Deliveries between plants located at the same site, a so-called campus site, are free of charges.
**Matrix Representation**

Figure 3-8 displays a block-schematic representation of the production location allocation model. The top row displays the transposed vector of decision variables. The type of each variable — real, integer or binary — is shown on top. The matrix of constraint coefficients is depicted below. Empty blocks contain only zeros. To the right, both the type of the inequality or equality relationship and the vector of RHS-constants are shown. The transposed vector of objective-function coefficients is depicted underneath the coefficient matrix.

The indices refer to three index sets: Product, Plant, and Site, where: part, prod ∈ Product, plant ∈ Plant, site, from, to ∈ Site. Different indices associated with the same set are used to distinguish semantic differences. For example, the inbound flow of the sites, which primarily serve as manufacturing sites, is stated in terms of parts. The outbound flow of the sites is stated in terms of manufactured products (which, in turn, may also serve as parts in subsequent manufacturing stages). Consequently, we use two flow balance constraints per site. The inbound flow of a site consists of parts bought locally, locally supplied manufactured products, and products transported from other sites. The total inbound flow is either sold directly to local customers or used to manufacture products. In the latter case, Bill-Of-Material quantities are used to relate the flow of manufactured products to component parts. The outbound-flow at a site

<table>
<thead>
<tr>
<th>real</th>
<th>real</th>
<th>real</th>
<th>real</th>
<th>binary</th>
<th>real</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacture (site,plant,prod)</td>
<td>Buy local (site,part)</td>
<td>Sale local (site,prod)</td>
<td>Sply local (site,prod)</td>
<td>Open plant (site,plant)</td>
<td>Transport (from,to,prod)</td>
<td>activity levels</td>
</tr>
<tr>
<td>(-)BOM-quantit (prod,part)</td>
<td>1</td>
<td>1: part=prod (prod,part)</td>
<td>1: part=prod (prod,part)</td>
<td>1: site=to, part=prod (prod,part, site,to)</td>
<td>= 0</td>
<td>inbound flow balance (site,part)</td>
</tr>
<tr>
<td>(site,plant,prod)</td>
<td>(site,part)</td>
<td>(site,prod)</td>
<td>(site,prod)</td>
<td>(site,from)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1: site=from (site,from)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(site,prod)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(site,prod)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mfg technology (plant,prod)</td>
<td>(-)Max open capacity (site,plant)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mfg technology (plant,prod)</td>
<td>(-)Min open capacity (site,plant)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-)Unit mfg cost (site,plant,prod)</td>
<td>(-)Unit part cost (site,part)</td>
<td>Unit sales profit (site,prod)</td>
<td>(-)Fixed mfg cost (site,plant)</td>
<td>(-)Transport cost (from,to,prod)</td>
<td>objective (maximise)</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3-8** Block-schematic representation of a production location allocation model.
matches the total of manufactured products. These products can be used for local supply, either for in-campus delivery or for local sale, or they can be transported to sites elsewhere.

3.4.7 AMPL

AMPL 2.0 [Fourer et al., 1990] is a commercially available algebraic (MI)LP-modelling system. The characterisation of the model solving, solution analyses, and report creation capabilities of AMPL presented in this section is derived from two surveys published in OR/MS Today: (1) a survey of algebraic modelling systems [Sharda and Rampal, 1995]; and (2) a survey of MIP-modelling packages [Saltzman, 1994].

SYNTAX

Figure 3-9 shows the location-allocation model from Section 3.4.6 using AMPL's algebraic modelling language. AMPL key words are depicted in bold. The '#' sign indicates a comment. Index expressions are depicted between braces. Instances of indexed parameters are depicted with brackets.

A1, A3: FORMULATE MODEL, SPECIFY MODEL INSTANCE

- **DIFFERENTIATION.** AMPL cannot be used to create the model statement; it does not contain a built-in editor. AMPL simply accepts the problem statements in AMPL format using ASCII flat files. Consequently, this differentiation criterion does not apply;

- **MODEL-DATA INDEPENDENCE.** AMPL separates the structure of a model from the data that is specific to model instances using separate model and data files;

- **INTEGRATION.** The creation of data files for the specification of model instances is completely disconnected from the representation of the model structure.

A1: FORMULATE MODEL

- **GENERALITY.** AMPL offers complete LP-generality and provides strong indexing and set-oriented expressions to obtain a high degree of practical generality;

- **LABOUR INTENSIVENESS.** Experienced model builders who are familiar with the mathematical notation will probably find AMPL both convenient and fast to work with. Less-experienced model builders may have trouble in using the system. Furthermore, in order to communicate the structure of the model, usually a different less abstract —and preferably graphical-oriented— representation of the model will be required.
• CONCRETENESS. The concreteness criterion does not apply;

# PRODUCTION LOCATION ALLOCATION MODEL

# SETS
set site;
set plant{site};
set prod;
set part within prod;

# geographical sites
# plants located at a site
# products
# products used as parts

# PARAMETERS
param max{plant{site,p in plant[s]}}; # max capacity
param min{plant{site,p in plant[s]}}; # min capacity
param max{site,p in plant[s]}; # max sales
param min{site,p in plant[s]}; # min sales
param max{site,p in plant[s]}; # max parts
param BOM{prod,part} = 0; # Bill Of Material
param utp{site,prod}; # unit transportation cost
param umc{site,p in plant[s]}; # unit manufacturing cost
param fmc{site,p in plant[s]}; # fixed manufacturing cost
param usp{site,prod}; # unit sales profit
param pcc{site,part}; # unit component cost
param prod,is_part{p in prod, c in part} := if (p <= c) then 0 else 1;

# VARIABLES
var open{site,p in plant[s]} binary; # open manufacturing at site
var make{site,p in plant[s]}; # manufacture volumes
var trns{site,prod} = 0; # transported volumes
var lda{prod} = 0; # local delivery
var buy {site,part} = 0; # number of parts bought
var sale{site,prod} = 0; # number of products sold

# OBJECTIVE
maximize net_profit:
+ sum {s in site, p in prod} (usp[s,p] * sale[s,p])
- sum {s in site, c in part} (pcc[c] * buy[s,c])
- sum {s in site, p in prod} (utp[t,f,p] * trns[site, part])
- sum {s in site, f in plant[s]} (fmc[f,s] * open[s,f] + sum {p in prod} (umc[s,f,p] * make[s,f,p]))

# CONSTRAINTS
subject to inbound_flow_balance{s in site, c in part}:
  + sum {f in plant[s], p in prod} (BOM{p, c} * make[s,f,p]) + buy[s,c]
  + sum {f in site, p in prod} (prod_is_part[p,c] * trans[site, p]) = 0;

subject to outbound_flow_balance{s in site, p in prod}:
  + sum {f in plant[s]} (make[s,f,p])
  - sum {to in site, p in prod} (trns[to, p])
  - lda = 0;

subject to maximum_sales{s in site, p in prod}: sale[s,p] <= max{site,p in plant[s]};
subject to minimum_sales{s in site, p in prod}: sale[s,p] >= min{site,p};
subject to maximum_buy {s in site, c in part}: buy[s,c] <= max{site, c};

subject to maximum_plant_capacity {s in site, f in plant[s]}:
  sum {p in prod} (make[s,f,p]) <= max_plant_capacity[s,f] * open[s,f];
subject to minimum_plant_capacity {s in site, f in plant[s]}:
  sum {p in prod} (make[s,f,p]) >= min_plant_capacity[s,f] * open[s,f];

Figure 3.9 AMPL representation of the reference model (cf. Section 3.4.6).
PART I - INTRODUCTION

- **CONCEPTUAL MODULARITY.** Not applicable;
- **COMPLETEENESS.** By definition, any algebraic model representation is complete.

**A3: SPECIFY MODEL INSTANCE**

- **DATA EXCHANGE CAPABILITIES.** Data can be supplied through ASCII flat files, direct linkages with external databases and spreadsheet files.

**A4: SOLVE MODEL**

- **SOLVER INDEPENDENCE.** The student version of AMPL includes MINOS (LP and non-linear programming) and CPLEX (LP and MIP). The professional package of AMPL does not include a Solver. AMPL supports various external Solvers. Solvers can be invoked through a command (menu) that is external to the ML.

**A5: ANALYSE SOLUTION**

- **SENSITIVITY ANALYSIS.** Full support.

**A6: CREATE REPORTS**

- **PRESENTATION.** AMPL comes with a report writing language to create customised reports containing non-graphic solution output.

### 3.4.8 Activity-Constraint graphs

Schrage (1987) briefly introduced the AC-graph formalism as part of the manual for the LINDO system. Greenberg (1983,p41) uses AC-graphs as the fundamental digraph for tracing flow paths in his ANALYZE system.

**SYNTAX**

AC-graphs contain two types of nodes (cf. Figure 3-10): activity nodes and constraint nodes. Activity nodes are depicted as boxes and represent decision variables. There is one activity node for each variable in the LP. Constraint nodes are depicted as circles. Constraint nodes are used to model both the constraints and the objective function. AC-graphs depict LP models in a standard maximisation format; the inequality relationships represent ≤-constraints. The constraint levels are written in the circles. Constraint nodes with zero values represent flow-balance equations. Nodes are connected by arrows. If an arrow points to a constraint, increasing its activity level causes the constraint level to increase and conversely, if an arrow incidents an activity,
increasing the level of the activity causes the constraint level to decrease. The values of
the coefficients for the transformation are added along the arrows. The conversion of
an AC-graph to an algebraic representation is straightforward. Following the
convention that terms on incoming arcs are positive while those on outgoing arcs are
negative, a constraint is formed by adding together terms involving each activity to
which it is connected. Each term is formed by multiplying the coefficient on the arc by
the symbol for the variable.

Figure 3-10 Activity-Constraint graph syntax.

A1, A3: FORMULATE MODEL, SPECIFY MODEL INSTANCE

- **DIFFERENTIATION.** Not applicable;
- **MODEL-DATA INDEPENDENCE.** In the original form AC-graphs do not distinguish
  between the representation of a model and a model instance;
- **INTEGRATION.** The representation of a model is used to specify a model instance
  (complete integration).

A1: FORMULATE MODEL

- **GENERALITY.** AC-graphs can represent any LP model;
- **LABOUR-INTENSIVENESS.** The AC-graph formalism does not support indexed
  parameters—all the parameters of a model instance must be specified which makes
  the formulation of even small-sized LPs a time-consuming task;
- **CONCRETENESS.** Most large-scale LP problems that involve, for example,
  production scheduling, physical distribution, and facility location contain
  transformations in form, time, or place of physical flows of goods. In these cases,
  the arcs in an AC-graph correspond to the flow of goods between activities.
  Because of this analogy with physical flows, AC-graphs provide an intuitively
  appealing representation of many real-world problems [Murphy et al.,1992a]. If the
  flow analogy is appropriate, it greatly simplifies the formulation of models. On the
other hand, when the flow analogy is lost, e.g. when modelling (generalised) lower bounds on activity levels, the AC-graph syntax is of little help to the model builder;

- **CONCEPTUAL MODULARITY.** The AC-graph technique does not provide support for hierarchical abstraction;
- **COMPLETENESS.** An AC-graph representation is complete.

A3: **SPECIFY MODEL INSTANCE**

- **DATA-EXCHANGE CAPABILITIES.** Not applicable.

### 3.4.9 glPS

glPS (graphical Linear Programming System) [Collaud et al., 1994] is designed for modelling experts capable of formulating LP models algebraically. glPS integrates Egli’s (1980) AC-graph based modelling scheme with Hürlimann’s (1988) (algebraic) Linear Programming Language (LPL). In fact, glPS uses Egli’s MS to build a graphical user-interface on top of LPL. LPL version 3.9 is available as a stand-alone non-commercial package. The characterisation of the model solving, solution analyses, and report creation capabilities of glPS is based on the outline of LPL that appeared in Sharda et al. (1995).

Unlike AC-graphs, the symbolism of glPS is a direct translation of the mathematical notation with support for both indexed parameters and subgraphs.

**Syntax**

Squares designate decision variables, circles either represent the equality sign or inequality sign associated with the LHS of a constraint or the objective function. Depending on their type, circles can be labelled as ‘≥’, ‘≤’, ‘=’, ‘Min’, or ‘Max’. Triangles represent the RHS-constant of a constraint and contain the associated data. The coefficient line, depicted as $\bigcirc \rightarrow \bigcirc$, always relates a square to a circle and, therefore, is used to define part of either the LHS of a constraint or objective function. The coefficient line contains the associated data. The RHS line, depicted as $\bigcirc \rightarrow \bigcirc$, always relates a triangle to a circle, thus defining the RHS of a constraint. Figure 3-11 shows the glPS representation of the canonical form of an LP minimisation problem: Min $c^T x$, subject to $Ax \leq b$, $x \geq 0$.

---

1 Contact the Institute of Informatics, Univ. Fribourg site Regina Mundi, rue de Faucigny 2, CH-1700 Fribourg, Switzerland

106
A1, A3: **FORMULATE MODEL, SPECIFY MODEL INSTANCE**

- **DIFFERENTIATION.** glPS does not separate the formulation of models from the specification of instances. During the formulation of a model, when the user introduces an index symbol which refers to a new index set, glPS requires the user to specify the members of the set. Although set membership is easily changed, glPS apparently provides no option to postpone the specification of index sets to the point at which the model is completed and specific model instances can be created. Similarly, at any time in the model formulation process, the user can specify data which may be introduced both manually and through external files;

- **MODEL-DATA INDEPENDENCE.** To create a new model instance from an existing model representation, the user simply has to store the model under a new name and change the contents of the index sets and the values of the data items. The structure of the model at the aggregate level remains unchanged;

- **INTEGRATION.** glPS can provide a completely disaggregated representation of the model (instance) which shows the numerical values associated with the model instance.

**A1: FORMULATE MODEL**

- **GENERALITY.** glPS offers complete LP-generality;

- **LABOUR-INTENSIVENESS.** glPS supports indexed parameters and the use of (predefined) subgraphs which reduce the effort of formulating models. The user, however, is responsible for the integration of the subgraph into the new model;

- **CONCRETENESS.** Although the glPS syntax is a direct translation of the mathematical notation and unlike AC-graphs it does not have an analogy with physical flows, it does support the creation of subgraphs;

- **CONCEPTUAL MODULARITY.** glPS supports conceptual modularity;
PART 1 - INTRODUCTION

- **COMPLETENESS.** The representation of an LP model in gLPS is complete.

A3: SPECIFY MODEL INSTANCE

- **DATA-EXCHANGE CAPABILITIES.** Data can be supplied through ASCII flat files and manually through the user interface at the disaggregate level.

A4: SOLVE MODEL

- **SOLVER INDEPENDENCE.** LPL includes a small LP Solver. LPL is linkable to XA, CPLEX, and LINDO (through MPSX).

A5: ANALYSE SOLUTION

- **SENSITIVITY ANALYSIS.** Full support.

A6: CREATE REPORTS

- **REPORTS.** LPL offers a set of pre-defined non-graphic solution reports which can be customised by specifying what data should be included (so-called ‘masking’ statements). LPL provides a graph of the constraint-coefficient matrix.

### 3.4.10 LOGS

LOGS[Brown et al, 1986] is designed on top of IBM’s MIP/370 (MI)LP Solver that requires models to be specified in MPS format. IBM’s ISPF package is used to facilitate the development of customised menu-driven user interfaces for viewing and changing data, submitting runs, and viewing results once models have proven to be satisfactory. In the first stages of using LOGS, only rudimentary interfaces for developing models are used.

Brown’s description of LOGS does not include an example of the syntax of the DML but only a description of its capabilities. We have not attempted to contact the authors. The outline of LOGS’ DML provided in this section therefore contains no examples.

**Syntax**

The representation of a model using LOGS’ DML amounts to the specification of a network of facility nodes, which correspond to geographically distinct facilities, such as suppliers, plants, machines, warehouses and markets, connected by directed arcs which represent flows of products. Each arc represents a single type of product. A
SURVEY OF EXISTING TOOLS

facility at which several activities occur may be modelled by introducing process activities within the facility nodes.

Activities, by default, are defined to be either the sum of the input arc flows or the sum of the output arc flows. For any non-supply, non-market nodes, the sum of the input flows is equivalent to the total output flows. Non-default material-balance equations can be defined by changing the relative weights of both the input and output flows thus preserving the equality of both weighted totals. A product activity is defined as the activity of a specified input or output arc. The user can define fixed ratios between pairs of input and output products and between a reference output product and other input and output products. Brown et al. refer to this latter type of relationship between the arcs associated with an activity as a ‘recipe’.

A logical (binary) variable may be associated with the activities at each node whose value indicates whether or not the activity is ‘in operation’. Consequently, LOGS can deal with fixed operating costs, close-down charges, and logical constraints on the logical variables associated with the facility nodes and process activities. LOGS also supports linear approximations representing convex cost structures.

LOGS’ DML supports upper and lower bounds on the activity levels and on the flows associated with the arcs.

A1, A3: FORMULATE MODEL, SPECIFY MODEL INSTANCE

- **DIFFERENTIATION.** Once the system is configured with models that have been validated and have proved to give satisfactory results, a dedicated user-interface is built on top of the models. Hence, the formulation of models and the specification of model instances are completely separated;

- **MODEL-DATA INDEPENDENCE.** Brown et al. (p223) view LOGS’ DML as a ‘data-input’ language in which the definition of data is expanded to include problem structure as well as numerical information. From this statement, we conclude that LOGS uses the DML to describe model instances;

- **INTEGRATION.** Depends on the actual implementation when using tailored interfaces.

A1: FORMULATE MODEL

- **GENERALITY.** The generality of LOGS is limited to problems that can be mapped on the underlying network representation. Although the applicability of LOGS is
not necessarily limited to logistics planning models, its DML is likely to lose much of its intuitiveness when dealing with models that do not have a network structure. Finally, the applicability of the system is limited to static, single-period, models which do not capture the timing and phasing of capacity expansion programs, and other activities with a time component [Bender,Northup,Shapiro,1981,p171];

- **LABOUR-INTENSIVENESS.** From Brown et al. it seems that LOGS does not support generalisations of relationships. The arcs, for example, represent specific flows of products. Each aggregate flow of products between any two sites A and B that is relevant to the model must be represented by a separate arc instead of being able to represent all product flows from A to B by a single directed arc;

- **CONCRETENESS.** The language primitives used in LOGS are based on network formalisms many people find intuitively appealing;

- **CONCEPTUAL MODULARITY.** LOGS does not support hierarchical abstraction;

- **COMPLETENESS.** LOGS’ DML is incomplete.

**A3: SPECIFY MODEL INSTANCE**

- **DATA-EXCHANGE CAPABILITIES.** No information available.

**A4: SOLVE MODEL**

- **SOLVER INDEPENDENCE.** LOGS interacts with IBM’s MIP/370 using MPS-format files.

**A5: ANALYSE SOLUTION**

- **SENSITIVITY ANALYSIS.** No information available.

**A6: CREATE REPORTS**

- **REPORTS.** LOGS produces reports from the output generated by the Solver that contain the same terms that were originally used to describe the problem. The Model Processor suppresses data that is generated by LOGS as part of the modelling process of which the user has no knowledge, for example, the values of binary variables used to model concave cost curves (linear approximation).
3.4.11 LPFORM

The MS that is used in LPFORM is essentially a hybrid scheme since it unites the expression capabilities of network representations with customisable icons representing collections of LP-activities, so as to incorporate the concepts of hierarchical abstraction, top-down decomposition and reuse of previously defined submodels.

**Syntax**

At the aggregate level, a network in LPFORM consists of activity blocks connected by directed links. Two types of links can be distinguished: logical links, which only have a meaning in the real-world system and not included in the LP-model, and flow links. Flow links represent either space or time links between two activity blocks. The ‘Def-Transport’ command enables the user to define the properties of a flow link including the commodities transported, capacity limits, gains or losses, etc.

Users can build the network in layers of increasing detail with the properties and connections of higher levels being inherited by lower levels as a default. This process of steps defining each next lower level of activity blocks and connecting links is repeated until the network structure of the problem is completely specified. To indicate the presence of activities within the activity blocks, the user inserts either a generic activity icon representing a new activity or an icon representing a previously defined sub model. An activity is defined by its activity set (which specifies the set of columns occupied by the variable in the LP matrix) and by its inputs and outputs (from which the rows intersected by the activity can be inferred), by the type of the objective function (cost/profit) and the objective-function coefficients, and optional (generalised) lower- and upper bounds on the activity level (cf. Figure 3-12). If the user specifies a previously defined sub model, using the ‘Call-Model’ command, the imported structure must be embedded within the current model by renaming the usually rather abstract identifiers of the index sets, variables, and coefficients. After specifying all activities, the user specifies the type of objects that form the index sets — not the members of these sets. Finally, the user specifies the direction of the optimisation process.

![Figure 3-12 Illustration of LPFORM’s modelling scheme from the activity viewpoint.](image-url)
PART I - INTRODUCTION

The above outline of formulating models in LPFORM does not mention the specification of constraints. The construction of the complete algebraic statement, which in fact comes down to the assembly of constraints, is left to the system. This concept of automated model construction from user-defined model components is what Murphy, Stohr and Asthana (1992) refer to as the ‘piecemeal’ approach. The index-based construction rules that are used by LPFORM to perform this task are the subject of Murphy, Stohr and Ma (1992).

In addition to the activity-oriented approach to modelling, LPFORM also provides the option to take a constraint-oriented viewpoint. Finally, the model builder is enabled to edit the algebraic statement of the complete model using LPSPEC\(^1\) which is a language similar to GAMS [Brooke et al., 1988]. After LPFORM has constructed the complete algebraic statement, it automatically generates a so-called ‘Data Schema’ which contains an entry for each of the exogeneous parameters of the model. The Data Schema forms the basis for the specification of model instances. Using the Data Schema, the user subsequently specifies the relational table, the columns that serve as the relation’s primary key fields, and the column containing the numerical values. This information and the specification of the members of the index sets enables LPFORM’s Model Processor to access the data in remote relational database systems.

A1, A3: FORMULATE MODEL, SPECIFY MODEL INSTANCE

- **DIFFERENTIATION.** LPFORM separates the formulation of models from the specification of model instances;

- **MODEL-DATA INDEPENDENCE.** LPFORM supports multiple model representations and uses a separate data-schema to specify the data associated with model-instances;

- **INTEGRATION.** LPFORM’s user interface provides access to the Data Scheme associated with a model.

A1: FORMULATE MODEL

- **GENERALITY.** Although the MS is most suited to represent models with a strong emphasis on physical flows, it provides the possibility to model any LP-model;

- **LABOUR-INTENSIVENESS.** The use of indexing expressions and the piecemeal approach of LPFORM reduces the effort that is needed to formulate models;

SURVEY OF EXISTING TOOLS

- **CONCEPTUAL MODULARITY.** LPFORM strongly supports conceptual modularity, it supports the grouping of logically related activities, its MS is compact and it is likely to attract inexperienced users;

- **COMPLETENESS.** The reason to deviate from LPFORM is that its MS is implicitly stated; although the representation of a model in LPFORM contains all the information that is necessary to construct the complete algebraic statement, the total picture in terms of activities and constraints remains hidden to the model builder. Murphy, Stohr and Asthana (1992) recognise this too: "(...) users may feel uncomfortable about leaving things 'up to the computer' and may not obtain as detailed an understanding about the way the model components relate." It is not the piecemeal concept itself that is the point here, but rather the fact that LPFORM's modelling scheme leaves the user without a choice.

A3: **SPECIFY MODEL INSTANCE**

- **DATA-EXCHANGE CAPABILITIES.** LPFORM contains a DBMS similar to IBM's SQL which interacts with remote relational database systems using ASCII files.

A4: **SOLVE MODEL**

- **SOLVER INDEPENDENCE.** LPFORM uses MPS-format files to communicate with LINDO [Schrage, 1987] which is a general purpose (MI)LP Solver;

A5: **ANALYSE SOLUTION**

- **SENSITIVITY ANALYSIS.** Full support.

A6: **CREATE REPORTS**

- **REPORTS.** LPFORM contains ANALYZE [Greenberg, 1983] which is a matrix and solution file analyser.
PART I - INTRODUCTION
4. OUTLINE OF THE ACTIVITY / CONSTRAINT - OBJECT MODELLER

OUTLINE
Chapter 1 stated the objective of this thesis and provided an outline of our approach to the pursuit of this objective but provided little information on the strategic and tactical production planning process and the applicability of (MI)LP-models in this area (cf. Chapter 2), or on the state-of-the-art in today's (MI)LP-based tools for decision making purposes (Chapter 3).

Against this background, we are able to outline the functional architecture of the system that is presented in this thesis, to contrast its design with existing (MI)LP-based tools for supporting strategic and tactical manufacturing planning, and to outline the design of its components. The proposed system is named Activity/Constraint - Object Modeller — ACOM for short.
PART I - INTRODUCTION

design of the system since it is used to integrate the representation of a modelled entity in terms of the (MI)LP-modelling scheme and the data-modelling scheme taking advantage of the strong support for modularity that has been consistently carried through in the design of both modelling formalisms.

The inherent flexibility and generality of the proposed system ensure that the way in which the system may be operated closely resembles the way of using an (MI)LP-modelling system. For this reason, it is natural to take the (MI)LP-model development process using a modelling system as outlined in Section 3.4.3 as a starting point to discuss the features of the components listed above. The global task description of these components within the context of an (MI)LP-modelling system was discussed in Sections 3.2.4 and 3.2.5. Figure 4-1 shows a copy of Figure 3-6 which depicts the decomposition of the model development process into clusters of activities using Structured Analysis (cf. Section 3.2).

![Diagram of model development process]

**Figure 4-1** Functional analysis of using an (MI)LP-modelling system (cf. Figure 3-6).

The inclusion of an (MI)LP-modelling facility, a data-modelling facility, and a model-data linkage mechanism that integrates these, enables us to differentiate the support to the first three model-development activities depicted in Figure 4-1. Consequently, the proposed architecture is in line with the differentiation criterion that was used to compare (MI)LP-modelling systems (cf. Section 3.4.5).
4.2.1 Outline of the (MI)LP-modelling scheme

In the previous section, four key words were used to typify the objective of this research. Three of these apply to the design of the (MI)LP-modelling scheme that is presented in this thesis: modularity, generality, and concreteness. (Flexibility is inherent to the inclusion of a modelling facility.) The implications of these key words to the design of the modelling scheme are now discussed:

- **Modularity.** The use of building blocks was introduced to facilitate the process of customising the system. Given the requirement that the system must be capable of dealing with tailored planning models, the concept of using building blocks may also be used to facilitate the formulation of (MI)LP models.

Section 3.4.3 indicated the practical value of a modular (structured) representation of the constraint-coefficient matrix by showing that many models contain groupings of logically related model activities that share the same structure in terms of zero and non-zero blocks. Activities within these groups have relatively few linkages with activities from other groups. Within the context of an (MI)LP-modelling scheme, a building block may be viewed as a model substructure that contains such a grouping of logically related model activities.

The proposed MS supports hierarchical abstraction, the grouping of logically related activities, top-down decomposition, reusability of previously defined model structures, and model templates that provide limited support, at the semantic level, to the composition of complete models from previously defined submodels;

- **Generality.** The generality requirement demands that the modelling scheme is capable of representing any (MI)LP model (cf. Section 3.4.5);

- **Concreteness.** The advantage of a graphical model representation is that it has the potential\(^1\) to provide a representation that is better suited for communicating the structure of the model to non-expert model builders than do algebraic model representations. Glover, Klingman, and Phillips (1992,p8) argue that network-based graphical modelling schemes in particular:

1. Facilitate feedback and review through the pictorial element;

2. Improve model accuracy through the increased visibility of problem interconnections, thus reducing the chances of leaving out important relationships and of inadvertently creating bogus ones.

\(^1\) Geoffrion's (1987,1989) Structured Modelling also provides a graphical network-based model representation. Many people find it difficult, though, to discern the structure of the models (cf. Section 3.4.1).
PART I - INTRODUCTION

Logistics planning models often contain a significant number of activities that affect the flow of commodities. The Activity/Constraint - graph formalism discussed earlier in Section 3.4.8 provides an analogy to physical flows. Moreover, AC-graphs are based on network primitives which have proven to be intuitively appealing [Brown et al.,1986; Glover et al.,1992]. The fact that both gLPS, LOGS, LPFORM, and Netforms use network-based model representations provides circumstantial evidence in favour of the above statement. For these reasons, the (MI)LP-modelling scheme is based on the AC-graph technique.

Section 3.4.5 listed a number of criteria for comparing (MI)LP-modelling schemes which were derived from the functional decomposition of the model development trajectory as depicted in Figure 4-1. These criteria include the five features that have already been discussed: differentiation, integration, modularity, generality, and concreteness. The implications of the criteria that remain are:

- **MODEL-DATA INDEPENDENCE.** The proposed modelling scheme differs from the AC-graph formalism in that it separates the structure of the (MI)LP models from the data that is specific to model instances;

- **COMPLETENESS.** The modelling scheme depicts all generalised relationships between the model components explicitly;

- **LABOUR-INTENSIVENESS.** The support for indexed model parameters together with the support for the assembly of models from pre-defined substructures (modules), reduces the effort that is needed to formulate models when compared to original AC-graphs. If the flow-analogy is appropriate, it enhances the speed of formulating models.

Chapter 5 presents the conceptual design of the AC-graph based (MI)LP-modelling scheme.

### 4.2.2 Outline of the data-modelling scheme

The design of the Data-Modelling Scheme (DMS) was largely motivated by the objective which was to use building blocks to facilitate the configuration of the system. In the introductory chapter (Section 1.7) it was argued that, given the requirement that the system must offer general (MI)LP-modelling capabilities and since, therefore, the data requirements (both the nature and the format of the data) cannot be known in advance, it was anticipated that a feasible approach to the implementation of the data structures associated with a building block would be to use a data-modelling scheme with strong support for conceptual modularity:
**OUTLINE OF THE AC-OBJECT MODELLER**

- **MODULARITY AND CONCRETENESS.** The (MI)LP-modelling scheme supports the grouping of logically related model activities. Likewise, the DMS supports the grouping of attributes to form a data abstraction of an actor or notion in the problem domain, e.g., a production plant, a transportation infrastructure consisting of sites connected by transport links, or a production schedule. The advantages of this approach are:

  1. That it provides the means to structure the database to conform with the structure of the underlying problem domain;

  2. That important parts of the modular arrangement of the models and the database, in particular those parts that represent actors in the problem domain, may run in parallel. This in turn provides special opportunities to integrate both types of representations of a modelled object. Section 0 elaborates on this subject in more detail when outlining the design of the model-data link.

The attributes that make up a data abstraction consist of both a key part for identification and a data-type specification. Attributes may represent properties which depend exclusively on the object being modelled, for example, the annual operating cost of a manufacturing plant. In a relational data model, each of these properties would be represented by a single column of a relation (in 1NF, cf. appendix B) representing that particular type of object. Alternatively, an attribute may represent a relation between the modelled entity and one or more other data abstractions. In its present form, the DMS restricts the permissible data types to common alpha numeric and numeric data types, i.e. strings, reals, and integers, and a referential data type that may be used to refer to an object of a pre-specified type.

To illustrate the differences between the proposed data-modelling scheme and the relational model, consider the relational model depicted in Figure 4-2. The model contains three relations: PLANT, PRODUCT, and UNIT MANUFACTURING COST. (It is

<table>
<thead>
<tr>
<th>PLANT RELATION</th>
<th>FIXED COST</th>
<th>data attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLANT:A</td>
<td>250,000.00</td>
<td></td>
</tr>
<tr>
<td>PLANT:B</td>
<td>120,000.00</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRODUCT RELATION</th>
<th>data attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT:1</td>
<td></td>
</tr>
<tr>
<td>PRODUCT:2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNIT MANUFACTURING COST RELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLANT #</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>PLANT:A</td>
</tr>
<tr>
<td>PLANT:A</td>
</tr>
<tr>
<td>PLANT:B</td>
</tr>
</tbody>
</table>

*Figure 4-2** Illustration of modelling references using the relational datamodel.

121
customary to append the term ‘relation’ to the name of the tables.) The latter represents a relationship between PLANTS and PRODUCTS. Both the PLANT relation and the PRODUCT relation contain two instances (rows). The rows of the unit manufacturing cost relation are uniquely defined by the first two attributes whose values must appear in the first columns of the two tables that form the relation: i.e., PLANT and PRODUCT.

The n columns of a relational table represent the attributes of the relation. Formally, a relation R defined over n—not necessarily distinct—sets D₁,..., Dₙ is a set of n-tuples (d₁,...,dₙ) such that d₁ ∈ D₁, ..., dₙ ∈ Dₙ. The Dᵢ (1 ≤ i ≤ n) define the domain of the attribute values dᵢ. The key of a relation is the minimum non-empty subset of its attributes, such that the values of the attributes comprising the key uniquely identify each tuple of the relation [Ötszu, 1991,p18]. A reference from relation R to relation R' is ‘modelled’ by including the key of R as part of the key of R' — a so-called ‘foreign’ key (cf. Figure 4-3). The relational scheme, however, does not support the notion of foreign keys explicitly. What it does support is the use of shared domains: the same set of permissible attribute values is used for the primary key of R and for the foreign part of the key of relation R'. In addition to the relational scheme, referential integrity constraints must ensure that the foreign key of a tuple R'(j) of R' actually corresponds to an existing tuple R(i) in relation R.

![Figure 4-3 The relational model: modelling references with foreign keys.](image)

In the relational model, logically related data is scattered across the database while references are not modelled explicitly. However, as was observed earlier, it is more natural to group the properties of the PLANT relation with the attributes of the UNIT MANUFACTURING COST relation into a single data abstraction: PLANT. In doing so, one automatically obtains the desired cohesion between the various logically related data attributes. The DMS that is presented in this thesis supports the creation of such data abstractions. Figure 4-4 shows a graphical illustration of a data model which may be created by using the proposed DMS. The data structures shown in Figure 4-4 may be used to store the relational data depicted in Figure 4-2 using the
same key format: it is the organisation of data and the mechanism for accessing the
data that makes the data models different.

The model shown in Figure 4-4 consists of three distinct data abstractions, each of
which contains a grouping of the attributes associated with the modelled real-world
object: PLANT, PRODUCT, and GEOGRAPHICAL SITE. The data abstraction PLANT
contains an additional referential attribute ‘geographic site’ to an object of type
GEOGRAPHICAL SITE to illustrate the use of referential data attributes.

![Figure 4-4 Illustration of grouping of data attributes.]

As illustrated in Figure 4-4, the data abstractions may contain multiple dissimilar
relations. The ID (identity) of the data abstraction PRODUCT serves as a domain
specifier for the permissible values of the second key field of the PLANT’s ‘unit
manufacturing cost’ property. The brackets indicate that the value of the key field is
drawn from a restricted domain. The identifier of each data abstraction, e.g. PLANT,
serves as an implicit key field to each of its attributes.

Using the relational model to implement the concept of building blocks containing
logically related data attributes would require additional information on cross
references among logically related tables at the conceptual level (cf. Figure 4-5) —
information that is an integral part of the DMS presented in this thesis.

![Figure 4-5 Inter-relational dependencies.]

To summarise: Logically related data attributes (key + data type) may be clustered
to form a data abstraction of a logical entity. A data attribute represents either an
object’s exclusive property or a relation that the object has with other objects.

Besides the importance of a modular design in enabling the creation of building blocks,
it is of equal importance that the design of the data-modelling scheme is compatible
with the characteristics of the system's (MI)LP-modelling scheme. As far as the concept of modularity is concerned, both requirements overlap. In addition, having a match between both modelling schemes implies:

- **Generality.** The data-modelling scheme must be general in the sense that it supports the data requirements of the (MI)LP models. Existing (MI)LP-modelling systems that contain a database management system use the relational data model. The relational approach to structuring the data matches perfectly with the associative way, using sets of indices, of representing the variables, coefficients, and RHS constants of (MI)LP models. These indices are used to establish a connection between the parameter and a column of a relational database table that contains the corresponding attribute. The actual values of the indices, which define an instance of the parameter, are used to compose a key tuple that uniquely identifies a row of the table. The intersection of the row and the column then determines the appropriate database entry.

As illustrated above under the caption 'modularity', the structure of the key part of distinct data attributes using the proposed DMS is similar to that of the relational model. This similarity has been used to design a mechanism for connecting model parameters to database entries that, in part, uses the relational mechanism for identifying database entries;

- **Data Type - Data Instance Independence.** The connection between the parameters of an (MI)LP model and a database requires at least the following two types of information:

1. A numerical data attribute, e.g., a column of a relational table;

2. A set of data objects (records, rows) to determine the data items needed to quantify individual instances of the parameter.

The first type of information is typically structural, in the sense that it should not depend on the properties of specific model instances, since it is generally considered good data-modelling practice to have a specific type of information stored in a single location of the data model.

The second type of information depends on the properties of model instances: more specifically on the contents of the index sets of the parameters.

In line with the functional decomposition of the model development trajectory as shown in Figure 4-1, the (MI)LP-modelling scheme separates the representation of the structure of a model from that of model instances. This requirement is reflected in the model-data independence criterion that was introduced in Section 3.4.4.
From the model builder's point of view, it is most convenient to be able to provide the first type of information for the definition of a database connection at the structural level of an (MI)LP model. This, however, requires that the data model is capable of separating the structural properties of the database from the data objects that contain the actual data. For that purpose, the proposed DMS supports the creation of:

- **Data types**, as illustrated in Figure 4-4, which define the key and data type of a cluster of logically related data attributes;

- **Data objects** that can be generated from a previously defined data type, a mechanism that is frequently referred to as object instantiation. The data type then defines the format of the data attributes of the data object whereas the data objects contain the actual data. Note the similarity to the distinction between a type definition and a variable declaration in conventional programming languages.

Thus, a data type defines an entire class of similar data objects, just as an (MI)LP-model defines an entire class of model instances.

Figure 4-6 illustrates part of the data model depicted in Figure 4-4 and some objects with their mutual dependencies generated from this particular model (for ease of exposition, the PLANT's referential attribute 'geographical site' is omitted). To improve the legibility of the data, attributes having 'foreign' keys like the 'unit manufacturing cost' property of the PLANT-s are depicted in separate tables.

We conclude this section with a few additional remarks. First, the DBMS that is included in some (MI)LP-modelling systems may be viewed as an external data model (cf. Section 3.2.4, Figure 3-3). The DBMS that is presented in this thesis serves as a tool for specifying the kinds of data and the data organisation that are permissible in the database. Hence, the DBMS defines an interface at the conceptual level. Note, however, that in either case the database may be an internal database or an external database system, using any kind of data model. A DBMS has the freedom to implement a datamodel which differs from the model that the database uses internally. The conceptual design of the proposed data-modelling scheme is presented in Chapter 6.

An important advantage of a relational DBMS is that it facilitates downloading data from external databases. Chapter 6, however, will demonstrate that the data model presented can be represented as a relational model. This 'mapping' secures the interfacing capabilities of the proposed system.
### 4.2.3 Outline of the model-data link: AC-objects

From the previous sections, it becomes clear that *Modularity* played a key role in the design of both the (MI)LP-modelling scheme and the data-modelling scheme. Due to this approach, the logical structure of the problem domain is made explicit in both the organisation of the models and the database as a result of which the modular arrangement of both the models and the database may be broadly parallel. This similarity makes it possible to integrate the representation of a modelled entity in terms of logically related model activities and associated constraints, and an accompanying data abstraction (data type) in an intuitive way. This integration amounts to using the graphical representation of the (MI)LP-model (sub)structure as a means to obtain direct access to the accompanying data. This approach is likely to improve the degree of concreteness of the system as a whole, since a collection of logically related activities representing some real-world entity, together with an associated data type, may be viewed as serving both functional abstraction and data abstraction. This corresponds to the notion of ‘objects’ that is becoming increasingly popular as a building block for the design of complex systems [Booch, 1992; Meyer, 1988].
The representation of a modelled entity in terms of the AC-graph based (MI)LP-modelling scheme, together with an associated data abstraction in terms of the DMS, is named an AC-object. In keeping with this terminology, we called the system as a whole: Activity/Constraint-Object Modeller — ACOM for short.

DATABASE NAVIGATION
The activities and constraints that are part of an AC-object’s (MI)LP-model representation should apply to the entity being modelled. Consequently, the data attributes that are necessary to create model instances are either part of the data abstraction of the AC-object itself or the data are accessible from the data abstraction through referential attributes. Each database connection between a model parameter and a data attribute uses the AC-object’s data abstraction as an Entry-Point (EP) to the database (cf. Figure 4-7).

![Figure 4-7 AC-objects: The data abstraction serves as an EP to the database.](image)

A relational DBMS is less suitable for the implementation of the concept of AC-objects where a single data abstraction serves as the entry point to the database for all database connections between the parameters of the (MI)LP-model substructure and the appropriate data attributes.

The process of finding a path through the database, relative to the data type that serves as EP, using references to move from one data abstraction to another is referred to as (database) navigation, in analogy to the terminology that is used with object-oriented databases. An AC-object provides a physical connection between the parameters of the (MI)LP-model (sub)structure and the database by fixing that particular part of the database path that the parameters have in common. That is the part of the database path prior to the data abstraction associated with the AC-object which serves as EP. The composition of the fixed part of the database path depends on the position of the AC-object within the overall hierarchy of the model. Chapter 7, which deals with the conceptual design of the model-data link, will elaborate on this subject.
PART I - INTRODUCTION

4.2.4 Overview

To conclude this outline of the functional architecture of the Activity/Constraint-Object Modeller, Figure 4-8 shows a structured analysis of the configuration process of ACOM with tailored models and an accompanying data model. Figure 4-8 also indicates the relationships with the activities depicted in Figure 4-1.

Figure 4-8 Functional analysis of the configuration process of ACOM with tailored (MI)LP-models and an accompanying data model.
PART TWO

Conceptual design
5. DESIGN OF THE (MI)LP-MODELLING SCHEME

OUTLINE
The (MI)LP-modelling scheme that is presented in this chapter is based on the Activity / Constraint graph formalism outlined in Chapter 3. Unlike AC-graphs, the modelling scheme that is presented in this chapter separates the representation of (MI)LP models from that of model instances and supports the use of indexed parameters, submodels (hierarchical abstraction), and model templates.
5.1 Introduction

The decomposition of the model development trajectory described in Section 3.4.4 is based on separation of the structural properties of a model, the required data structures, and the numerical data that is specific to model instances. To formalise the gradual development of an (MI)LP model and related model instances when using the modelling system that we propose, we need a notation which resembles the algebraic notation that is customarily used to denote (MI)LP models. The usual algebraic notation, as will be explained shortly, is best suited to represent a model after the specification of its structure, the data structures, and the contents of the index sets which define, in part, the data set for a related model instance. Section 5.2, therefore, introduces two notational variants which will be used to denote the result of the first and the second stage of the model-development trajectory respectively: i.e., (A1) formulate (MI)LP model; and (A2) define data model (cf. Figure 3-6). Note, however, that the formal notation remains hidden to the user of the modelling system. Finally, in Section 5.3, the AC-graph based modelling scheme will be presented.

5.2 Formal notation

In Section 3.4.3 we observed that in many cases the structure of the problem domain underlying an (MI)LP model may be used to partition the activities and constraints into groupings, each of which represents a particular actor. We refer to a model representation that embodies such a superimposed structure as a structured representation of an (MI)LP model. A non-structured representation views the activities and constraints without a superimposed partitioning.

5.2.1 A non-structured referential representation of an LP-model

A non-structured matrix representation of an LP model in minimisation format can be denoted as:

\[ \text{Min. } c^T x, \text{ subject to: } Ax \diamond b, x \geq 0. \]  

(5-1)

where: \( c \) denotes a vector of objective-function coefficients, \( x \) a vector of decision variables, \( A \) a matrix of constraint coefficients, \( b \) a vector of RHS constants, and \( \diamond \) a vector-like ordering of (in)equality signs whose elements equal \( \leq, \geq, \) or \( = \).
DESIGN OF THE (MI)LP MODELLING SCHEME

If we suppose that the activities of the model shown in (5-1) are indexed over \( J \) and that the constraints are indexed over \( I \), its parameters may be denoted as: \( x_i, c_j, b_j, \) and \( a_{ij}, \) where \( i \in I \) and \( j \in J \). An index represents an arbitrary member of an index set. An index set is a set of objects pertinent to a model [Fouger et al., 1990]: e.g., periods of time, products, or plants. The algebraic formulation of the non-structured form of an LP model may be denoted as:

\[
\text{Min. } \sum_{j \in J} c_j \cdot x_j, \text{ subject to: } \forall i \in I : \sum_{j \in J} a_{ij} \cdot x_j \leq b_i, x_j \geq 0. \quad (5-2)
\]

The parameters of a model may be conceived as functions which have the Cartesian product of the index sets, in the same order as the tuple of indices, as their domain: e.g., \( x:J \rightarrow \mathbb{R}^n \), or \( a:I \times J \rightarrow \mathbb{R}^n \). The tuple of indices associated with an indexed parameter then defines an arbitrary element of the function domain. A function domain may contain the same index set more than once. In such cases, the indices alone may not be sufficient to distinguish between the different roles that the members of these index sets play. Then it is the ordering of the indices, as implied by the composition of the index tuples, that must provide this type of essential information.

The notation \( i \in I \) may be used to denote an element of a set of known objects. Alternatively, the same expression may be used to denote an element of a set whose contents is intentionally left undefined. The contents of an index set is typically concerned with the specification of a model instance and is irrelevant to the description of the structure of a model. In order to make this distinction part of our notation, we introduce the term index domain to refer to an index set with undefined contents and an index symbol as an unambiguous reference to an index domain. Let \( I \) denote an index domain, then the declaration of an index symbol \( i \) with \( I \) as its domain is denoted as: \( I \rightarrow i \). Using index symbols, we can denote the non-structured form of an LP model as:

\[
\text{Min. } \sum_{J \rightarrow j} c(j) \cdot x(j), \text{ subject to: } I \rightarrow i : \sum_{J \rightarrow j} a(i,j) \cdot x(j) \cap(i) b(i), x(j) \geq 0. \quad (5-3)
\]

The 'summation' expression shown in (5-3) must be interpreted as a symbolic description of a summation that results after the modelling system has constructed the 'equivalent' algebraic formulation.

In our approach, the second model-development activity, i.e. A2: "Define data model", includes the establishment of a connection between the parameters of the model and the database. (Since at this stage we are still concerned with the structural properties of the model, the database connection is specified at the conceptual level of the database which deals with the structure of the data, not with the data itself.) At the level of model instances, a tuple of indices will be used to locate the numerical values for the parameters. For that purpose, the order of the indices should match the order of the key
fields of the database relation. The ordering of the index symbols as shown in (5-3) emerges from the establishment of a database connection. To indicate the absence of an ordering among the index symbols prior to this stage, i.e. stage A1: “Formulate (M1)LP model”, the index symbols on the model parameters are denoted as sets. Using both index symbols and the set notation, we can denote the non-structured form of an LP model as:

\[ \text{Min. } \Sigma \{ J \rightarrow j \} \ c \{ j \} \cdot x \{ j \}, \text{ s.t.: } \{ I \rightarrow i \} : \Sigma \{ J \rightarrow j \} \ a \{ i, j \} \cdot x \{ j \} \cdot b \{ i \}, \ x \{ j \} \geq 0. \quad (5-4) \]

We refer to the notation shown in (5-4) as the referential notation. Note that the use of a set notation implies that the index symbols must have unique identifiers.

As depicted in Figure 5-1, our approach using both the referential and ordered-referential variants of the algebraic notation is completely in line with the gradual development of an (M1)LP model during the first three stages of the model development trajectory. Stage A1 concerns the formulation of the model using the referential notation. Stage A2 deals with the specification of the database connections for the model parameters. The resulting ordered referential notation embodies an ordering of the index symbols on the model parameters. The brackets denote a tuple of index symbols. Usually, the brackets are omitted. Finally, in stage A31, the actual contents of the index sets is specified.

\[
\begin{align*}
\text{Formulate (M1)LP-model} & \quad \downarrow & \quad \text{referential notation} \\
A1 & \quad \downarrow & \quad I, J \quad I \rightarrow i, J \rightarrow j \quad a \{ i, j \}, b \{ i \}, c \{ j \}, x \{ j \} \\
\text{Define data model} & \quad \downarrow & \quad \text{ordered referential notation} \\
A2 & \quad \downarrow & \quad I, J \quad I \rightarrow i, J \rightarrow j \quad a \{ i, j \}, b \{ i \}, c \{ j \}, x \{ j \} \\
\text{Specify index sets} & \quad \downarrow & \quad \text{algebraic notation} \\
A31 & \quad \downarrow & \quad I = \{ 1, 2 \} \quad i \in I \quad \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, b \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, c \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, x \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}
\end{align*}
\]

**Figure 5-1** Three stages in model development.

The order of the index symbols on the model parameters does not affect the syntactical correctness of an (M1)LP model. Hence, we can use the referential formulation of a model to verify its syntactical correctness. For that purpose, the referential form is decomposed in a way similar to the decomposition of the algebraic formulation of LP models described by Murphy, Stohr, and Ma (1992). The paper of Murphy et al.
focuses on the rules that guide the assembly of a complete model from a set of algebraic pieces. Basically the same rules can be used to verify the syntactical correctness of a complete algebraic statement.

Following Murphy et al., we term an RHS constant, together with its (in)equality sign, an RHS piece. The left-hand side (LHS) of a constraint consists of one or more LHS pieces. An LHS-piece is made up of the product of a coefficient and a variable together with its associated summations (if any). For example, the LHS piece of the structural constraints denoted in (5-4) equals \( +\sum\{J \rightarrow j\} \cdot a\{i,j\} \cdot x\{j\} \) whereas the RHS piece equals \( \emptyset \{i\} \cdot b\{i\} \).

Also following Murphy et al., we require that the set of index symbols on the LHS pieces and the RHS piece that make up a constraint are equal. We refer to this requirement as the Matching Pieces Requirement (MPR). The set of index symbols of an LHS piece, without the associated summation, is defined as the union of the sets of index symbols for the variable and the coefficient. The effect of a summation is to remove the summed index symbols. The set of index symbols on the RHS piece equals that of the RHS-constant. Hence it follows that, for example, the set of index symbols on both the LHS piece and the RHS piece of the structural constraints denoted in (5-4) equals \( \emptyset \{i\} \) thus satisfying the MPR.

### 5.2.1.1 Index-symbol expressions

Index-symbol expressions constitute an essential part of the referential notation. In the algebraic notation, the meaning of the index expression,

\[
+\sum\{j \in J, k \in K: j + k = n\} \cdot a\{j,k\} \cdot x\{j\},
\]

for given \( n \in N \), is evident: a summation over indices \( j \) and \( k \) restricted to values satisfying \( j + k = n \). In terms of the referential notation, it is natural to write:

\[
+\sum\{J \rightarrow j, K \rightarrow k: j + k = n\} \cdot a\{j,k\} \cdot x\{j\}, \tag{5-5}
\]

where \( j + k = n \) represents an expression on the index symbols \( j, k \), and \( n \) with \( N \rightarrow n \). The relationship between referential index-symbol expressions and algebraic index expressions is similar to the relationship between the expressions in a computer program and the executable form that results after compilation. The modelling system converts the index-symbol expressions into 'executable' index expressions for specific sets of index values. The modelling system then uses these index expressions, which
PART II - CONCEPTUAL DESIGN

are embedded in the structure of the model, to produce an output statement of the model instance.

From the nature of the elements of the index domains, we distinguish two types of operations on index symbols:

1. **ARITHMETIC OPERATIONS** which only apply to index symbols that refer to integer-valued index domains;

2. **BOOLEAN OPERATIONS** which concern the (in)equality of index symbols that refer to index domains that share a common ‘higher level’ domain universe: an index domain that has the index domains of the index symbols as sub domains.

For ease of implementation and without loss of generality, the model builder is required to use a Boolean index operation to explicitly name the result of arithmetic operations which involve multiple index symbols: e.g., \( i + j = n \). It is also required that only one operand of a Boolean expression may consist of an expression containing arithmetic operations.

Index-symbol expressions are often necessary for modelling optimisation problems. Consider, for example, equation (5-6) which shows an inventory-balance constraint that is typically part of many aggregate production planning models. All parameters are measured in the same aggregate units, e.g. direct labour hours: \( i\{t-1\} \) and \( i\{t\} \) represent the inventory at the end of period \( t-1 \) and period \( t \), respectively, that is available for use in subsequent future periods, \( x\{t\} \) represents the production during period \( t \), and \( d\{t\} \) represents the exogenous demand for that period.

\[
T\rightarrow t: t \cdot x\{t\} + 1 \cdot i\{t - 1\} - 1 \cdot i\{t\} = d\{t\} .
\]  

(5-6)

Equation (5-7) shows an example of a constraint the summation of which prescription contains an index-symbol expression. Constraint (5-7) requires that the cumulative production over periods \( t' \) prior to period \( t \) equals the cumulative demand \( D\{t\} \) for that period. The summation prescription denotes a partial sum [Murphy et al., 1992]: index symbol \( t' \) runs over a subset of its domain.

\[
T\rightarrow t: \sum_{t' : t' \leq t} T \rightarrow t' \cdot x\{t'\} = D\{t\} .
\]  

(5-7)

Another application of index expressions concerns the specification of coefficients that have a structural value which is based solely on the values of the indices, for example, an incidence relationship like:

\[
a\{i,j\} := 1 \text{ for } i = j , \text{ and } a\{i,j\} := 0 \text{ for } i \neq j , \text{ where } I \rightarrow i , J \rightarrow j . \]

(5-8)
INDEX-SYMBOL EXPRESSIONS AND THE MPR

An algorithm that verifies the syntactical correctness of a referential formulation of an (M)LP model by using the Matching Pieces Requirement must account for the impact of index-symbol expressions. An MPR-evaluation algorithm which applies the above procedure to determine the set of index symbols on LHS pieces would mistakenly reject referential structures containing index-symbol expressions. For example,

- The set of index symbols associated with the LHS piece shown in equation 5-5 should equal \{n\} instead of the empty set;
- Likewise, the set of index symbols on the LHS piece shown in equation (5-7) should equal \{t\}; not \{\};
- The LHS piece shown in equation (5-6) containing variable \(i\{t-1\}\) belongs to a constraint concerning period \(t\), not to a period \(t-1\) constraint.

In order to maintain the MPR as a basis for the evaluation of the syntax of a referential formulation, two modifications should be made to the procedure for determining the set of index symbols on LHS pieces so that it will be capable of dealing with index expressions:

1. If, at some stage, the algorithm encounters a unary arithmetic expression (acting on a single index symbol), like the expression \(t-1\) used in equation (5-6), then it should be replaced by the index symbol which is acted upon;

2. If, at some stage, the algorithm encounters a Boolean (in)equality expression, say—for ease of exposition—of the form \(\varphi(i_1,\ldots,i_n) \hat{\diamond} j\), where \(\varphi(i_1,\ldots,i_n)\) denotes an arithmetic operation on the index symbols \(i_1,\ldots,i_n\) and \(\hat{\diamond}\) an (in)equality sign, then the \(i_1,\ldots,i_n\) should be replaced by index symbol \(j\).

Figure 5-2 illustrates the modified procedure for determining the set of index symbols on the LHS piece of constraint (5-7). Step 1 determines the set of index symbols on both the coefficient and the variable. In step 2, the second modification is applied in order to deal with the index-symbol expression in the summation prescription.

\[
\begin{align*}
1 \cdot \{t\}' \\
{} \cup \{t\}' = \{t\}' \\
+\Sigma\{T \rightarrow t' : t' \leq t\} 1 \cdot \{t\}' \\
\downarrow \quad \quad \quad \quad \quad \downarrow (modification 2) \\
\{t\} \\
\end{align*}
\]

*Figure 5-2* Illustration of the second modification to the procedure for determining the set of index symbols on LHS pieces.

Again, we remark that the referential notation constitutes the formal basis of the proposed modelling scheme, but that, with the exception of index-symbol expressions, it remains hidden to the user of the system.
5.2.2 The structured form

Figure 5-3 depicts the constraint-coefficient matrix of an LP model with a superimposed structure that results from a partitioning of the activities and the constraints. Suppose that the constraints and activities are partitioned into groups indexed over \( i^0 \) and \( j^0 \) respectively. Let \( b(i^0), j^0 \in i^0 \), denote the \( i^0 \)-th subset of constraints and let \( x(j^0), j^0 \in j^0 \), denote the \( j^0 \)-th subset of activities. Then the submatrix of constraint coefficients associated with \( b(i^0) \) and \( x(j^0) \) can be denoted as \( A(i^0, j^0) \). Figure 5-3 also displays an arbitrary element of \( A(i^0, j^0) \): \( a(i^0, i^1, j^0, j^1) \) with \( i^1 \in I^1(i^0) \) and \( j^1 \in J^1(j^0) \) where \( I^1(i^0) \) and \( J^1(j^0) \) denote the set of row labels and column labels of submatrix \( A(i^0, j^0) \) respectively.

![Figure 5-3 Schematic representation of an LP model with a superimposed structure.](image)

A structured algebraic representation of the LP model depicted in Figure 5-3 can be denoted as:

Minimise \( \sum \{ j^0 \in J^0, j^1 \in J^1(j^0) \} c(j^0, j^1) \cdot x(j^0, j^1) \), \hspace{1cm} (5-9)

subject to:

\[
\forall i^0 \in I^0, i^1 \in I^1(i^0):
\]

\[
\sum \{ j^0 \in J^0, j^1 \in J^1(j^0) \} a(i^0, i^1, j^0, j^1) \cdot x(j^0, j^1) \odot (i^0, i^1) b(i^0, i^1).
\]

In the referential description, as shown in (5-9), \( I^0 \) and \( J^0 \) represent index domains instead of index sets because the actual number of blocks may depend on the properties of a particular model instance. Similarly, the modelling scheme only describes the structure of the blocks in terms of zero and non-zero elements. The actual number of rows and columns for each block depends on the model instance, so again we use index domains: \( I^1(i^0) \rightarrow i^1, J^1(j^0) \rightarrow j^1. \) A final difference between the referential notation...
and the algebraic notation is that when e.g. \( I^0 \) actually denotes a Cartesian product of index sets, then \( I^0 \in I^0 \) denotes a tuple of indices with a similar ordering. At the referential level, however, this ordering is lacking so we cannot use the concise tuple notation. Hence, each of the index domains and the associated index symbols must be denoted explicitly.

\[
\text{Minimise } \sum \{ j^0 \rightarrow j^0, j^1 (j^0) \rightarrow j^1 \} c \{ j^0, j^1 \} \cdot x \{ j^0, j^1 \},
\]

subject to:

\[
I^0 \rightarrow i^0, I^1 (i^0) \rightarrow i^1:
\]

\[
\sum \{ j^0 \rightarrow j^0, j^1 (j^0) \rightarrow j^1 \} a \{ j^0, j^1 \} \cdot x \{ j^0, j^1 \} \left( \bigodot \{ j^0, j^1 \} b \{ j^0, j^1 \} \right).
\]

### 5.3 Compacted Activity / Constraint graphs

In its original form, AC-graphs depict each variable, RHS-constant, and model coefficient together with their mutual relationships. This appears to be unworkable even for small LPs. The nodes and arrows of the compacted AC-graphs that we propose, however, represent indexed parameters. Unlike modifications to the AC-graph syntax presented previously by Murphy et al. (1992a) for representing indexed parameters and ‘side’ constraints, the extensions that we propose are graphically, rather than symbolically, and cover also the support for subgraphs (conceptual modularity), binary and integer variables, and relationships among instances of indexed variables.

#### 5.3.1 Syntax for representing constraints and activity bounds

Logistic planning models often include binary and integer decision variables. The compacted AC-graphs formalism, therefore, contains special icons representing integer and binary decision variables (cf. Figure 5-4). Furthermore, there are distinct icons for each type of (in)equality relationship (cf. Figure 5-4). In practice it appeared that, in order to depict the type of inequality relationship, analogues to lower and upper bounds work more intuitively than mathematical symbols.

![Activity Nodes](image)
![Constraint Nodes](image)

**Figure 5-4** Compacted AC-graph icons.
PART II - CONCEPTUAL DESIGN

Constraint (5-10), using the referential notation that was introduced in Section 5.2, comprises a single LHS-algebraic piece:

\[ l \rightarrow i; + \sum_{j} a[i,j] \cdot x[j] \leq b[i]. \]  \hspace{1cm} (5-11)

Figure 5-5 depicts the compacted AC-graph representation of constraint (5-10). Figure 5-5 also shows the components of the graph as they might appear in a block-schematic representation of the model.

![Figure 5-5 Syntax for representing constraints."

The compacted AC-graph formalism is based on the MPR (Matching Pieces Requirement): the set of index symbols on the LHS-pieces and the RHS-piece of a constraint are equal. The index symbols associated with the LHS pieces of a constraint which are not part of the set of index symbols on the RHS piece are interpreted as summed indices.

If the sign of the LHS-algebraic piece is positive then the arc representing the constraint coefficient incidents the activity node. In the opposite case, when the sign of the LHS piece is negative, the arc incidents the constraint node.

Figure 5-6 depicts a direct translation of the inventory-balance constraint (cf. equation 5-6) into the compacted AC-graph formalism. (For brevity, we have omitted the index symbols on the constraint coefficients.) It is difficult, however, to discern the structure of the constraint from the graph shown in Figure 5-6. It is, therefore, unlikely that a model builder would come up with this graph.

![Figure 5-6 Compacted AC-graph of the inventory constraint: direct translation."

138
Figure 5-7 shows an alternative representation of the inventory-balance constraint which, as opposed to the graph shown in Figure 5-6, uses the analogy with physical flows. From this graph, it is easy to discern the structure of the inventory-balance constraint. The translation into an equivalent referential formulation yields two constraints which, after substitution, can be reduced to the inventory-balance constraint.

\[
\begin{align*}
- x(t) - t(t-1) + i(t) + s(t) &= 0(t) \\
+ s(t) &= d(t) \\
+ x(t) + t(t-1) - i(t) &= d(t); \quad T \rightarrow t.
\end{align*}
\]

**Figure 5-7** Compacted AC-graph of the inventory constraint: intuitive model.

Figure 5-7 contains an undirected arc which represents the bound on the sales activity. AC-graphs depict lower and upper bounds on activity levels as constraint nodes connected to a single activity node. The arc has, however, no parallel with physical flows. In these cases, the use of arrows works counter-intuitively, so we use undirected arcs to represent bounds on activity levels. Undirected arcs yield LHS pieces with positive signs (cf. Figure 5-8).

\[
I \rightarrow i : l[i] \leq \sum_{J \rightarrow j} x[i, j] \leq u[i] \quad \Leftrightarrow
\]

**Figure 5-8** Representing lower bounds and upper bounds.

Figure 5-9 shows an alternative syntax that may be used to model bounds on individual real-valued activities.
Figures 5-5 through 5-7 also illustrate the way in which index-symbol expressions may be embedded in compacted AC-graphs: index symbols and index-symbol expressions may be used interchangeably. Figure 5-7 also illustrates the use of index expressions, e.g. ‘1’, instead of symbolic identifiers for the constraint-coefficients to denote the value of those coefficients that have a structural value which is based solely on the values of the indices. When the index summation associated with the LHS piece involves an index operator, its function prescription, preceded by the summation sign, is added along the arc representing the constraint coefficient (cf. Figure 5-10).

\[ N \rightarrow n, \sum_{|j \rightarrow j, K \rightarrow k} j + k = n \]
\[ a[j, k], b[n], x[j] \geq b[n] \]

**Figure 5-10** Representing index summations with index operators.

### 5.3.2 Uniqueness of activity nodes

The inventory-balance constraint shown in equation (5-6) contains two LHS pieces, \( +1 \cdot i[t] \) and \( -1 \cdot i[t-1] \), the variables of which represent different instances of the same class of variables: \( i \). To be able to depict this type of constraint, both \( i[t] \) and \( i[t-1] \) must have an activity node representation. For the modelling system, however, it must be clear, that both nodes represent essentially the same set of activities. To this end, both the identifier and the set of index symbols of an activity node are used for identification. The modelling system considers two activity nodes that have the same ID and the same set of index symbols as equal. When the procedure for the composition of the set of index symbols on a LHS-algebraic piece as described in Section 5.2.1 is applied, it then follows that the activity nodes for \( i[t] \) and \( i[t-1] \) have equal identifying properties.

### 5.3.3 Syntax for representing the objective function

AC-graphs represent the objective function of an LP model as a constraint node. As decision variables usually have a cost or profit associated, most activity nodes are
connected to the constraint node representing the model's objective function. Consequently, AC-graphs look unnecessarily complex. As an LP model contains only a single objective-function, we may assume that it exists, without first representing it as a constraint node. In the absence of a constraint node representing the model's objective function, we represent the objective-function coefficient associated with an activity by an arrow which is only connected to the activity node. Activity nodes may be connected to only one such arrow.

Equation (5-11) depicts the two LHS pieces which can be composed from an objective-function coefficient $c(j)$ and a variable $x(j)$:

$$ +\sum_{j \rightarrow i} c(j) \cdot x(j), \quad -\sum_{i \rightarrow j} c(j) \cdot x(j) $$

(5-12)

Which LHS piece will be generated depends on whether the model is formulated as a maximisation problem or as a minimisation problem (cf. Figure 5-11).

Figure 5-11 Syntax for representing the objective-function.
5.3.4 A compacted AC-graph of the reference model

Figure 5-12 depicts a compacted AC-graph of the production location allocation model which was introduced in Section 3.4.5. The index symbols along the arcs have been omitted in order to reduce the complexity of the graph.

![AC-graph](image)

*Figure 5-12 A compacted AC-graph of the production location allocation model.*

5.3.5 Relationships among instances of indexed variables

Because indexed activity nodes are used as the basic units of the MS, compacted AC-graphs are not capable of representing relationships among subsets of variables which belong to the same class. We can derive a subset of instances from an indexed variable by replacing one or more index symbols with indices that refer to index domains that are subsets of the domains they replace.

For example, the graph depicted in Figure 5-12 contains an activity node *manufacture* which represents a variable, say $x$, indexed over $Site\rightarrow site$, $Plant\rightarrow plant$, and $Product\rightarrow prod$. Suppose that the index domain $Site$ is intended to contain sites from several countries and that, for some reason, the value of the gross production in a particular group of countries must exceed the value of the production allocated to the remaining countries. If we partition the $Site$-s into sites located in, e.g., the European Union, $EU-Site \subseteq Site$, and sites located elsewhere, $Non-EU-Site \subseteq Site$, the constraint stated above can be denoted as:

142
DESIGN OF THE (MI)LP MODELLING SCHEME

\[ \Sigma \{ \text{EU-Site} \rightarrow \text{eus}, \text{Plant} \rightarrow \text{plant}, \text{Product} \rightarrow \text{prod} \} \cdot v(\text{prod}) \cdot x(\text{eus,plant,prod}) \]

\[ \geq \]

\[ \Sigma \{ \text{Non-EU-Site} \rightarrow \text{neus}, \text{Plant} \rightarrow \text{plant}, \text{Product} \rightarrow \text{prod} \} \cdot v(\text{prod}) \cdot x(\text{neus,plant,prod}) \],

with: \text{EU-Site} \rightarrow \text{eus}, \text{Non-EU-Site} \rightarrow \text{neus}, and where \( v(\text{prod}) \) denotes the monetary value of a product on a common cost basis. If we would represent both subsets of variables by an activity node, the modelling system would interpret the original and the derived nodes as variables referring to distinct classes, since we are using different index symbols in some parts. To be able to represent this type of constraints, one would have to change the criteria for establishing the identity of activity nodes. However, the compacted AC-graph is not a very suitable formalism for representing relationships among instances of variables that belong to the same class, since they often concern logical relationships. Usually, the analogy with physical flows does not apply. To enable the intuitive modelling of logical expressions and to avoid the analogy with physical flows, a modelling scheme that differs from the AC-graph formalism is proposed. To stimulate the notion of a single modelling environment the modelling scheme resembles that of AC-graphs. Figure 5-13 shows the representation of constraint (5-13) which may be obtained by a double mouse click on the icon for the activity. The border of the modelling window that appears on the screen represents the icon of the activity. In this way, the activity node acts as the context to the modelling window. The arcs represent instances of the activity node. To avoid the flow analogy we do not use arrows. Constraint nodes depict the type of (in)equality relationship explicitly. The user-interface allows only the borders to the left and right to be used to connect the constraint node.

![Diagram](image)

**Figure 5-13** Syntax for representing instances of decision variables.

### 5.3.6 Subgraphs

To reduce the complexity of compacted AC-graphs which contain several relatively independent substructures, an octagonal icon is used to represent an AC-subgraph containing logically related activities. The inclusion of an icon representing an AC-
subgraph implies that constraints associated with multiple activities which should preferably be allocated to different subgraphs must be shared by these subgraphs. We term these constraints, which constitute the interface between activities located in different subgraphs, *global constraints*. In contrast to global constraints, *local constraints* have meaning only within the activity-context of an AC-graph. The MS depicts local constraints by circles of solid lines (conforming to the AC-graph syntax) and global constraints by circles of dashed lines.

![Figure 5-14 Icons for subgraphs and shared constraints.](image)

Figure 5-15 shows the creation of a subgraph with a single activity (shaded box) through a bottom-up aggregation procedure. Note that the arcs which connect the subgraph icon with the global constraint node do not represent constraint coefficients — they only indicate the use or production of a shared resource.

![Figure 5-15 Syntax for representing compacted AC-sub graphs.](image)

If a subgraph both uses and produces a shared resource, then two arcs are used to indicate the bi-directional flow, connecting the global constraint node at the aggregate level with the subgraph icon. The AC-graph depicted in Figure 5-16, for example, shows the creation of a subgraph containing two activities that share the same global constraint. Two arcs connect the subgraph icon and the shared constraint node at the aggregate level.
Subgraphs which occur several times within a model, e.g. a subgraph representing a manufacturing plant, may be associated with an index symbol. Assigning an index symbol to the subgraph has the effect that the index symbol automatically becomes part of the set of index symbols associated with the components of the subgraph. Figure 5-17 illustrates the use of indexed subgraphs with the help of the referential notation. The subgraph, labelled as $G$, is indexed over $N \rightarrow n$. Note that, with the exception of the shared constraint node, the index symbol of the subgraph is omitted from the set of index symbols on the components of the subgraph.

$$I \rightarrow i: \sum \{N \rightarrow n, J \rightarrow j\} a(n, i, j) \cdot x(n, j) \leq b(i).$$

**Figure 5-17 Syntax for representing compacted AC-sub graphs.**

**SUBGRAPH TEMPLATES**

To create a subgraph, a model builder would typically put an icon for a subgraph at an empty area on the screen. At that time, the interior of the icon contains a cross to indicate the absence of activities within the subgraph. In subsequent steps the model builder would define the interface of the subgraph: At the aggregate level, the model builder can create a link between the constraint-node icon and the icon representing the subgraph using common direct manipulation techniques. Within the context of the subgraph, when the remainder of the graph cannot be manipulated directly, pop-up menus enable the model builder to import constraint nodes defined at higher levels. Finally, the model builder specifies the contents of the subgraph.

In a top-down procedure, where the user specifies the graph in layers of increasing detail, the model builder may decide to specify the contents of a subgraph by using a
predefined subgraph. Evidently, both subgraphs must share the same interface. An empty subgraph defines in fact a subgraph template. From a syntactical point of view, the contents of any predefined subgraph may be used to instantiate a subgraph template, provided that both share the same interface. If the interface of the predefined subgraph contains any constraint nodes which are not part of the template’s interface (apparently, the template was not intended for that particular use), they become available at the aggregate level. To facilitate the creation of subgraphs which fit a particular template, the modelling system supports the creation of subgraphs from the definition of subgraph templates. The semantics of both AC-graphs must ensure that using the contents of a particular subgraph to instantiate a subgraph template actually makes sense.

Subgraph templates are expected to facilitate the formulation of complete AC-graphs as experienced model builders may configure the system with a library of templates and subgraphs which may be used by inexperienced users to create or assemble complete graphs. Today’s graphical user-interfaces enable an intuitive instantiation of subgraph templates through direct manipulation techniques, e.g., drag-and-drop, as illustrated in Figure 5-18.

![Figure 5-18 Instantiation of a template-sub graph.](image)

### 5.3.7 A modular AC-graph representation of the reference model

The compacted AC-graph depicted in Figure 5-19 is derived from the graph shown in Figure 5-12 displaying the Reference Model. The graph shown in Figure 5-19a serves as the root graph for the graphs shown in Figure 5-19b and c. It contains a subgraph for the activities that take place at a ‘Campus Site’. Apart from the icon representing the subgraph, it also contains the activity node representing the transportation of commodities between different sites. Note that the inbound-flow and the outbound-flow balance constraints are now global. Figure 5-19b shows the subgraph for the ‘Campus Site’. The graph contains a subgraph ‘Mfg Plant’ for the activities that relate to the manufacturing activities at the site. The subgraph is depicted in Figure 5-19c.
Figure 5-19a-c Modular AC-graph of the Production Location Allocation Model.
6. DESIGN OF THE DATA MODEL

OUTLINE
This chapter presents the conceptual design of the data model. That is, the data-modelling scheme, the allowable operations for manipulating both the conceptual model of the database and the data, and the structural (non-application specific) integrity rules that apply. The proposed data model, unlike the relational data model, supports the grouping of logically related data.
6.1 Introduction

An AC-object consists of a compacted AC-graph, containing a collection of logically related activities representing a real-world entity and an associated data abstraction (cf. Section 4.2). The data abstraction contains a grouping of the data attributes that apply to the entity that is being modelled.

A compacted AC-graph only defines the structural properties of an (MI)LP model; different sets of numerical data for the endogeneous parameters of an AC-(sub)graph define different model instances. In order to obtain a similar distinction between the structural and casual properties of the data, the data abstraction associated with an AC-object must contain a specification of the properties of the data attributes that are similar for all model instances: i.e. the identifying label and the data type; not the numerical data that is specific to model instances. The data abstraction associated with an AC-object thus defines the format of a whole class of similar data objects. Therefore, it may be viewed as a (compound) data type. The data objects contain the actual data.

This chapter introduces the conceptual design of the data model. The division of this chapter into sections is largely based on the components of a data model as identified by Codd (cf. Section 3.2.4). Section 6.2 introduces the building blocks of the Data-Modelling Scheme (DMS) and the terminology that applies to these building blocks. Section 6.3 discusses the syntax of the data-modelling scheme. The DMS is not a graphical data-modelling language. Therefore, we have used a pseudo programming language and the familiar syntax diagrams to explain the syntax of the DMS. Although the graphical user-interface of ACOM supports the greater part of the functionality of the DMS, the pseudo language could be used to construct an ASCII character-based user interface for the system's DBMS. Section 6.4 discusses the operations that enable the user and/or the modelling system to manipulate the data structures and to query data. Section 6.5 discusses the integrity rules that must ensure the consistency of the database. After discussing both the syntax, the operations, and the integrity constraints we completed the conceptual design of the data model. Section 6.7 outlines two types of data views, graphics and tabular data summaries, which are based upon the data model. Strictly speaking, the support for data views is not part of the design of a data model or database management system. However, because the prototype limits its support for data views to data-entry forms, we discuss data views in this chapter. Section 6.8 discusses the differences between the data model that is presented in this chapter and the object-oriented data model (cf. Appendix B). Finally, Section 6.9 contains a summary of the terms and notation presented in this chapter.
6.2 The building blocks of the data-modelling scheme

The data-modelling scheme that is presented in this chapter is based on one type of building block or complex data type: a DAT-dictionary. A DAT-dictionary specifies both the identifying label—the key—and the data type of each of the attributes of a class of related data objects: Data Attribute Tables (DATs). A DAT-dictionary may thus be viewed as a reference list, a dictionary, for the entries of the associated DATs. The entries of a DAT may contain both numerical values and references to other DATs to enable both the user and the system to navigate through the database according to the logical structure of the data.

An AC-object consists of a compacted AC-graph and a DAT-dictionary. The entries of the DAT-dictionary represent the data format of the attributes of the associated DATs with the same level of abstraction as is used by the components of the AC-graph to depict the parameters of (MI)LP-model instances. Owing to the absence of a data set, instances of indexed model parameters, e.g. \( x(i) \) with \( i \in I = \{1,2\} \), do not exist at the level of abstraction that is used by compacted AC-graphs to represent an (MI)LP model. Instances of indexed parameters exist for model instances. It is for this reason that DAT-dictionaries cannot distinguish between the data representation of, for example, a variable \( x_1 \) and that of a variable \( x_2 \), so both variables must be represented by a single entry for all \( x(I) \) with \( I \rightarrow i \). The database connection between an indexed model parameter and the attribute of a DAT-dictionary is part of the structural properties of a model description. To construct a model instance, the Model Processor subsystem uses the contents of the index domains to convert the compacted AC-graph into a representation that contains all parameter instances — i.e. all coefficients, variables, and RHS-constants. To quantify a coefficient or an RHS-constant, both the database connection specified at the DAT-dictionary level and the contents of the index domains are used to determine the appropriate DAT and the right entry containing the data (cf. Chapter 7). At the level of DATs, as opposed to the DAT-dictionary level, different entries are used to store the numerical specification of different instances of indexed parameters. Thus the single entry at the DAT-dictionary level representing an indexed model parameter thus defines the data format of several DAT entries.

Figure 6-1 illustrates the links between an AC-graph and a DAT-dictionary on the one hand, and the links between a related (MI)LP-model instance and a set of DATs on the

---

1 In the area of object-oriented programming, it is customary to refer to the type specification of a set of similar objects as an object class. Strictly speaking, an AC-object should be termed an AC-class or AC-type. In that case, the term AC-object would apply to a model instance and an associated data object. The (MI)LP-modelling scheme, however, deals with models and not with model instances just as an object-oriented program manipulates objects and not classes. For this similarity, we have chosen not to use the term AC-class.
other hand. The compacted AC-graph shown in Figure 6-1 represents a constraint that ensures that the capacity utilisation of a manufacturing facility $F \rightarrow f$ does not exceed a given aggregate upper bound $c(f)$. The variable $x(f,p)$ represents the quantities of products $P \rightarrow p$ being manufactured by the facility $f$. The technology coefficient $t(f,p)$ depicts the capacity utilisation per unit quantity of products $p$ at facility $f$. At the level of DATs, Figure 6-1 shows two manufacturing facilities: ‘AMS’ (Amsterdam) and ‘BRX’ (Brussels). The instantiated AC-graph only applies to facility ‘AMS’.

![Diagram](image)

Figure 6-1 Two levels of abstraction in (MI)LP-modelling and data modelling.

The DAT-dictionaries, together with their mutual relationships, make up the conceptual model of the database. The data model can be depicted by a graph in which the nodes represent the dictionaries and the links depict the interdependencies (i.e. cross-references) that exist among the DAT-dictionaries. Bearing this visualisation in mind, the dictionaries serve as direction signs that help the user to navigate through the database. This also explains the choice of the icon that is used to represent a DAT-dictionary; a visualisation of a data table (the icon of a DAT) with a direction sign in front (cf. Figure 6-2).

![Diagram](image)

Figure 6-2 The database schema and the database: DAT-dictionaries and DATs.
IDENTIFICATION
Unique labels help to identify DAT-dictionaries and DATs. A DAT identifier needs to be unique only within the set of DATs associated with a particular dictionary as the dictionary’s ID is an integral part of the DAT’s identifier.

6.3 The data-modelling scheme

The identification mechanism of the DMS deals with keys. Keys uniquely identify the entries of DAT-dictionaries and DATs. The key of a DAT-dictionary entry specifies the format of the keys of the corresponding entries of the associated DATs.

6.3.1 The declaration of DAT-dictionaries

Keys consist of one or more key fields. At the DAT-dictionary level, we discern two types of key fields: fixed key fields and key-field templates.

FIXED KEY FIELDS
A fixed key field contains a non-empty string of characters. A key which consists of only a fixed key field suffices to identify data attributes which depend exclusively on the entity being modelled. Consider, for example, both entries of DAT-dictionary PLANT shown in Figure 6-3. The keys of both entries consist of only a single fixed key field. The first entry represents a plant’s annual operating cost in Monetary Units (MUs). The second data attribute contains a reference to a geographical site containing the data that relates to the plant’s location.

![Figure 6-3 Data attributes with fixed key fields.](image)

KEY-FIELD TEMPLATES
The second type of key field contains a reference to a DAT-dictionary using the dictionary’s unique identifier. This type of key field is termed a key-field template. Key-field templates are used to model relations between the data abstraction of an AC-object and other DAT-dictionaries. Figure 6-4 illustrates the use of a key-field template. A plant’s unit manufacturing cost depends on the type of product being manufactured. The key of the third data attribute of DAT-dictionary PLANT therefore
PART II - CONCEPTUAL DESIGN

consists of two fields. The first key field, which is a fixed key field, indicates the nature of the property being modelled. The second key field represents the relation between the plant and the DAT-dictionary PRODUCT.

![Diagram](image)

**Figure 6-4** The use of key-field templates to model relations.

A key-field template associated with an attribute of a DAT-dictionary may contain a reference to that same dictionary. For example, Figure 6-5 shows a DAT-dictionary PRODUCT of which the first entry represents the BOM "goes-into" quantities (if any) per unit for that particular type of product.

![Diagram](image)

**Figure 6-5** ‘Self’ references at the DAT-dictionary level.

ROLE IDENTIFIERS

Role identifiers are used in cases where a key template contains multiple key-field templates which refer to the same DAT-dictionary. The identifiers depict the role of a particular key-field template within the association being modelled by the data attribute. Figure 6-6 illustrates the use of role identifiers to model the source (‘from’) and destination (‘to’) of a transportation link within the context of a transportation-cost relation.

![Diagram](image)

**Figure 6-6** Role identifiers for key-field templates.
DATA-ATTRIBUTE TYPES
The data-modelling scheme supports basic numerical data attributes, integer and real, and references to DATs. Owing to the level of abstraction, references to DATs are specified in terms of DAT-dictionaries. To facilitate the implementation of the DMS, binary attributes are represented as integer data attributes.

PSEUDO CODE
To explain the syntax of the DMS, we introduce a specification language, a pseudo code, that uses the concepts and terms of the DMS outlined in the previous section. Figure 6-7 shows a pseudo-code representation of the data model depicted in Figure 6-4. Next follows a brief explanation of the syntactical specification of the pseudo-code; the syntax diagrams may be found on the following pages. Key words and punctuation marks which are part of the syntax are depicted in bold. Remarks are denoted between (*...*). Identifiers are placed between quotation marks. All declarations are terminated by semicolons. Declarations of DAT-dictionaries have the following format:

DAT-dictionary "DAT-dictionary identifier" { attribute list }:

Attribute lists are optional and may be omitted. Data attributes can be specified as follows:

"fixed key-field identifier" key-field template identifiers : data type;

Key fields are separated by dots. Key-field templates (if any) are preceded by a <D> label indicating that the identifier is that of a previously defined DAT-dictionary. The data type of a numerical attribute is specified as either ‘real’ or ‘integer’. The data type of a referential data attribute is specified by using the identifier of the referenced DAT-dictionary, also preceded by a <D> label.

DAT-dictionary "Product": (* No attributes defined! *)

DAT-dictionary "GeoSite"
{
    "direct labour cost [mhr]" : real;
};

DAT-dictionary "Plant"
{
    "fixed operating cost[MU/year]" : real;
    "unit manufacturing cost[MU]".<D>"Product" : real;
    "location" : <D>"GeoSite";
};

Figure 6-7 Declaration of DAT-dictionaries using pseudo code.
Figures 6-8 through 6-15 show the syntax diagrams of the pseudo code. To read a syntax diagram, follow the arrows. Alternative paths are often possible; paths that begin at the left and end with an arrow on the right are valid. A path traverses boxes that hold the names of elements used to construct that portion of the syntax. The names in rectangular boxes stand for actual constructions. Terms in circular boxes —reserved words, and punctuation— are the actual terms to be used in the pseudo code.

Character strings are not allowed to contain quotation marks or end-of-line characters. Quotation marks are used to mark identifiers. End-of-line characters are not allowed because the standard single-line edit box is unsuited for displaying these type of strings.

![Diagram](image)

**Figure 6-8 Syntax diagram: character string.**

Identifiers must be placed between quotation marks to allow blanks to be part of the identifier. A GUI-based implementation, unlike a character-based implementation, provides the most convenient support for the use of blanks. The use of quotation marks lowers the readability of, for example, a piece of programming code. Character-based implementations therefore often prohibit the use of blanks and encourage the use of underscores or a mixture of upper and lower case characters.

![Diagram](image)

**Figure 6-9 Syntax diagram: identifiers.**

The syntax diagram for the declaration of a DAT-dictionary supports the declaration of a dictionary without an associated set of attributes. This option makes it possible to use DAT-dictionaries to refer to classes of items for which it is sufficient that the associated DATs only have an identity.

![Diagram](image)

**Figure 6-10 Syntax diagram: declaration of a DAT-dictionary.**
Figure 6-11 Syntax diagram: format of a DAT-dictionary's list of attributes.

Figure 6-12 shows the syntax diagram for the composition of keys at the dictionary level. The diagram shows that the first key field of any key must be a fixed key field. This fixed key field indicates the nature of the data attribute.

Figure 6-13 Syntax diagram: fixed key field.

Figure 6-14 Syntax diagram: identifier of a key-field template.

Figure 6-15 Syntax diagram: data-type specification.

### 6.3.2 The creation of DATs

A **key template**, i.e. a key containing one or more key-field templates, defines a single entry of a DAT-dictionary. A key template also defines the format of the key of a whole class of DAT-entries. **If the database is viewed as being composed of DATs, then keys containing references must refer to DATs.** As a key-field template contains a reference to a DAT-dictionary it defines a generalised type of reference. A key at the DAT level fits a template specification if the fields that correspond to the key-field
templates contain references to DATs which are associated with the dictionary that is referred to by the template fields. Such a key is termed a \textit{template key}\textsuperscript{1}. We refer to the key fields containing references to DATs as \textit{reference key fields}. Figure 6-16 depicts a possible instantiation of the key-template representing a PLANT’s unit manufacturing cost for a specific manufacturing facility ‘AMS’.

![Diagram of key templates and data types](image)

\textbf{Figure 6-16} Key templates (DAT-dictionary level) vs. template keys (DAT level).

The design of this part of the identification mechanism is based on the idea that the index domains associated with a compacted AC-graph can be composed from \textit{homogeneous} sets of DATs. The elements of a homogeneous set of DATs are all associated with the same DAT-dictionary. Alternatively, we can say that the elements of a homogeneous set of DATs are all of the same type.

Key-field templates specify the type of DAT, i.e. the DAT-dictionary, and \textit{not} the set of DATs whose members may be used to instantiate derived template keys. This level of abstraction is comparable to that of AC-graphs. AC-graphs contain indexed model parameters. The index symbols refer to sets of DATs the contents of which is

\textsuperscript{1} The C++ ANSI 3.0 standard supports the use of class templates and function templates. An instantiation of a class template is referred to as a template class. In this thesis we use the term template in a similar fashion.
undetermined during the model-formulation process. The specification of set membership is part of the creation of model instances. Once the members of the index domains are known, these can be used to compose template keys that fit the template specification defined at the dictionary level to search for the appropriate location, i.e. a DAT-entry, in the database (cf. Chapter 7).

PSEUDO CODE

Figure 6-17 shows an example of a piece of pseudo code for the creation of a DAT derived from the DAT-dictionary PLANT. Next follows a brief explanation of the syntax used in Figure 6-17; figures 6-18 through 6-27 show the syntax diagrams for creating DATs — diagrams 6-23 through 6-27 can be found in any reference manual of Pascal. Declarations of DATs have the following format (note the similarity to the declaration of variables in Pascal):

```dat
"DAT identifier" : "DAT-dictionary identifier" { attribute list };
```

The attribute list is optional and may be omitted. DAT entries can be specified as follows:

```dat
"fixed key-field identifier" referential key-field identifiers := data value ;
```

The key of a DAT attribute is denoted in basically the same way as the key part of data attributes on DAT-dictionaries except that key-field templates are replaced by referential key fields whose identifier consists of a previously defined DAT instead of the ID of a previously defined DAT-dictionary. Referential key fields and ‘value’-references to DATs are denoted in a similar way:

```
<d:D> DAT identifier
```

DAT identifiers are preceded by a `<d:D>` label. The values of numerical data attributes are denoted in the usual way.

```dat
DAT "AMSTERDAM" : "GeoSite";
DAT "p1", "p2" : "Product";
DAT "Plant-AMS" : "Plant"
{
  "fixed operating cost[MU/year]" := 1.2E06;
  "unit manufacturing cost[MU]"."<d:D>p1" := 121.25;
  "unit manufacturing cost[MU]"."<d:D>p2" := 152.30;
  "location" := <d:D>"AMSTERDAM";
};
```

Figure 6-17 Example of representing a DAT-dictionary using pseudo code.
PART II - CONCEPTUAL DESIGN

Figure 6-18a shows the syntax diagram for the creation of a DAT without specifying the values of its attributes. By default, the attributes are assigned a null value which is indicated as ‘n/a’. Figure 6-18b depicts the creation of a DAT of which some or all data attributes are assigned a non-null value.

\[
\text{DAT} \rightarrow \text{DAT} \rightarrow \text{DAT-identifier} \rightarrow \text{DAT-dictionary identifier} \rightarrow (;) \rightarrow \\
\text{DAT} \rightarrow \text{DAT} \rightarrow \text{DAT-identifier} \rightarrow \text{DAT-dictionary identifier} \rightarrow \text{Attribute specification} \rightarrow (;)
\]

Figure 6-18a-b Syntax diagrams: two ways of creating DATs.

The syntax for specifying attribute values resembles the Pascal syntax for assigning values to variables using the familiar ‘:=’ sign.

\[
\text{Attribute specification} \rightarrow \text{Key specification} \rightarrow (=) \rightarrow \text{Value specification} \rightarrow (;)
\]

Figure 6-19 Syntax diagram: specification of DAT attributes.

The specification of keys at the DAT level (cf. Figure 6-20 and Figure 6-21) follows the key declaration at the DAT-dictionary level. The compulsory first fixed key field of the key of a DAT entry matches the first fixed key field of the corresponding key template. Additional key-field templates at the DAT-dictionary level are mirrored by reference key fields at the DAT level that contain references to DATs of the appropriate type.

\[
\text{Key specification} \rightarrow \text{Fixed key field} \rightarrow (;) \rightarrow \text{Reference key field}
\]

Figure 6-20 Syntax diagram: specification of keys.

\[
\text{Reference key field} \rightarrow (<d:D>) \rightarrow \text{DAT-identifier}
\]

Figure 6-21 Syntax diagram: specification of a DAT-reference key field.
6.3.3 Representing the DAT-based data model as a relational data model

The possibility to exchange data with external database systems is generally considered as a key requirement of a modelling system and a DSS in particular (cf. Section 3.2.4). Virtually all major database systems offer a relational DBMS as an external interface. This section shows how the DAT-based data model presented in the previous section
may be mapped on the relational data model. This mapping constitutes the basis for the
design of an interface between the DAT model and the relational model.

The attributes of a DAT-dictionary can be categorised into attributes that have a fixed
key and attributes that have a key template. Within the context of a relational data
model, it is customary to represent the attributes of a DAT-dictionary that have fixed
keys by a single relation. The order of the columns of the relation is irrelevant, just as
the order of the attributes of a DAT-dictionary is irrelevant. This is because both
models use attribute identifiers as a reference to the attributes and not the relative
position of the attribute. The rows of the relation (i.e. tuples) correspond to the DATs.
So, the value of an attribute of a DAT-dictionary that has a fixed key can be found on
the intersection of the column bearing the fixed attribute’s identifier and the row that is
tagged with the DAT’s identifier as a primary key. Figure 6-28 shows the relation for
the fixed-key attributes of the DAT-dictionary PLANT as shown in Figure 6-4. It is
customary to add the term ‘relation’ to the identifier of each table. The abbreviation
FOC denotes the plant’s Fixed Operating Cost in Monetary Units (MUs) per year.

<table>
<thead>
<tr>
<th>GEO_SITE_ID</th>
<th>fixed-key attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMSTERDAM</td>
<td>PLANT_RELATION</td>
</tr>
<tr>
<td>BRUSSELS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PLANT_ID</th>
<th>FOC[MU/YEAR]</th>
<th>GEO_SITE_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLANT-AMS</td>
<td>250,000.00</td>
<td>AMSTERDAM</td>
</tr>
<tr>
<td>PLANT-BRX</td>
<td>120,000.00</td>
<td>BRUSSELS</td>
</tr>
</tbody>
</table>

**Figure 6-28** Representing attributes having fixed keys.

The attributes of a DAT-dictionary that have a key template represent a relation
between the DAT-dictionary and the data abstraction of other AC-objects. Within the
context of a relational data model, each such attribute is modelled as a separate
relation. The columns of this relation consist of:

1. An attribute for the DAT identifiers;
2. The key fields of the key template;
3. A column for the value that is to be assigned to the attribute.

The key of the relation consists of the first column containing the DAT identifier and
the fields of the key template. Figure 6-29, for example, shows a relation which
contains the unit manufacturing cost for each type of product being manufactured by
each plant.
DESIGN OF THE DATA MODEL

<table>
<thead>
<tr>
<th>PRODUCT_RELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT_ID</td>
</tr>
<tr>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
</tr>
<tr>
<td>PLANT-AMS</td>
</tr>
<tr>
<td>PLANT-AMS</td>
</tr>
<tr>
<td>PLANT-BRX</td>
</tr>
</tbody>
</table>

Figure 6-29 Representing attributes with template keys.

The mapping described so far does not contain information that relates to the composition of the DAT-dictionaries, such as which relation contains the set of attributes with fixed keys and which relations represent the attributes that have a key template.

To complete the relational description, we need an additional relation which contains a description of the contents of the DAT-dictionaries. The key of this relation consists of a single field containing the label of a dictionary that is part of the conceptual model of the database (the database schema). The remaining attributes contain the necessary data for locating the relations that represent the entries of the DAT-dictionary. The number of columns that is designated for storing the identifiers of the attribute relations must be at least as large as the largest number of attributes that is associated with one of the dictionaries in the database schema. A single column is sufficient for storing the identifier of the relation that holds the fixed-key attributes. The remaining columns contain the identifiers of the relations that hold the data of attributes having key templates (or an expression that is equivalent to ‘0’, ‘nil’, or ‘n/a’). The cardinality of this relation, i.e. number of records that this table will hold, will be limited because the number of DAT-dictionaries will be relatively small. The waste of storage capacity that may be induced by the relation, due to large differences in the number of attributes associated with the dictionaries, will be within acceptable bounds. Figure 6-30 shows what such a table might look like. The column labelled ‘FAR’ (Fixed Attribute Relation) contains the ID of the single relation containing the fixed attributes. The TAR#n denotes the column containing the identifier of the the n-th Template Attribute Relation.

<table>
<thead>
<tr>
<th>DAT_DICTIONARY_CONFIGURATION_RELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DICTIO_ID</td>
</tr>
<tr>
<td>PLANT</td>
</tr>
<tr>
<td>PRODUCT</td>
</tr>
<tr>
<td>GEO_SITE</td>
</tr>
</tbody>
</table>

Figure 6-30 Representing meta data on the contents of DAT-dictionaries.


PART II - CONCEPTUAL DESIGN

OUTLINE OF A RELATIONAL INTERFACE

A relational interface based on the mapping described above that is easy to implement, operates on a single ASCII file or database table which either contains: (1) a subset of the attributes of a DAT-dictionary that have a fixed key; or (2) a single attribute that has a key template. Here we outline the process of importing data from an ASCII file. Most spreadsheet and database systems can create ASCII files in a tabular format that is suitable for downloading the data into the database of the modelling system.

The first step is to specify the DAT-dictionary entries to which the data that is stored in the ASCII file applies.

The second step concerns the specification of the file format. In the case of multiple attributes with fixed keys, the specification of the file format amounts to the specification of the sequence of the data attributes. For example, the format of an ASCII file containing the data for the entries of the DAT-dictionary plant with fixed keys might look like:

```
DAT-ID, "GEO-SITE", "FOC[MU/YEAR]".
```

In the case of a single key template, the specification of the file format concerns the order of the key fields.

The third step concerns the specification of the file location, the name of the ASCII file, and the separator character (delimiter) that is used to indicate the transition from one column to another. Finally, the interface algorithm parses the ASCII file, creates the appropriate DATs if necessary, and stores the data in the database. The contents of the ASCII file for the fixed-key attributes of the dictionary PLANT might look like:

```
"PLANT-AMS", "AMSTERDAM", 2.500.000.00
"PLANT-BRX", "BRUSSELS", 1.200.000.00
```

6.4 Operations

Section 6.4.1 outlines basic operations that may be used to manage the conceptual model of the database (schema manipulation). Section 6.4.2 outlines the operations for manipulating the contents of the database: i.e. creation and deletion of DATs, editing the values of the entries of the DATs and defining queries for subsetting the set of DATs associated with a DAT-dictionary.

164
DESIGN OF THE DATA MODEL

To illustrate the way in which the DBMS executes the operations, we first introduce a concise notation for representing the database schema, the dictionaries that make up the database schema, their properties, and the set of DATs associated with a DAT-dictionary. The notation shows much resemblance to Borland's object-oriented extension to Pascal [Borland International, 1990]. Let $S$ denote the conceptual model of the database, the database schema. $S$ consists of a set of DAT-dictionaries together with their mutual relations: the referential attributes and the attributes with key templates. The identifiers of DAT-dictionaries are denoted with capitals. The notation,

$$D \in S,$$  \hspace{1cm} (6-1)

indicates that DAT-dictionary ‘$D$’ is part of the database schema $S$. An attribute or property ‘$p$’ of ‘$D$’ is denoted as

$$D.p.$$  \hspace{1cm} (6-2)

We refer to the number of properties associated with a DAT-dictionary $D$ as the degree of $D$. Hence, we may denote the set of properties $P(D)$ associated with a DAT-dictionary $D$ as:

$$P(D) = \{ D.p_i : \forall 1 \leq i \leq \text{degree}(D) \}.$$  \hspace{1cm} (6-3)

It is natural to view the set of DATs associated with a DAT-dictionary as a property of that particular dictionary. In the field of object-oriented databases, this property is sometimes referred to as the ‘allset’. In keeping with this terminology, we denote the set of DATs associated with a dictionary $D$ as:

$$D.allset = \{ d_i : \forall 1 \leq i \leq \text{cardinality}(D) \},$$  \hspace{1cm} (6-4)

where $\text{cardinality}(D)$ denotes the number of DATs associated with DAT-dictionary $D$. Note the similarity to the terminology that is used in relational database modelling (cf. Appendix B, Section B.1) where the cardinality of a relation $R$ refers to the number of tuples of $R$.

The key part and the $i$-th key field of an attribute $p$ of DAT-dictionary $D$ are denoted as:

$$\text{key}(D,p),$$  \hspace{1cm} (6-5)

and

$$\text{key-field}(\text{key}(D,p),i),$$  \hspace{1cm} (6-6)

respectively.
The data type of an attribute \( D.p \), i.e. integer, real, or a reference to a member (a DAT) of the \( .allset \) of a DAT-dictionary \( D' \), is denoted as:

\[
\text{type}(D.p).
\] (6-7)

To indicate the type of a referential data attribute, we use the type identifier of the DAT reference, i.e. the DAT-dictionary: e.g., \( \text{type}(D.p) = D' \). The notation,

\[
d:D,
\] (6-8)

denotes a DAT \( d \) associated with DAT-dictionary \( D \). This notation is quite similar to the declaration of a variable ‘\( d \)‘ of type ‘\( D \)‘ in Pascal. The key of an attribute \( p \) of DAT \( d:D \) is denoted as \( \text{key}((d:D).p) \). If attribute \( D.p \) has a fixed (non-template) key, then it follows that:

\[
\text{key}(D.p) = \text{key}((d:D).p).
\] (6-9)

If the key of \( D.p \) is actually a key template, then equality (6-9) does not hold. In that case, it must be true that the key of \( (d:D).p \) fits the key template of property \( D.p \). The value of an attribute \( p \) of a DAT \( d:D \) is denoted as:

\[
\text{value}((d:D).p).
\] (6-10)

The following sections use flow charts to outline the execution of the DBMS operations. The symbols of a flow chart represent groups of elementary steps or operations portraying the behaviour of the system being analysed. The arrows indicate the order of execution. The statements within the symbols identify what is occurring at each step. The process is read in top-to-bottom order. A rounded box depicts the beginning or the end of the process being modelled. Rectangles denote a processing step. A diamond represents a decision point. At each decision point, there is a branch to alternative paths. The lines leaving the decision box are labeled “yes” or “no” according to whether or not the Boolean condition is satisfied or not. The circle depicts an entry from, or exit to, another part of the flow chart.

The flow charts contain specified ‘backup’ and ‘restore’ activities which are needed to maintain the consistency of the database. These activities were added for the sake of completeness and to stress the importance of this aspect of DBMS operations. However, I have paid little attention to the efficiency of these operations so it is therefore very likely that equally effective, much more refined and efficient procedures can be designed. This area, however, is far beyond the scope of this thesis and has thus been left out of consideration.
6.4.1 Schema manipulation (data definition)

- Create DAT-dictionary $D$

  Pre-condition: TRUE.

  Post-condition: $D \in S$.

  Outline of the algorithm: Straightforward, cf. the flow chart depicted in Figure 6-31.

![Flow chart: Create DAT-dictionary.](image)

- Create DAT-dictionary property $D.p$

  Pre-condition: $D \in S$.

  Post-condition: $p \in P(D)$.

  Outline of the algorithm: Figure 6-32 shows a flow chart of the process of defining a data attribute $D.p$. The first two steps are to define both the key and the data type of the new attribute (not necessarily in this order). Next, the algorithm checks to see if the key is unique, after which the DBMS appends the new attribute to the type definition of the DAT-dictionary. The process is completed by setting the value for the new attribute (provided that is has a non-template key) for each of the DATs $d:D$ to 'n/a' (null).
Delete DAT-dictionary property \( D.p \)

**Pre-conditions:** \( D \in S, p \in P(D) \).

**Post-condition:** \( p \notin P(D) \).

*Outline of the algorithm:* Figure 6-33 shows a flow chart of the process of deleting a data attribute at the DAT-dictionary level. After deleting the corresponding entries from the DATs associated with the DAT-dictionary, the attribute is deleted from the database scheme.
_DELETE D.p

pre-conditions satisfied?

backup D, D.alset

key(D,p) is template?

∀ d:D delete (d:D).p

∀ d:D delete (d:D).p

key((d:D).p) fits key(D,p)?

delete (d:D).p

Post-conditions satisfied?

End.

Figure 6-33 Flow chart: Delete data attribute.

- Delete DAT-dictionary D

Pre-conditions: D ∈ S.

Post-condition: D ∉ S.

Outline of the algorithm: Figure 6-34 shows a flow chart of the process of deleting a DAT-dictionary D. The algorithm first checks whether DAT-dictionaries D' ∈ S with data attributes that contain references to D, type(D'.p) = D, exist. If so, these attributes are deleted. The next step is to check for DAT-dictionaries that contain data attributes with key-field templates that refer to D: key-field(key(D'.p),j) = D. These data attributes are also deleted from their respective DAT-dictionary type specification. At this point, the database contains no references from DATs d:D' to DATs d:D, through either referential data attributes or through template keys. The following steps deal with the deletion of the set of DATs associated with D and the subsequent removal of the type specification of D from the database schema.
6.4.2 Data manipulation

- Create DAT $d:D$

Pre-condition: $D \in S$.

Post-condition: $d \in D.allset$.

Outline of the algorithm: Straightforward, cf. the flow chart depicted in Figure 6-35.
Figure 6-35 Flow chart: Create DAT.

- Delete DAT \( d:D \)

  **Pre-conditions:** (1) \( D \in S \); (2) \( d \in D.allset \).

  **Post-condition:** \( d \not\in D.allset \).

  **Outline of the algorithm:** Figure 6-36 shows a flow chart of the process of deleting a DAT \( d:D \). The first step deals with deleting references from DATs \( d':D' \) to \( d:D \). These references are set to null. The next step concerns the removal of DAT entries with template keys that contain key fields with references to \( d:D \). These attributes are also deleted. Finally, DAT \( d:D \) is removed from the set of DATs associated with \( D \) and thus from the database.
Create template-key attribute \((d:D)_p\)

**Pre-conditions:** (1) \(D \in S\); (2) \(d \in D\)allset.

**Post-condition:** \(p \in P(d:D)\).

**Outline of the algorithm:** Figure 6-37 displays a flow chart of the process of creating a DAT attribute \((d:D)_p\) that fits the format of a property \(D.p\) with a key template. The first two activities specify both the template key and the value for the attribute. The algorithm then checks to see whether the template key fits the format of the key template defined at the DAT-dictionary and whether or not the newly created template key does already exist. If both conditions are satisfied the new attribute is then added to the DAT.
Figure 6-37 Flow chart: Create template-key attribute.

- Delete template-key attribute (d:D).p
  
  **Pre-conditions:** (1) D ∈ S; (2) d ∈ D.allset; (3) p ∈ P(d:D).
  
  **Post-condition:** p ∉ P(d:D).

- Update the value of (d:D).p
  
  **Pre-conditions:** (1) D ∈ S; (2) d ∈ D.allset; (3) p ∈ P(d:D).
  
  **Post-condition:** value((d:D).p) = v.

**QUERIES**

Queries are used to create sets of DATs that share certain properties. A query Q defines a mapping between two sets of DATs, say S and Q(S) where Q(S) ⊆ S. The members of S must all be derived from the same DAT-dictionary (i.e., homogeneous sets of DATs, cf. Section 6.3.2): S ⊆ D.allset. Because the (MI)LP-modelling scheme only deals with homogeneous sets of DATs, heterogeneous sets are left out of consideration.

In order to store a query specification in the system’s database it must be assigned a unique identifying label. The definition of a query Q does not involve the specification of the domain S nor the identifier of the resulting set Q(S). It suffices to specify the
PART II - CONCEPTUAL DESIGN

DAT-dictionary to which the query applies. The application of a query specification Q only requires the specification of the ‘target’ set of DATs S. As a default, the resulting subset is labelled as: Q(S).

A simple query consists of a single Boolean condition for evaluating whether or not a particular DAT is considered a member of Q(S). A Boolean condition consists of the specification of a DAT-dictionary entry, which we denote as D.p, a value specification, and a comparison operator. For example, SQ1 specifies a query for selecting plants that have a fixed annual operating cost less than 200,000.00 MUs. Query SQ2 selects all plants that are located near Brussels.

Query SQ1("Plant") { "FOC[MU/year]" <= 200000 };

Query SQ2("Plant") { "geographical site" = "Brussels" };  

The specification of a simple query containing a Boolean condition that refers to a DAT-dictionary property with a key template is more complex. There are two ways to deal with key-field templates:

1. A query may specify a template key by specifying a particular DAT associated with the DAT-dictionary that is used in the key-field template specification. For example, query SQ3 selects all manufacturing plants that manufacture product ‘P1’ for less than 2.00 MUs per unit:

Query SQ3("Plant")
{
    "unit manufacturing cost[MU/unit]".<D:"P1" < 2.00
}

2. A query may specify all DAT entries that fit the template specification by specifying the DAT-dictionary that is used in the key-field template (a qualified ‘wild card’). For example, to select all plants that have a unit manufacturing cost no greater than 2.00 MUs per unit, a query specification would amount to:

Query SQ4("Plant")
{
    "unit manufacturing cost[MU/unit]".<D:"Product" < 2.00
}

A compound query consists of multiple Boolean conditions. Logical (binary) operators, i.e. AND, OR, and XOR (eXclusive OR), link up the Boolean conditions.
Table 6-1 displays the results of applying the logical operators with different values for the Boolean arguments b1 and b2.

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b1 AND b2</th>
<th>b1 OR b2</th>
<th>b1 XOR b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Compound query CQ1, for example, comprises three conditions. CQ1 selects all plants located near the capitals of the BENELUX:

```
Query CQ1("Plant")
{
  ("geographical site" = <d:D>"Amsterdam" ) OR
  ("geographical site" = <d:D>"Brussels" ) OR
  ("geographical site" = <d:D>"Luxembourg")
};
```

The examples given above reveal that the specification of a query shows much resemblance with the declaration of a DAT-dictionary. Figures 6-38 through 6-42 display the syntax diagrams for specifying queries.

**Figure 6-38** Syntax diagram: format for specifying queries.

**Figure 6-39** Syntax diagram: query specification.
6.5 Integrity constraints

Integrity constraints define consistent states of the database. Integrity constraints can be structural or behavioural [Öszu, 1991, p.25]. Structural constraints impose requirements on data relationships that cannot be modelled directly, but which can be expressed in terms of the components of the data model without any application-specific information. Behavioural constraints capture the semantics of a specific application of the data model. This section focuses on structural integrity constraints. Section 6.6, discusses a mechanism for representing non-structural integrity constraints.

The structural constraints relate to the domain specification of key-field templates and the data type specification of DAT-dictionary entries:
1. Key fields, at both the DAT-dictionary and the DAT level, are not allowed to contain null values:

1.1 Fixed key fields must contain non-empty strings;

1.2 Key-field templates must contain references to DAT-dictionaries that are part of the database schema;

1.3 Referential key fields which are used to compose template keys at the DAT level — keys which fit a particular key template — must contain references to DATs which are members of the set of DATs associated with the DAT-dictionary that is used to specify the corresponding key-field template at the DAT-dictionary level;

2. The DAT-dictionary that is used to specify a referential data attribute must be part of the database schema. This second constraint is basically the same as rule number one, except that it deals with the data-type specification of DAT-dictionary entries instead of key-fields;

3. Finally, the third constraint deals with the correctness of the values assigned to DAT entries. The data-type specification at the DAT-dictionary level either defines the range of valid numerical values (integer or real) or the range of valid references to DATs: i.e. any DAT that is a member of the .allset associated with the DAT-dictionary. The values assigned to the DAT entries must fall within these ranges.

6.6 Applications of constraints on role identifiers

Key templates are used to model relationships between the data abstractions of multiple AC-objects. Key templates consist of one or more key-field templates with a reference to a DAT-dictionary. In Section 6.3.1, we introduced the use of role identifiers in cases where some ambiguity as to what role particular key-field templates play as part of the relation that is being modelled might exist.

Another way of looking at role identifiers is to view role identifiers as symbolic representations of arbitrary members of the set of DATs associated with the referenced DAT-dictionary. This view enables the modelling of non-structural integrity constraints (cf. Section 6.5) and extends the modelling capabilities of the data-modelling scheme to sets of referential data attributes.

Modelling semantic constraints

Role identifiers can be used to specify behavioural constraints that capture the semantics of a specific application of the DMS. For example, the quantities of products
of some type that are transported between two geographical sites may be represented by an entry of a DAT-dictionary ‘Infrastructure’,

\[
\text{"transport[unit]"} \\
\text{."f"<D>"G-Site"} \\
\text{."t"<D>"G-Site"} \\
\text{.<D>"Product" : real;}
\]

where ‘f’ denotes the source of the transport link, ‘t’ its destination, and \text{PRODUCT} the type of commodity being transported. Many (MI)LP models use different variables to represent flows between sites (inter-site) and intra-site flows. The above entry represents inter-site flows. Hence, the source and destination of the transportation link must be different sites: \( f \neq t \).

Role identifiers provide an intuitive mechanism for representing non-structural integrity constraints. Only two comparison operators are allowed: the equality operator (=) and its logical counterpart (\(!=\)). The pseudo-code representation of non-structural integrity constraints should be part of the DAT-dictionary entry declaration (cf. Figure 6-11).

\text{Attribute List} \quad \text{key declaration} \quad \text{type specification} \quad \text{behavioural integrity rule} \\

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image}
\caption{Figure 6-43 Syntax diagram: DAT-dictionary entry declaration with behavioural integrity constraints.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image}
\caption{Figure 6-44 Syntax diagram: Non-structural integrity constraints.}
\end{figure}

When the syntax of diagrams Figure 6-43 and Figure 6-44 is applied the DAT-dictionary entry declaration (6-11) becomes:

\[
\text{"transport[unit]"} \\
\text{."f"<D>"G-Site"} \\
\text{.\texttt{ Product" : real \{ f \\ t \}};}
\]
The first of the two following DAT entries is compatible with the DAT-dictionary entry specification shown in (6-12). The second DAT-entry does not fit the DAT-dictionary entry declaration:

```
"transport[unit]"
  .<d:D>"Amsterdam"
  .<d:D>"Brussels"
  .<d:D>"Product-A" : 4.50E06;

"transport[unit]"
  .<d:D>"Amsterdam"
  .<d:D>"Amsterdam"
  .<d:D>"Product-B" : 2.25E03;
```

**Modelling sets of references**

Role identifiers are also used when representing sets of referential data attributes. For example, so-called campus sites may contain multiple manufacturing facilities. Suppose that we have a database schema that contains both a DAT-dictionary CAMPUS-SITE and a dictionary PLANT. The set of references to plants located at the site can be modelled as:

```
(*SET of references to campus facilities:*)
"campus facility".<D>"Plant" : <D>"Plant" ;
```

The key template provides a uniform identification mechanism for the members of the set of plants located at the campus site. The data-type specification contains the actual reference to the data abstraction of the plants. The above entry specification, however, does not require that the same DAT is used to create an instance of the key template and the actual reference to the PLANT. For example, the template key

```
"plant".<d:D>"Plant-A" : <d:D>"Plant-B";
```

fits the above template specification. To ensure that the DAT that is used to compose the template key is the same DAT that is used as a reference value, we again use the role-identifier mechanism (although the name is not very appropriate). At the DAT-dictionary level, we allow ‘role’ identifiers for both key-field templates and the data-type specification of referential data attributes. Using role identifiers, DAT-dictionary property (6-13) becomes:

```
"plant"."r"<D>"Plant" : "r"<D>"Plant";
```

(6-14)
PART II - CONCEPTUAL DESIGN

Note that the equality operator is valid for equal role identifiers. Consequently, the next two DAT entries are valid instantiations of the DAT-dictionary attribute declaration shown in (6-14):

"plant".<d:D>"Plant-A" : <d:D>"Plant-A";

"plant".<d:D>"Plant-B" : <d:D>"Plant-B";

6.7 Data Views

Sprague and Carlson (1982, p98) observed that when making or explaining a decision, decision makers seem to rely on illustrations of concepts, such as pictures and charts. From this observation, they inferred that the capabilities of a DSS derive, amongst other things\(^1\), from its ability to provide representations to help conceptualise and communicate the problem or the decision situation.

One of the major objectives in designing the graphical (MI)LP-modelling scheme that was introduced in the previous chapter was to make it relatively easy for the decision maker to discern the structure of the underlying models and thus to clarify the trade-offs involved. The options that the system provides to create views of the data that is stored in the database are equally important. The support for data views that is offered by the prototype (cf. Chapter 8) is limited to data-entry forms for both DAT-dictionary and DAT objects and so-called ‘contents views’ for the database at the level of DAT-dictionaries and DATs. This section indicates how the support for illustrations of data, e.g. bar charts, line graphs, and pie charts, and cross-DAT data summaries may take shape in an actual implementation based upon the data model presented in this chapter.

DATA ILLUSTRATIONS

It is easy to visualise the implementation of basic two-dimensional graphs (bar charts, line graphs, pie charts, histograms, and scatter plots) in the case of a system that is supported by a relational DBMS. Each numerical attribute of a database relation may be used to define a series of data. In a bar chart or a line graph, each point on the x-axis corresponds to a single record of the relation and may be labelled with the key for that record. The values for the selected attribute define the y-coordinates of the graph. The

---

\(^1\) The capabilities of DSS from a user's point of view, according to Sprague and Carlson, derive from its ability to provide representations, operations to analyse and manipulate those representations, memory aids to assist the user in linking the representations and operations, and control aids to handle and use the entire system. This process-independent approach for identifying the necessary capabilities of a DSS is referred to as the ROMC approach.
set of records may include the complete range of records from the relation or a selected subset which may be specified either manually or by executing a query. Histograms require a partitioning of the range of values for the attribute where each group corresponds to a point on the x-axis. The number of records for each group defines the coordinates along the y-axis. Scatter plots require two numerical attributes: For each record, the values for the two attributes define a point on the plot. The user-interface may include special graph-type dependent algorithms that provide general statistics, facilities that link a point of the graph to the corresponding database record, and for scaling the ranges along the axes.

As illustrated in Figure 6-45, in the data model that is presented in this chapter, the values of corresponding data attributes of DATs associated with the same DAT-dictionary may be used to define a series of numerical data that may be illustrated in some way. However, instead of specifying an attribute of a relation, the user must specify an attribute of a DAT-dictionary. In the case of a DAT-dictionary attribute with a fixed (non-template) key, each DAT has a single corresponding entry. The DAT-dictionary attribute then defines a series of data in an unambiguous way. This, however, is not the case for a DAT-dictionary attribute with a key template. Each DAT may have a number of entries that correspond to a DAT-dictionary attribute with a key template. The definition of a series of conformable data requires that the parameters of the key template are filled in. That is, each key-field template must be replaced by a referential, i.e. fixed, key field that fits the template specification thus creating a so-called template key (cf. Section 6.3.2).

![Figure 6-45 Bar chart of an attribute D.p from a subset of DATs d.D.](image-url)
PART II - CONCEPTUAL DESIGN

Figure 6-46 depicts a sketch of a SPMD (Single Property Multiple DATs) bar-chart object for displaying the values from multiple DATs for a DAT-dictionary entry with a template key. It includes fields for the specification of the DAT-dictionary, the set of DATs (alldat, manual subset, or by query), and the data attribute labelled as ‘DAT-property’. This area of the screen contains a drop-down listbox for each key-field template from which to select the DATs that instantiate the corresponding templates. Figure 6-46 shows the unit manufacturing cost [MU/unit] of a set of ‘Plant’-s. The key of the data attribute contains a single key-field template: <D>‘Product’.

![Image of a bar chart]

Figure 6-46 Sketch of the GUI-implementation of a 2D bar-chart view.

So far, we have discussed series of numerical data, derived from multiple DATs, for a single DAT property. On many occasions a decision maker might also be interested in a visual representation of a series of attribute values from a single DAT all of which are instances of a single DAT-dictionary entry with a key template. For example, to investigate the relation between the unit manufacturing cost for a range of products all of which are manufactured by a single ‘Plant’ or to plot a time series of production quantities for a single ‘Plant’ and a specific product, but for a range of periods. A series of conformable data can then be obtained by fixing (filling in) all but one of the key-
field templates. A range of DATs may be used to instantiate the remaining key-field template (allset, manual, or by query).

DSSs for logistics planning often provide a way of displaying data on a geographical map, including, for example, the location of plants and the spatial distribution of demand. In order to be able to support these types of data views, it will probably be necessary to have the required data available in the form of pre-defined data structures. The system must then be configured in such a way that application-specific data structures actually use these pre-defined structures.

**DATA SUMMARIES**

Figure 6-47 depicts a sketch of a data summary that shows much resemblance with the usual tabular form of displaying the contents of a relational table. Each column is associated with a single DAT entry that corresponds to a DAT-dictionary attribute with either a fixed key, e.g., ‘fixed operating cost [MU/year]’, or a key-template: ‘unit manufacturing cost [MU/unit’.<D>‘Product’. In the case of the latter, a drop-down list box is used to select the DAT that instantiates the template. Each row corresponds to a single DAT object. Here again, the set of DATs may include all DATs associated with the DAT-dictionary, or a subset that may be specified either manually or by query.

The advantage of this type of illustration is that it is concise and that it enables a direct comparison between the values of several data attributes associated with different DATs.

![Figure 6-47 Sketch of the GUI-implementation of a data-summary view.](image-url)
PART II - CONCEPTUAL DESIGN

6.8 Comparison with the object-oriented data model

This section briefly discusses the similarities and the differences between the data model that is presented in this chapter and the object-oriented approach to data modelling (cf. Appendix B). Hughes (1991) mentions five abstraction concepts that form the fundamental basis of the object-oriented data model: (1) classification and instantiation; (2) identification; (3) aggregation; (4) generalisation and specialisation; (5) association. We will use these five abstraction concepts to outline the similarities and the differences between both models.

6.8.1 Differences

- **BUILDING BLOCKS.** The underlying building blocks constitute the most obvious difference between both data models. The object-oriented model deals with classes (object-type specifications) which contain the definition of the properties and operations of a whole class of similar objects. The properties may refer to arbitrary previously defined object types or any basic data type that is supported by the object-oriented programming language or data-manipulation language that is used to implement the class operations and to write application programs which access the object database. A DAT-dictionary specification does not include operations. Furthermore, the data-modelling scheme supports only a limited number of data types: integer, real, and references to previously defined DAT-dictionaries.

Another important difference between these data models is that, in contrast with the data that is stored in DATs, the data that is stored in an object database is not globally accessible. The operations associated with the objects define the interface between the application program and the data that is stored in the objects. The only way to manipulate the data of an object that is stored in the database is through its operations. The interface of an object protects the consistency of the object. By doing so, it also facilitates maintaining the consistency of the entire database;

- **AD-HOC QUERIES.** Data manipulation in an object-oriented database system is accomplished by means of the operations defined in the class definitions and through the programming language. Despite the effort that is put into the design of the operations, it is always necessary to accommodate to ‘ad-hoc’ or ‘one-time’ queries. It is difficult to anticipate all the information requirements that apply to a specific database system and some users simply do not want to interact with the database through a procedural programming language. Many object-oriented database systems therefore provide a high-level query language interface, usually one that is based on SQL. An example of an object-oriented query language is
Object SQL (OSQL) that is part of ONTOS\textsuperscript{1}, formerly known as Vbase. OSQL employs the familiar SQL syntax

```
SELECT ... FROM ... WHERE ...
```

except that the FROM clause uses the identities of the objects and that the SELECT clause deals with object properties. Appendix B contains an example of the use of OSQL. In general, as the object model does not impose any limitations on the data structures that are used to implement the properties of an object, it is difficult to design a object-oriented query language that provides full support to the data that is stored in an object database. As yet, there is no well-defined, widely accepted query language for object-oriented database systems which fulfils the general purpose role that SQL plays in the relational environment [Hughes, 1991, p153].

The DAT-based data-modelling scheme that is presented in this chapter uses less complex data structures. The more uniform way of representing the data leads to a corresponding uniformity in the operators that are necessary to implement a general-purpose query mechanism;

- **GENERALISATION AND SPECIALISATION.** Generalisation is an abstraction in which the common properties and operations of a group of similar object types are grouped together to form a generic object type. In practice, either the generic object-type specification is derived from an existing set of rather similar class definitions or an existing generic class definition is used to create subtypes which add their type-specific properties and operations. The data-modelling scheme presented in this chapter does not support the creation of a generalisation hierarchy;

### 6.8.2 Similarities

- **CLASSIFICATION AND INSTANTIATION.** Classification concerns the grouping of objects with similar properties into classes. Instantiation deals with the generation of distinct objects of a class;

- **IDENTIFICATION.** Booch (1991, p84) defines the identity of an object as: "(...) that property of an object which distinguishes it from all other objects." Identification concerns the process whereby both abstract concepts (i.e. classes) and concrete objects (i.e. instances) are uniquely identifiable. The identity of an object should be immutable despite changes to the properties of the object. For this reason an object's identity may not be based on the values of certain of its properties. Object-oriented programming languages use unique variable names to identify classes. The

\textsuperscript{1} Ontos Inc, Three Burlington Woods, Burlington, MA 01083. The 'mini-FAQ' on the newsgroup comp.databases.object contains a comprehensive list of object databases and the addresses of their vendors.
identity of an object often consists of the object’s memory address. The problem
with using memory addresses, however, is that persistent objects which are stored
in a database continue to exist, even when the application is terminated and the
object has been removed from memory. Many object-oriented database systems
therefore assign unique labels to the objects. These are stored together with the
object’s properties and hidden from the user or application program.

DAT-dictionaries and DATs also have identities. The user may specify the
identifiers for both the DAT-dictionary and the DAT. The identifier cannot be
changed. The label of a DAT has to be unique only with respect to the set of DATs
associated with the DAT-dictionary because the system stores DATs as members of
‘allsets’ each of which is associated with a distinct DAT-dictionary;

• **AGGREGATION.** Aggregation deals with a combination of objects that are related
and which thus form a higher level aggregate object. This is particularly useful if
the aggregate object is itself to be related to other objects. Aggregation provides a
way to deal with so-called “is-part-of” relationships among objects. Aggregation
differs from classification because classification concerns the aggregation of
attributes to form an entire object. The data-modelling scheme supports the
modelling of “is-part-of” relationships through referential data attributes. The DMS
does not support the creation of composite objects that contain nested sub-objects.

### 6.9 Summary of terms and notations

**DAT.** Acronym of Data Attribute Table. DATs contain the values of a group of
logically related data attributes. Each DAT is associated with a single DAT-dictionary.

**DAT-DICTIONARY.** Specifies the key and the data type of the attributes of a whole class
of DATs.

**HOMOGENEOUS SETS OF DATS.** A set of DATs $d$ of which the elements are all
associated with the same DAT-dictionary $D$. In short: \( \{d:D\} \)

**KEY.** Uniquely identifies the entries of both the DAT-dictionaries and the DATs. The
key of a DAT-dictionary entry specifies the format of the keys of the corresponding
data attributes of the associated DATs. Keys consist of one or more key fields, the first
of which is a fixed key field.

**FIXED KEY FIELD.** Consists of a non-empty string of characters. Corresponding data
attributes at both the DAT-dictionary and the DAT level have the first key field, which
is always a fixed key field, in common.
**KEY-FIELD TEMPLATE.** Contains a reference to a DAT-dictionary. Key-field templates may be used to model relations among data abstractions. Key-field templates are used at the DAT-dictionary level; the key of corresponding entries at the DAT level instantiate key-field templates by using referential key fields.

**KEY TEMPLATE.** A key at the DAT-dictionary level containing one or more key-field templates.

**TEMPLATE KEY.** A key at the DAT level which 'fits' the format of a key template. That is, the referential key fields contain references to DATs which are associated with the DAT-dictionary that is referenced by the corresponding key-field templates.

**REFERENCE KEY FIELD.** Contains a reference to a DAT. A referential key field instantiates a key-field template referencing a DAT-dictionary $D$ using a reference to a DAT $d:D$.

**PSEUDO CODE**
Table 6-2 contains a summary of pseudo-code statements which are described in detail in sections 6.3.1, 6.3.2, and 6.6 (role identifiers).

---

DAT-dictionary declarations (attribute lists are optional and may be omitted):

```
DAT-dictionary "DAT-dictionary identifier" { attribute list };
```

DAT-dictionary attributes:

```
"fixed key-field identifier" key-field template identifiers : data type;
```

Key-field templates, references to DAT-dictionaries (role identifiers are optional and may be omitted):

```
"role identifier"<D>"DAT-dictionary identifier"
```

DAT declarations:

```
DAT "DAT identifier" : "DAT-dictionary identifier" { attribute list };
```

DAT attributes:

```
"fixed key-field identifier" referential key-field identifiers := data value ;
```

References to DATs, reference key fields (role identifiers are optional and may be omitted):

```
"role identifier"<d:D>"DAT identifier"
```

---

Table 6-2 DMS pseudo code.
CONCISE NOTATION

Table 6-3 contains a summary of the concise notation for representing the elements of the data-modelling scheme which is introduced in Section 6.4.

<table>
<thead>
<tr>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database schema</td>
<td>$S$</td>
</tr>
<tr>
<td>DAT-dictionary ‘$D$’</td>
<td>$D (D \in S)$</td>
</tr>
<tr>
<td>All DATs associated with DAT-dictionary ‘$D$’</td>
<td>$D.allset$</td>
</tr>
<tr>
<td>Property (attribute) ‘$p$’ of DAT-dictionary ‘$D$’</td>
<td>$D.p$</td>
</tr>
<tr>
<td>Key part of $D.p$</td>
<td>$key(D.p)$</td>
</tr>
<tr>
<td>$i$-th Key-field of $key(D.p)$</td>
<td>$key-field(key(D.p), i)$</td>
</tr>
<tr>
<td>Data type of $D.p$ (i.e., integer, real, or DAT-reference)</td>
<td>$type(D.p)$</td>
</tr>
<tr>
<td>DAT ‘$d$’ associated with DAT-dictionary ‘$D$’</td>
<td>$d:D, d \in D.allset$</td>
</tr>
<tr>
<td>Property ‘$p$’ of DAT $d:D$</td>
<td>$(d:D).p$</td>
</tr>
</tbody>
</table>

*Table 6-3 Concise notation for representing the elements of the DMS.*
7. DESIGN OF THE MODEL - DATA LINK

OUTLINE
Chapter 5 introduced the compacted AC-graph modelling scheme for representing (MI)LP models. Chapter 6 introduced the DAT-based data modelling scheme. This chapter concerns the composition of a database path which provides a functional connection between both modelling schemes. The components of a compacted AC-graph, which is part of an AC-object, all have part of their database of their database connection in common. The added value of an AC-object is not only that it integrates an AC-graph representation and a data abstraction of a real-world entity, but that it also enforces this commonality by fixing the shared part of the database connection of the components of its AC-graph.
7.1 Introduction

A C-objects integrate a compacted AC-graph representation of logically related (MI)LP-model activities and an accompanying data abstraction in the form of a DAT-dictionary (DAT is an acronym of Data Attribute Table).

Chapter 5 described the compacted AC-graph modelling scheme for representing (MI)LP models. A compacted AC-graph specifies the structural properties of an (MI)LP model by using the concept of an index domain and an index symbol. An index symbol is an unambiguous reference to an index domain. An index domain constitutes an abstraction of a set of similar objects pertinent to a model. The contents of the set is irrelevant at the level of abstraction that is used to depict the structure of an (MI)LP model. However, to create a model instance, the user must specify the actual contents of the index domains, which are then referred to as index sets. The index symbols are now referred to as indices as they represent arbitrary members of the index set.

\[
\text{index domain} \\
\text{Plant} \rightarrow p \quad \text{index symbol} \\
p \in \text{Plant} = \{\text{Plant-AMS:1}, \text{Plant-AMS:2}, \text{Plant-BRX}\} \\
\text{index set} \\
\]

**Figure 7-1** Index symbols and index domains vs. indices and index sets.

Chapter 6 introduced the DAT-based data modelling scheme. Within the context of the DAT-based data model, data abstractions take the shape of DAT-dictionaries. A DAT-dictionary defines both the format of the key part and the data type of the properties that are shared by a whole class of related DATs. The DATs contain the actual data.

**Figure 7-2** DAT-dictionary ‘Plant’ and its associated set of DATs.
To integrate both modelling schemes, an index set associated with a model instance is viewed as a homogeneous set of DATs. A homogeneous set of DATs consists of DATs of the same type: i.e. all elements are associated with the same DAT-dictionary (cf. Section 6.3.1). Using the notation introduced in Section 6.4, a homogeneous set of DATs may be denoted as: \{d:D\}

Each DAT has a unique identifying label, which is the DAT's identity. This identity is used to denote the contents of an index set. A DAT-dictionary may be used to indicate the type of the intended members of an index set without specifying its contents. A DAT-dictionary may therefore be used to define an index domain.

For example, Figure 7-3 shows the DAT-dictionary ‘Plant’ and the current contents of its associated set of DATs. Two plants, ‘Plant-AMS:1’ and ‘Plant-AMS:2’, are located near Amsterdam; ‘Plant-BRX’ is located near Brussels. Figure 7-3 also shows two index symbols that share the same index domain: ‘Plant’→f-ams referring to plants near Amsterdam and ‘Plant’→f-brx referring to plants near Brussels. Different index symbols which share the same domain are either used to distinguish between the different roles the index symbols play in the model (different semantics) or to refer to different index sets: i.e. different subsets of the set of DATs associated with a DAT-dictionary. In Figure 7-3, we have at the model instance level: f-ams ∈ {‘Plant-AMS:1’, ‘Plant-AMS:2’} and f-brx ∈ {‘Plant-BRX’}.

![Diagram](image)

**Figure 7-3 An example of the specification of index sets.**

Viewing index sets as homogeneous sets of DATs implies that an index symbol may be conceived as a type specification of a database ‘pointer’ (a reference) to a DAT. The domain of the index symbol specifies the DAT-dictionary D, i.e. the type, whereas the index symbol itself represents an abstraction of a subset of DATs d:D with as yet an undefined contents. An index associated with a model instance may thus be viewed as a database-pointer with a specified range; the set of DATs \{d:D\} assigned to the index.
PART II - CONCEPTUAL DESIGN

After the user has specified the contents of the index sets for each of the index symbols that is used in the (MI)LP-model, the Model Processor composes the instances of the indexed model parameters from the contents of the database by replacing the index symbols with the identifiers of the permitted DATs. Figure 7.4 illustrates this process for a small part of the compacted AC-graph of the reference model as depicted in Figure 5.11 (the ‘#’-sign denotes a remark).

```
# c : Fixed Operating Cost
# y : Open Plant (binary)

Maximise:
- c(‘Amsterdam’, ‘Plant-AMS:1’) y(‘Amsterdam’, ‘Plant-AMS:1’)
- c(‘Amsterdam’, ‘Plant-AMS:2’) y(‘Amsterdam’, ‘Plant-AMS:2’)
# - e(‘Brussels’, ‘Plant-AMS:1’) y(‘Amsterdam’, ‘Plant-AMS:1’)
# - e(‘Brussels’, ‘Plant-AMS:2’) y(‘Amsterdam’, ‘Plant-AMS:2’)

Subject to:
+ y(‘Amsterdam’, ‘Plant-AMS:1’) binary;
+ y(‘Amsterdam’, ‘Plant-AMS:2’) binary;
# + y(‘Amsterdam’, ‘Plant-BRX’) binary;
# + y(‘Brussels’, ‘Plant-AMS:1’) binary;
# + y(‘Brussels’, ‘Plant-AMS:2’) binary;
+ y(‘Brussels’, ‘Plant-BRX’) binary;
```

Figure 7.4 Illustration of the process of constructing an (MI)LP-model instance.

The Model Processor, as illustrated in Figure 7.4, generates all possible instances of the indexed model parameters. However, from the contents of the database we know that campus site ‘Amsterdam’ contains only two plants: ‘Plant-AMS:1’ and ‘Plant-AMS:2’. Similarly, the contents of the database reveals that plant ‘Plant-BRX’ is located near Brussels, not near Amsterdam, so the entries that have been struck out should not appear in the representation of the model instance. Hence it follows that in order to generate the correct set of parameter instances, the contents of the database is of vital importance.

192
The same set of DATs that is used to compose an instance of an indexed model parameter suffices to locate the DAT entry that contains the numerical value of the parameter (see also Section 6.1).

** STRUCTURE OF THIS CHAPTER  
**Section 7.2 discusses how to specify the database connection between a component of a compacted AC-graph, representing an indexed model parameter, and an entry of a DAT-dictionary and how the Model Processor uses the contents of the index sets to establish a database connection. Section 7.3 deals with the creation of AC-objects, where part of the database connections associated with the model parameters coincide. Section 7.4 outlines the support for multi-model experiments.

### 7.2 The composition of a database path

Referential data attributes make it possible to navigate through the conceptual model of the database, from one data abstraction to another, and to navigate through the data itself by moving from one DAT to another.

In general, the database schema will not be structured as a hierarchy of DAT-dictionaries. Usually, the database schema will take the form of one or more mutually disconnected networks rather than that of a tree. An immediate consequence of this layout is that there is no highest level or ‘root’ DAT-dictionary from which all other dictionaries can be accessed. Each DAT-dictionary, and hence each of its associated DATs, may therefore serve as a starting point of a database path and thus as an entry point to the database.

At the DAT level, a database path either consists of a single DAT (the degenerate case) or it forms a connection between two distinct DATs that are linked by a sequence of references. A database path can be defined as a sequence of links that consist of a DAT and a data attribute or property associated with the DAT. Except for the last data attribute, which must be numerical, all attributes must contain references to DATs. Using the notation introduced in Section 6.4 (see also Section 6.9), DATs associated with a DAT-dictionary ‘D’ are denoted as: \(d:D\). A data attribute ‘\(p\)’ associated with a DAT \(d:D\) is denoted as: \((d:D).p\). The links of a database path may then be denoted as:

\[
(d:D, (d:D).p), \quad (d':D', (d':D').p'), \quad (d'':D'', (d'':D'').p''),
\]

*Figure 7-5 Illustration of a database path at the DAT level.*

193
Clearly, two successive links must satisfy the condition that the \((l-1)\)-th link refers to the DAT that is used in the \(l\)-th link. Using the notation introduced in Section 6.4, this condition may be denoted as:

\[
(d:D , (d:D),p)_l \to (d':D' = value((d:D),p) , (d':D'),p'L)_{l-1}
\]

The Model Processor uses a database path to locate the DAT entry that contains the numerical value associated with a parameter that is part of an (M)LP-model instance. The components of a compacted AC-graph, however, represent aggregate indexed parameters, not the model parameters that appear in the final representation of model instances. The database connection for these indexed parameters is, therefore, defined at the DAT-dictionary level (cf. Section 6.1). We term a database path at the DAT-dictionary level a database-path template.

**DATABASE-PATH TEMPLATE.** A database-path template defines a whole class of database paths at the DAT level, using the structural dependencies between DAT-dictionaries that exist in the conceptual model of the database. Database-path templates define the database connection of indexed model parameters. Each path that fits the structure of a database-path template may be used to locate the numerical value of an instance of the parameter. The links of a database-path template can be denoted as:

\[
(D , D,p).
\]

In accordance with condition (7-1), two successive links must satisfy the condition that the DAT-dictionary that is used to specify the type of the referential property \(D,p\) of link \(l-1\) is the same dictionary that is used in link \(l\):

\[
(D , D.p)_{l-1} \to (D' = \text{type}(D,p), D',p')_l.
\]

A database path fits a database-path template if it satisfies condition (7-1) and the two following conditions for each pair of corresponding links (cf. Figure 7-6):

1. The DAT \(d:D\) that forms the link of the database path must be associated (7-4) with the DAT-dictionary \(D\) that forms the link of the database-path template;

2. The same property \(D,p\) that is used to extent the database-path template must (7-5) be used to extend the database path: \((d:D),p\).

![Figure 7-6 Matching links.](image)
To make the concepts so far introduced more concrete, we show what the database-path template and an associated database path might look like for the objective-function coefficient ‘fixed operating cost’ depicted in Figure 7-7 and the database schema shown in Figure 7-8.

Figure 7-7 shows part of the compacted AC-graph of the reference model. The graph represents the decision to establish a plant at a specific location and the cost associated with doing so.

![Diagram of fixed operating cost](image)

**Figure 7-7 Compacted AC-graph (derived from Figure 5.11).**

Figure 7-8 shows a database schema whose structure is based on the same distributed production system that served as a model for the compacted AC-graph depicted in Figure 7-7. The database schema differs from the database schema that was introduced in the previous chapter because the DAT-dictionary ‘Site’ now contains a single reference to a ‘Plant’ instead of a set of references using key templates, as would be the case for a so-called campus site.

```dat
Dat-dictionary "Plant"
{
  "fixed operating cost (M€/year)" : real;
  (" other properties ")
};
```

![Diagram of database schema](image)

**Figure 7-8 A database schema without key templates.**
PART II - CONCEPTUAL DESIGN

By using the format of (7-2) for denoting the links of a database-path template, the database connection for the objective-function coefficient 'fixed operating cost' can be denoted as:

\[
\langle \text{D}"\text{Site","plant":}\langle \text{D}"\text{Plant"),}\langle \text{D}"\text{Plant","fixed operating cost [MU/year]"}\rangle : \text{real} \rangle.
\]

A database path that fits this template specification for an instance of the objective-function coefficient, e.g., the fixed operating cost of a plant 'Plant-BRX' located near Brussels, may be denoted as:

\[
\langle \text{d:D}"\text{Brussels",} \text{"plant":}\langle \text{d:D}"\text{Plant-BRX"),}\langle \text{d:D}"\text{Plant-BRX",} \text{"fixed operating cost [MU/year]"} \rangle := 1.2E6 \rangle.
\]

Although the following sequence of links at the DAT level satisfies conditions (7-4) and (7-5), it does not satisfy condition (7-1):

\[
\langle \text{d:D}"\text{Brussels",} \text{"plant":}\langle \text{d:D}"\text{Plant-BRX"),}\langle \text{d:D}"\text{Plant-AMS",} \text{"fixed operating cost [MU/year]"} \rangle := 2.5E6 \rangle.
\]

As illustrated in Figure 7-8, once we have selected the DAT that serves as the starting point of a database path, we can move along the path simply by following the references stored in the database. With the exception of data attributes that have key templates, as will be explained shortly, there are no points (links) at which there are multiple possibilities to extend the path while maintaining the compatibility with the structure of the database-path template. So, given a database-path template that does not consist of data attributes that have a key template, compatible database paths differ in the DAT that defines the first link. Therefore, this initial DAT must be uniquely related to the instance of the indexed model parameter of which the corresponding DAT entry is to be located. To create the instances of an indexed model parameter, the Model Processor replaces the index symbols by references to DATs. Each tuple of DATs depicts a particular instance of the indexed model parameter. Because a database path that fits the format of a database-path template without any key templates is completely specified by the DAT that comprises the first link, while the path must also be unique for that particular model instance, it follows that the the tuple of DATs for the parameter only contains a single DAT and, hence, that the set of index symbols for the indexed parameter comprises only a single index symbol. Therefore, we may use this index symbol to specify the permissible range of DATs for the first link and thus to specify the first link of the database-path template for that parameter.
The index symbol that is used to specify the first link of a database-path template is selected from the set of index symbols associated with the model parameter. As the set of index symbols may contain several index symbols that share the same domain, i.e. the same DAT-dictionary, the format of (7-2) for denoting the links of a database-path template is inappropriate. To avoid any ambiguity, we use the index symbol to specify the first link of the database-path template instead of the identifier for the DAT-dictionary. Using the referential notation introduced in Chapter 5, the first link of a database-path template can be denoted as:

$$ (D ightarrow i, D, p), \quad (7-6) $$

where $i$ denotes an index symbol and $D$ the DAT-dictionary that serves as the domain for the index.

A database-path template, which is defined as a sequence of links of which the first has the format of (7-6) and successive links have the format of (7-2), does not necessarily define an unambiguous database path at the DAT level. If a link contains a data attribute $D, p$ with a key template, then an associated DAT $d:D$ is likely to have a set of corresponding data attributes. In that case, the specification of the DAT is not sufficient to determine which one of the entries is to be selected.

![Diagram](image)

$$(d:D),(d:D),p)\ , \ (d':D),(d':D'),p')$$

**Figure 7-9 Key templates.**

DAT entries that correspond to a DAT-dictionary property that has a key template can be identified by verifying whether or not their key fits the format of the key template. In Section 6.2.2 such keys were termed template keys. To obtain a template key, the key-field templates must be replaced by reference key fields. A reference key field contains a reference to a DAT. This DAT must belong to the DAT-dictionary that is used to specify the key-field template.
To turn a database path template into an unambiguous recipe for creating database paths at the DAT level, for each link we must specify the tuple of DATs that must be used to compose the key that fits the format of the key template. This tuple must be derived from the tuple of DATs that serves as a unique identification of the parameter. Each DAT is taken from a set of DATs which instantiates an index domain at the structural level. The index domain is identified with a DAT-dictionary and an index symbol so we must assign an index symbol to each key-field template of a DAT-dictionary property that forms a link of a database-path template to guide the composition of database paths at the DAT-level. Then it follows that the format of (7-2) and (7-6) for denoting the links of a database-path template is only sufficient for paths which do not contain data attributes with key templates. To be able to cope with key templates, the format of the links of a database path as shown in (7-2) changes into:

\[(D, D.p, \text{key-field}(key(D.p), 2) \rightarrow i(1), \ldots, \text{key-field}(key(D.p), K) \rightarrow i(K-1)),\]  
(7-7)

where \(\text{key-field}(key(D.p), j)\) denotes the \(j\)-th key field (cf. Section 6.3), \(i(j)\) the \(j\)-th index symbol, and \(K\) the total number of key-field templates associated with the key of property \(D.p\). We start counting key-field templates at 2 because the first field of a key must be a fixed key field (cf. Section 6.2.1). Similarly, when the key of the data attribute that forms the first link is a key template, (7-7) changes into:

\[(D \rightarrow i(1), D.p, \text{key-field}(key(D.p), 2) \rightarrow i(2), \ldots, \text{key-field}(key(D.p), K) \rightarrow i(K)).\]  
(7-8)

At the level of a compacted AC-graph representation, no ordering among the index symbols associated with an indexed model parameter exists. The ordering among the index symbols emerges from the process of selecting index symbols to compose the sequence of links that makes up the database-path template.

Figure 7-10 shows part of the compacted AC-graph of the reference model which represents the decision to manufacture a number of products of type \(p\) at a plant \(f\) located near a site \(s\). Figure 7-11 (next page) displays the conceptual model of a database that supports the graph depicted in Figure 7-10. From the database schema

![Diagram](image)

**Figure 7-10** Compacted AC-graph (derived from Figure 5.11).
shown in Figure 7-11, it appears that there are two ways to compose a database path that ends with the ‘unit manufacturing cost’ property of DAT-dictionary ‘Plant’. The first option is to start the database path directly at the DAT-dictionary ‘Plant’. By using the format of (7-2) to denote a database-path template, this first path results in:

\[
\langle D \rangle "Plant", "unit manufacturing cost [MU/unit]", \langle D \rangle "Product" : real). 
\]

The second option is to use the DAT-dictionary ‘Campus Site’ as the starting point for the database path, then the reference to DAT-dictionary ‘Plant’ and finally connecting to the ‘unit manufacturing cost’ property of DAT-dictionary ‘Plant’:

\[
\langle D \rangle "Campus Site", "plant"."p" \langle D \rangle "Plant", "p" \langle D \rangle "Plant"), 
\langle D \rangle "Plant", "unit manufacturing cost [MU/unit]", \langle D \rangle "Product" : real). 
\]

The set of index symbols associated with activity ‘manufacture’ depicted in Figure 7-10 contains three elements: ‘Campus Site’→s, ‘Plant’→f, and ‘Product’→p. Apparently, all three index symbols are considered necessary to obtain a proper representation of the underlying system. Therefore, we require that all the index symbols associated with a model parameter are used to compose the database-path template, so we will focus on the composition of the second database-path template.

![Diagram showing database schema with key templates](image)
Figure 7-12 displays the steps necessary for the completion of the second database-path template. The first step concerns the specification of the DAT-dictionary that will serve as an entry point to the database and, hence, as a starting point of the database-path template. Index symbol ‘Campus Site’ $\rightarrow$ is used to specify this first link. The second step deals with the selection of the property that contains the reference to the DAT-dictionary ‘Plant’. As shown in Figure 7-11, the data attribute which contains the reference to DAT-dictionary ‘Plant’ has a key template because the ‘Campus Site’ may contain several manufacturing facilities. Index symbol ‘Plant’ $\rightarrow$ is used to guide the composition of keys that fit the format of the key template. The index symbols are placed between brackets to stress that the index symbols refer to the respective key-field templates in that particular order. The third stage, deals with the completion of the second and final link of the database-path template. It concerns the specification of the data attribute that contains the required numerical data. The key of this data attribute must be a key template because there is still an index symbol left to use: ‘Product’ $\rightarrow$.

After mounting this third index symbol, the database-path template is complete.

![Diagram](image)

('Campus Site' $\rightarrow$)  \[ \text{Campus Site} \]

('Campus Site' $\rightarrow$)  \[ \text{Campus Site} \]

('Plant' $\rightarrow$)  \[ \text{Plant} \]

('Campus Site' $\rightarrow$)  \[ \text{Campus Site} \]

('Plant' $\rightarrow$)  \[ \text{Plant} \]

('Product' $\rightarrow$)  \[ \text{Product} \]

\[ 'plant'. 'p' <D> 'Plant': 'p' <D> 'Plant' \]

\[ 'plant'. 'p' <D> 'Plant': 'p' <D> 'Plant' 'unit mfg cost'. <D> 'Product': real \]

**Figure 7-12 Composition of a database-path template.**

**INDEX SYMBOLS EXCLUSIVELY FOR DATABASE NAVIGATION**

Occasionally, there is only a partial match between the relation or association that is denoted by the set of index symbols on the parameter and the structure of the database. In the database, relationships between DAT-dictionaries are modelled either as a series of references, by means of a DAT-dictionary property with a key template, or as a combination of both. A database-path template that connects to a relation must have index symbols available for the DAT-dictionary that ‘contains’ the relation and for each of the DAT-dictionaries that participates in the relation by means of key-field templates as well. If the DAT-dictionary that ‘contains’ the relationship serves as the
domain of an index symbol on the parameter, that part of the structure of both the (MI)LP model and the database are parallel. Sometimes though, the logical structure of the data requires the introduction of an additional DAT-dictionary to contain the relation. For example, in the case of the reference model, the cost of transportation between two sites may be modelled as part of the DAT-dictionary ‘Site’, where the site containing the data attribute serves as the origin of the transport link. However, allocating the cost of transportation to a DAT-dictionary ‘Infrastructure’ or ‘Transport Link’ more closely resembles the logical structure of the data.

In order to be able to uncouple the structure of the (MI)LP-models from the structure of the database to a certain extent, the model builder has the option to introduce index symbols solely for the purpose of database navigation. As these index symbols do not interfere with the structure of the model, they are not part of the graphical representation of the set of index symbols on a model parameter.

7.3 Activity / Constraint objects

An AC-object represents a class of real-world entities whose structural behaviour can be modelled by a single (MI)LP-model structure. As a consequence, with the exception of shared constraint nodes, the components of the compacted AC-graph either relate exclusively to that particular real-world entity or they represent a relationship between the entity and one or more other real-world objects. The top part of Figure 7-13, for example, shows a compacted AC-subgraph that represents the activities of the reference model which take place at a ‘Campus Site’ (The index symbols associated with both the constraint coefficients and the objective-function coefficients have been omitted). None of the activities, coefficients, and constraints relates exclusively to the site. The parameters represent associations between the site and a product, and associations between the site, a plant, and a product. The DAT-dictionary associated with an AC-object, therefore, either contains a data attribute that represents the model parameter or it contains a reference to a DAT-dictionary that is part of the association. This DAT-dictionary, in its turn, again either contains the attribute that corresponds to the model parameter or a reference to a DAT-dictionary that is part of the association. In both cases, however, the first link of the database-path template deals with the same DAT-dictionary.

An AC-object provides a physical connection between a compacted AC-graph and a DAT-dictionary by fixing the first part of the database-path template, i.e. the DAT-dictionary that will serve as Entry Point (EP) to the database, associated with each of the components of the AC-graph. To identify the EP dictionary, the AC-graph is
PART II - CONCEPTUAL DESIGN

associated with an index symbol (and thus an index domain) which automatically becomes part of the set of index symbols associated with the components of the model. For reasons of brevity, this index symbol is omitted from the set of index symbols on the components of the graph.

Figure 7.13 AC-object 'Campus Site'.

7.3.1 Nested AC-objects

The compacted AC-graph formalism supports the creation of a hierarchy of subgraphs. Each of these graphs, at any level in the hierarchy, may be associated with a data
abstraction, thus forming a hierarchy of AC-objects. In the previous section, we saw that the physical connection between a compacted AC-graph and a DAT-dictionary consists of a shared starting point for the database-path templates associated with the components of the graph. The inclusion of an AC-object within an AC-object therefore leads to an elaboration of the common part of the database-path template.

Figure 7-14 shows a two-level hierarchy of AC-objects. Some of the activities of the AC-object ‘Campus Site’ shown in Figure 7-13 are clustered and allocated to the AC-sub object ‘Plant’. The database connection has been extended to DAT-dictionary

\[\text{DAT-dictionary } '\text{Plant}'\]

\[
\{
\text{unit manufacturing cost [MU/unit]}: <D>'\text{Product'} : \text{real};
\text{fixed operating cost [MU/year]} : \text{real};
\text{max capacity [LHR/year]} : \text{real};
\text{min capacity [LHR/year]} : \text{real};
\text{capacity occupation[LHR/unit]} : <D>'\text{Product'} : \text{real};
\text{manufactured qty [unit/year]} : <D>'\text{Product'} : \text{real};
(* other properties *)
\};
\]

\text{Figure 7-14 A two-level hierarchy of AC-objects.}
PART II - CONCEPTUAL DESIGN

‘Plant’ using the set of references to plants that is part of the ‘Campus Site’ DAT-dictionary.

7.4 Support for multi-model experiments

MANAGING THE DATA
An experiment with the aid of either an (MI)LPS or modelling system is typically concerned with running one or several models for a selected range of data with specific settings for the exogeneous parameters. The range of data objects that serve as input to a model define a state space of the system under design. In the field of logistics planning, the collection of data objects that serves as input to a model is often referred to as a ‘configuration’, whereas the settings for the exogeneous parameters are collectively termed a ‘scenario’. For example, in the case of a production location-allocation model, a configuration might comprise a network of geographical locations (regions) mutually connected by transport links, sets of possibly hypothetical suppliers, customers, and manufacturing facilities for each of the sites, the means of transportation, and the product types. Decision makers often consider different scenarios, e.g., an optimistic, normal, and pessimistic scenario, for both the spatial distribution, the quantities, and the price levels of the demand for end items, the availability of equipment, etc. The goal is to find a robust distribution of facilities and an allocation of products to these facilities that is expected to perform well under each of these scenarios. In a multi-model experiment, the models may be used sequentially, as in hierarchical planning systems (cf. Chapter 2), where the output of a higher-level model serves as input to a lower-level model. Alternatively, it is possible to use different models side by side, in a complementary fashion, where each model views the system under design from a different angle.

The application of a model in order to optimise the modelled system for a specific scenario requires its own set of data, separated from the data that are used by other experiments. As illustrated by Figure 7-15, one approach to the management of these different sets of data, would be to make the relation between the value of a parameter and the experiment to which it applies part of the conceptual model of the database. The biggest advantage of this approach is that it will not be necessary to store multiple sets of data for each of the experiments in the database. This approach results in less redundancy. Disadvantages of this approach are that the structure of the database becomes more complicated and that the key part of the data attributes increases. This has an important negative impact on the performance of the Model Processor when generating model instances.
A less sophisticated, less complicated approach is to organise the database by means of a tree-structured organisation of the data objects. As illustrated in Figure 7-16, each level in the hierarchy contains a copy of a set of data objects, each of which represents the same set of real-world entities. At the ‘configuration’ level, the contents of the sets is specified along with the values for the data attributes which are invariant to different scenarios. The ‘scenario’ level contains copies of the sets of data at the configuration level. In addition, the values for the data attributes that represent the exogeneous parameters are specified. The ‘experiment’ level, in turn, contains copies of the set of data objects at the ‘scenario’ level, including the values for the exogeneous parameters. The sets of data objects at the ‘experiment’ level are actually used to instantiate models and to store the values of the variables.

**Figure 7-16** A hierarchy of sets of DATs.

**MANAGING THE MODELS**

Much of the strength of existing DSSs based on multiple (M)LP-models stems from the design of the user interface. It is customary to base the design of the user interface upon a model of the decision-making process and to tailor the user interface to the data-interdependencies that exist between the underlying models as implied by the order in which the models may be applied.

205
PART II - CONCEPTUAL DESIGN

For obvious reasons, the generality of both the (MI)LP-modelling facility and the data-modelling facility that we propose makes the design of experiments that involve multiple models part of the configuration stage of the system. According to Sprague et al. (1982), a DSS should provide so-called 'memory aids' that support the user, in this case, to conduct experiments and to preserve the links to both the input data and the results as they are produced by running the models\(^1\).

One way of implementing a way to group logically related (MI)LP-model instances would be to incorporate the concept of a workspace in the design of the user interface. A workspace is usually designed by means of a so-called 'parent' window surrounding several 'child'-windows each of which contains the representation of a model instance. Another approach is to use a 'notebook' [IBM Corporation, 1992] control. A notebook contains pages separated into sections by tabbed divider pages. Notebooks are probably more suited to illustrate a relative order of application among the models. Figure 7-17 on the next page displays a possible design for a notebook control for multi-model experiments. The relative order of the models is reflected in the sequence of sections as indicated by the tabs. The notebook shown in Figure 7-17 contains both an 'activity view' section and a 'data view' section:

- The activity view on the front page shows a 'tree-view' of the hierarchy of activities and submodels of the modular AC-graph of the Production Location Allocation Model as presented in Section 5.3.7, Figures 5-18a-c. Clicking on the '+' will expand that part of the tree. Conversely, when the '-' is clicked the tree will be displayed by the icon for the subgraph. For each activity, the linked DAT-dictionary entry is also listed.

- The data view supports the composition of a data set that instantiates the compacted AC-graph. The user has the option to automatically associate the same set of DATs with index symbols with identical names from different models. The data view also supports automated queries for dependent sets. For example, the tactical Aggregate Production Planning (APP) model (cf. Chapter 2) only uses those facilities that are appointed as 'open' facilities by the production location allocation model. The drop-down list boxes make it possible to select a value for the index and to view the value of the indexed parameters with these current settings.

\(^1\) In practice, it can be observed that this statement also applies to systems that are not considered as decision support systems. For example, spreadsheets and compilers which, almost without exception, offer the possibility to create workspaces which can be managed as independent modules and which can be saved for later use.
Figure 7-17 Sketch of a notebook control for multi-model experiments.
PART THREE

Implementation & Validation
8. DESIGN & IMPLEMENTATION OF THE PROTOTYPE

OUTLINE

The object-oriented design and implementation of ACOM is presented. First, the tools that have been used to implement the prototype are discussed. The remainder of this chapter is structured conform the discussion of the functional architecture of (MI)LP-based decision support systems and (MI)LP-modelling systems presented in Chapter 3.
8.1 Introduction

In the course of this project, we have implemented a prototype which provides the functionality of an (MI)LP-modelling system based on AC-objects. The purpose of implementing a prototype has been threefold:

1. The implementation of a prototype provides the necessary feedback that helps ensure that the conceptual design constitutes a sound basis for the construction of an (MI)LP-modelling system based on AC-objects;

2. The prototype provides a means for experimenting and thus for validation of the claimed advantages of the (MI)LP-modelling scheme;

3. The prototype shows how the conceptual design may take shape in an actual implementation.

The functionality of the prototype covers the greater part of the conceptual design as described in the second part of this thesis. As far as the (MI)LP-modelling scheme is concerned, the support for relationships among instances of activity nodes (cf. Section 5.3.5) has not been implemented and, while the index operators and the query mechanisms have been implemented and tested as part of the prototype’s kernel, they have not yet found their way into the system’s user interface. The prototype does, however, support the use of index expressions (so-called ‘idx-functions’) for representing incidence structures. As far as the support for data views is concerned (cf. Section 6.7), the functionality of the prototype is limited to DAT-dictionary and DAT entry forms. Finally, multi-model experiments, as outlined in Section 7.4, are not supported explicitly.

It is well known that the duration of projects that focus on the implementation of a piece of software often far exceeds the anticipated period of time. With this in mind, the implementation was started as soon as the conceptual design appeared to be relatively stable. It took about two man-years to implement the prototype. The design and implementation of the prototype has been done by the author. Parts of the user interface have been implemented in cooperation with Barakat (1995) who also documented the greater part of the implementation of the user interface. Maryniak (1992) assisted with the design of the communication mechanisms between the machine running the prototype and external workstations running (MI)LP-solvers.

The prototype has been implemented by using object-oriented programming methods. According to Booch (1991,p36), an object-oriented program is organised as a cooperative collection of objects, each of which represents an instance of some class,
and whose classes are all members of a hierarchy of classes united via inheritance relationships. In the broadest sense, a class represents a generalisation of a set of objects that have certain properties in common. Within the context of an object-oriented program, a class declaration defines a collection of data structures in a way similar to the conventional ‘record’ in Pascal or the ‘struct’ in C, and a set of methods (also known as functions or procedures) through which the data may be manipulated.

We have used C++ to implement the prototype. C++ supports four kinds of relationships among classes:

1. **INHERITANCE RELATIONSHIPS.** Inheritance relationships represent so-called ‘kind-of’ or ‘is-a’ relationships. For example, a referential key field and a fixed key field (cf. Section 6.2.1) are both specialisations of the more general notion of a key field. An object-oriented implementation may use a so-called ‘base’ class ‘key field’ which defines the properties that are common to all key fields whereas both class ‘referential key field’ and ‘fixed key field’ represent specialised subclasses;

2. **CONTAINMENT RELATIONSHIPS.** Containment relationships represent so-called ‘has-a’ relationships where a composite or aggregate object consists of several ‘sub’-objects which existentially depend on the aggregate object which is said to ‘own’ the subobjects. For example, a DAT-dictionary object contains several DAT objects. If the DAT-dictionary object ceases to exist, so do the associated DATs;

3. **USES RELATIONSHIPS.** A uses relationship refers to a dependency between two objects where one object depends on another object for the execution of its tasks;

4. **INSTANTIATION RELATIONSHIPS.** Instantiation relationships concern relationships between a parameterised class which serves as a class template and a class which instantiates the class template by filling its parameters by means of other non-template classes. A parameterised class must be instantiated before any instances (objects) can be created. Parameterised classes are typically used for implementing container classes: e.g. lists, arrays.

In this chapter, we present the object-oriented decomposition of the prototype. An object-oriented decomposition breaks the system into classes and objects. “Object-oriented design” is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design” [Booch,1991,p37]. This chapter provides a largely static picture of the broad outlines of the logical design of the prototype, using the notation proposed by Booch (1994). Class Diagrams (CD) depict the class structure. Object Diagrams (OD) show the interplay between objects during a time-lapse snapshot of an otherwise transitory event. Object diagrams are prototypical
in the sense that they represent the interactions that may occur among a collection of objects, no matter what specifically named instances participate in the mechanism. The syntax of both diagrams is the subject of Section 8.3.

STRUCTURE OF THIS CHAPTER
Section 8.2 provides an overview of the programming tools that we have used to implement the prototype. Section 8.3 outlines the syntax of the Booch notation. Section 8.4 discusses two general topics that concern the design and implementation of the prototype. The remainder of this chapter is structured conform the decomposition of (MI)LP-modelling systems outlined in Chapter 3. Section 8.5 discusses the Model Management subsystem. Section 8.6 outlines the design of the DBMS. Section 8.7 discusses the design of the Model Processor subsystem. Finally, Section 8.8 outlines the design and implementation of the prototype’s graphical user-interface.

8.2 Tools for implementation

The prototype has been implemented with Borland C++ v1.0 for OS/2 [Borland, 1993]. The large number of C++ programming tools that is available made C++ into an attractive language for implementing the prototype. To facilitate and to speed up the task of implementing the prototype, we used both an Object-Oriented DataBase Management Subsystem (OODBMS) and a graphical User-Interface Management System (UIMS). The vendors of both subsystems offer versions of their products for each of the major operating environments. Provided that the programmers produce standard platform independent C++ code, porting the prototype across different platforms, i.e. different windowing systems, is relatively easy. It requires a C++ compiler that supports the same C++ implementation as the Borland compiler that we have used, and the proper versions of the database management subsystem and the user-interface management subsystem.

THE DATABASE SUBSYSTEM
POET v2.0 [POET Software, 1993] is an OODBMS. An OODBMS may be used to store the persistent part of the class structure and the objects and their mutual interdependencies as they appear, at some snapshot in time, in an object-oriented program.

POET offers an extension to the common C++ syntax that includes, amongst others, the key word ‘persistent’ that may be used as part of a class declaration. An accompanying precompiler converts the extended syntax into an equivalent C++ syntax. It appears that persistent classes are derived from a common base class called ‘PtObject’.
THE USER-INTERFACE MANAGEMENT SUBSYSTEM

StarView v2.0 [Star Division, 1993] is a C++ library that contains classes for creating a graphical user interface: e.g. 'MDIWindow', 'MouseEvent', 'Button', and 'MenuBar'. StarView is a graphical UIMS. If the application is executed, the StarView system manages the entire flow of messages between the application and the graphical windowing system which, in turn, manages the screen, the keyboard and the mouse. To enable the StarView system to take control of the execution of events in response to user input, the application must be derived either from class Application or from class MDIApplication which complies with IBM's Multiple Document Interface (MDI) standard for user interfaces.

8.3 The syntax of class diagrams and object diagrams

Class diagrams and object diagrams are part of the object-oriented design method proposed by Booch (1991, 1994). The syntax of the diagrams that we use in this thesis is described in Booch (1994). Class diagrams show the existence of classes and their relationships in the logical view of the system. Figure 8-1 shows the icons that represent a class, an abstract class, a parameterised class, an instantiation of a parameterised class, and a class utility, respectively. An abstract class has no instances. Derived classes, at some hierarchical level, must complete the implementation of the typically incomplete methods of an abstract base class in order to become concrete classes. A class utility denotes a collection of procedures or functions which are not part of a class declaration. A class utility may, for example, be used to denote a collection of operators for a class in C++ which are defined as global functions and not as class methods.

![Figure 8-1 Icons for classes, parameterised classes, and instantiated classes.](image)

Figure 8-2a shows the icons for representing inheritance, containment, uses, and instantiation relationships among classes. Inheritance relationships are depicted by a directed arc from the derived class to the base class. Containment relationships are depicted by an arc with a filled circle near the composite or aggregate class. A filled square on the opposite site of the link indicates a 'has by value' (subobject) containment relationship. An unfilled square indicates a 'has by reference' (pointer reference) relationship. An unfilled circle near class A denotes that class A uses certain methods of class B but objects of both classes exist independently.
PART III - IMPLEMENTATION & VALIDATION

Figure 8-2a Icons for class relationships.

Labels may be attached to a class relationship to indicate its role (cf. Figure 8-2b). The number of objects that is involved in a particular relationship (the cardinality of the relationship) is specified on either side of the link for the relationship. The symbol ‘n’ indicates an unlimited number of objects. By default, the cardinality of a relationship equates to ‘n’. Ranges are specified by using two dots separating the lower bound and the upper bound: e.g., ‘0..1’, indicates a range of zero or one object. The symbol \( \forall \) depicts a static class relationship between two classes. A static class relationship may be viewed as a property of the class. Whereas each object contains a copy of the non-static data attributes, there exists only one copy of a static property which is shared by all the objects of a class. The \( \forall \) depicts a friend class or class utility which has unrestricted access to the data stored in an object. Using friend classes is often considered bad programming practice as it neglects the class’ interface and may thus endanger the consistency of an object. Programmers, however, use friend classes and class utilities to speed up the implementation process. Finally, the symbol \( \forall \) depicts a virtual inheritance relationship. If a class is derived from multiple base classes (multiple inheritance) and if some or all of these base classes, in turn, have a higher-level base class in common, then an object of the derived class contains an instance of this higher-level base class for each path that connects the derived class with the base class. When a base class is specified as being virtual, it will be represented by only a single object of that class. For more information, refer to Lipmann (1993) or Stroustrup(1993).

Figure 8-2b Labels for class relationships.

214
Object diagrams show the existence of objects and their relationships in the logical view of a system. Object diagrams are used to outline the execution of a particular sequence of method calls termed a ‘scenario’. By themselves, object diagrams are static: they do not show the creation and deletion of objects nor the flow of control. Figure 8-3 shows the icon to represent an object, multiple similar objects, and an object containing multiple subobjects respectively. The objects in an object diagram must denote an instance of a class which is defined in a class diagram.

Figure 8-3 Icons for objects

Figure 8-4 shows the graphical syntax for representing relationships between objects. A relationship between two objects means that one of the two or both objects can invoke methods from the other object. Calling up a method and sending an object or value in return is often referred to as the interchange of messages between objects. The link between two icons therefore represents a path for the exchange of messages between two objects. An object S can only call upon a method of another object R if R is ‘visible’ to S. Special icons may be used to denote how an object S sees the object R. The visibility adornment is placed along the relationship between R and S, near the object R. An object R may be visible to S if it is either globally accessible, which is shown as a \( G \), if it is passed as a parameter, which is depicted by a \( P \), if the object is declared in the body of a method or in the same module (i.e. file) in which the method was implemented, which is shown as an \( L \) (local scope), or if S contains a field with a reference to R, which is indicated by an \( F \).

Visibility:
- \( G \) global
- \( P \) parameter
- \( L \) local scope
- \( F \) field

Figure 8-4 Icons for class relations.

The relative order in which the method-calls that appear in an object diagram are invoked can be indicated by attaching labels. Although this mechanism only works well for an unconditional flow of control, we think that object diagrams provide an adequate picture of the most important mechanisms that involve the interplay of objects.
8.4 General design topics

This section discusses two basic issues that relate to the implementation of run-time type checking and the implementation of container classes.

8.4.1 A base class for run-time type checking

C++ does not provide a standard way of doing run-time type inquiries — only virtual function calls [Stroustrup, 1993]. Run-time type checking provides a useful mechanism for establishing the type of an object that was dynamically created. Methods that operate on a base class but which contain portions that act differently, depending on the actual type of object (as it was originally created), then have a way of determining the type of object that was passed through.

The prototype uses a mechanism for run-time type checking that is proposed by Stroustrup (1993, cf. Section 13.5). The abstract base class ‘DSS-IDE BASE’ contains a virtual function ‘is_a()’ that returns a reference to an instance of class ‘TYPE INFO’. Concrete classes that inherit from class ‘DSS-IDE BASE’ contain a static reference to a ‘TYPE INFO’ object which contains a unique identifier for the class. The derived class also implements the ‘is_a()’ method by returning a reference to the static ‘TYPE INFO’ object (cf. Figure 8-5). The object diagram depicted in Figure 8-6 shows a call to the ‘is_a()’ method of ‘an object’ of a concrete class which is derived from class ‘DSS-IDE BASE’.

![Diagram](image_url)

Figure 8-5 CD: Run-time type checking.

![Diagram](image_url)

Figure 8-6 OD: Run-time type checking.
8.4.2 Container classes

For ease of implementation, all non-persistent container classes were modelled as specialised subclasses of the parameterised class ‘LIST’ (cf. Figure 8-7). Class ‘LIST’ offers the functionality of a double-linked list. ‘LIST’’s provide an easy way for dynamically allocating storage capacity. On the other hand, however, in some cases the use of double-linked lists will yield a significant reduction of performance.

Figure 8-7 CD: List template.

POET forces the programmer to use specialised persistent container classes. The class diagrams shown in this chapter, however, consistently use class ‘LIST’ or a specialised subclass to illustrate the use of container classes.

The type of object that is stored in a container class should not affect its implementation. Therefore, we have implemented class ‘LIST’ as a parameterised class. Following Stroustrup (1993, cf. Section 8.3.2), class ‘LIST’ does not require that its elements provide facilities to help the implementation of the class: e.g., a pointer referencing the ‘next’ object in the list. In other words, the elements of a list need not be derived from some base class ‘LINKABLE’. Instances of class ‘LINK’ actually make up the elements of a list (cf. Figure 8-8). ‘LINK’ objects contain the necessary link fields and a reference to the generic object that is stored in the list.

Figure 8-8 OD: Insert item in a list.
PART III - IMPLEMENTATION & VALIDATION

The class ‘ITERATOR’ contains a pointer to an element of the list and provides the necessary facilities for looking into a ‘LIST’ object. ‘ITERATOR’ objects may be used to traverse the list, to access the ‘current element’ (the ‘cupo’), and to point at an element of the list that serves as a point of reference for inserting and deleting elements. It is possible to associated several ‘ITERATOR’ objects with a single list simultaneously. The object diagram shown in Figure 8-9 depicts a typical sequence of method calls for getting a reference to the i-th element of the list.

Figure 8-9 OD: Get item in a list.

8.5 The model management subsystem

The early start of the implementation has had an impact on the design of the prototype. The design of the prototype had to embody enough flexibility to withstand major changes in the design of the (MI)LP-modelling scheme. For that purpose, we implemented the Modelling Scheme (MS), i.e. the compacted AC-graph formalism, on top of an Implementation Scheme (IS) which implements the referential notation. The object-oriented design of the IS is based on the decomposition of the algebraic formulation of LP models as proposed by Murphy, Stohr, and Ma (1992). Section 5.2.1 has already introduced the concepts of LHS-algebraic pieces, RHS-algebraic pieces, and the Matching Pieces Requirement on which Murphy et al. have based their model construction algorithm in support of their piecemeal approach to the composition of LP-models. Although the time needed to implement to prototype would have been reduced if the Model Processor subsystem could deal with AC-graphs directly, it would require a major effort to implement a change in the compacted AC-graph formalism. Moreover, because the algebraic-oriented IS is based on the work of Murphy et al., the database representation of (MI)LP-models can be kept relatively uncomplicated. The IS provides full support for automatic model construction so LHS and RHS-algebraic pieces may be stored separately which reduces the need for storing cross references in the database. The full support for automatic model construction may be put to use to facilitate the model formulation process by possible future extensions to the prototype. Finally, the use of a separate algebraic-oriented IS offers the advantage that the kernel of the prototype may be used as part of other (MI)LP-based applications.

218
8.5.1 Object-oriented decomposition of the Modelling Scheme

Class 'NETWORK' serves as a base class for class 'AC-GRAPH'. Class 'NETWORK' contains a set of 'NODE'-s and a set of 'NETWORK LINK'-s (the name 'link' is already part of the name space of the StarView library) and implements common network operations for addition and deletion of nodes and links and for displaying the network on a window. Class 'NETWORK' also contains the date and time of the last storage operation and a non-persistent flag that indicates whether or not the network was modified after the network was last saved.

Class 'AC-GRAPH' inherits the basic network functionality from class 'NETWORK'. In addition, it protects the integrity of the graph (e.g., unique name identifiers for the links and nodes, compliance with the compacted AC-graph formalism) and it is capable of converting itself into the format of the algebraic-oriented IS (cf. Section 8.5.2, class '(MI)LP-MODEL'). Class 'AC-SUBGRAPH' contains one or more references to the subgraph-nodes which contain the graph as a subgraph. This bi-directional link allows the subgraph to trigger the subgraph node at the aggregate level to update its graphical appearance (i.e. its icon view, cf. Section 8.8.2).

![Diagram of network structure of a compacted AC-graph.](image)

**Figure 8-10** CD: The network structure of a compacted AC-graph.

Class 'AC-GRAPH' may only contain instances of subclasses 'AC-GRAPH LINK' and 'AC-GRAPH-NODE'. As shown in Figure 8-11, both the AC-graph and the associated links and nodes contain a single instance of class 'IDX COLLECTION' which serves as an
abstraction of a set of index symbols. The implementation ensures that the set of index symbols associated with the links and nodes of an AC-graph are subsets of the set of index symbols associated with the AC-graph (cf. Figure 8-12).

Figure 8-11 CD: AC-graph (components) and their sets of index symbols.

Figure 8-12 OD: Compacted AC-graph components and their sets of index symbols.

Class ‘AC-GRAPH-NODE’ serves as an abstract base class for several node classes: ‘ACTIVITY-NODE’, ‘CONSTRAINT NODE’, ‘SUBGRAPH NODE’, and ‘OBJECTIVE-F NODE’. Each of these derived classes has a distinct graphical representation. Class ‘OBJECTIVE-F NODE’ serves as an objective-function node. The compacted AC-graph formalism represents the contribution or reduction of the objective-function value of an activity node by a link which is connected only to the activity node. The reason for having a class representing objective-function nodes stems from the fact that we have chosen to implement links consisting of two nodes in order to be able to modify the graphical appearance of a link. An objective-function node is represented by a small filled square that allows it to be selected with the mouse pointer. A global constraint node is part of two or more AC-graphs. The prototype stores a copy of the constraint-node object in each subgraph. Another approach would have been to insert the same constraint-node object, not a copy, in each subgraph. That, however, would have made it more difficult
to implement a mechanism for deleting nodes from a network. The copy of the constraint-node object is an instance of class ‘CONRAINT-NODE CLONE’ which is derived from class ‘CONRAIN-NODE’. Each instance of class ‘CONRAIN-NODE CLONE’ contains a reference to the global constraint node which was introduced first, labelled as the ‘original’ node. The majority of the methods by which the data of a constraint-node clone can be inspected and modified actually call upon the corresponding methods of the original constraint node object (cf. Figure 8-13). By inspecting the set of clones associated with a constraint node, it is possible to determine whether or not a constraint node is actually a global (non-empty set) or a local constraint node (empty set).

Figure 8-13 OD: inspecting the name of a constraint-node clone.

8.5.2 Object-oriented decomposition of the Implementation Scheme

Figure 8-14 shows the class diagram for index sets. Amongst other operations, abstract class ‘IDX-SET’ implements the basic operations for sets: i.e. union ‘+’, difference ‘-‘, and intersection ‘*’. The elements of class ‘IDX-SET’ are integers. To improve run-time performance, the contents of an index set are stored in a non-persistent double-linked list as pairs that comprise both a lower bound and an upper bound for each of the subrange of successive elements that can be identified. For example, index set

\{1, 41, 50, ..., 100\}

is stored as:


- Class ‘INT-ENUM-IDX-SET’ represents sets of integers. Their contents are specified simply by enumeration of the subranges in the format as shown above;

- Class ‘DAT-ENUM-IDX-SET’ represents sets containing arbitrary sets of DAT objects which are all associated with the same DAT-dictionary. The elements of these sets are specified manually;

- Class ‘DAT-ALLSET-IDX-SET’ represents index sets which contain the complete set of DAT objects that is associated with a particular DAT-dictionary. It suffice to specify the DAT-dictionary in order to create an instance of this class;
• Class ‘DAT-QUERY-IDX-SET’ represents index sets which contain a subset of the DAT objects associated with a particular DAT-dictionary that results from a query.

The integer elements of the ‘IDX-SET’ type definition that is part of the derived classes which contain references to DAT objects —i.e. ‘DAT-ENUM-IDX-SET’, ‘DAT-ALLSET-IDX-SET’, and ‘DAT-QUERY-IDX-SET’— refers to the relative order of the DAT objects in the set of DAT objects (the ‘allset’) associated with an instance of class ‘DAT-DICTIONARY’. The relative order may change when DAT objects are either being created or deleted. Therefore, the list of integers is updated each time the ‘IDX-SET’ object detects that the set of DAT objects associated with the DAT-dictionary has been modified.

Figure 8-15 shows the class diagram for an ‘index-set iterator’. An ‘index-set iterator’ provides the facilities to traverse the elements of an index set.

Figure 8-16 shows the class diagram for class ‘IDX-SYMBOL’ and class ‘IDX-COLLECTION’. Class ‘IDX-SYMBOL’ contains a reference to a single instance of class ‘IDX-SET’. The existence of instances of class ‘IDX-SYMBOL’ does not depend on the existence of instances of class ‘IDX-COLLECTION’. Class ‘IDX-COLLECTION’ serves only as a specialised container class which does not own the instances of class ‘IDX-SYMBOL’. The most-aggregate compacted AC-graph (the root graph) controls the
lifetime of instances of class `IDX-SYMBOL'. Class `IDX-COLLECTION' implements the basic set operations union `+', difference `-', and intersection `*'.

Figure 8-15 CD: Index-set iterator.

Figure 8-16 CD: Index symbols and index collections (sets of index symbols).

Abstract class `idx-function' (cf. Figure 8-17) models a structural relationship between a number of index symbols (≥2). These index symbols form the domain of the index function. The index function returns either a real value or an integer value, depending on the type of index function and the actual values for the indices. Index functions may
be associated with a link of a compacted AC-graph, for example, to model an incidence structure.

A function call to an instance of class ‘BINARY-IDX-FUNCTION’ returns either zero or one. The user defines one or more conjunctions of index symbols, each of which must be part of the domain of the index function that must have equal values for the index function to return a value of one. An instance of class ‘CONSTANT-IDX-FUNCTION’ always returns the same value, irrespective of the values of the indices.

Figure 8-17 CD: Index functions.

Figure 8-18 (next page) shows the class diagram for abstract class ‘IDX-OPERATOR’ (cf. Section 5.2.1) and class ‘COMPOUND-IDX-OPERATOR’. Class ‘IDX-OPERATOR’ captures both the domain and the range of an index operator, but not the operator prescription (in the form of an index expression).

The specialised subclasses shown in Figure 8-19 (next page) model the elementary index expressions: ‘SUM-IDX-OPERATOR’ (i+j), ‘EQUALITY-IDX-OPERATOR’ (i==j), ‘DIFFERENCE-IDX-OPERATOR’ (i-j), ‘GTE-IDX-OPERATOR’ (i\geq j), ‘LTE-IDX-OPERATOR’ (i\leq j), ‘SHIFT-IDX-OPERATOR’ (i-1), and ‘MULTIPLE-IDX-OPERATOR’ (2i).

The method apply() operates on a single tuple of integer values. The operand tuple must be part of the domain of the index operator. In accordance with the operator’s function prescription, it will generate either a single integer value or a set of integers (‘\geq’ and ‘\leq’), each of which must be a member of the range of the operator. A call to apply_reverse() will do the opposite. It takes a single integer value as operand and generates either a single tuple of integers or a whole set of integer tuples, each of which must be part of the operator’s domain. The result of a call to either apply() or apply_reverse() is stored a in freely accessible temporary buffer.
Class 'COMPOUND-IDX-OPERATOR' contains a list of instances of class 'IDX-OPERATOR'. Each of these operators is defined on a subset of the domain of the compound operator. The order of execution is the same as that in which the index operators were stored in the list. For example, suppose that we have the following index summation prescription:

\[ \forall i \in I, j \in J, k \in K, l \in L, m \in M, n \in N: \sum \{i + j = m, 2k + l = n\}(\ldots). \]
PART III - IMPLEMENTATION & VALIDATION

Using the specialised subclasses for index operators shown in Figure 8-19, the composition of the list of index operators for the index summation shown above would look similar to:

\[
\text{apply()}
\]

\[
\text{multiply: } K \rightarrow K, \quad multiply(k) = 2k.
\]

\[
(i, j, k, l) \in I \times J \times K \times L
\]

\[
\text{sum: } K \times L \rightarrow N, \quad sum(k,l) = k+l.
\]

\[
(i, j, 2k, l) \in I \times J \times K \times L
\]

\[
\text{sum: } I \times J \rightarrow M, \quad sum(i,j) = i+j.
\]

\[
(i, j) \in I \times J \times N
\]

\[
\text{apply reverse()}
\]

\[
(m, n) \in M \times N
\]

Note that the implementation of a compound index operator precludes index expressions containing sets of mutually dependent equations.

Instances of class 'INDEXED ELEMENT', as shown in Figure 8-20, represent the indexed parameters of an (M)LP-model: i.e. the decision variables, the model coefficients, and the RHS-constants. The 'IDX-COLLECTION' contains the set of index symbols associated with the indexed parameter. Each index symbol is associated with a 'COMPOUND IDX-OPERATOR' object. Compound index operators can contain only unary index operators. In order to have a uniform mechanism for dealing with index operators, each compound index operator contains an equality (identity) operator \(i=i\) by default. Indexed elements which represent constraint coefficients may contain an instance of class 'IDX-FUNCTION'. 'INDEXED ELEMENT' objects that do not have an index function associated have a 'DAT-DICTIONARY NAVIGATOR' object that provides the physical connection with the database (cf. Section 8.7).
Figure 8-20 CD: Indexed elements.

Figure 8-21 shows the class diagram for class ‘IDX-SUMMATION’ which represents the optional summation expression that is associated with a LHS-algebraic piece. Each ‘IDX-SUMMATION’ object contains a set of ‘INDEXED ELEMENT’s which make up the summed arguments. Usually, an index summation is associated with only two indexed elements: a constraint or objective-function coefficient and a decision variable.

Each instance of class ‘IDX SUMMATION’ contains two ‘IDX-COLLECTION’ objects to store the summed index symbols and non-summed index symbols separately. The set of ‘COMPOUND-IDX-OPERATOR’s contains the index operators which form the index summation prescription.

A call upon ‘apply()’ requires a tuple of integers that is derived from the sequence of indices that is associated with the RHS of an (in)equality relationship. The method apply() returns a set of tuples of integers that matches with the union of the index sets associated the indexed elements (including all non-summed indices) that form the arguments of the summation. The tuples are stored in a temporary buffer. For example, a call to ‘apply(4)’ associated with the ‘IDX-SUMMATION’ object in the following inequality relationship,

\[ \forall k \in K = \{1,2,3\}, l \in L = \{1,2,3\}, n \in N = \{1,2,3,4\}: \sum \{k,l:k+l = n\}a(k,l)x(l) \leq b(n), \]

will yield the following set of tuples \((k,l) \in K\times L\): (1,3), (2,2), (3,1). Class ‘IDX-SUMMATION’ provides the necessary facilities to traverse the buffer and to access its elements.
Figure 8-21 CD: Index summation.

Figure 8-22 shows the class diagram for an (MI)LP-model and its components. Class ‘(MI)LP-MODEL COMPONENT’ serves as an abstract base class for class ‘LHS-ALGEBRAIC PIECE’ and class ‘RHS-ALGEBRAIC PIECE’ which correspond to the concepts introduced in Chapter 5 by Murphy et al (1992). Class ‘(MI)LP MODEL’ contains a set of unrelated model components. It is not until after a call upon ‘create_mps_file()’ that the model construction algorithm assembles the complete constraints.
8.5.3 The interface between the MS and the IS: Model instances

The class diagram depicted in Figure 8-23 shows the interface between the AC-graph based MS and the algebraic-oriented IS. AC-graph objects represent (MI)LP-models. Class ‘(MI)LP-MODEL’ is used to represent (MI)LP-model instances. Hence, the change from model to model instance is used to change the representation of the structure of the model.
PART III - IMPLEMENTATION & VALIDATION

Instances of class ‘(MI)LP-MODEL’ contain a single reference to the ‘AC-GRAHAND’ object from which they originated. This reference may be used to integrate the numerical data with the AC-graph representation to obtain an AC-graph based interface to the data. This approach is also applied by the gLPS system (cf. Section 3.4.8).

MODEL-INSTANCE DATA
Instances of class ‘(MI)LP-MODEL’ do not ‘own’ their sets of data, for the same data may be used by multiple model instances. The data for a model instance is stored indirectly through the ‘DAT-DICTIONARY NAVIGATOR’ objects (cf. Section 8.7) associated with the indexed elements which embody the connections with the database at the structural level (the so-called database-path templates, cf. Section 7.2) and the sets of DATs which are used to compose the database paths through which the appropriate database entries can be located. The user is thus responsible for having the right data available in the database by the time the data is actually needed, for example, to display the model instance or to generate an MPS file (cf. Section 8.5.4).

MODEL-VERSION MANAGEMENT
‘AC-GRAHAND’ objects maintain a list of references to instances of class ‘(MI)LP-MODEL’. The model instances depend for their existence on the existence of the ‘AC-GRAHAND’: deleting the graph from the database will also delete the associated model instances. Once the user has created a model instance, the ‘AC-GRAHAND’ is labelled as ‘read only’. Neither the structure of the graph, nor the structural values of objective-function coefficients and RHS-constants, nor the connections between the components of the AC-graph and the database may be changed. Otherwise, the correspondence between the structure of the model and the input-output relationship for the model instances will be lost because the AC-graph is used to display the model instance. If the model builders decide to make a change to the AC-graph, then they must either delete all existing model instances or store a copy of the AC-graph and make the change to that copy.

8.5.4 MPS files
Instances of class ‘(MI)LP-MODEL’ can create an MPS file. The performance of this operation depends to a large extent on the number of disk accesses (read operations).

The performance of the creation of an MPS file improves significantly if the content of the file is written to disk in a single step. Therefore, we store the content of the MPS file in memory first. Figure 8-24 shows the class diagram for an MPS file. Class ‘MPS FILE’ is implemented as a formatted text file. The contents of an MPS file is stored as
an ordered sequence of MPS records. Figure 8-25 shows the class diagram for MPS records.

Figure 8-24 CD: MPS file.

Figure 8-25 CD: MPS records.
8.6 The database management subsystem

This section discusses the implementation of DAT-dictionaries, DATs, sets of DATs, and DAT queries. Section 8.8 outlines both the graphical representation of the elements of the data-modelling scheme and the implementation of the container objects (folders) which may be used to store compositions of (MI)LP-model representations and data objects.

Figure 8-26 shows the class diagram for the key structure of both DAT-dictionary attributes and DAT attributes. Instances of both class ‘FIXED KEY’ and class ‘KEY TEMPLATE’ can identify DAT-dictionary entries. The key of a DAT entry must be an instance of class ‘FIXED KEY’.

Figure 8-28 shows the class diagram for DAT-dictionary attributes. Each instance of class ‘DAT-DICTIONARY ATTRIBUTE’ owns an instance of a specialised subclass of class ‘KEY’. Instances of concrete subclasses of class ‘NUMERICAL DAT-DICTIONARY ATTRIBUTE’ specify the format of DAT entries which contain either an integer or a real at the DAT level. The referential DAT-dictionary attributes specify the format of a DAT entry with a reference to a DAT.

Figure 8-28 displays the class diagram for class ‘DAT-DICTIONARY’. As shown in the diagram, class ‘DAT-DICTIONARY’ is actually a dedicated container class for instances of class ‘DAT-DICTIONARY ATTRIBUTE’. Figure 8-28 also shows that each DAT-dictionary object owns a set of DATs. Deleting the DAT-dictionary from the database will also destroy the set of associated DATs.
Figure 8-27 CD: DAT-dictionary attributes.

Figure 8-28 CD: DAT-dictionaries.

Figure 8-29 shows the class diagram for DAT attributes. Note the similarity to the class diagram for DAT-dictionary attributes depicted in Figure 8-27. The same is true for the class diagram for class ‘DAT’ shown in Figure 8-30. Its layout is almost identical to the class diagram for class ‘dat-dictionary’ shown in Figure 8-28.

The connection between a DAT attribute and its associated DAT-dictionary attribute is made implicitly — the relative order of a ‘DAT’-object’s ‘DAT ATTRIBUTE’'s matches the relative order of ‘DAT-DICTIONARY ATTRIBUTE’ objects in the associated ‘DAT-DICTIONARY’. This mechanism, however, only works well if there is a one-to-one relationship between ‘DAT-DICTIONARY ATTRIBUTE’'s and ‘DAT ATTRIBUTE’'s. However,
PART III - IMPLEMENTATION & VALIDATION

a DAT-dictionary entry with a key template defines the format of a whole class of attributes with template keys at the DAT level (cf. Chapter 6). Each such set of template attributes is stored separately as part of a ‘TEMPLATE DAT’ object. Each ‘DAT’ contains a list of ‘TEMPLATE DAT’s that matches the relative order of the attributes at the DAT-dictionary level that have a key template.

![Diagram](image)

**Figure 8-29 CD: DAT attributes.**

![Diagram](image)

**Figure 8-30 CD: DATs.**

Figure 8-31 depicts the class diagram for a set of DATs. The reference to an instance of class ‘DAT-DICTIONARY’ is used to ensure that each set of DATs contains ‘DAT’-objects of the same type. Instances of the derived class ‘DAT ALLEST’ contain all ‘DAT’-objects associated with the DAT-dictionary.
Instances of specialised subclasses of class ‘DAT QUERY’ (cf. Figure 8-32) define a query on a set of DATs, each of which must be associated with the DAT-dictionary that is referenced by the ‘DAT QUERY’ object. The referenced ‘KEY’ object is used to select the DAT-dictionary attribute to query for. Instances of class ‘NUMERICAL DAT QUERY’ define a query condition (‘=’, ‘≠’, ‘≤’, ‘<’, ‘≥’, or ‘>’) for a numerical data attribute. Instances of class ‘DAT-REF DAT QUERY’ define a query condition (‘=’, ‘≠’) for a referential data attribute. Class ‘COMPOUND DAT QUERY’ defines a sequence of ‘DAT QUERY’ objects, using set operators (implemented as an enumerated type): ‘and’, ‘or’, and ‘xor’ (exclusive or).
PART III - IMPLEMENTATION & VALIDATION

8.7 The interface between the models and the data

Figure 8-33 shows the class diagram for class ‘DAT-DICTIONARY NAVIGATOR’ whose instances may be used to store a database connection using a database-path template (cf. Chapter 7). Figure 8-34 shows the connection between an instance of class ‘DAT-DICTIONARY NAVIGATOR’ and the components of an AC-graph.

Figure 8-33 CD: DAT-dictionary Navigators.

Figure 8-34 CD: The database connection for AC-graph components.
8.8 The model processor subsystem

In this section, we present an outline of the model construction algorithm that we use to convert an (MI)LP-model representation in terms of the algebraic-oriented IS into a complete model in order to create an MPS file which is accepted by an external (MI)LP-solver. Section 8.8.2 outlines the transmission mechanism that we use to send the MPS file to the solver and the solution file in the opposite direction.

8.8.1 Outline of the model construction algorithm

Figure 8-36 on the next page shows an outline of the model construction algorithm using a pseudo programming language. ‘Key words’ are depicted in bold. Identifiers are underlined to avoid an excessive use of underscores. The familiar ‘dot’-notation is used to denote a method or an attribute of an object.

8.8.2 The interface between the Model Processor and external (MI)LP solvers

If the (MI)LP solver and the prototype are not running on the same machine, there must be a network connecting both machines. The MPS-file transmission mechanism of the prototype is built upon the availability of an FTP (File Transfer Protocol) server on both sides of the network connection. Figure 8-35 shows the class diagram for class ‘WORK STATION’. Class ‘WORK STATION’ contains the necessary information for establishing an FTP connection between the local machine running the prototype and the remote workstation (except for the user’s password). This includes its IP-address (Internet Protocol) and/or the machine’s host name and its domain name, the user’s name on the external machine, and the directory containing the data files.

Figure 8-35 CD: Work Station.
PART III - IMPLEMENTATION & VALIDATION

Figure 8-36 Outline of the model construction algorithm.

for ( all (MILP-model RHS-algebraic pieces) do
{
    current_constraint create RHS( current_RHS-piece )
}

for ( all (MILP-model LHS-algebraic pieces) do
{
    (* Check for matching sets of index symbols *)
    if ( current_LHS-piece.idx-collection == current_RHS-piece.idx-collection )
    {
        current_constraint create LHS( current_LHS-piece )
    }
}

(* Now that we have assembled a complete indexed constraint: generate the corresponding rows of the matrix in MPS-file format *)

for ( all current_RHS-piece idx-symbols ) do
{
    (* create tuple of (integer) index values for the row (summed) indices *)
    for ( current_idx-symbol idx-set lowerbound to current_idx-symbol idx-set upperbound ) do
    {
        current_row-idx tuple insert( current_idx-symbol idx-set current_value )
    }

    create MPS rows-section data record( current_MPS-row identifier, current_RHS-piece RHS-constant get value( current_row-idx tuple ) )

    for ( all current_constraint LHS-pieces ) do
    {
        (* LHS-piece summation apply () produces a list of tuples with index values which are all compatible with the union of the sets of index symbols for the coefficient and the variable *)

        current_LHS-piece summation apply( current_row-idx tuple )

        for ( all union_a x compatible idx-tuples ) do
        {
            (* create tuple of index values that is compatible with the set of index symbols for the constraint coefficient *)
            x-compatible idx-tuple := current_LHS-piece create x-compatible idx-tuple( union_a x compatible idx-tuple )

            create MPS columns-section data record( current_MPS-row identifier, current_MPS-column identifier, current_LHS-piece coefficient get value( x-compatible idx tuple ) )
        }
    }
}

(MILP-model sort MPS-data records)

write MPS-file

238
DESIGN & IMPLEMENTATION OF THE PROTOTYPE

The FTP protocol is only sufficient for transferring files — not for invoking the external solver. To call upon the external solver, we use the RPC (Remote Procedure Call) facility that is available on most machines which are connected to a network. For that purpose, we send a batch file with control commands that guide the optimisation algorithm and which ensure that the external machine sends the solution file to the local machine running the prototype as soon as it is ready (cf. the object diagram shown in Figure 8-37).

Figure 8-37 OD: FTP connections between the Model Processor and an (MI)LP-solver.

Standard ASCII control files were used to manage XA. To control an OSL optimisation session, we sent a standard C-code control program which uses OSL’s callable libraries, which we compiled on the external machine using the C-compiler that is available on any machine running Unix, after which it would be executed.

8.9 The user interface

This section first briefly discusses IBM’s Common User Access (CUA) guidelines for object-oriented user-interface design [IBM Corporation, 1992]. The design of the prototype’s user-interface complies to a large extent to these guidelines. The remaining parts of this section present the various parts of the user-interface as they relate to the prototype’s working environment, the (MI)LP-modelling facility, and the DBMS.
PART III - IMPLEMENTATION & VALIDATION

8.9.1 User interfaces and object orientation

In the CUA environment, an object is any visual component of a user interface that a user can work with as an independent unit to perform a task. A database, one record in a database, a bar chart, one bar in a bar chart, and a printer are all objects. Each object can be represented by one or more icons that a user can interact with. However, an object need not always be represented by an icon, and not all interaction is accomplished by way of icons. A user can interact with an object by opening a window that displays more information about the object and which contains a variety of mechanisms for interacting with the object.

OBJECT-ACTION PARADIGM

The *object-action paradigm* is a pattern for interaction in which a user first selects an object, then selects an action [IBM Corporation, 1992]. To reduce the memory load for the user when operating the system when an object is selected the system can then present a list of actions that can be applied to that object. The object-action paradigm is a continuum, with *direct manipulation* at one end and *indirect manipulation* at the other.

![Diagram: Object-Action Continuum](image)

*Figure 8-38* The object-action continuum (derived from: IBM Corporation, 1992, p54).

Direct manipulation is a way of interacting with an object via a pointing device. Indirect manipulation is interaction with an object through *choices*, text or graphics that a user can select to modify or manipulate an object, and *controls*: standard GUI-mechanisms that provide a way to present the available options and to input the user’s choices, e.g., menus, list boxes, entry fields, buttons, check boxes, etc. During indirect manipulation, an object and an action are separated. A user selects an object first, then the interface immediately tailors and presents a list of appropriate actions, which are displayed as choices that the user can apply to that particular object.

TYPES OF OBJECTS

The CUA environment includes three types of object classes: (1) *container objects* the purpose of which is to provide a way for a user to group related objects for easy access and retrieval; (2) *data objects* whose primary purpose is to convey information, such as text or graphics; and (3) *device objects* which either represent a physical object in the real world, e.g., a printer, or a logical object in a user’s computer system, for example, a wastebasket object.

240
VIEW TYPES

The CUA interface distinguishes four basic types of views for objects in the CUA environment (objects can have more than one view):

1. **COMPOSED VIEWS.** A composed view presents the components of an object in relative order. This relative order contributes to the overall meaning of the object: changing the order of the objects changes the meaning of the entire object;

2. **CONTENTS VIEWS.** A contents view lists the components of an object. The order of the components does not affect the meaning of the object containing the information. An icons view is a type of contents view in which each object is displayed as an icon. An icon view is appropriate if the user must have an easy way to directly manipulate the objects;

3. **SETTINGS VIEWS.** A settings view displays information about the characteristics, attributes, or properties of an object, and it provides a way for a user to change the settings of some of these properties. Typically, a settings view is provided for each type of object;

4. **HELP VIEWS.** A help view provides information that can assist a user in working with an object.

The CUA guidelines emphasise that these four types of views represent idealised views along a continuum of possible view types. The CUA guidelines also indicate that the distinction between a composed view and a contents view can be somewhat blurred (p53), depending on the object being viewed. A designer should primarily be concerned with creating views that convey information in a form that is meaningful to users, regardless of where the views fall in the continuum of view types. The name that is given to a view should be based on the user's conceptual model.

### 8.9.2 Design of the object-action interaction mechanism

In the CUA interface, an icon is a small graphic image that represents an object. The user can act upon the object by pointing at the icon with a pointing device, and then pressing a button on the pointing device to select the icon. The response of the object depends on which button was pressed, and for how many times or for how long (single-click, double-click, or a drag operation). A single-click on the left-hand side button typically selects the icon view, a single-click on the right-hand side button typically displays a context-sensitive pop-up menu for the object, a double-click on the left-hand side displays the default view for the object, and finally, a movement with the right-hand side button pressed commences a drag and drop operation.
PART III - IMPLEMENTATION & VALIDATION

An object can have several icon views at the same time. For example, an instance of class ‘CONSTRAINT NODE’ may be both visible as part of a compacted ‘AC-GRAPH’ and it may be part of the so-called interface view of an ‘AC-GRAPH’ which contains all of the graph’s global constraint nodes.

The appearance of an icon view can change when some attribute of the corresponding object changes. For example, the icon of an ‘AC-GRAPH’ that is part of a container object changes when the graph is displayed in a separate window. The icon of a container object, or a device object changes when it is select as a target object in a drag and drop operation.

Figure 8-39 shows the class diagram for class ‘GUI OBJECT’, class ‘ICON VIEW’ and class ‘SET OF ICON VIEWS’. Class ‘GUI OBJECT’ serves as a base class for classes that must have an icon view representation in support of direct manipulation interaction. Each instance of class ‘GUI OBJECT’ contains a set of ‘ICON VIEW’ s, each of which has either one or two icons associated. The use of two icons supports graphical associations of basic icons. For example, the icon view for the Model Base subsystem combines the icon for an AC-graph and the icon of a folder container object (cf. Section 8.9.3).

![Diagram showing relationships between GUI OBJECT, SET OF ICON VIEWS, ICON VIEW, WINDOW, STRING, and LIST.]

Figure 8-39 CD: GUI objects and icon views.

Figure 8-40 shows an object diagram which depicts the interplay between objects at the sending end of a drag-and-drop operation. After the StarView UIMS has detected a mouse event, it dispatches a mouse-event object to the appropriate window. This window, in turn, starts up an inquiry among the objects it contains to determine if there is an object the icon view of which occupies a window region that surrounds the coordinates of the mouse event. If such an object exists, the drag-and-drop enabled window commences a drag-and-drop operation.
8.9.3 The working environment

For each StarView-based MDI-application, there must be a specialised subclass of class ‘MDIApplication’ with exactly one instance (‘static’). Class ‘MDIApplication’ contains a virtual method ‘Main()’ which replaces the function main() of a normal C++ program. Class ‘MDIApplication’ provides, amongst others, functions for initialisation of the application data, and for starting and terminating input processing. Class ‘DSS-IDE’, as shown in Figure 8-41, represents this specialised subclass for the prototype. Class ‘DSS-IDE’ also serves as a container object whose components, most of which are also container objects, give access to all other objects that exist in the application:

- The ‘MODEL_BASE’ which gives access to all compacted AC-(sub)graphs;
- The ‘DAT(A) BASE’ which gives access to all DAT-dictionaries and DATs;
- The ‘TEMPLATE BASE’ which contains templates for the creation of AC-graphs, DAT-dictionaries, and folders;
- The ‘MAIN-WINDOW’ (application window) which ‘owns’ the components of the GUI.

In addition, class ‘DSS-IDE’ contains a number of device objects. ‘MILP SOLVER’ objects provide the facilities of external solvers. The ‘DSS-IDE SERVER’ object represents the local machine running the prototype. Finally, the ‘WASTE BASKET’ takes care of deleting objects.
Figure 8-41 CD: MDIApplication and DSS-IDE.

The window area of the 'MAIN WINDOW', as shown in Figure 8-42, contains a menu bar, a toolbox, an MDI-(child)window region, a so-called 'work bench' which holds the container, and device objects mentioned above, and an application-status bar.

Figure 8-42 Layout of the prototype's working environment.
CREATING OBJECTS

Objects may be created from the application's menu bar, the toolbox, or by means of the 'TEMPLATE BASE' which contains designated objects that act as templates for new objects. The template objects have a special visual representation which is based on the way object templates are represented in OS/2. By dragging the object template on the icon of a folder, its icon view or its contents views, and releasing it there, a new object will be created which is stored in that particular folder object.

Figure 8-43 Drag and drop.

DELETING OBJECTS

Objects may be deleted by either selecting the 'delete' option from the object's pop-up menu or by dragging the object to the icon of the 'WASTE BASKET' and releasing it there.

8.9.4 Container objects

Figure 8-44 shows the class diagram for abstract class 'FOLDER'. A folder is a container for AC-graphs, DAT-dictionaries, and other folders. Figure 8-45 depicts the hierarchical folder structure. 'SYSTEM FOLDER's serve as 'root' folders and have a direct linkage with an instance of class 'system base' which is either the Modelbase subsystem, the DAT-base subsystem, or the Template Base subsystem (cf. Section 8.9.3).

Class 'folder icon view' is a subclass of StarView class 'MDIWindow'. A 'folder icon view' object displays an icon view—a special type of contents view (cf. Section 8.9.1)—of the contents of a folder (cf. Figure 8-46). Class 'folder browser view' is based on StarView class 'Browser'. It provides more information about the objects contained while this type of view is also more concise and better suited handling large amounts of objects (cf. Figure 8-47).
Double clicking on the icon representation of a folder object will display the folder's icon view. The browser view and the composed view (only for system folders) of a folder become visible after selection of the corresponding entry of the pop-up menu.
8.9.5 User-interface design for compacted AC-graphs

Compacted AC-graphs can be created and displayed within the context of an instance of class ‘AC-GRAPH WINDOW’ (cf. Figure 8-48). Class ‘AC-GRAPH WINDOW’ is derived from class ‘NETWORK WINDOW’ which captures the more general functionality for creating and displaying networks.

![Diagram](image)

**Figure 8-48** CD: AC-graph window (composed view).

**Creating nodes and links**

Nodes can be created by first selecting the toolbox button for the node type and then by clicking on the centre of an empty window area which has the required size. Links can be created by first selecting the toolbox button for the link type (constraint-coefficient, activity-bound, or objective-function coefficient), then selecting the icon of the source
PART III - IMPLEMENTATION & VALIDATION

node and finally by dragging the mouse pointer on the icon of the target node and releasing it there.

The input processing routines of class ‘AC-GRAPH WINDOW’ are based on the syntactical rules for creating compacted AC-graphs. For instance, it is not possible to create a link between two nodes of the same type. The mouse pointer will take the shape of a ‘not-allowed’ sign while the mouse pointer is on the icon view of the target node. However, the input routines do not ensure the syntactical correctness of the AC-graph under construction completely. For example, it is still possible to create a graph which contains isolated (unconnected) constraint or activity nodes. The editor which is part of the prototype of Networks [Jones, 1990] completely ensures the syntactical correctness of the graphs under construction. Jones refers to this type of editor as a syntax directed editor. A syntax directed editor for Pascal would not allow the programmer to input an ‘if-then-else’ block character by character, but it would provide a single key stroke or a specialised button that inserts an ‘if-then-else’ template. The implementation of a syntax directed AC-graph editor, however, would have been more complicated and would have taken more time than the current implementation, which is why this implementation was used. The correctness of the AC-graph is checked automatically just prior to the creation of a model instance or at any time the model builder explicitly instructs the system to do so.

VIEW TYPES

Figure 8-49 shows the views associated with class ‘AC-GRAPH’ with the exception of the graph’s composed (network) view. An ‘AC-GRAPH INSTANCES VIEW’ shows the graph’s (MI)LP-model instances. An ‘AC-GRAPH SETTINGS VIEW’ provides a means for modifying the graph’s identifier and for viewing several status attributes, including the date and time the graph was last saved, which also appear on the statusbar of the ‘AC-GRAPH WINDOW’. The ‘idx-collection view’ shows the set of index symbols associated with the components of the model. Finally, a graph’s ‘AC-GRAPH INTERFACE VIEW’ contains the global constraint nodes that are part of the graph. Figure 8-50 contains an example of each type of view.
Figure 8-49 CD: Views for an AC-graph.

Figure 8-50 Examples of AC-graph views.

The nodes and links of an AC-graph, as shown in Figure 8-51, also have dedicated views for their settings, their sets of index symbols, and their database connection (if applicable), respectively.
PART III - IMPLEMENTATION & VALIDATION

Figure 8-51 CD: Views of AC-graph components.

MAKING A DATABASE CONNECTION
Figure 8-53 on the next page shows three successive 'NAVIGATOR VIEW's which are necessary to connect the objective-function coefficient 'unit manufacturing cost' of the reference model (cf. Figure 5-11) to the appropriate DAT-dictionary entry.

CREATING MODEL Instances
After having specified the structural properties of a model, the model builder can create a model instance by selecting the corresponding entry from the AC-graph's pop-up menu. Amongst others, the dialog shown in Figure 8-52 will then be displayed. It contains the set of index symbols associated with the graph (and any of its subgraphs). The user is prompted to define the sets of DATs for each of the index sets.

Figure 8-52 Pop-up menu of an AC-graph object.
PART III - IMPLEMENTATION & VALIDATION

INVOKING THE SOLVER
After creating an (MI)LP-model instance, the user can invoke an (MI)LP solver either by dragging the icon for the model instance to the icon of the solver (device) object or by means of the pop-up menu associated with the model-instance object.

8.9.6 User-interface design for the data-modelling scheme

Figure 8-54 shows the class diagram for the view types associated with both class 'DAT-DICTIONARY' and class 'DAT'.

![Class diagram](image)

**Figure 8-54** CD: DAT-dictionary and DAT views.

The contents view of a DAT-dictionary displays the set of DATs associated with the dictionary. The composed view of a DAT-dictionary displays its data attributes. As shown in Figure 8-55, each row corresponds to a key field of a data attribute. Key-field templates have a '<T>' indicator which precedes the identifier of a DAT-dictionary. As the key of each data attribute may have a different number of key fields, we decided not to display the key-fields in a separate column, but rather to use a separate row for
each key field instead. However, as the number of key-fields rarely exceeds four, a column-wise representation is more concise and likely to be clearer.

To create a DAT-dictionary entry, the user is presented a dialog screen for inputting the sequence of key fields (the relative order is of importance), the entry's data type, and in the case of a referential attribute, the referenced DAT-dictionary. To modify or delete a data attribute, the user must select the entry from the composed view, after which a pop-up menu will be displayed from which the appropriate choice can be selected.

![Figure 8-55 DAT-dictionary composed view.](image)

Figure 8-56 shows an example of the contents view of a DAT which displays both the key and the value of its attributes. As shown in Figure 8-56, the entries that correspond to a DAT-dictionary entry with a key template are listed separately.

![Figure 8-56 DAT composed view.](image)
9. CASE STUDIES AND EVALUATION

OUTLINE
Two case studies are presented. The models focus on logistics planning issues and have been used to implement prototype MIP-based planning support systems.

For each case, this chapter outlines the context and trade-offs underlying the models. A block-schematic representation that matches the original formulation is presented, followed by an equivalent modular AC-graph representation and an associated data model. The last section discusses our findings when using ACOM.
9.1 Introduction

The case studies presented in this chapter were used to gain practical experience with the current implementation of ACOM. On the basis of this experience, the intended advantages of the conceptual design underlying the prototype and the pros and cons of the current implementation can be verified. At the outset of the evaluation, several questions were raised which may be subdivided into two main categories:

1. **Questions Regarding the Intended Advantages of the Conceptual Design:**
   A1. Formulate (MI)LP models
   1. Does the modular design of the (MI)LP modelling scheme support a natural hierarchical arrangement of the representation of models?
   2. Can the flow analogy be put into use effectively? Can relationships between activities and constraints to which the flow analogy does not apply be formulated easily?
   3. Are there basic rules or guidelines that facilitate the formulation of such model structures?
   4. Does the compacted AC-graph formalism attract the model builder to use certain intuitively appealing constructions and is the use of these constructions appropriate or are there any pitfalls?
   5. Is it easy to discern the structure of complex models from the compacted AC-graph representation?
   6. Does the modelling scheme provide a proper balance between the information provided at the level of AC-graphs and the information that may be obtained through pop-up menus?

A2. Define data model
   1. Does the data-modelling scheme support the creation of appropriate data structures?
   2. How does the DAT-dictionary Navigator mechanism perform in practice? Is it easy to establish a database connection?

3. **Questions Regarding the Current Implementation of ACOM:**
   Algorithms

256
1. Is the conceptual design underlying the current implementation complete? Does it work correctly?

User interface

1. How does the representation of hierarchical AC-graphs using separate MDI (Multiple Document Interface) windows to view distinct subgraphs perform?

2. Does the support for data entry meet the requirements of large models?

A convincing means which, to some extent, supports an objective validation of the intended advantages of the implementation of ACOM and the underlying conceptual design is the use of control groups. A typical set-up of such an experiment would require:

1. Two groups of users with similar (MI)LP modelling capabilities, who are not accustomed to using a specific modelling system. This will avoid biased results owing to large differences in learning curves. The members of the first group would use ACOM. The members of the second group would use a popular modelling system, e.g. AlMMS or AMPL;

2. A number of optimisation problems that can be readily formulated and solved as (MI)LP models. All participants would be given the same set of problems. The problems should be stated in a form that does not allow for a direct translation (i.e. transcription) into the modelling languages that are used by the two modelling systems.

The participants would be asked to formulate the corresponding models and to solve the appropriate model instances. The data that would allow for a comparison between both modelling systems would be the fraction of accurate model formulations (those that yield correct solutions) and the time spent to formulate the models and to create model instances.

The need for a fair comparison and the use of a third party for evaluation purposes imposes some requirements on the maturity of the implementation, which should be stable. Unfortunately, the current prototype is not stable and only infrequently will it be possible to pass through all stages of the model development trajectory without running into an error that forces the application to terminate. If the prototype is terminated due to an error, the database may be corrupted thus preventing any further use. A feasible safeguard is to backup the database on a regular basis, typically 10-15 times during a complete model-development cycle. Clearly, this working procedure would be unacceptable to third-party users.
PART III - IMPLEMENTATION & VALIDATION

Debugging the code revealed that errors which cause the application to terminate most frequently occur after intensive use involving many user-system interactions and many database insertion and updating operations. It also appeared that errors occur in parts of the code that have been properly executed many times before. In such cases, there is usually a memory allocation problem, e.g. buffers running out of space or obsolete pointer references. The irregular pattern of the occurrence of errors makes it difficult to determine the causes of these errors, although there are strong indications that they may be due to the interaction with the database subsystem. This observation is partly supported by the fact that backups of databases that worked properly on previous occasions sometimes caused errors which could always be solved simply by resetting the machine. The limited time that was left at the end of the project prevented further efforts to overcome this problem.

From the practical experience gained by using ACOM (cf. 9.5.3, Table 9-4), it appears that the performance of the POET database subsystem is poor. The time that is required to insert a number of data attributes consisting of a total of \( n \) fields, both key fields and data fields, is of the order \( n(n+1) \). From these figures, it becomes evident that the current prototype can only be used for solving small problems. Therefore, only small-sized problems were used to validate the correctness of the algorithms which support the complete model-development trajectory, i.e.:

1. The conversion of a compacted AC-graph into an equivalent referential form consisting of LHS and RHS pieces;
2. The assembly of a complete representation of a model from these pieces;
3. The merging of the complete representation with the appropriate numerical data;
4. The production of an MPS-file;
5. Downloading the results into the database subsystem.

To evaluate ACOM with respect to both its support for the model-development trajectory up to the point of inputting data for model instances and with respect to the merits of the current implementation, complex models were used which have been published in the literature. Consequently, it was not necessary to validate the structure of these models. The accuracy of the compacted AC-graph representation can be easily verified using the elementary rules for translating a compacted AC-graph into an equivalent algebraic formulation.

---

1 Meanwhile POET Software announced that its future releases would no longer support the Borland C++ compiler for OS/2. POET v2.1 for OS/2 [Poet.1994] supports the IBM CSet C++ compiler.

258
The models that were used in the case studies focus on logistics planning issues. The reference model which was introduced in Chapter 3 will be used to demonstrate the creation and solving of model instances. The Deployment Model (DM) [Mourits, 1995] and the Coal-Chain model [Savenije, 1992; Posthumus, 1992] were both used to implement prototypes of MIP-based planning support systems. The complexity of the DM and the size of the model formulation, which contains a great number of constraints to which the flow analogy does not apply, constitute a major challenge to input the model in ACOM. Section 2.1.4.2 already outlined the major characteristics of the Deployment Model.

PLATFORM
ACOM is running on an Intel 486DX2-66 based PC, running OS/2 v3.0 (‘Warp’), with 16MB of RAM, and approximately 100MB free disk space.

OUTLINE OF THIS CHAPTER
In the remainder of this chapter, three case studies are presented. In each case, first the context and the trade-offs are introduced, then a block-schematic representation of the model is presented, followed by an equivalent compacted AC-graph representation. The block-schematic representation matches the original formulation which can be found in the references. Finally, after the presentation of the data model, Section 9.5 contains our findings with applying ACOM on each of these cases.

9.2 Case 1: The reference model

Section 3.4.5 introduced both the objective, the trade-offs, and the block-schematic representation of the Production Location Allocation Model.

DATA MODEL
Figure 9-1 and Figure 9-2 show two alternative data models. The first model uses DAT-dictionary Infrastructure to model the data that is related to transport links (T-Links) between sites. T-Links are represented implicitly as part of the key of the data attributes using an ordered pair of key-field templates with references to dictionary Geo Site. The data model that is depicted in Figure 9-2 may be viewed as a more accurate portrayal of the problem domain, since it uses a separate DAT-dictionary ‘T-Link’. The parameters of the reference model that relate to transport links have two subscript index symbols for the sites on either site of a transport link: ‘Geo Site’→from,to. Therefore, in the second data model there is still a need for a DAT-dictionary Infrastructure which contains a list of references to T-Links using two key-field templates referring to both the source and sink of the T-Link. Using the ‘T-Link’ data attribute of dictionary Infrastructure, T-Link objects can be accessed by using the
PART III - IMPLEMENTATION & VALIDATION

two index symbols from and to. Because of its less complicated structure, to improve ACOM’s performance, the first data model was used.

Figure 9-1 Data model: T-Links modelled implicitly.

Figure 9-2 Data model: T-Links modelled explicitly.

OUTLINE OF THE CASE

Figure 9-3 illustrates the spatial setting of a rather uncomplicated case consisting of two sites: Amsterdam (AMS) and Brussels (BRX). Three optional facilities are considered, two of which, AMS:1 and AMS:2, are located near Amsterdam. Plant BRX is located near Brussels. The total manufacturing capacity of plants AMS:1, AMS:2, and BRX is 75, 125, and 75 respectively. Manufacturing technology differentials are
not taken into account. Figure 9-3 also shows the customer demand levels for products P1, P2, and P3, respectively. No profit differentials are considered: all three products sell at 10 MU (Monetary Unit) each. Product P3 is required as component part of products P2 in a ratio of 1:1. Transport costs amount to a single MU for each product. Fixed manufacturing costs are not taken into account. Figure 9-3 also shows the anticipated unit manufacturing costs for each facility. AMS:2 cannot manufacture P3 while plant BRX is not equipped for manufacturing P2. From a comparison of these figures and the unit transportation costs, it is evident that the plant located near Brussels has a net cost advantage with respect to the production of products P3.

![Figure 9-3 Spatial setting of the case.](image)

**INPUT**

Figure 9-4 and Figure 9-5 show a number of data-entry forms for inputting the case-data into the database subsystem of ACOM. Both figures show instances of data attributes with key-templates. The format of the key template is shown in the title bar of the windows.

![Figure 9-4 Representing Campus Site data.](image)

As shown in Figure 9-5, numerical data is displayed with four decimal digits conforming with the default representation of numerical values in the MPS file format.
PART III - IMPLEMENTATION & VALIDATION

<table>
<thead>
<tr>
<th>Nr.</th>
<th>(Fixed)</th>
<th>&lt;T&gt;Site</th>
<th>&lt;T&gt;Site</th>
<th>&lt;T&gt;Product</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>transport cst</td>
<td>Site:AMS</td>
<td>Site:BRX</td>
<td>Product:1</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>transport cst</td>
<td>Site:AMS</td>
<td>Site:BRX</td>
<td>Product:2</td>
<td>1.0000</td>
</tr>
<tr>
<td>3</td>
<td>transport cst</td>
<td>Site:AMS</td>
<td>Site:BRX</td>
<td>Product:3</td>
<td>1.0000</td>
</tr>
<tr>
<td>4</td>
<td>transport cst</td>
<td>Site:BRX</td>
<td>Site:AMS</td>
<td>Product:1</td>
<td>1.0000</td>
</tr>
<tr>
<td>5</td>
<td>transport cst</td>
<td>Site:BRX</td>
<td>Site:AMS</td>
<td>Product:2</td>
<td>1.0000</td>
</tr>
<tr>
<td>6</td>
<td>transport cst</td>
<td>Site:BRX</td>
<td>Site:AMS</td>
<td>Product:3</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Figure 9-5 Representing transport costs.

CREATION OF MODEL INSTANCE

It takes ACOM about 2 minutes to assemble the complete algebraic representation of a model instance on the basis of the given case from the referential LHS and RHS pieces. The given instance consists of 54 activities and 42 constraints. The assembly of the objective function takes about 13 seconds, while the assembly of the constraints requires approximately 110 seconds. Running in debug mode, the construction of an MPS file of about 400 lines takes about 3 minutes: i.e. with assertion checking activated, creation of non-zero and zero MPS entries and the generation of a log file. ACOM automatically generates the 8-character names which are required for the variables and constraints in the MPS file representation.

SOLVING THE MODEL INSTANCE

The status of a model instance, e.g. if and when an optimal solution was obtained, can be found from the corresponding entry of the instance’s pop-up menu. Figure 9-6 shows an example of the dialogue that will be displayed once this menu option is activated. The dialogue shows that the optimal solution to the model instance for the given case was obtained by using XA in 10 iterations with an objective-function value of 1156.25 MU. Note that the value of the optimal solution is depicted with 4 decimal digits, conforming with the MPS standard for representing numerical values.

262
Figure 9-6 (MI)LP model instance solution status information.

Figure 9-7 shows a selection of the solution data extracted from the XA solution report. From these data, it follows that the total manufacturing capacity of plant BRX is used for the production of P3. The demand for products P1 at the Brussels site is satisfied by plant AMS:1.

![Figure 9-7 Output data.](image-url)
9.3 Case 2: The coal-chain model

9.3.1 Introduction

To meet the demand for electricity in the Netherlands, the SEP regularly determines the amounts of fossil fuels that each power utility will require for use in the next period (typically 1 year). The optimal quantities of coal, natural gas, and oil are based upon the anticipated cost of each type of fuel and the cost of starting up and closing down power utilities (which also depends on the primary source of fuel). The optimal quantities of coal for each power utility for the next period, as determined by the SEP, serve as firm targets for the purchase of coal. The GKE is responsible for the purchase and delivery of coal.

To guarantee coal supplies, the long-term delivery contracts that are signed typically cover a period between two and five years. Besides providing a hedge against delivery uncertainties, long-term contracts offer lower prices. On the other hand, long-term contracts provide fewer opportunities to fine-tune coal deliveries when more accurate estimates of actual demand do become available, so additional contracts are necessary to satisfy final demand. New contracts may both be long-term contracts and short-term contracts (which typically span less than 1 year). Delivery contracts specify minimum and maximum supply quantities, quality levels (e.g., minimum calorific value, maximum levels of sulphur and chloride), price per ton, and the point of transfer of ownership. If ownership of the coal is transferred directly at the mine, then GKE is responsible for further transport, handling, and intermediate storage and thus for hiring the necessary transport and handling capacity by contracting terminals and carriers. Alternatively, the point of transfer may be located at a terminal.

The Coal-Chain model focuses on the planning of coal deliveries. The Coal-Chain model may be used to determine the supply quantities for existing contracts, within agreed-upon levels, and the implementation of new contracts from a set of pre-screened alternatives, given the required amounts of coal and the quality levels per power utility. The model views the Coal Chain as being composed of four different types of chains (cf. Figure 9-8): mines, export terminals, import terminals, and power utilities. In new contracts the mines and both types of terminals can serve as transfer points. The chains are mutually connected by transport links, e.g., sea links, railway, barge. Mines and export terminals are usually linked by overland transport connections. Transport between export and import terminals is usually carried out by ship. Import terminals are located in the Netherlands (Rotterdam, Amsterdam, Ijmuiden). Most of the transport between the import terminals and the power utilities is by barge.
The Coal-Chain model is split into two layers: a blended layer and an unblended layer. Coal deliveries that pass through the unblended layer comply with the quality standards imposed by the power plants without first having to be blended with other deliveries. However, in addition to transport and handling capacity, the blended layer requires temporary storage and blending capacity at the terminal where the component deliveries meet.

The decision whether or not to implement new contracts gives rise to the use of binary variables. The Coal-Chain model is formulated as a static model: the planning horizon is made up of a single-period. The objective of the model is to minimise total costs.

### 9.3.2 Block-schematic representation

Figure 9-9a-b shows a block-schematic representation of the unblended layer of the Coal-Chain model. The index symbols that have been used have the following meaning:

<table>
<thead>
<tr>
<th>Index Symbol</th>
<th>Index Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Mine</td>
</tr>
<tr>
<td>et, it</td>
<td>Terminal (export &amp; import)</td>
</tr>
<tr>
<td>u</td>
<td>Power Utility</td>
</tr>
<tr>
<td>c</td>
<td>Coal Type</td>
</tr>
</tbody>
</table>

*Table 9-1 Index symbols and associated index domains.*

The activity ‘Compulsory Delivery’ denotes the supply quantities for existing contracts. The subscripts $m$, $et$, and $it$ refer to the agreed upon points of transfer respectively. The ‘Additional Delivery’ activity denotes newly implemented supplies. The binary activity ‘Contract Additional Delivery?’ represents the decision to enter new contracts. A similar modelling approach is taken to deal with both contracted and additional handling capacity at export and import terminals and contracted and additional transport capacity for each type of transport link. For brevity, the activities representing additional transport capacity and signing contracts with carriers are therefore not included.
Figure 9.9a Block-schematic representation of the Coal-Chain model (unblended layer).
**CASE STUDIES AND EVALUATION**

<table>
<thead>
<tr>
<th>binary</th>
<th>real</th>
<th>real</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>(et,c)</td>
<td>(u,c)</td>
<td>(et,t,c)</td>
<td>(t,u,c)</td>
</tr>
<tr>
<td>(m,et,t,c)</td>
<td>(et,u,c)</td>
<td>(t,u,c)</td>
<td></td>
</tr>
</tbody>
</table>

**Objective:** Minimise Cost of Transportation (Existing LT-contracts) (m,et,c) (et,t,c) (t,u,c)

**Activity Levels:**
- Max Delivery (m,c) (5-1)
- Min Delivery (m,c) (9-2)
- Max Delivery (et,c) (9-3)
- Min Delivery (et,c) (9-4)
- Max Delivery (t,c) (9-5)
- Min Delivery (t,c) (9-6)
- 0 (m,c) (3-7)
- 0 (et,c) (3-8)
- 0 (et,c) (9-9)
- 0 (et,c) (9-10)
- 0 (et,c) (9-11)
- 0 (et,c) (9-12)
- Max Handling (et,c) (9-13)
- Min Handling (et,c) (9-14)
- Max Handling (t,c) (9-15)
- Min Handling (t,c) (9-16)
- 0 (et,c) (9-17)
- 0 (et,c) (9-18)
- 0 (et,c) (9-19)
- 0 (t,c) (9-20)
- 0 (t,c) (9-21)
- Min Demand (u) (9-22)
- 0 handle IBF (et,c) (9-23)
- 0 handle IBF (t,c) (9-24)
- 0 (m,c) (9-25)
- 0 (t,c) (9-26)
- 0 (u) (9-27)
- 0 (m,c) (9-28)
- 0 (et,c) (9-29)
- 0 (t,c) (9-30)
- 0 (u) (9-31)
- 0 (u) (9-32)

**Figure 9-9b** Block-schematic representation of the Coal-Chain model (unblended layer).
Constraints (9-1) to (9-6) define both upper and lower bounds on compulsory deliveries. Constraints (9-7) to (9-12) specify upper bounds ‘MxAD’ and lower bounds ‘MnAD’ on short-term deliveries provided that the new contract is selected to be implemented. Similar to constraints (9-1) to (9-12), constraints (9-13) to (9-20) define bounds on compulsory and additional handling capacity for export and import terminals respectively. Next follows an explanation of constraints (9-21) to (9-32):

(9-21) Utilities must receive their pre-specified quantities. Supplies must cover at least these quantities;

(9-22) Handling capacity for both export and import terminals must be adequate to (9-23) accommodate total inbound shipments;

The ‘Post-Transfer Supply-Chain Flow’ activity encompasses coal supplies, from the point of transfer, i.e., the mine, the E(export) terminal, or the I(import) terminal, to the power utility. The sets of index symbols associated with the three activities correspond to the representation of the downstream supply chain, from the transfer point onwards. Note that constraints (9-24) to (9-30) only apply to sites that are connected by transport links. The existence of these links must be determined by inspection of the database.

The ‘Post-Transfer Supply-Chain Flow’ activity is used to formulate the flow balance equations that are related to the transport activity:

(9-24) The downstream flow of coal, from the transfer point onwards, must equal the (9-25) quantities being delivered at the mines, the E-terminals, and the I-terminals (9-26) respectively;

(9-27) The total flow of coal (which originated from deliveries being transferred at either the mines, E-terminals, and I-terminals) being received at a power utility must be equal to the utility’s total supplies;

(9-28) The amount of coal that is being transported between a mine and an E-terminal must be equal to the deliveries being transferred at that particular mine;

(9-29) The flow between an export and import terminal must be equal to deliveries being transferred at the E-terminal plus those being transferred at the mine;

(9-30) The flow between an I-terminal and a power utility must be equal to the deliveries being transferred at the I-terminal, the E-terminal, and the mine;
(9-31) Deliveries must comply with certain quality standards which vary per power (9-32) utility. These quality standards apply to properties which can differ per coal type. For each property, the total supplies to a utility must be within prespecified bounds.

### 9.3.3 AC-graph representation

Figures 9-10 to 9-12 show a compacted AC-graph representation of the unblended layer of the Coal-Chain model. As in the block-schematic representation, the activities for additional transport capacity and contracting carriers have been omitted for the sake of brevity. The AC-graph representation does not contain the ‘Post-Transfer Supply-Chain Flow’ activity. Instead, the equivalent AC-graph representation balances the flows directly. Figure 9-11 shows the E-terminals subgraph. The structures of the E-terminals subgraph and the I-terminals subgraph are identical. For the sake of brevity, the I-terminals subgraph has been omitted.

![AC-graph - Coal Chain](image)

![AC-graph - Utilities](image)

Figure 9-10 Coal-Chain model (unblended layer) - Overall graph & Utilities subgraph.
Figure 9-11 AC-graph: Coal-Chain model (unblended layer) - E-terminals subgraph.
9.3.4 Data model

The database graph depicted in Figure 9-13 provides an image of the data model that has been used to support the modular AC-graph representation of the Coal-Chain model. Figure 9-13 is a manual composition of ACOM’s option to provide a composed database view and several DAT-dictionary composed views.

Figure 9-13 Data model in support of the Coal-Chain model.
9.4 Case 3: The deployment model

9.4.1 Introduction

The Deployment Model (DM) is based on a given spatial arrangement of Suppliers, Distribution Centres (DCs), and Customers. Depending on each of the supply chains in which a DC participates, a DC may serve three purposes: products may be kept in inventory, products may be transhipped, and products may be assembled into end-items. Figure 9-14 shows the possible flows of products inside a DC.

![Diagram of flows inside a DC](image)

**Figure 9-14** Possible flows of products inside a DC. (Source: Mourits, 1995, p120.)

Unlike location-allocation models, the DM considers the timing of product flows by means of supply frequencies and order-reception lead times. The DM searches for the best places to keep inventory, to assemble, or to tranship each product type, such that the given customer service requirements, with respect to the timing and sizing of demand, are satisfied. The trade-offs include those between transportation economies of scale and the cost of keeping inventory throughout a supply chain while complying to lead time and supply frequency restrictions. The cost factors that are accounted for include:

1. **TRANSPORTATION COSTS**: the fixed costs of employing a transport link and the variable costs per tour;

2. **INVENTORY COSTS**: the fixed cost of installing inventory capacity, the handling costs of stocking items and preparing them for shipment, and the capital outlet associated with cycle stock and safety stock (if any);

3. **TRANSHIPMENT COSTS**: the variable costs of handling and transshipping products through a DC;

4. **FINAL-ASSEMBLY COSTS**: the fixed costs of installing final assembly capacity and the variable cost of assembly plus the handling costs.

272
The MIP-formulation of the Deployment Model is subdivided into three distinct aspect models:

1. **FLOW-ASPECT MODEL**, which describes the commodity flows through the network as well as the consistency requirements which balance the inbound and outbound flows of DCs;

2. **FREQUENCY-ASPECT MODEL**, which must ensure that customers get their products with the minimum required frequency. If a DC serves as a transhipment centre for a product, then the largest required order reception frequency of the customers of a DC (both end-customers and DCs) is propagated upstream and serves as the demanded reception frequency for the DC. If the DC keeps the product in inventory, then the DC’s demanded reception frequency is based on the minimum inventory replenishment frequency. The supply frequency determines the cycle-inventory cost. The frequency aspect model also determines the number of tours between DCs and the number of tours from DCs to customers using both the demanded reception frequency and vehicle capacity. The variable costs of transportation are based upon the number of tours;

3. **LEAD-TIME ASPECT MODEL**, which, for each node and for each product, intends to find a match between the required downstream order lead times (starting with the demanded customers lead time), and the upstream order lead-time that is offered by that part of the supply chain that precedes the current node (starting with the minimum lead time offered by suppliers). The presence of inventory at a particular node resets the calculated order lead time to zero. Replenishment lead times are used to calculate the cost for maintaining safety stock (if any).

### 9.4.2 Block-schematic representation

**FLOW-ASPECT MODEL**

Figure 9-16a-b displays a block-schematic representation of the flow-aspect model. Unlike those of the block-schematic representation of the reference model (cf. Figure 3-8), the elements of Figure 9-16a-b are denoted by using the referential notation introduced in Section 5.2. The index symbols refer to the following index domains:

<table>
<thead>
<tr>
<th>Index Symbol</th>
<th>Index Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Supplier</td>
</tr>
<tr>
<td>dc, dc-f (from), dc-t (to)</td>
<td>DC</td>
</tr>
<tr>
<td>c</td>
<td>Customer</td>
</tr>
<tr>
<td>prod, asmbly</td>
<td>Product</td>
</tr>
</tbody>
</table>

*Table 9-1 Index symbols and associated index domains.*
PART III - IMPLEMENTATION & VALIDATION

It may be easier to discern the mechanics of the DM when taking into account that block elements with a positive sign indicate that the associated activity lowers the constraint level while block elements with a negative sign increase the level of the associated constraint.

![Diagram](image)

**Figure 9-15** Direction of 'flows' in a block-schematic representation.

The constraints in Figure 9-16a-b are labelled. Next follows a brief explanation of these constraints. Constraints (9-33) through (9-36) protect the product flows that pass through a DC against inconsistencies.

(9-33) The supplies of a product plus, in the case of end-items, the quantities of the product assembled at the DC, must be equal to the outbound flow for that product, either in the same form as that in which the product arrived, or as part of an assembly;

(9-34) Two subsets of the same logical variable 'F.Assembly Flow', indicating the quantities of a particular product being assembled at a DC, are distinguished: one indexed over \{dc,asmbl\}, the other indexed over \{dc,prod\}. Constraint (9-2) ensures that when 'prod' and 'asmbl' both refer to the same product, both variables must have equal values. Constraint (9-2) is added because both subsets of variables will have a distinct activity node representation;

(9-35) The outbound flow of a product for a DC must not exceed the flows for that product that pass through inventory, transhipment, and final assembly. Because end-items may be stocked after being assembled at the DC the constraint is formulated as an inequality;

(9-36) Provided that a product is kept in stock, the flow that passes through the inventory must be equal to the external supplies of that product plus the quantities of that product being assembled at the DC. If no inventory is kept the constraint should be invalidated. For this reason, the constraint is formulated as an inequality relationship. The 'M' indicates a value that must exceed the external supplies plus the quantities that pass through final assembly. Constraint (9-4) can be paraphrased as follows:

\[
\text{Final Assembly} + \text{Inbound Transport} \leq \text{Inventory} + M \cdot (1 - \text{Keep Inventory})
\]
CASE STUDIES AND EVALUATION

(9-37) Supplies can neither exceed a supplier’s maximum delivery capacity nor can (9-38) they be larger than the agreed-upon minimum purchase level;

(9-39) Customer demands must be met. This constraint actually pulls the flow of products through the network;

(9-40) The binary variable ‘Open T-Link’ indexed over \{s,dc\} indicates whether or not a transport link between a supplier and a DC is used for shipments of specific products as indicated by the variable ‘Use T-Link’. Constraint (9-40) ensures that the ‘Open T-Link’ variable is activated when the link is used for one or more products. The inclusion of this variable makes it possible to charge the fixed cost associated with the employment of a transport link only once;

(9-41) The product quantities that pass along a transport link may not exceed the maximum allowed throughput;

(9-42) see: (9-40);

(9-43) see: (9-41);

(9-44) see: (9-40);

(9-45) see: (9-41);

(9-46) The product quantities that are assembled at a DC must not exceed the maximum assembly throughput capacity. The FA capacity claim per unit denotes the claim on FA capacity per unit volume of products ‘prod’;

(9-47) The flow of products that passes through the inventory must not exceed the maximum inventory throughput capacity. The inventory claim per unit denotes the claim on storage capacity per unit volume of products ‘prod’;

(9-48) The flow of products that is transhipped must not exceed the maximum allowed transhipment throughput capacity. This constraint is activated once the ‘Keep Inventory’ variable for the product is set to zero. Keeping inventory and serving as transhipment centre are mutually exclusive options. Constraint (9-48) can be restated as follows:

Transhipment ≤ Max. Transhipment Throughput \times (1 - \text{Keep Inventory})\).
**Figure 9-16a** Block-schematic representation of the flow-aspect model.
### CASE STUDIES AND EVALUATION

#### Detailed Tables

<table>
<thead>
<tr>
<th>real</th>
<th>binary</th>
<th>activity levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport flow</td>
<td>Open T-link</td>
<td>(s, dc, prod)</td>
</tr>
<tr>
<td>-1</td>
<td>+1: Open (dc-f, dc-t, prod)</td>
<td></td>
</tr>
<tr>
<td>(s, dc, prod)</td>
<td>(dc-f, dc-t, prod)</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>(dc-c, prod)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>real</th>
<th>binary</th>
<th>activity levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport flow</td>
<td>Use T-link</td>
<td>(s, dc, prod)</td>
</tr>
<tr>
<td>+1: Open (dc-f, dc-t, prod)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s, dc, prod)</td>
<td>(dc-f, dc-t, prod)</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>(dc-c, prod)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>real</th>
<th>binary</th>
<th>activity levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport flow</td>
<td>Use T-link</td>
<td>(s, dc, prod)</td>
</tr>
<tr>
<td>+1: Open (dc-f, dc-t, prod)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s, dc, prod)</td>
<td>(dc-f, dc-t, prod)</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>(dc-c, prod)</td>
<td></td>
</tr>
</tbody>
</table>

| M | Process Inv. | (s, prod) |
| Maximum Ext. | Supply | (s, prod) |
| Minimum Ext. | Supply | (s, prod) |
| Customer demand | (c, prod) |
| M | Enable Use | (s, dc) |
| Throughput | (s, dc, prod) |
| Enable Use | (s, dc) |
| Throughput | (s, dc, prod) |
| Enable Use | (d, c) |
| Throughput | (d, c, prod) |
| Enable Use | (d, c) |
| Throughput | (d, c, prod) |
| Throughput | (d, c, asmby) |
| Max. Trs. | Troughput | (d, c, prod) |

### Objective

**Minimise**

**Fixed Transport cost**

| (d, c, prod) |

---

**Figure 9-16b** Block-schematic representation of the flow-aspect model.

---

277
PART III - IMPLEMENTATION & VALIDATION

LEAD-TIME ASPECT MODEL

Figure 9-17 depicts a block-schematic representation of the lead-time aspect model. Index symbol Product→ corresponds to index symbol prod that was used previously. The remaining index symbols correspond to those shown in Table 9-1. Next follows an explanation of the constraints displayed in Figure 9-17:

(9-49) The dispatch lead time of a DC at least equals the time that is needed to collect the products from stock and to prepare them for shipment: T(Inv.);

(9-50) If a DC assembles a product to order, thus without keeping the end-items in stock, in order to determine the dispatch lead time, the production lead time must then be accounted for: T(Assembly). However, if the products are kept in stock, this constraint is disabled;

(9-51) If a DC only serves as transhipment centre, then the DC's dispatch lead time is based upon the reception lead time of the products at the DC plus the time required to perform the transhipment operations: T(Trnshp.);

(9-52) The reception lead time of a product at a DC is equal to the order lead time of (9-53) its suppliers plus the time that is required to transport the products from the (9-54) supplier to the DC: T(Transp.);

(9-55) Customers can enforce a maximum order reception lead time (Max-RLT);

<table>
<thead>
<tr>
<th>binary</th>
<th>binary</th>
<th>real</th>
<th>real</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep Inventory?</td>
<td>Do Assembly?</td>
<td>Dispatch Lead time (DLT)</td>
<td>Reception Lead time (RLT)</td>
<td>Use T-link?</td>
</tr>
<tr>
<td>(dc.p)</td>
<td>(dc,asmbly)</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
<td>(dc,f,p)</td>
</tr>
<tr>
<td>-T(Inv.)</td>
<td>(dc,p)</td>
<td>+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+M</td>
<td>-T(Asmbly)</td>
<td>(dc,p)</td>
<td>+1</td>
<td></td>
</tr>
<tr>
<td>(dc,p)</td>
<td>(dc,asmbly)</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
<td>-1</td>
</tr>
<tr>
<td>+M</td>
<td>+M</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
</tr>
<tr>
<td>(dc,p)</td>
<td>(dc,asmbly)</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>(dc-f)</td>
<td>(dc-f)</td>
<td>T(Transp)</td>
</tr>
<tr>
<td>(dc)</td>
<td>(dc)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>(s)</td>
<td>(s)</td>
<td>T(Transp)</td>
</tr>
<tr>
<td>(c)</td>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
<td>(dc,p)</td>
<td>Max-RLT</td>
</tr>
<tr>
<td>(s,p)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

activity levels

| ≥ | 0 | (9-49) |
| ≥ | 0 | (9-50) |
| ≥ | T(Trnshp.) | (9-51) |
| ≥ | T(Transp) | T(Transp) | (9-52) |
| ≥ | (dc-f,dc,prod) | (9-53) |
| ≥ | T(Transp) | T(Transp) | (9-54) |
| ≥ | (dc,prod) | (9-55) |
| ≥ | Max-RLT | Max-RLT | (9-56) |
| ≥ | (dc,p) | Min-DLT | (9-57) |
| ≥ | (s,p) | | |

Figure 9-17 Block-schematic representation of the lead-time aspect model.
(9-56) DCs can enforce a maximum replenishment lead time (Max-RLT) for products kept in inventory;

(9-57) The minimum lead time for products ordered at a supplier.

**FREQUENCY-ASPECT MODEL**

Figure 9-18a-b depicts a block-schematic representation of the frequency-aspect model. Next follows an explanation of the constraints shown in Figure 9-18a-b:

(9-58) The dispatch frequency of a DC must be at least equal to the largest required reception frequency of any of its customers;

(9-59) The dispatch frequency offered by a supplier must not exceed the maximum attainable dispatch frequency for the supplier;

(9-60) The dispatch frequency of a DC or supplier for some product must be at least as large as the reception frequency for that product demanded by any of the DCs and/or customers being served (as indicated by the binary variable ‘Use T-link’);

(9-62) Constraint (9-62) translates the dispatch frequency that is required of a DC into a reception frequency which the DC demands from its suppliers. The reception frequency demanded by a DC equals the maximum dispatch frequency that is required from the DC, provided that the DC does not keep inventory of the product or uses the product as part of an assembled end-item;

(9-63) If the DC keeps inventory of a product, then the reception frequency which it demands from its suppliers is based upon the minimum inventory replenishment frequency for the product;

(9-64) The number of tours between suppliers and DCs as well as between DCs must be at least equal to the maximum reception frequency for the receiving party. Again, the Use T-link indicates whether there is a supply relationship between the two parties involved for that particular product;

(9-66) Vehicle capacity partly determines the least required number of tours. The abbreviation VCCU denotes the Vehicle Capacity Claim per Unit-quantity. This factor consists of the quotient of the volume per unit product and the vehicle capacity;

(9-69) The dispatch frequency which customers demand from the DCs by which they are being served must be at least equal to their required reception frequency:
(9-70) The total number of tours for both suppliers and DCs is derived from the (9-71) demanded dispatch frequency by multiplying the dispatch frequency by the number of addresses present in each customer zone (NAZ), divided by the number of addresses that is visited in one tour (NAT);
(9-72) Vehicle capacity also determines a lower bound on the total number of tours for (9-73) a supplier and a DC.
9.4.3 AC-graph representation

Figures 9-13 to 9-15 depict a compacted AC-graph representation of the flow-aspect model (cf. Figure 9-18). Figure 9-19 displays both the root graph and the Suppliers subgraph. Figure 9-20 shows the Distribution Network subgraph. Figure 9-21 contains the activities that relate to DCs. If applicable, the Entry Point (EP) index symbol is displayed as part of the title bar of the window. Table 9-2 contains a list of abbreviations which are used in the Distribution Network and DC subgraphs respectively.

<table>
<thead>
<tr>
<th>Abbrev. Explanation</th>
<th>Abbrev. Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAC</td>
<td>MxS Max. throughput transhipment</td>
</tr>
<tr>
<td>Fixed cost of installing assembly capacity</td>
<td>MxT Max. throughput transport link</td>
</tr>
<tr>
<td>FIC</td>
<td>UAC Unit Assembly Cost</td>
</tr>
<tr>
<td>Fixed cost of installing inventory capacity</td>
<td>UIC Unit Inventory Cost</td>
</tr>
<tr>
<td>FTC</td>
<td>USC Unit Transhipment Cost</td>
</tr>
<tr>
<td>Fixed cost of maintaining a transport link</td>
<td>USCC Unit claim on transhipm. capacity</td>
</tr>
<tr>
<td>Mxl Max. Inventory Throughput</td>
<td></td>
</tr>
<tr>
<td>MxA Max. Assembly Throughput</td>
<td></td>
</tr>
<tr>
<td>UACC Unit claim on assembly capacity</td>
<td></td>
</tr>
<tr>
<td>UICC Unit claim on inventory capacity</td>
<td></td>
</tr>
</tbody>
</table>

Table 9-2 List of abbreviations used in Figures 9-14 to 9-16.

[AC-graph - Deployment Model]

[AC-graph - Suppliers (s:Supplier)]

Figure 9-19 AC-graph: Flow-aspect model - Overall graph and Supplier graph.
Figure 9-20 Flow-aspect model: Distribution Network subgraph.
9.4.4 Data model

The database graph shown in Figure 9-22 provides an image of the structure of the data model that was used to support the AC-graph representation of the flow-aspect model. DAT-dictionary ‘Global Values’ contains entries for the various large numbers ‘M’.

Figure 9-22 Data model in support of the Deployment Model.
9.5 Evaluation

The evaluation of our experiences with ACOM with respect to the three cases presented is divided into three subsections. The first of these deals with the evaluation of the (MI)LP-modelling scheme, while the evaluation of the data modelling facility is the subject of Section 9.5.2. Section 9.5.3 contains our experiences with ACOM which relate to the technical features of the current implementation.

9.5.1 Evaluation of the (MI)LP-modelling scheme

CASE 2: THE COAL-CHAIN MODEL. The Coal-Chain model primarily deals with flows. Logical constraints play only a minor role, so the Coal-Chain model is an excellent case to use for evaluating the benefits of the flow-analogy that is inherent to an AC-graph based modelling scheme.

The use of ACOM for formulating the Coal-Chain model proved that the intended advantages of the proposed (MI)LP-modelling scheme appeared to exist. Inputting the model is an easy task. This is due to the following reasons:

1. The coal chain consists of a fixed number of consecutive stages: both the number and the relative order in which the various actors participate is fixed. Figure 9-10 shows that the compacted AC-graph provides an intuitively appealing representation of the coal chain. The transport activities connect the various actors in a most natural way;

2. The delivery, transport, and handling activities directly affect the flow of coal. The arcs of the AC-graph, therefore, naturally coincide with the physical flows of coal. This intuitive feature ensures that the formulation of the model requires very little effort;

3. The modular arrangement of the compacted AC-graph naturally coincides with the actors that participate in the coal chain;

4. The AC-graph representations of export terminals and import terminals are virtually identical. The EP index symbol constitutes the only difference between the two subgraphs: i.e. e-term and i-term, respectively. For instance, after completion of the AC-graph representation for the E-terminals, the same structure can be reused for I-terminals.

As observed in Section 9.3.3, the structure of the AC-graph representation of the Coal-Chain model differs from the block-schematic formulation depicted in Section 9.3.2. Unlike the block-schematic representation, the AC-graph representation contains the
"Post-Transfer Supply Chain Flow" activity. This activity equates to the flow between a transfer point and a power utility. As illustrated by Figure 9-23, three types of flow balance equations ensure that:

1. The post-transfer flow for each type of coal is equal to the delivered quantities at the point of transfer (cf. constraints 9-24 to 9-26);

2. All post-transfer flow for a particular utility equates with the minimum demand for that utility (cf. constraint 9-27);

3. The flow that is being transported between actors is equal to the post-transfer flow between the two sites (cf. constraints 9-28 to 9-30).

When using ACOM, flow-balance equations are most naturally modelled for separate actors (cf. Figure 9-24).

CASE 3: THE DEPLOYMENT MODEL. The structure of the DM is complex: it contains a large number of activities and constraints with many relationships among them. Moreover, many constraints represent logical relationships among activities using one
or more constraint ‘enabling’ or ‘disabling’ mechanisms which contain binary variables, like, for example:

\[ x_1 \geq x_2 \iff y = 0 \iff x_1 \geq x_2 - M \cdot y; \quad (9-74) \]

\[ x_1 \geq x_2 \iff y = 1 \iff x_1 \geq x_2 - M \cdot (1 - y); \quad (9-75) \]

where both \( x_1, x_2 \geq 0 \) and \( y \in \{0,1\} \) represent decision variables and \( M > y^2 \) denotes a sufficiently large number.

Although the impression which some people might experience when first seeing the compacted AC-graph of the flow-aspect model might suggest otherwise, our experience taught us that the creation of this graph is fairly easy. One procedure in particular appeared to be of great assistance when formulating and ‘reading’ the compacted AC-graph representation of logical constraints (constraints to which the flow analogy does not apply). This is that the direction of the arrows corresponds to the direction in which the activity levels of the source nodes, plus the autonomous constraint level, is passed on as either a lower or upper bound to the activities at the receiving end. If multiple activities constrain the activity at the receiving end, the total value of their activity levels serves as a bound. Similarly, if the constraint acts upon multiple activities, then the constraint serves as a bound on the total value of their activity levels. With this in mind, the representation of the logical constraints denoted in equation (9-74) and (9-75) is straightforward:

![Figure 9-25 Examples of logical constraints.](image)

The Distribution Network subgraph shows that the complex structure of the Deployment Model also makes the AC-graph representation look complicated. The large number of activities, constraints, and relationships yields rather crowded graphs. The disadvantage of this visual clutter is evident. It is more difficult to isolate pieces of the graph in order to understand the underlying mechanisms.
PART III - IMPLEMENTATION & VALIDATION

The generality of the building blocks of the modelling scheme is one source of complexity. The components of a compacted AC-graph have a one-to-one relationship with the elements of an equivalent referential representation. Therefore, in so far as the number of elements is a source of complexity, the modelling scheme preserves this cause of complexity. The possibility to create modular AC-graphs does not always provide sufficient means to keep the complexity of an AC-graph within practical limits.

It also appeared that the current implementation of the user interface of ACOM does not provide sufficient ways to organise the AC-graph in such a manner that the complexity of the graph is kept within reasonable proportions. For example, one approach would be to allow the model builder to create multiple views of a graph, each of which describes part of the model structure. An overall view would then present the entire model structure in keeping with the current implementation of the user interface, while separate views focus on complicated formulations or, in line with the presentation of the structure of the DM, different aspect models. The overall view may then be used to depict the interplay of various parts of the model structure while the separate views provide the user with the necessary detail and focus to discern the structure of basic mechanisms. Visually, different views may be grouped by using a notebook control as shown in Figure 7-17.

From our experience with the flow-aspect model, it became clear that the user-interface could be improved in a number of ways. Some of these have already been implemented so that some case studies could be continued, e.g., to enable the creation of large graphs, the user can drag the icons across the boundaries of the AC-graph window to enlarge the drawing area. Scroll bars allow the user to select the appropriate area of the rectangle that is occupied by the graph. Besides the addition of the option to create multiple views of an AC-graph, other possible improvements might include:

1. The possibility to depict links representing flows of commodities in a way that differs from other kinds of links;

2. Constant values for exogenous constraint levels should be part of the icon representing of the constraint node (cf. Figure 9-25). Currently, constant values for constraint levels are displayed as part of the 'settings' dialogue associated with constraint nodes.

9.5.2 The data-modelling scheme

In all three case studies, it required very little effort to put together the data model. This is true because of the following reasons:
1. The structure of the data models is less complex than the structure of the (MI)LP models;

2. The model builder can focus to a large extent on the logical structure of the data. The DAT-dictionary Navigator mechanism provides a flexible means for connecting the parameters of an (MI)LP model with the appropriate database entries, so, when there are several options to put together the data-model, the system can accommodate to whichever of these data models the user decides to implement;

3. The compacted AC-graph formalism supports, at least for aggregate levels, a modular 'actor-oriented' approach to modelling, i.e., the representation of the problem domain in terms of an AC-graph. The modular arrangement of an AC-graph facilitates the composition of a data model because the modular arrangement of both an AC-graph and an associated data model may largely parallel. Moreover, the object-oriented structure of the data models simply implies the existence of certain data attributes. Data attributes were added when the need for it developed which was only infrequently.

ACOM provides several options to view data models. We feel that these views provide the user with a clear image of the data. The clustering of logically related data makes it clear to the user where specific data entries can be found. The clustering of related data and the compatibility between the modular arrangement of AC-graphs and the associated data model makes the composition of database-path templates an easy and straightforward task when using the DAT-dictionary Navigator mechanism. The use of EP index symbols on AC-objects greatly improves the efficiency of this process.

9.5.3 Evaluation of the current implementation of ACOM

This section describes our experience with the current implementation of ACOM that need special attention in light of future releases.

THE USER INTERFACE

It is our experience that although it works intuitively, direct manipulation of objects that appear on the screen turns out to be efficient only for non-repetitive tasks: i.e., tasks that need to be executed only a small number of times. For instance, creating AC-graphs and DAT-dictionaries, consulting/editing the settings of an activity node, and dragging an (MI)LP-model instance to the icon of a Solver in order to solve the problem. The creation of DATs and the creation of large numbers of DAT-entries that comply to the format of a specific data-attribute template, however, typically require automated support.
PART III - IMPLEMENTATION & VALIDATION

THE DATABASE SUBSYSTEM

It appeared impossible to experiment with instances of the Deployment Model within reasonable time. Table 9-3 contains some empirical data that relates to the size of five different copies of the POET database subsystem which were obtained during our efforts to input the necessary numerical data. The columns of Table 9-3 deal with the following:

1. **T-Size**: The size of the POET database expressed in bytes. These figures denote the size of the ‘objects.dat’ file, not the index files. The size of an ‘empty’ database, i.e. a database which only contains the class structure but no objects, equals 56320 bytes;

2. **#DATs**: The number of DAT objects that is stored in the database;

3. **A-Size**: The size of the database that is attributed to data attributes (DAT entries);

4. **#Fields**: The number of data attributes expressed in terms of the number of key fields and data fields these data attributes contain. For example, a data attribute with 4 key fields, one fixed and three referential key fields, consists of five fields in total. The number of fields is used as an equivalent unit for analysis of the contents of the database copies;

5. **Avg. Size**: The average size per field expressed in bytes. This figure is not an exact measurement. POET allocates storage space in chunks of 5120 bytes a time. The difference between the size of two subsequent copies of the database therefore always equals a multiple of 5120 bytes. Hence, the storage space that was really needed may either be smaller or larger than this difference.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>158,720</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>II</td>
<td>184,320</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>III</td>
<td>225,280</td>
<td>24</td>
<td>40,960</td>
<td>93</td>
<td>460</td>
<td>89.0</td>
</tr>
<tr>
<td>IV</td>
<td>240,640</td>
<td>24</td>
<td>56,320</td>
<td>133</td>
<td>620</td>
<td>90.8</td>
</tr>
<tr>
<td>V</td>
<td>440,320</td>
<td>24</td>
<td>256,000</td>
<td>525</td>
<td>2580</td>
<td>99.2</td>
</tr>
</tbody>
</table>

Table 9-3 Size-related data on different copies of the POET database.

The last column of Table 9-3 shows that the average storage capacity per field is quite significant compared to what one would normally expect. Most key fields contain a reference to a DAT (8 bytes: POET assigns unique non-recycable 64 bit, 8 bytes, IDs to database objects) whereas the numeric fields either contain an integer or a real (4 bytes). In the unlikely case that POET has implemented container structures as double linked lists, then each field is associated with two additional object references (16 bytes). These figures leave about 66 (90-24) bytes, on average, for:
1. Data structures and object-references that link: (1) the key fields with the key object; (2) the key and the numeric field with the data attributes; and (3) the data attributes with the DAT object (cf. Figure 8-28). The storage capacity that is needed for these structures is used by the fields associated with the data attribute;

2. Database overhead: e.g., the object’s ID, class (type) information, index tables.

Additional experiments show that POET uses about 64 bytes to store key-field objects. Data-attribute ‘container’ structures which do not yet contain a key and a numeric field also requires about 64 bytes. With an average of four fields per attribute, and taking into account the storage capacity needed for object references to both the data attribute as a whole (as part of the DAT object) and to fields separately (as part of the data-attribute object), we obtain an estimated field size of 90 bytes which comes close to the figures shown in the last column of Table 9-3.

The last column of Table 9-3 shows that the average size per field slightly increases. This increase may be due to additional overhead structures needed to cope with the increasing demand for storage capacity.

Table 9-4 contains some of the same data as Table 9-3. In addition, Table 9-4 includes the time that was needed to add a data attribute comprising 4 key fields and a numeric field. Insertion times were measured manually, so an inaccuracy of 1s was accounted for. The columns next to those containing absolute values contain relative figures compared to the value for case III which was set to one.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>III</td>
<td>225,280</td>
<td>1.2</td>
<td>40,960</td>
<td>1.0</td>
<td>460</td>
<td>1.0</td>
<td>13-15</td>
<td>1.0-1.0</td>
</tr>
<tr>
<td>IV</td>
<td>240,640</td>
<td>1.3</td>
<td>56,320</td>
<td>1.4</td>
<td>620</td>
<td>1.3</td>
<td>15-17</td>
<td>1.1-1.4</td>
</tr>
<tr>
<td>V</td>
<td>440,320</td>
<td>2.4</td>
<td>256,000</td>
<td>6.3</td>
<td>2580</td>
<td>5.6</td>
<td>79-81</td>
<td>4.6-6.8</td>
</tr>
</tbody>
</table>

*Table 9-4 Time required to insert a single numeric data attribute with a four-fold key.*

Table 9-4 shows that the increase of the size of the database that is attributed to the storage of data attributes (‘A-Size’), the number of fields (‘#Fields’), and the ‘Insertion Time’ follow the same pattern. The time needed to insert the data attribute grows linearly with both the size of the database and the number of fields already contained. Hence, appending \( n_1 - n_0 \) fields to a database that contains \( n_0 \) fields already, requires a time of:

\[
\sum_{k=n_0}^{n_1} k \left( \frac{f_0}{n_0} \right) = \frac{f_0}{2n_0} (n_1 - n_0 + 1)(n_1 + n_0),
\]

(9-76)
PART III - IMPLEMENTATION & VALIDATION

where \( t_0 \) denotes the time (in seconds) needed to append the data attribute to a database comprising \( n_0 \) fields. Quadratic performance is generally considered efficient. However, as can be inferred from Table 9-4, database performance rapidly decreases below acceptable limits. Extrapolation of (9-44) yields an initial insertion time, starting with database II \((n_0 = 0)\), of approximately 0.03s per field which seems acceptable.
PART FOUR

Conclusions
10. CONCLUSIONS AND RECOMMENDATIONS

OUTLINE
This last chapter is divided into two sections. The first section recapitulates the main points of this thesis and presents the conclusions that follow from this research. The second section contains a survey of topics that will merit attention in a continuation of the present research.
10.1 Main points and conclusions

The first part of this thesis focuses on the availability and scope of (MI)LP models for strategic and tactical planning and the merits of existing (MI)LP-based tools for decision supporting purposes in these areas. To start by considering the introductory part of this thesis, we conclude that:

MODELS

- The literature contains a large number of (MI)LP models that focus on strategic and tactical manufacturing planning.

Models for strategic manufacturing planning focus on ‘structural’ decisions (cf. Section 2.1.3) — decisions that deal with the ‘bricks, mortar, and machinery’ [Skinner, 1978]. Four broad categories were distinguished: (1) facility location models; (2) allocation models; (3) capacity acquisition, replacement, and disposal models; and (4) technology selection models.

DSSs based upon facility location and/or allocation models have been successfully applied in practice. The literature on capacity acquisition, replacement, and disposal models and technology evaluation models is in such a state of rapid innovation that a consolidation into a few dominant models and paradigms does not seem to have occurred yet [Fine, 1993]. Existing models may, however, contribute to an improved understanding of the interplay of phenomena that affect the decisions being made in these areas. A potential route for the application of these models in the short term lies in the development of decision support systems that could be used by practitioners [Fine, 1993].

Practical models for tactical manufacturing planning separate aggregate planning decisions from detailed planning decisions. The hierarchical approach to aggregate production planning has proven to be a viable practical approach. Nevertheless, the information technology approaches that lead towards MRP(-II)-like systems seem to be most commonly applied in practice.

EXISTING (MI)LP-BASED DECISION SUPPORT SYSTEMS

- Existing (MI)LP-based DSSs for strategic production planning focus on facility location and/or allocation decisions.

- Using existing (MI)LP-based DSSs is basically a matter of specifying the values of a coherent set of input parameters in order to conduct experiments the objective of which is to search for solutions that perform well in a number of anticipated
settings. Users need to understand the mechanisms (trade-offs) that play a role in the underlying models. The mathematical formulation of the models remains hidden from the user.

- Much of the potential strength of (MI)LP-based DSSs stems from the design of the user interface:
  - The user interface can apply terms, concepts, and illustrations that are familiar to the intended group of users (decision makers) who are generally considered to be untrained management scientists. Unlike (MI)LP-modelling systems, (MI)LP-based DSSs do not use a representation of the underlying models as a basic means to communicate with the user.
  - The structure of the user interface may be based on a model of the decision-making processes that is supported by the DSS. The existence of this model is reflected in the choice of the models underlying the DSS, the data dependencies between these models, and the relative order in which the models may be applied. Such user interfaces can serve as a reference framework for the decision maker thus broadening the support from Simon's (1965) 'choice' phase, the traditional focus of management science, to the 'design' phase.

- An important disadvantage of existing (MI)LP-based DSSs is that the user cannot change the structure of the underlying models. This characteristic imposes a serious limitation on the applicability of these systems. The options that are available to customise the system must adhere to the basic structure of the model. A survey of facility location models by Brandeau and Chiu (1989), however, revealed that at least forty-five types of problems with ten different types of objective-functions have appeared in the literature. Hence, the options to tailor these systems are intrinsically limited.

- Another disadvantage of existing DSSs for strategic manufacturing planning is their narrow focus on facility location models. The implications of this property are threefold:
  1. Owing to their computational intractability, facility location models do not provide a detailed representation of the problem domain. For instance, it is customary to model variable costs of transportation as linear functions of the throughput of commodities via a transport link by using unit-cost measures based on the distance between sites. A more accurate basis for the variable costs of transportation, however, is the number of transport movements (trips) between sites [Mourits, 1995]. If transport costs constitute the predominant factor in the decision to locate new facilities, a more thorough evaluation of the costs of transportation will be essential. For that purpose, it will be necessary to use an additional model
PART IV - CONCLUSIONS

which will depart from the distribution of facilities proposed by the facility location model and which will focus on transport costs in more detail. The results obtained from such an evaluation might provide the necessary feedback to direct the facility location model.

2. Usually, as a result of a decision to establish one or more new manufacturing facilities, the pattern of commodity flows changes. New supply relationships emerge while existing ones cease to exist. Consequently, the implications of the decision to establish a new facility on the co-ordination of suppliers, facilities, and customers need to be evaluated. Questions that may arise include:

- How do existing supply characteristics (supply batches, order lead-times, supply frequencies) perform in the intended configuration of the production system?
- What is the effect on inventory levels?
- What capital outlays are necessary to modify supply performances?
- How do these modifications affect the costs of aggregate production plans?

A DSS for strategic production planning which supports the making of aggregate production plans, not as part of a routine tactical production planning process, but in order to evaluate the strategic proposal, could assist the decision maker to gain insight in the implications of strategic decisions on tactical levels.

3. Capacity acquisition, replacement, and disposal models and technology selection models are beyond the reach of the user of existing DSSs for strategic production planning. The above-cited opportunity [Fine, 1993] that these models offer is not considered.

EXISTING (MI)LP-MODELLING SYSTEMS

- The objective of declarative non-procedural (MI)LP-modelling systems is to express a problem in a way in which human modellers understand it rather than the form in which algorithms solve it [Fourer et al., 1990; Fourer, 1983]. The elements of the modelling schemes that these systems offer serve as metaphors of the components of the algebraic notation (i.e., variables, coefficients, and right-hand side constants), as metaphors of algebraic structures, or as metaphors of templates of algebraic structures.

- Chapter 3 contains a survey of declarative (MI)LP-modelling schemes. The decomposition of the model-development process into clusters of activities served as point of departure for the survey of (MI)LP-modelling systems. In Chapter 3, a number of criteria for comparison were presented. Criteria that directly relate to the
properties of an (MI)LP-modelling scheme are: (1) model-data independence; (2) conceptual modularity; (3) concreteness; (4) completeness; (5) generality; and (6) labour-intensiveness.

- Experienced model builders often prefer algebraic modelling languages because of their similarity with the algebraic notation that is customarily used to represent (MI)LP models. Algebraic modelling languages are non-intuitive: the physical structure of the underlying problem is not made explicit [Murphy et al., 1992b]. Inexperienced model builders prefer intuitive modelling schemes — modelling schemes which elaborate on analogies that relate more closely to the problem domain rather than to the algebraic notation. (MI)LP-modelling systems which offer an intuitive model representation either fail to offer complete generality or are not capable of representing all relationships between the components of a model.

In this research, it was decided not to base the intended system upon an (MI)LP-modelling scheme that offered limited generality. This approach guaranteed that the final system would, at least, feature an intrinsic capacity to deal with customised (MI)LP models. We also decided not to use a modelling scheme incapable of representing all relationships between the components of a model. On this point, we agree with Murphy, Stohr, and Asthana (1992) who observed that: "(...) users may feel uncomfortable about leaving things 'up to the computer' and may not obtain as detailed an understanding about the way the model components relate."

- Some (MI)LP-modelling systems that we encountered did not contain a model-independent database subsystem. For example, gLPS [Collaud et al., 1994] uses the representation of the structure of a model to store the data that are to be used in it. MPL only provides linkages with external databases. A disadvantage of the use of a DBMS as an interface to external databases instead of using a database subsystem is that it encourages the use of existing high-volume transaction-oriented databases which are shared by various applications. In the area of decision support systems, despite the obvious advantages of using existing databases (less redundancy, less maintenance, lower costs), the advantages of a database subsystem have long been recognised. In fact, a database subsystem is generally considered an indispensable part of any DSS. The principal advantages concern the possibility to focus on relevant parts of the data and to (re)structure the data conform the needs of the user of the DSS: i.e., to use a tailored datamodel.

- None of the (MI)LP-modelling systems that we encountered contains a model-independent data dictionary which enables users to model, maintain, and view the structural relationships between the elements of the database in agreement with the logical structure of the data. These structural relationships may consider referential integrity constraints or semantic rules at the record level.
PART IV - CONCLUSIONS

- Current (MI)LP-modelling systems typically offer the user an interface to the data that is in keeping with the relational data model which today's major database systems use as an external data model. The support for conceptual modularity is generally considered an essential property of (MI)LP-modelling schemes that focus primarily on inexperienced model builders because it reduces the complexity of creating and interpreting model structures. Precisely this notion of modularity, which is typical of the object-oriented data model where logically related pieces of data are grouped into data objects, is lacking in the relational approach to data modelling. Logically related data is scattered across the database in seemingly unrelated tables. If the (MI)LP-modelling scheme and the data-modelling scheme both applied the concept of conceptual modularity, the structure of the underlying problem domain would be reflected in the organisation of both the models and of the database. Because the organisation of both the models and the database would then be broadly identical, it would become possible to integrate the representation of a real-world entity in terms of the (MI)LP-modelling scheme and the data-modelling scheme in a highly intuitive manner. Such a tight coupling of both types of representations would improve both the degree of concreteness of the system and its manageability.

Part II describes the conceptual design of the Activity/Constraint - Object Modeller (ACOM). ACOM may be viewed as an attempt to obtain a synergy between the flexibility offered by general-purpose (MI)LP-modelling systems and the user-friendliness and decision-support functionality offered by (MI)LP-based DSSs. The proposed system is aimed at shortening the process of implementing flexible, intuitive, and tailor-made DSSs based on (MI)LP models. ACOM differs from existing (MI)LP-modelling systems in that it integrates an intuitive graphical (MI)LP-modelling language, a data-modelling language, and an intuitive model-data linking mechanism that integrates both and which all consistently support, and even enforce, a modular approach to modelling. Next follows a summary of the main points and the conclusions that can be drawn from the second part of this thesis:

(Ordered) REFERENTIAL NOTATION

- The algebraic notation that is customarily used to denote (MI)LP models is not suitable to depict the gradual development of (MI)LP models and related model instances using a modelling system that separates the formulation of the structural properties of a model from the specification of model instances.

- The referential notation and the ordered referential notation presented in Chapter 5 constitute two variants of the algebraic notation which comply with the
decomposition of the model-development trajectory described in Chapter 3. The referential notation constitutes the formal basis of the (MI)LP-modelling scheme that is contained in ACOM. With the exception of index symbol expressions, it remains hidden from the user of the system.

- The modular design of ACOM separates the AC-graph based (MI)LP-modelling scheme (the user’s form) from the internal representation of the models which is based on the referential notation. Consequently, it is relatively easy to supplement the system with alternative modelling schemes.

THE (MI)LP-MODELLING SCHEME: COMPACTED ACTIVITY/CONSTRAINT-GRAPHS

- The (MI)LP-modelling scheme that is presented in Chapter 5 is based on the Activity/Constraint (AC)-graph formalism [Schrage, 1987]. Most logistics planning models contain a significant number of activities that affect the flow of goods. AC-graphs provide an intuitively appealing representation of such models.

- The proposed modelling scheme differs from the AC-graph formalism in several major aspects:
  - Unlike AC-graphs, the components of the compacted AC-graph formalism represent indexed model parameters.
  - The proposed modelling scheme contains special icons for binary and integer decision variables.
  - Compacted AC-graphs may consist of a hierarchical arrangement of subgraphs containing logically related activities.

- Using the criteria for comparing (MI)LP-modelling schemes presented in Chapter 3, the evaluation of the compacted AC-graph formalism results in:
  - **MODEL-DATA INDEPENDENCE**: The MS separates the structural properties of (MI)LP models from the data that are specific to model instances.
  - **GENERALITY**: Compacted AC-graphs provide complete (MI)LP generality.
  - **LABOUR INTENSIVENESS**: The use of indexed parameters and compacted AC-subgraphs greatly reduces the effort that is needed to formulate models when compared to original AC-graphs. If the flow-analogy is appropriate models can be formulated more quickly.
  - **CONCRETENESS**: The integration of a subgraph containing logically related activities and an accompanying data abstraction yields a high level of concreteness.
PART IV - CONCLUSIONS

- **CONCEPTUAL MODULARITY**: The MS supports top-down decomposition and reuse of predefined subgraphs.

- **COMPLETEENESS**: The MS depicts all generalised relationships between the aggregate elements of the model explicitly. Logical relationships between instances of variables are depicted separately.

THE DATA-MODEL: DAT-DICTIONARIES AND DATS

- Just as the compacted AC-graph formalism separates the structural properties of an (MI)LP model from the numerical data of model instances, the proposed data model distinguishes between *data types* (DAT-dictionaries) and *data objects* (DATs). A DAT-dictionary specifies the identifying labels (the keys) and the data types of a set of logically related data attributes, whereas the associated DATs contain the actual data. DATs of a similar type may be aggregated to form homogeneous *classes* of DATs.

- The data-modelling scheme supports basic numerical data types, i.e. integers and reals, and referential data attributes. Referential data attributes make it possible to *navigate* through the conceptual model of the database, from one DAT-dictionary to another, and hence through the data itself, moving from one DAT to another.

- In Section 6.3.3, a mapping between the proposed data model and the relational data model is presented. This mapping may be used to design an interface between the proposed data model and the relational model in order to be able to exchange data with external database systems.

THE MODEL-DATA LINK: AC-OBJECTS

- To integrate the (MI)LP-modelling scheme and the data-modelling scheme, index sets are made up of homogeneous sets of DATs (i.e., DATs associated with the same DAT-dictionary). The values of the indices, i.e. the DAT identifiers, can be used to compose the key of a DAT entry and thus to establish a connection between the model parameters and corresponding database entries.

- AC-objects consist of a compacted AC-graph and an associated DAT-dictionary.

- AC-objects provide a physical connection between an AC-graph and a DAT-dictionary by fixing that part of the database connection which the components of the graph have in common. A brief explanation: An AC-object represents a class of real-world entities whose structural behaviour can be modelled by a single (MI)LP-model structure. As a consequence, the components of the compacted AC-graph — with the exception of shared constraint nodes — either relate exclusively to that.
CONCLUSIONS & RECOMMENDATIONS

particular entity or they represent an association between the entity and one or more other objects. The DAT-dictionary associated with the AC-object, therefore, contains either the data attribute for the parameter or a reference to a DAT-dictionary that is part of the association. In both cases, however, the components of the AC-graph have the first part of the database connection in common.

- AC-objects can be nested. The inclusion of an AC-object within an aggregate or ‘higher-level’ AC-object leads to an elaboration of the shared part of the database connection.

The third part of this thesis outlines the design and implementation of the current prototype of ACOM. Part III also presents three case studies. Next follows a summary of our main conclusions and findings which follow from our experiences with ACOM (cf. Section 9.5):

THE (MI)LP-MODELLING SCHEME

- MODULARITY. In the case of the Coal-Chain model, it appeared that when different types of actors were mutually connected by transport activities, the modular arrangement of the compacted AC-graph coincides with the connection between the actors that exists within the problem domain. In this particular case, it appeared obvious that this arrangement had to be applied in a top-down decomposition. For the Deployment Model this is only partly true. The way in which the Distribution Network could be subdivided into an overall graph and a Distribution Center subgraph only became clear after the Distribution Network graph had been completed. In that particular case we used a bottom-up formulation procedure. The prototype does not impose a particular working procedure: top-down and bottom-up formulation strategies may be mixed, as circumstances require.

- FLOW ANALOGY. Activities and constraints which act upon flows of commodities are represented in a most natural way. If the flow analogy is appropriate, the formulation of these constraints requires very little effort.

- LOGICAL CONSTRAINTS. The flow analogy does not apply to logical constraints. However, from our experiences with the Deployment Model we learnt that there is an easy ‘rule’ which facilitates the creation and interpretation of AC-graph structures that represent logical constraints.

THE DATA-MODELLING SCHEME

- In all three case studies, it required very little effort to create a data model. Primarily, this is due to the fact that for aggregate levels the compacted AC-graph
formalism supports an ‘actor-oriented’ approach to modelling and hence to the formulation of AC-graphs. The same high-level representation may be used to structure the data model the result of which is that the modular arrangement of the (MI)LP model and the data model broadly parallel.

- The clusturing of related data makes it intuitively clear where specific data should be located.
- The DAT-dictionary Navigator mechanism provides a natural and flexible means for establishing database connections.

THE USER INTERFACE

- The user interface does not currently provide enough means to organise AC-graphs in such ways that the complexity of the graph is kept within reasonable limits. Section 9.5.1 describes one approach to dealing with this deficiency which is to allow the model builder to create multiple views of a graph, each of which describes part of the model structure.
- Direct manipulation of objects that appear on the screen, although working intuitively, is only efficient in the case of non-repetitive tasks. Manipulating large numbers of data objects requires more efficient support.

THE DATABASE SUBSYSTEM

- The performance of the POET 2.0 database subsystem appeared to be intolerably slow.

CONCLUDING REMARKS
ACOM reduces the effort that is needed to implement tailored (MI)LP-based DSSs. The applicability of the system to models other than logistics planning models was not tested. Our experience, however, shows that the expressive power of compacted AC-graphs shows up best in models that embody a strong flow component.

First responses to brief demonstrations with earlier prototypes of ACOM supported the view that compacted AC-graphs provide an intuitively appealing representation of many models which may be used directly as a basic means of communication between model builders and their clients. ACOM may be used to short-circuit this dialogue, possibly by using rapid prototyping, thus contributing to these clients’ understanding and confidence.
ACOM may encourage untrained model builders to attempt to use (MI)LP-technology for decision making and analytical purposes. Finally, we expect that ACOM may prove a valuable tool for educating linear programming.

### 10.2 Topics for further research

If the present line of research is to be continued, the following topics should be given attention:

**MULTIPLE OBJECTIVES**

The addition of multi-objective capabilities to the compacted AC-graph formalism is straightforward. As illustrated in Figure 10-1, in order to represent multiple objective functions, the links representing the objective-function coefficients could be tagged with the identifier of the corresponding objective function.

![Figure 10-1 Multiple objectives.](image)

Besides the above-mentioned extension to the AC-graph syntax, the addition of multi-objective capabilities to the current implementation of ACOM requires the modification of the algorithms for producing output statements for external solvers. The production of output statements for solvers capable of dealing with multi-objective (MI)LP models is straightforward. In order to apply a single-objective (MI)LP solution algorithm, the model must first be reformulated as a single-objective (MI)LP model. Evidently, single-objective (MI)LP solution algorithms cannot discriminate between the progress towards each of the objectives.

**DEDICATED SOLUTION ALGORITHMS**

The current implementation of ACOM only supports ‘standard’ (MI)LP solvers which include general-purpose solution algorithms. These algorithms, unlike dedicated algorithms, do not exploit the special structure of a specific class of problems.
PART IV - CONCLUSIONS

Dedicated solution algorithms can deal with much larger problems by using special decomposition techniques or general rules of thumb (heuristics).

Heuristics often require only access to the data of a model instance. However, to solve a problem using a special decomposition technique, the structure of the problem must fit the structure of the class of problems for which the algorithm was intended.

One approach that may be used to support dedicated algorithms would be to link the algorithm to an AC-graph which represents the class of problems to which the algorithm applies. This approach, however, would be highly inflexible. Deviating AC-graphs, which can be mapped onto the structure of the problems to which the algorithm applies, would require a tailored interface with the algorithm. The same is true for the addition of minor extensions to the AC-graph associated with the algorithm, which the algorithm is capable of dealing with: e.g., adding capacity constraints to the incapacitated facility location problem.

Another approach would be to add information on the structural properties of the problem to which the algorithm applies to AC-graphs whose structure falls within the scope of the algorithm. For example, the activities and constraints of the problem type to which the algorithm applies could be tagged with labels. The same labels could then be used to tag the activities and constraints of AC-graphs within the scope of the algorithm. The interface between the AC-graph and the algorithm could then use these labels to convert the structure of the graph in the required format.

LINEAR APPROXIMATIONS

Practical optimisation problems frequently involve non-linear behaviour that must be taken into account: e.g., decreasing marginal profits due to market saturation, learning curves and price discounts which give rise to decreasing marginal costs, and higher costs of working overtime which increase the marginal costs of labour.

Non linear cost functions can be approximated by using piecewise linear functions. Piecewise linear approximations fit into the required format for MIP models. Only piecewise linear approximations to concave objective functions (decreasing slopes) which are used in maximisation problems and convex objective functions (increasing slopes) which are used in minimisation problems fit into the format of LP models.

It is difficult to model piecewise linear objective functions with the help of AC-graphs, since the formulation of piecewise linear functions is a mathematical procedure. Moreover, the formulation of a piecewise linear function should not depend on the
density of the grid for the variable\(^1\) (activity level) because this is not a structural property. To support the formulation of piecewise linear objective functions, the AC-graph formalism could be extended by adding a ‘macro’ type of link. As part of the process of producing output statements of model instances, this link would be automatically expanded to the necessary structures for representing a specific piecewise linear function. Clearly, the database would have to support a dedicated data type for specifying the grid and the slopes of piecewise linear functions.

**PRE-SOLVER PROCESSING**
In Chapter 5, it was shown that an intuitive representation of the inventory-balance constraint (cf. equation 5-6) required an extra activity node and an extra flow-balance constraint node. The transcription of the ‘intuitive’ AC-graph yielded two constraints which could be reduced to the single inventory-balance constraint denoted in equation 5-6. From this example, it appears that the algebraic transcription, from which ACOM generates MPS files, leaves room for improvement. Removing superfluous activities and constraints prior to invoking the solver could help to increase performance.

**INHERITANCE**
Currently, the proposed data-modelling scheme does not support the creation of basic and derived types. ‘Inheritance’ of the data attributes of a basic DAT-dictionary by a derived type could improve the expression capabilities of the data model.

**SUMMED RHS-CONSTANTS**
Consider the case where ACOM is configured with both an annual and a monthly production planning model. The annual planning model contains a constraint for the annual demand of a customer. Similarly, the monthly model contains a constraint representing the customer’s monthly demand. With the current version of ACOM, it would be necessary to use two distinct entries (partly redundant) on the DAT-dictionary for a customer to model both types of demand:

```plaintext
DAT-dictionary "Month","Product":
DAT-dictionary "Customer"
{
  "annual demand","Product" : real:
  "monthly demand","Product","Month" : real:
}
```

\(^1\) The grid for the variable determines the number of additional variables (one for each line segment).
PART IV - CONCLUSIONS

The AC-graph formalism does not offer the possibility to define RHS pieces comprising summations of RHS-constants like, for example:

\[ \text{Customer} \rightarrow c, \text{Product} \rightarrow p : \ldots \leq \sum \{ \text{Month} \rightarrow m \} \ demand \{ c, p, m \}. \ (*\text{annual demand}*) \]

The algebraic-oriented modelling scheme that is part of the current implementation of ACOM already supports the creation of RHS pieces comprising summations of RHS constants.
REFERENCES


Barakat, Y., Eindverslag Werken met Produktie Planning Ondersteunend Systeem, Stageverslag Opleiding Informatica en Informatiekunde, Haage Hogeschool, Sector Informatica, 1995


Benjaafar, S., Morin, T.L., Talavage, J.I., The strategic value of flexibility in sequential decision making, Working Paper, University of Minnesota, Minneapolis, 1993


Bishop, J., Paragon Decision Technology, P.O. Box 3277, 2001 DG, Haarlem, The Netherlands


Codd, E.F., *Data Models in Database Management*, IBM Research Laboratory, San Jose, California, 1980


Cplex Optimization, *Cplex version 1.2 Documentation*, CPLEX Optimization, Inc., Houston, TX 77095, 1990


REFERENCES


Hax, A.C., Meal, H.C., "Hierarchical Integration of Production Planning and Scheduling", *Studies in Management Science*, 1, 1975, 53-69


REFERENCES


Kraan, E.A.F., Evers, J.J.M., "Modelling with AC-Objects", 1995, conditionally accepted for publication by Decision Support Systems


Lootsma, F.A., Extensions of Linear Programming, Lecture Notes A85C, 1986


Murphy, F.H., Stohr, E.A., and Ma, P-C., “Composition rules for building linear programming models from component models”, Management Science, 38, 7, 1992, 948-963


Numetrix, Decision Support for Supply Chain Management, Numetrix NV-SA, Brussels, Belgium, 1995


Oliff, M.D., Burch, E.E., “Multiproduct Production Scheduling at Owens-Corning Fiberglas”, Interfaces, 15, 5, September-October 1985, 24-35


Paragon Decision Technology, P.O. Box 3277, 2001 DG, Haarlem, The Netherlands

POET Software, POET programmer’s and reference guide, Release 2.0, POET Software GmbH., Hamburg, Germany, 1993

POET Software, POET programmer’s and reference guide, Release 2.1, POET Software GmbH., Hamburg, Germany, 1995


Savenije, R., Het Optimaliseren van de Steenkolenketen, Delft University of Technology, 1992 (in Dutch)
REFERENCES


Shapiro, J.F., Mathematical Programming Models and Methods for Production Planning and Scheduling, in: Handbooks in OR & MS, Ch. 8, Vol. 4, Graves, S., Rinnooy Kan, A.H.G., Zipkin, Ph.H., (Eds.), 1993, Elsevier Scientific Publishers


Sijbrands, M., Strategische en Logistieke Besluitvorming: Een empirisch onderzoek naar informatiegebruik en instrumentele ondersteuning, Thesis Vrije Universiteit, Amsterdam, 1993 (in Dutch)


Star Division, StarView C++ Klassenbibliothek, v2.0, Benutzer-handbuch (in german), Star Division Softwareentwicklung und Vertrieb, Hamburg, Germany, 1993


Wouterse, K., *The Design of a Management Supporting System for Distributed Production Planning*, Delft University of Technology, Delft, 1990

Appendix A: (MIXED INTEGER) LINEAR PROGRAMMING

This appendix introduces the basics of Mixed-Integer and Linear Programming (MI)LP models, the simplex method, duality and the application of duality to sensitivity analysis.

A.1 LP basics and vocabulary

Equation A-1 shows a standard form of an LP model. The $x_j$ are decision variables. The constraint coefficients $a_{i,j}$, the Right-Hand Side (RHS) constants $b_i$, and the objective-function coefficients $c_j$ may be referred to as the input parameters of the LP. $I$ and $J$ denote index sets. Let $I = \{1,\ldots,n\}$ and $J = \{1,\ldots,n\}$. Then $A$ represents an $(m \times n)$ matrix; $c$ and $x$ represent $n$-vectors whereas $b$ denotes an $m$-vector.

$$\begin{align*}
\text{Max. } Z &= \sum_{j \in J} c_j x_j \\
(P): \quad \text{s.t. } &\sum_{j \in J} a_{i,j} x_j \leq b_i; \quad i \in I \\
x_j &\geq 0; \quad j \in J
\end{align*}$$

(A-1)

$$\begin{align*}
\text{Max. } Z &= c^T x \\
\text{s.t. } &Ax \leq b, \\
x &\geq 0.
\end{align*}$$

Equation A-1 defines a set of $m+n$ inequality relationships, usually referred to as restrictions or constraints. The first $m$ restrictions describe the structural properties of the LP and are sometimes called functional constraints. The $x_j \geq 0$ are called nonnegativity constraints.

Each of the constraints depicted in equation A-1 defines a $n$-dimensional half space. The intersection of all half spaces defines the feasible region. Any $n$-vector $x$ satisfying all $m+n$ inequality relationships is called a feasible solution. Each half space is bounded by a $n$-dimensional plane: the hyperplane. The hyperplane's algebraic formulation, the boundary equation, can be obtained by converting the functional constraint into an equality.

![Figure A-1 2-Dimensional and 3-dimensional hyperplanes.](image-url)
A.2 The simplex method

A corner-point feasible solution is a feasible solution that satisfies \(n\) constraint boundary equations simultaneously. As there are only \((m+n)\) constraint boundary equations, the number of different subsets of \(n\) equations out of a total of \((m+n)\) equals:

\[
\frac{(m+n)!}{m!n!}
\]  \hspace{1cm} (A-2)

Although not every subset of \(n\) constraint boundary equations yields a feasible corner-point solution, if the LP has a unique optimal solution it is a feasible corner-point solution. If there are multiple optimal solutions to the problem, then at least two of them must be adjacent corner-point feasible solutions: i.e. the line segment connecting them lies on an edge of the feasible region.

**OUTLINE OF THE ALGORITHM**

The simplex method is a commonly applied procedure for solving LP models. The simplex method searches the finite set of corner-point feasible solutions until it reaches an optimal solution.

The simplex method deals with sets of equality relationships. Therefore, with the exception of the nonnegativity constraints, the inequality constraints depicted in equation A-1 must be converted into equivalent equations. This is done by introducing nonnegative slack variables: \(x_{n+i}\). After introducing slack variables, the functional constraints can be denoted as follows:

\[
\sum_{j \in J} a_{i,j} x_j + x_{n+i} = b_i; \ i \in I
\]  \hspace{1cm} (A-3)

For each of the functional constraints depicted in equation A-1 the values of the corresponding slack variables depicted in equation A-3 indicate whether or not the constraint serves as one of the \(n\) ‘active’ constraint boundary equations of which the intersection defines the optimal feasible corner-point solution. If the slack variable equals zero, then the associated constraint is active. Similarly, if the value of a decision variable equals zero, then its associated nonnegativity constraint is active for the current optimal solution.

Equation A-3 consists of \(m\) equations. Because there are \(n+m\) decision variables, we may assign arbitrary values to \(n\) out of the \(n+m\) decision variables while the remaining variables still solve the system of \(m\) equations. Customarily, these decision
variables are set to zero and referred to as nonbasic variables. The other \( m \) variables are called basic variables. Basic variables do not necessarily have nonzero values which is the case where a feasible corner-point solution satisfies more than \( n \) constraint boundary equations (redundancy). Therefore, it is necessary to keep track of the set of nonbasic variables rather than relying upon their zero values.

A basic solution has \( n \) nonbasic variables set to zero. The remaining \( m \) basic variables are the simultaneous solution of the system described in equation A-1. A basic feasible solution is a basic solution with all basic variables being nonnegative.

The simplex method starts with a basic feasible solution. In subsequent iterations it moves to better adjacent basic feasible solutions until it reaches an optimal solution. Note that two adjacent basic feasible solutions both lie on the same edge of the feasible region. This edge is in fact a line which, in an \( n \)-dimensional space, results from the intersection of \( (n-1) \) constraint boundary equations. Adjacent feasible basic solutions therefore share \( (n-1) \) constraint boundaries. To move towards a better adjacent basic feasible solution, the simplex method selects the most promising nonbasic variable. By increasing the variable’s activity level from zero (by which it becomes a basic variable) the constraint associated with the variable is inactivated. Hence, the simplex method moves in a feasible direction which is determined by the intersection of the remaining \( (n-1) \) constraint boundaries. Just how much the variable can be increased depends on how quickly the simplex method reaches the adjacent feasible corner point solution. By moving in this direction, one basic variable gradually reaches zero. This basic variable can be identified easily by rewriting the basic variables in terms of the most promising nonbasic variable. The basic variable that imposes the smallest upper bound on the increase of the nonbasic variable is the one whose constraint boundary equation is the first to become active when the variable reaches zero. If the simplex method has reached a corner-point feasible solution that outperforms all of its adjacent corner-point feasible solutions (as measured by \( Z \)), then it scores better than all other corner-point feasible solutions which means that it is optimal.

A.3 Duality

This section outlines the relationships between the primal and dual formulation of an LP problem, the interpretation of the dual problem, and the applications of duality.

Equation A-1 depicts a standard form of an LP model. This form is sometimes referred to as the primal formulation to distinguish between the dual formulation of the LP that is exhibited in equation A-4.
APPENDIX A

\[
\begin{align*}
\min \, W &= \sum_{i \in I} u_i b_i \\
\text{s.t.} \quad \sum_{i \in I} a_{i,j} u_i &\geq c_j, \quad j \in J \\
u_i &\geq 0
\end{align*}
\]

\[
\begin{align*}
\min \, W &= u^T b \\
\text{s.t.} \quad A^T u &\geq c, \\
u &\geq 0.
\end{align*}
\]

(A-4)

The primal problem as depicted in equation A-1, and its associate dual problem, shown in equation A-4, are *not* different representations of the same problem. The use of the dual problem stems from the following two properties:

1. The primal LP has an optimal feasible solution only if the dual problem has an optimal feasible solution;
2. The optimal objective-function values of the primal problem and the dual LP are equal.

**INTERPRETATION OF THE DUAL PROBLEM**

The primal formulation in the form of equation A-1 is usually conceived as maximising a profitability measure \( Z \) under the limited availability of the necessary resources. The constraint coefficients associated with an activity are thought of as the resource requirements for undertaking a unit-quantity of that particular activity. The functional primal constraints ensure that the consumption of resources by the activities at the present levels does not exceed the total supply of resources.

The dual problem focuses on the deployment of resources. The dual-optimal \( \bar{u}_i \) \((i \in I)\) can be interpreted as the *current contribution to profit* per unit of resource \( i \) by having an amount of \( b_i \) of resource \( i \) available for the primal problem. In other words: the \( \bar{u}_i \) indicate how much the value of the objective function changes with a unit change of the associate RHS-constant provided that the current optimal solution remains feasible (cf. equation A-5).

\[
Z^* = c^T \bar{x} = \bar{u}^T b, \quad \text{so,} \quad \frac{dZ^*}{db_i} = \bar{u}_i.
\]

(A-5)

The nonnegativity requirement on the \( \bar{u}_i \) follows immediately from this interpretation: resources whose average unit costs exceed their potential contribution to profits need not be considered.

If the primal LP is a maximisation problem, the dual model is a minimisation problem. The objective of the dual problem can be viewed as minimising the total value of the resources that are consumed to obtain the profits.
RELATIONSHIPS BETWEEN THE PRIMAL AND DUAL PROBLEM

Following Lootsma (1986), we derive the relationships between the solutions of the primal and dual problems. Assume that an optimal solution \( x^* \) for the primal LP (cf. equation A-1) exists. Then, for any feasible solution \( x, \ y = x - \bar{x} \) is a feasible direction relative to \( \bar{x} \). Since no feasible direction can improve the value of the objective function at \( \bar{x} \), we have:

\[
c^T y = c^T (x - \bar{x}) = c^T x - c^T \bar{x} \leq 0,
\]

Moreover, if we denote the \( i^{th} \) row of the coefficient matrix \( a^i \), then for any active functional constraint \( a^i \) at \( \bar{x} \) and for any active nonnegativity constraint \( e^j \) at \( \bar{x} \) we have:

\[
\begin{align*}
(a^i)^T y &\leq 0, \quad \text{if} \quad (a^i)^T \bar{x} = b_i, \\
(e^j)^T y &\geq 0, \quad \text{if} \quad (e^j)^T \bar{x} = 0.
\end{align*}
\]

Farkas' theorem (cf. the parenthesis) states that if, and only if, for any \( y \) satisfying \( Qy \geq 0 \), the inner-product of \( p \) and \( y \) is nonnegative, i.e. \( p^T y \geq 0 \) then \( p \) is a nonnegative linear combination of the rows of \( Q \), i.e. there exists a vector \( u \geq 0 \) such that \( p = Q^T u \). If we let \( p = -c \) and the rows of \( Q \) equal \( (-a^i)^T \) and \( (e^j)^T \) corresponding to the active constraints at \( \bar{x} \), then there exist nonnegative coefficients \( \bar{u} \) and \( \bar{v} \) such that:

\[
c = -\sum_i \bar{u}_i a^i + \sum_j \bar{v}_j e^j.
\]

If we force \( \bar{u}_i = 0 \) and \( \bar{v}_j = 0 \) for the inactive constraints by stipulating that:

\[
\bar{u}_i \left[ \sum_i a_{i,j} \bar{x}_j - b_i \right] = 0, \quad i \in I, \quad \text{and} \quad \bar{v}_j \bar{x}_j = 0, \quad j \in J,
\]

the necessary and sufficient conditions for \( \bar{x} \) to be a maximum solution to the primal problem can be reformulated to:

\[
-c = -A^T \bar{u} + \bar{v}, \quad \text{with} \ \bar{u} \geq 0, \ \bar{v} \geq 0,
\]

where:

\[
\begin{align*}
\bar{u}^T (A \bar{x} - b) &= 0, \\
\bar{v}^T \bar{x} &= 0.
\end{align*}
\]
If we eliminate vector $\tilde{v}$ we obtain necessary and sufficient conditions for $\tilde{x}$ to be maximum solution to the primal LP: there must exist an $m$-vector $\tilde{u}$ such that:

\begin{align}
A^T\tilde{u} & \geq c, \quad \text{(A-12)} \\
\tilde{u} & \geq 0, \quad \text{(A-13)} \\
\tilde{u}^T(A\tilde{x} - b) & = 0, \quad \text{(A-14)} \\
\tilde{x}^T(c - A^T\tilde{u}) & = 0. \quad \text{(A-15)}
\end{align}

Equation A-12 and A-13 are called the dual constraints. Equations A-14 and A-15 are called the complementary slackness relations. To show that $\tilde{x}$ maximises the primal problem and $\tilde{u}$ minimises the dual constraints (as measured by $b$), we use the complementary slackness relations to obtain:

\begin{equation}
c^T\tilde{x} = \tilde{u}^T A\tilde{x} = \tilde{u}^T b. \quad \text{(A-16)}
\end{equation}

And thus for any primal-feasible $x$ we have that $\tilde{x}$ maximises the primal problem:

\begin{equation}
c^T x \leq \tilde{u}^T Ax \leq \tilde{u}^T b. \quad \text{(A-17)}
\end{equation}

Similarly we find that $\tilde{u}^T b$ minimises the dual problem since for any primal-feasible $x$ we have that $x$ measured by $c$ provides a lower bound for any dual feasible $u$ measured by $b$:

\begin{equation}
c^T x \leq u^T Ax \leq u^T b. \quad \text{(A-18)}
\end{equation}

\textbf{Parenthesis} Farkas’ theorem

Farkas’ theorem states that only if for any $y$ satisfying $Qy \geq 0$, the inner-product of $p$ and $y$ is nonnegative, i.e. $p^T y \geq 0$ then $p$ is a nonnegative linear combination of the rows of $Q$, i.e. there exists a vector $u \geq 0$ such that $p = Q^T u$. This theorem can also be formulated as a theorem of alternatives since $(A \Leftrightarrow B) \Leftrightarrow A \lor B$: either the system $p^T y < 0$, $Qy \geq 0$ or the system $p = Q^T u$, $u \geq 0$ has a solution, but never both.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{Graphical illustration of the Theorem of Alternatives.}
\end{figure}

320
APPLICATIONS OF DUALITY

The dual problem is primarily used for sensitivity analysis (cf. section A.4). A second advantage of the dual problem arises when the dual problem has fewer functional constraints than the primal problem \((m > n)\). In that case, solving the dual instead of the primal problem reduces the time that is needed to solve the problem\(^1\).

A.4 Sensitivity analysis

The purpose of sensitivity analysis is to trace how changes of the values of input parameters affect the current optimal solution to the primal LP. Does changing the values of the \(c_p, a_{ij}, \text{ or } b_i\) have a major effect on the activity levels of the current compound optimal solution or does the current solution become infeasible thus requiring that the model should be resolved?

CHANGING RIGHT-HAND SIDE CONSTANTS

If the value of a RHS-constant \(b_i\) is changed to \(b_i + \Delta b_i\), then the feasibility of the current optimal solution \(\bar{x}\) to the primal problem \(P(b_i)\) might be affected. If \(\bar{x}\) remains a feasible solution to the primal problem \(P(b_i + \Delta b_i)\), then the set of basic variables remains unchanged and so does the optimal solution to the dual problem: \(\bar{u}\). (Note that the solution to the dual problem may change only if either the objective-function of the primal problem changes or its set of active constraints changes.) The new value of the objective function can then be calculated by using the dual variables:

\[
Z^* (b + \Delta b) = Z^* (b) + \bar{u}^T \Delta b = Z^* (b) + \bar{u}_i \Delta b_i. \tag{A-19}
\]

If the current optimal solution to the primal problem turns out to be infeasible then, in order to find the new optimal solution, the primal problem has to be reoptimised. In that case, the dual simplex method can be used. Unlike the ‘primal’ simplex method, which iterates through primal-feasible suboptimal solutions until it finds an optimal solution, the dual simplex method seeks for a primal-feasible solution among solutions whose associate dual solution satisfies the dual constraints. The non-optimal primal solutions generated by the dual simplex method do not satisfy the constraints of the primal LP. Therefore, the current optimal solution can be used as an advanced starting point for this algorithm.

\(^1\) LP solution times using the simplex method are proportional to the number of variables. Solution times are proportional to the cube of the number of functional constraints \(O(m^3)\). The computation time per iteration depends heavily on the proportion of coefficients that is not zero. The number of iterations is roughly twice the number of functional constraints [Hillier et al., 1989,p92].
APPENDIX A

Figure A-3a shows a typical result of the effect of gradually increasing a RHS-constant $b_i$ on both the objective function value and the dual price $\bar{u}_i$ associated with the constraint. If the primal problem is a maximisation problem, the curve of the objective-function $Z^*$ value has the following characteristics:

- The curve is a piece-wise linear and continuous function;
- The curve is concave (i.e. its gradient is non-increasing);
- The slope of the curve corresponds to the dual variable associated with the constraint;
- The breakpoints correspond to changes in the set of basic variables in the optimal feasible solution to the primal problem.

To show that the curve is concave, we show that the slope of the curve is non-increasing:

$$\frac{d\bar{u}_i}{db_i} \leq 0. \quad (A-20)$$

If we increase $b_i$ from $b_i^1$ to $b_i^2$, then the optimal solution to the dual problem corresponding with $b = b_i^1$, $\bar{u}_i^1$ remains feasible. The question is whether $\bar{u}_i^1$ is still optimal. What we know is that, because $\bar{u}_i^1$ is also a feasible solution to the dual problem with $b = b_i^2$, the dual-optimal solution for $b = b_i^2$, $\bar{u}_i^2$ must score at least as good as $\bar{u}_i^1$: $\bar{u}_i^2 b_2 \leq \bar{u}_i^1 b_1$. As $b_i^2 > b_i^1$, it follows that: $\bar{u}_i^2 \leq \bar{u}_i^1$.

The simplex method requires only minor adjustments to trace the sequence of breakpoints of the curve [Schrage,1987]. The amount of extra work is roughly equivalent to a single iteration of the simplex algorithm.

CHANGING OBJECTIVE-FUNCTION COEFFICIENTS

Changing the values of objective-function coefficients does not affect the feasibility of the current optimal solution. The question that remains is whether the current solution

![Figure A-3a and Figure A-3b Parametric curves.](image-url)
\( \bar{x} \) is still optimal. It suffices to check the feasibility of the current optimal solution to the dual problem. If the current optimal solution to the dual problem still satisfies the dual constraints, \( \bar{x} \) maintains its optimality. If not, the model has to be reoptimised using the current basic feasible solution as a starting point.

Figure A-3b shows a typical plot of the effect of gradually increasing an objective-function coefficient \( c_j \) on both the objective function value \( Z^* \) and the level of the corresponding activity. If the primal problem is a maximisation problem, the curve has the following properties:

- The curve is a piece-wise linear and continuous function;
- The slope of the curve corresponds to the level of the corresponding activity;
- The curve is convex (i.e. its gradient is nondecreasing): it pays to increase the activity level if the corresponding objective-function coefficient is increased;
- The breakpoints correspond to changes in the set of basic variables in the optimal feasible solution to the primal problem.

To show that the curve is concave, we show that the slope of the curve is non-decreasing:

\[
\frac{d\bar{x}_j}{dc_j} \geq 0. \tag{A-21}
\]

Let \( \bar{x}_1 \) be the optimal solution to the primal problem \( P(c^1_j) \) and let \( \bar{x}_2 \) be the optimal solution to the primal problem \( P(c^2_j) \), with \( c^1_j < c^2_j \). Since we know that both \( \bar{x}_1 \) and \( \bar{x}_2 \) are basic feasible solutions to \( P(c^1_j) \) and \( P(c^2_j) \), we can obtain the following:

\[
(c^1_j)^T \bar{x}_1 \geq (c^1_j)^T \bar{x}_2 \iff (c^1_j)^T (\bar{x}_1 - \bar{x}_2) \geq 0, \quad \text{and} \tag{A-22}
\]

\[
(c^2_j)^T \bar{x}_2 \geq (c^2_j)^T \bar{x}_1 \iff (c^2_j)^T (\bar{x}_2 - \bar{x}_1) \geq 0. \tag{A-23}
\]

So, by adding both inequalities, we proof that the curve must be convex as its slope increases for increasing \( c \):

\[
(c^1_j)^T (\bar{x}_1 - \bar{x}_2) + (c^2_j)^T (\bar{x}_2 - \bar{x}_1) \geq 0 \iff \left( (c^2_j)^T - (c^1_j)^T \right)^T (\bar{x}_2 - \bar{x}_1) \geq 0. \tag{A-24}
\]

**CHANGING CONSTRAINT COEFFICIENTS**

To investigate the effect of a change of the value of a constraint coefficient \( a_{ij} \) we must distinguish between basic and nonbasic variables (having zero values). Changing the
value of a constraint coefficient associated with a nonbasic variable does not affect the feasibility of the current optimal solution. It suffices to check the dual feasibility of the current optimal solution to determine whether the current optimal solution is still optimal. If not, the current optimal solution can be used as an advanced starting point for the primal simplex method. Changing the constraint coefficients of basic variables generally makes the current solution infeasible. If not, simply apply the same procedure as for changing the coefficient of a non-basic variable. If the current optimal solution has become infeasible, it can be used as an advanced starting point for the dual simplex method (cf. Changing RHS-constants).

**Introducing Additional Decision Variables**

Introducing new variables requires the addition of appropriate coefficients to the functional constraints and the objective function. The dual problem changes less radical. In fact, the only change to the dual problem is a new constraint (the $u_i$'s define a linear combination of the rows of $A$). The question then arises if the current optimal solution to the primal problem expanded with the new nonbasic variable — having its values set to zero — would still be a feasible primal solution? Using the dual problem, this can be answered quickly by investigating whether the optimal solution to the original dual problem still satisfies the newly added constraint. If so, the current optimal solution to the primal problem remains optimal as is.

**Adding Extra Constraints**

Adding additional constraints to an already solved problem may affect the feasibility of the current optimal solution. If the solution is still feasible, it remains optimal — the additional constraints only reduce the number of feasible solutions, they do not add any. If the current optimal solution is not feasible, then the current basic solution can be used as an advanced starting point for the dual simplex method.

### A.5 Mixed-integer programming

Equation A-10 depicts a standard algebraic form of a mixed-integer program where the integer variables are required to be zero or one only. This type of problem is referred to as either a zero-one program or a binary program.

Max. $Z = \sum_{j \in J} c_j x_j$

**MIP:**

$s.t. \sum_{j \in J} a_{i,j} x_j \leq b_i$ for $i \in I$

$x_j \geq 0$ for $j \in N = \{1, \ldots, n\}$

$x_j \in \{0,1\}$ for $j \in I \subseteq N$.  

\[(A-25)\]
SOLUTION METHODS

Usually, the values corresponding to the integer variables in the optimal solution to the LP obtained by relaxing the integrality requirements are not integral. To identify integral optimal solution(s), MIP algorithms systematically enumerate all possible combinations of values for the integer variables.

The branch and bound strategy systematically fixes the values of subsets of the integer variables. Assuming that the objective is to be maximised, the branch and bound algorithm determines an upper bound for each subset and the fixed values which applies to all solution with the variables fixed at those values. If the algorithm has already found a lower bound for all solutions that exceeds the upper bound on the optimal solution for the current subset, then all solutions with the given fixed values can be ignored. Subsets can also be excluded (also referred to as: fathomed) from further consideration if the subset has no feasible solutions or if its best feasible solution has been found. A lower bound is determined by the value of the objective function for the best feasible solution to the original MIP identified thus far. The integer solution corresponding to the best lower bound encountered in the search is the optimum integer solution. An upper bound on the value of the optimal solution to the original MIP is determined by the optimal solution to the LP relaxation. The systematic enumeration of solutions is accomplished by fixing variables and adding constraints to obtain disjoint subregions, then recursively applying the same bounding rule to each subproblem. Eventually all integer variables will be fixed or the LP solution will be integral.

To obtain disjoint subregions, bounds can be incorporated on an integer variable with a fractional value in the relaxation, forcing the value to the next higher or next lower integer value to create two new subproblems.

The performance of the branch and bound method depends on the tightness of the upper and lower bounds, the separation strategy for obtaining disjoint sub problems, and the branch rule for selecting a new sub problem from those that are still live. The lower bounds of the LP relaxation can be tightened by adding so-called cutting planes to the constraint set which cut off feasible solutions to the LP relaxation but no integer solutions. The introduction of strong cutting planes is the most important recent development in MIP/IP technology [Saltzman,1994]. The branching strategy can have a dramatic impact on the time required to solve the model. Unfortunately, it is not possible to tell in advance what will work well for a particular problem.
A.6 Sensitivity analysis for MIP- models

The interpretation of the shadow prices and dual variables in linear programming is based on the fact that the optimal solution and the optimal value are continuous functions of the coefficients defining the constraints. Hence, small changes in the data lead to small changes in the results. MIP models have no dual variables with an interpretation comparable to that in linear programming [Geoffrion and Nauss, 1977].

Only few papers have been published on MIP sensitivity analysis. To illustrate some of the research in this area, we will outline an upper bound developed by Schrage and Wolsey (1984) for the optimal value of $MIP(b + \Delta b)$. This upper bound is exact for small variations in $b$ and requires the dual prices obtained from solving the LP relaxation of the MIP.

The branch and bound solution of a MIP constructs a tree of LP problems. Each node $i$ (node 1 is the root of the tree) of the branch and bound tree is associated with a linear program $Q^i(b)$:

$$
\begin{align*}
\max & \quad \bar{Z}'(b) = \sum_{j \in I} c_j x_j \\
\text{s.t.} & \quad \sum_{j \in I} a_{ij} x_j \leq b_i \quad \text{for } i \in I \\
& \quad x_j \geq 0 \quad \text{for } j \in N = \{1, \ldots, n\} \\
& \quad L'_j \leq x_j \leq U'_j \quad \text{for } j \in I
\end{align*}
$$

(A-26)

where:

$$
L'_j = U'_j = 0 \quad \text{for } j \in F'_o \\
L'_j = U'_j = 1 \quad \text{for } j \in F'_1 \\
L'_j = 0, \quad U'_j = 1 \quad \text{for } j \in I - F'_o - F'_1
$$

and $F'_o$, $F'_1$ are the sets of the variables in $I$ fixed to 0 and 1 respectively. Let $P^i(d)$ be the mixed-integer program obtained from $Q^i(b)$ by replacing the constraints $0 \leq x_j \leq 1$ by binary integrality constraints for $j \in I - F'_o - F'_1$.

Let $\pi'$ denote the vector of dual prices of the $|I|$ functional constraints. Let $\theta'_j$ ($j \in N$) denote the dual price associated with the $j$-th non-negativity constraint. Let $\mu'_j$ denote the dual price associated with constraints $x_j \geq L'_j$ and $x_j \leq U'_j$ with no ambiguity as at most one of these constraints can be binding for each $j$. 

326
From duality theory (cf. equation A-16) we know that $C'(d)$ defined as,

$$C'(d) = x' + \sum_{j \in F_d - F_d'} \max \left\{ 0, \mu_j \right\} + \sum_{j \in F_d'} \mu_j,$$

(A-27)

corresponds to the linear objective function of the dual problem of $Q'(b)$ for small values of $d$ which maintain the optimality of the current optimal solution. In addition, $C'(d)$ also provides an upper bound for the objective-function value of $Q'(b): \bar{Z}'(b)$.

For each terminal node with a feasible solution to $Q'(b)$ define $B'(d) = C'(d)$. For a terminal node with no feasible solution, define:

$$B'(d) = \begin{cases} -\infty : C'(d) < 0, \\ +\infty : C'(d) \geq 0. \end{cases}$$

Assume that any nonterminal node $t$ has two offspring: L(t) and R(t). For this node, define:

$$B'(d) = \min \{ C'(d), \max \{BL(t)(d), BR(t)(d) \} \}.$$ 

$Z \leq B^I(d)$: Then we can proof that $B^I(d)$ is a upper bounding function on the objective-function value of $P'(d)$ for the non-trivial case where $t$ is a nonterminal node. Let $BL(t)(d), BR(t)(d)$ denote upper bounding functions for $P^L(t)$ and $P^R(t)$ respectively. Then from duality: $Z'(d) \leq \bar{Z}'(d) \leq C'(d)$. From the branch and bound tree it follows that: $Z'(d) = \max \{ Z^L(t)(d), Z^R(t)(d) \} \leq \max \{ B^L(t)(d), B^R(t)(d) \}$. Combining both expressions yields: $Z'(d) \leq \min \{ C'(d), \max \{ BL(t)(d), BR(t)(d) \} \}$, and hence $Z'(d) \leq B^I(d)$.

In addition to $B^I(d)$ being an upper bound for the objective-function value of $MIP$, Schrage and Wolsey (1984) proved that if the original MIP has been solved to optimality and a new binary variable $x_j$ is added to the problem to give problem $\overline{MIP}$, then an optimal solution to $\overline{MIP}$ with $x_j = 0$ if $c_j \leq B^I(b) - B^I(b - a_j)$ exists. Suppose that the new variable is part of the optimal solution to the new binary program $\overline{MIP}$. This means that its activity level in the optimal solution equals 1. By fixing the value of $x_j$ to $\ell$, we can find the complete optimal solution to $\overline{MIP}$ by solving the original program with a right-hand side of $b - a_j$. From the first theorem we know that the optimal solution value to this problem is upper bounded by $B^I(b - a_j)$. To obtain the optimal solution value to $\overline{MIP}$, we must add $c_j$ to the optimal value obtained from solving $\overline{MIP}(b - a_j)$. Consequently, the optimal solution value of $\overline{MIP}$ can have a solution value no greater than $c_j + B^I(b - a_j)$. Therefore, $B^I(b)$ equals the optimal solution to $\overline{MIP}(b)$, considering solutions with
$x_j = 1$ only makes sense if $c_j + B^1(b - a_j) > B^1(b)$ which proves the second theorem of Schrage et al.
Appendix B: THE RELATIONAL AND OBJECT-ORIENTED DATA MODEL

This appendix outlines the components of the relational data model and the object-oriented data model.

B.1 The relational data model

A relational data model represents the data as a collection of relations. Each relation represents an entity type. A relation $R$ defined over $n$ — not necessarily distinct — sets $D_1, ..., D_n$ is a set of $n$-tuples $(d_1, ..., d_n)$ such that $d_i \in D_i$, ... , $d_n \in D_n$. The $D_i$ ($1 \leq i \leq n$) are the domains of $R$. It is convenient to depict a relation as a table (cf. Figure B-1). The $n$ columns of a relational table $R$ represent the attributes associated with the entity type. The number of attributes defines the degree of $R$. The rows of the table refer to the entities associated with the entity type. The ordering of both the columns and the rows is irrelevant. The number of tuples of $R$ defines its cardinality.

Figure B-1 depicts a SUPPLIES relation which may be used to keep track of deliveries of raw materials and component parts from external suppliers. There are five attributes: Supplier ID, Product ID, Delivery Date, the Quantity Received, Price (unit sales price), and Discounts (a percentage of the total value of the supplies). An attribute represents the use of a domain within a relation. The values of Supplier ID, for example, come from the domain of all valid supplier IDs. To emphasise the distinction, attributes may be given names that are distinct from those of the underlying domains. For ease of exposition, the same name will be used for both the attribute and its domain.

<table>
<thead>
<tr>
<th>SUPPLIER ID</th>
<th>PRODUCT ID</th>
<th>DELIVERY DATE</th>
<th>QUANTITY RECEIVED</th>
<th>PRICE</th>
<th>DISCOUNTS %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company A</td>
<td>X</td>
<td>10.01.1996</td>
<td>1200</td>
<td>5.50</td>
<td>5.00</td>
</tr>
<tr>
<td>Company B</td>
<td>Y</td>
<td>10.01.1996</td>
<td>5500</td>
<td>2.30</td>
<td>2.50</td>
</tr>
<tr>
<td>Company A</td>
<td>X</td>
<td>16.01.1996</td>
<td>1200</td>
<td>5.50</td>
<td>5.00</td>
</tr>
<tr>
<td>Company A</td>
<td>X</td>
<td>22.01.1996</td>
<td>-</td>
<td>5.45</td>
<td>5.00</td>
</tr>
</tbody>
</table>

Figure B-1 SUPPLIES relation.

As shown in Figure B-1, the last tuple represents a delivery d.d. January 22, 1996. Apparently, the delivery was registered before the actual quantities received were verified. The quantity attribute is therefore undefined, as indicated by the ‘-’ sign. This special value is generally referred to as the null value. The representation of the null value must be different from any other domain value.
APPENDIX B

KEYS: PRIMARY AND CANDIDATE KEYS
The key of a relation is the minimum non-empty subset of its attributes such that the values of the attributes comprising the key uniquely identify each tuple of the relation. The key of SUPPLIES consists of the attributes Supplier ID, Product ID, and Delivery Date (depicted with underlines). Each relation has at least one key. Sometimes, there may be more than one option for the key. In such cases, each alternative is considered a candidate key, and one of the candidate keys is chosen as the primary key.

FOREIGN KEYS (REFERENCES)
An association between two relations $R$ and $R'$ is 'modelled' by including the primary key of $R$ as part of the key or as an ordinary attribute of $R'$ — a so-called 'foreign' key (cf. Figure B-2). The relational scheme does not store references to tuples of $R$ explicitly. What it does support is the use of shared domains: the same set of permissible attribute values is used for the primary key of $R$ and for the foreign-key attributes of relation $R'$. Referential integrity constraints (cf. section B.1.2), in addition to the relational scheme, must ensure that the foreign key of a tuple $R'(j)$ of $R'$ actually corresponds to an existing tuple $R(i)$ in relation $R$.

<table>
<thead>
<tr>
<th>key</th>
<th>data attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$D_1,...,D_k$</td>
</tr>
<tr>
<td></td>
<td>$D_{k+1},...,D_n$</td>
</tr>
<tr>
<td>$R(i)$</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>foreign key</th>
<th>data attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R'$</td>
<td>$D'_1,...,D'_j$</td>
<td>$D_{j+1},...,D_k$</td>
</tr>
<tr>
<td></td>
<td>$D'_{j+1},...,D'_m$</td>
<td></td>
</tr>
<tr>
<td>$R'(j)$</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Figure B-2 Modelling references with foreign keys.

B.1.1 Normalisation
The following four problems might exist in a relational database scheme [Özsu and Valduriez, 1991]:

1. REPETITION. Unnecessary repetition of data. In consequence, storage is wasted and it becomes more difficult to maintain database consistency;

2. UPDATING. Performing updates may be troublesome due to the repetition of data;

3. INSERTION. It may not be possible to add new information to the database because insufficient information is available to create the key of a new tuple;

4. DELETION. It may not be possible to delete a piece of data from the database without deleting information that should remain in the database.
Normalisation is a step-by-step reversible process of resolving inconsistencies before implementing the data model. This entails replacing a given set of relations by relations which have a progressively simpler and more regular structure and which do not have the problems listed above. Normalisation theory is built around the concept of normal forms. Normal forms are based on three types of dependency structures [Özsu et al., 1991]:

1. **FD: FUNCTIONAL DEPENDENCY.** Given a relation $R$ defined over the set of attributes $A = \{A_1, \ldots, A_n\}$ and let $X \subseteq A$, $Y \subseteq A$ denote subsets (not necessarily distinct) of the attributes of $R$. If, at any instant in time, each value of $X$ has exactly one value of $Y$ associated with it, then "$X$ functionally determines $Y$" or "$Y$ is functionally dependent on $X$". Notationally, this is shown as $X \rightarrow Y$. The key of a relation functionally determines the nonkey attributes of the same relation.

   $Y$ is fully functionally dependent on $X$ if it depends on the association of attributes in $X$ and not on distinct attributes. $Y$ is partially dependent on $X$ if it is functionally dependent on a proper subset of $X$. Finally, let also $Z \subseteq A$, then $Z$ is transitively dependent on $X$ iff. $X \rightarrow Y \rightarrow Z$;

2. **MVD: MULTI-VALUED DEPENDENCY.** Let $R$ be a relation defined over the set of attributes $A = \{A_1, \ldots, A_n\}$ and let $X \subseteq A$, $Y \subseteq A$, $Z \subseteq A$. If the set of $Z$-values matching a given $(X,Y)$-pair in $R$ depends only on the $X$-value, then "$X$ multi-determines $Z$" or "$Z$ is multi-dependent on $X$". Notationally, this is shown as $X \rightarrow Z$. Note that every FD is also an MVD, but the reverse is not necessarily true;

3. **JD: JOIN DEPENDENCY.** Let $R$ be a relation defined over the set of attributes $A = \{A_1, \ldots, A_n\}$ and let $X \subseteq A$, $Y \subseteq A$, $Z \subseteq A$. Then, if $R$ is equal to the join of $X$, $Y$, and $Z$, $(X,Y,Z)$ constitutes a join dependency for $R$.

**NORMAL FORMS**

The most popular approach to normalising a relational database scheme is the decomposition approach where, starting with a relation which contains all attributes, at each iteration a relation is split into two or more relations of a higher normal form. During this decomposition process, the replacement of a relation should not result in loss of information.

- **1NF**: The attributes of relations in 1NF contain atomic, non-decomposable values only, never a set of values;

- **2NF**: A 1NF relation is in second normal form if every non-key attribute is fully functionally dependent on the key of the relation.
APPENDIX B

To illustrate 2NF, consider the MAKES relation depicted in Figure B-3. It records two types of information: (1) the set of products being manufactured by the various plants; and (2) to which cluster of products, i.e. product type, the products belong. The MAKES relation is not in 2NF due to the partial functional dependency PRODUCT→PRODUCT-TP.

| MAKE | RELATION | | | |
|------|----------|------------------|
| PLANT| PRODUCT  | PRODUCT-TP       |

*Figure B-3* MAKES relation not in 2NF.

- 3NF: A 2NF relation $R$ is in third normal form if every non-key attribute of $R$ is non-transitively dependent on the key of $R$.

To illustrate 3NF, consider the SELLS relation shown in Figure B-4. This table records two types of information: (1) which plants export which products to which countries; and (2) under which brand labels are these products sold. Evidently, both types of information should be stored separately in distinct tables. The SELLS relation is not in 2NF: the BRAND attribute depends only partially on the key attributes (depicted with underlines): $(\text{COUNTRY},\text{PRODUCT}) \rightarrow \text{BRAND}$. The two tables that result from a decomposition of the SELLS relation depicted in Figure B-5 are in 3NF.

| SELLS | RELATION | | | |
|-------|----------|------------------|
| PLANT | COUNTRY  | PRODUCT          | BRAND |

*Figure B-4* Relation in first normal form.

<table>
<thead>
<tr>
<th>EXPORTS RELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLANT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BRANDS RELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
</tr>
</tbody>
</table>

*Figure B-5* Relations in third normal form.

- BCNF: Boyce-Codd Normal Form. BCNF is a stronger form of 3NF. A relation in 3NF is in BCNF if every attribute, or group of attributes, on which some other attribute is fully functionally dependent is a candidate key.

To illustrate BCNF, consider the MAKES relation (in 3NF) depicted in Figure B-6. This relation is not in BCNF since we have the FD PRODUCT→PRODUCT-TP while PRODUCT is not a candidate key. A nonloss decomposition of the relation into MAKES(PLANT,PRODUCT) and PRODUCT-TYPES (PRODUCT-TP,PRODUCT) respectively yields two relations in BCNF.

| MAKES | RELATION | | | |
|-------|----------|------------------|
| PLANT | PRODUCT-TP | PRODUCT |

*Figure B-6* MAKES relation (3NF) not in BCNF.
4NF: A relation $R$ in BCNF is in fourth normal form if all multi-valued dependencies $X \rightarrow \rightarrow Y$ in $R$, are also functional dependencies.

BCNF does not protect against redundancies caused by multiple MVDs. Consider the Sells relation shown in Figure B-7 which records which companies sell which products in which countries. Suppose that plants sell all their products in every country to which they export (Plant $\rightarrow \rightarrow$ Product and Plant $\rightarrow \rightarrow$ Country), then it is clear that this relation contains a great deal of redundancy (by simply enumerating all possible options). This redundancy would be eliminated by a decomposition of the sells relation into two relations: Makes(Plant, Product) and Exports(Plant, Country).

<table>
<thead>
<tr>
<th>Sells Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plant</td>
</tr>
</tbody>
</table>

Figure B-7

- 5NF: A relation $R$ in 4NF is in fifth normal form if every join-dependency defined for the relation is implied by the candidate keys of $R$: i.e., the subsets of attributes (or projections) should all contain the same candidate key.

There are algorithms that provide a lossless decomposition of an 1NF relation into a 3NF relation. Relations in 5NF cannot be further decomposed without loss of information. For practical reasons, full normalisation may not always be desirable. For example, addresses and postcodes are almost always used together as a unit while the partial functional dependencies Postcode $\rightarrow \rightarrow$ Street and Postcode $\rightarrow \rightarrow$ City imply that such relations are not in 3NF.

### B.1.2 Integrity rules

Integrity rules are assertions about the logical structure of a valid database. Integrity constraints thus define consistent states of the database. Integrity rules can be divided into three categories:

1. **Domain Integrity Rules**, which are concerned with maintaining the correctness of attribute values within relations. The entity integrity rule, which is necessary to enforce the uniqueness of keys and which dictates that each attribute of the key is non null, falls into this category;

2. **Intra-Relational Integrity Rules**, which focus on maintaining the correctness of relationships among the attributes of a relation. One of the most important of these is key uniqueness;
APPENDIX B

3. REFERENTIAL INTEGRITY RULES, which are concerned with maintaining the correctness and consistency of relationships between relationships which the relational model cannot represent. Referential integrity may be defined as follows: Let $R$ be a relation with an attribute, or group of attributes, $A$ which forms the primary key of another relation $R'$ (foreign key). Then, at any given time, $A$ must either be equal to the primary key value of some tuple in $R'$, or it must be null, but in that case $A$ cannot be part of the key of $R$.

B.1.3 Relational algebra

Relational algebra is a procedural data manipulation language: the user specifies how the result of a query is to be obtained. Relational algebra is derived from set theory. In its basic form, it consists of five fundamental operators and five derived operators that can be defined in terms of these. The operations operate on entire relations and do not depend on the order of the attributes or the tuples. The fundamental operators are:

1. **SELECTION.** Selection produces a subset of tuples of a given relation, with the same degree as the degree of the original relation, which all satisfy a formula (condition). The formula is made up of a sequence of terms, linked by logical connectors, of the form $A \hat{o} c$, where $A$ is an attribute of $R$, $c$ a value from the associated domain, and $\hat{o}$ an (in)equality sign;

2. **PROJECTION.** Projection produces a subset of tuples of a given relation, with a degree less than or equal to the degree of the original relation, where the result relation contains only those attributes of the original relation over which the projection is performed. The result of a projection might contain identical tuples. It is possible to specify projection with or without the elimination of duplicate tuples;

3. **UNION.** The union of two relations $R$ and $S (R \cup S)$ is the set of all tuples in $R$, or in $S$, or in both. $R$ and $S$ should be *union compatible*. Union may be used to insert new tuple in an existing relation;

4. **SET DIFFERENCE.** The set difference of two relations $R$ and $S (R - S)$ is the set of all tuples in $R$ but not in $S$. $R$ and $S$ should be *union compatible*. The operation is asymmetric. This operation allows the deletion of tuples from a relation;

5. **Cartesian Product.** The Cartesian product of two relations $R$ of degree $k_1$ and $S$ of degree $k_2$, denoted as $R \times S$, is the set of $k_1 + k_2$ tuples, where each result tuple is a concatenation of one tuple of $R$ with one tuple of $S$, for all tuples of $R$ and $S$. In case $R$ and $S$ have attributes with the same name, the attribute names are prefixed with the relation name so as to maintain uniqueness of attribute names.

---

1 Two relations $R$ and $S$ are union compatible iff. they are of the same degree $n$ and if the $i$-th attribute (1st$n$) of each relation is defined over the same domain.
The five derived operators that can be defined in terms of the fundamental operators are:

1. **INTERSECTION.** The intersection of two relations \( R \) and \( S \) \( (R \cap S) \) consists of the set of all tuples in both \( R \) and \( S \). In terms of the basic operators: \( R \cap S = R - (R - S) \);

2. **\( \theta \)-JOIN.** Join is a derivative of Cartesian product. The \( \theta \)-join, commonly called the join, of two relations is equivalent to performing a selection over the Cartesian product of the two operand relations. The selection formula is made up of a sequence of terms, linked by logical connectors, of the form \( A \hat{\circ} B \), where \( A \) is an attribute of \( R \), \( B \) an attribute of \( S \), and \( \hat{\circ} \) an (in)equality sign;

3. **NATURAL JOIN.** A natural join is a special case of the \( \theta \)-join of two relations over attributes with the same domain where the selection formula contains an equality sign (a so-called *equi-join*);

4. **SEMI-JOIN.** The semi-join of relation \( R \), defined over the set of attributes \( A \), by relation \( S \), defined over the set of attributes \( B \), is the subset of the tuples of \( R \) that participate in the join of \( R \) with \( S \).

5. **DIVISION.** The division of relation \( R \) of degree \( r \) with relation \( S \) of degree \( s \) \( (r > s \text{, } s \neq 0) \) is the set of \((r - s)\)-tuples \( t \) such that for all \( s \)-tuples \( u \) in \( S \), the tuple \( tu \) is in \( R \).

**B.1.4 Relational calculus**

Relational calculus is non-procedural (declarative): the user specifies the relationships that should hold in the result of a query, not how to obtain the result. This section briefly discusses *tuple* relational calculus for it serves as the basis of SQL (Structured Query Language). SQL uses English key words in place of the quantifiers and logical connectors.

**TUPLE RELATIONAL CALCULUS.** The primitive variable used in tuple relational calculus is a tuple variable which specifies a tuple of a relation. In tuple relational calculus queries are specified as \( \{ t | F(t) \} \) where \( t \) is a tuple variable and \( F \) a so-called 'well-formed' formula containing atomic formulas of two forms:

1. Tuple-variable membership expressions usually specified as \( R.t \) or \( R(t) \) meaning to say "tuple \( t \) belongs to relation \( R \);

2. Conditions which can be defined as \( s[A] \hat{\circ} t[B] \) where \( s \) and \( t \) are tuple variables associated with relations \( R \) and \( S \), \( s[A] \) and \( s[B] \) are elements of \( s \) and \( t \), respectively, which correspond to attributes \( A \) and \( B \) of relations \( R \) and \( S \),
respectively, and \( \Diamond \) an (in)equality sign. Alternatively, conditions can be defined as
\[
s(A) \Diamond c,
\]
where \( s, A, \) and \( \Diamond \) as defined above and \( c \) is a constant.

### B.2 The object-oriented data model

To facilitate the transition from the functional requirements of a software tool that must
be built to the design of an implementation and finally to the implementation of the
software, the product(s) of each stage should be used as a starting point for subsequent
stages. Therefore, much research is focused on the design and integration of system-
analysis techniques, software-design techniques, and programming languages which
are founded on the same conceptual building blocks.

Conventional system analysis techniques consider the functionality of a system to be
separate from the data requirements of the system. Processes are introduced that
represent transformations of input data to output data and by the principle of stepwise
refinement, processes are successively decomposed into simpler sub-processes. This
approach groups functions that may operate on completely different data stores. The
most important disadvantage of these conventional software design methods and
programming languages like, e.g., C and Pascal which follow essentially the same
approach, is that the computer implementations that emerged from this approach lack a
grouping of logically related procedures and data. This approach significantly increases
the complexity of the software, especially when programs get large.

Breaking down an application area into objects and relationships among these objects
is another common technique in systems analysis [Hughes,1991,p79]. In an object-
oriented perspective, the clustering of tasks is centred on the underlying data
abstractions. That is, every function must be associated with a particular object and
functions are grouped if they operate on the same data abstraction. The advantage of
this approach is that the objects provide an intuitive means to group the tasks and the
data that play a role in the system.

**Object-oriented design** methods focus on mapping the tasks of a computer
implementation onto sequences of operations, each of which is uniquely identified with
a specific cluster of objects. Functions which are derived from a higher level process
may reside in different objects and a sequence of messages (‘function calls’) between
the objects is necessary to perform a higher level process. Hence an object has a set of
operations, representing the allowable operations for the entity that it describes, and a
set of data attributes whose values define the state of the object. An object can be
RELATIONAL AND OBJECT-ORIENTED DBMS

manipulated like a data structure but it defines its own manipulation protocol by means of its operations. By invoking an operation on an object, its state may change.

It is desirable to be able to classify objects which have the same structural and behavioural pattern. Instead of describing individual objects, the object-oriented approach concentrates on properties and operations that are common to an entire class of objects. This so-called class definition, which is essentially an object-type definition, is the natural unit of abstraction in object-oriented systems. An arbitrary number of instances (i.e. objects) of a given class may co-exist.

Class definitions provide a good facility for modular decomposition. In object-oriented systems, the concept of inheritance permits objects to be organised in layered structures (single inheritance structures can be represented as hierarchies or trees, multiple inheritance structures can be represented as networks) in which specialised objects inherit the properties and functions of more generalised objects. Similar classes of objects can be modelled by specifying a base class, which defines the common part, and then deriving specialised sub-classes from this base class. This feature provides powerful support for reusability and extensibility since new class definitions can be based on existing class definitions.

OBJECT-ORIENTED PROGRAMMING is a method of implementation in which programs are organised as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of an hierarchy of classes united via inheritance relationships [Booch, 1991, p36]. A class in an object-oriented programming language thus defines a structure, i.e. the set of properties, and a set of operations which are common to an entire class of objects. Object-oriented programming languages have a rich set of features for creating data types and representing the relationships among data which are not supported in conventional relational database systems.

AN OBJECT-ORIENTED DATABASE is a database which fully supports the object-oriented model. That is, in an object-oriented database every object is an instance of a class (type) definition. Unlike relational databases, the properties of an object are not restricted to non-decomposable data types and may in fact be complex objects themselves.

B.2.1 Object-oriented abstractions

Five abstraction concepts form the fundamental basis of the object-oriented (data) model:
APPENDIX B

1. CLASSIFICATION. Objects of the same type are collectively described by a class definition. Instatiation is the inverse of classification and concerns the creation of distinct objects of a class;

2. IDENTIFICATION. Booch (1991,p84) defines the identity of an object as: "(...) that property of an object which distinguishes it from all other objects." Identification concerns the process whereby both abstract concepts (i.e. classes) and concrete objects (i.e. instances) are uniquely identifiable. The identity of an object should be immutable despite changes to the properties of the object. An object's identity may not therefore be based on the values of certain properties of the object. Object-oriented programming languages use unique variable names to identify classes. The identity of the objects is often provided by the object's memory address. The problem with using memory addresses is, however, that persistent objects which are stored in a database continue to exist even when the application is terminated and the object has been removed from memory. Many object-oriented database systems therefore assign unique labels to the objects. These labels are stored together with the object's properties and hidden from the user or application program;

3. AGGREGATION. Aggregation is an abstraction in which a relationship among objects is represented by a higher level, aggregate object (type). Aggregation provides a convenient mechanism for modelling so-called "is-part-of" relationships among objects. Aggregation differs from classification. Classification concerns the aggregation of attributes to form an entire object whereas aggregation deals with the creation of composite objects;

4. GENERALISATION. Generalisation is a type of abstraction in which a cluster of properties and functions that is shared among rather similar object types is used to form a new generic class definition. The new generic object type is explicitly named and may be used as a property in other class definitions allowing the modeller to specify relationships in which generic objects participate. The differences among the set of derived objects are reflected in different object-type specifications. Each of the derived classes defines a specialisation of the more generic type. Objects of derived types inherit the properties of the generic class definition;

5. ASSOCIATION. Association captures relationships among object types where, in order to perform their tasks, objects of class A use certain operations of objects of class B. Hence, association abstractions capture relationships other than "is-part-of" (aggregation) and "is-a" relationships (generalisation).
B.2.2 Integrity control in object-oriented databases

An object-oriented database is a collection of persistent objects, each identified by a unique identifier which may be visualised as a pointer to a persistent object. Persistent objects must be allocated in persistent store and they exist beyond the lifetime of the program that created them. If persistent capabilities are added to an object-oriented programming language, as is the case for POET[POET Software, 1993] and C++, the programming’s language typing mechanisms can help to guarantee, to a certain extent, database integrity and consistency.

TYPING. Typing is the enforcement of the type of an object, such that objects of different types may either not be interchanged or may be interchanged in only very restricted ways [Booch, 1991]. A strongly typed language offers the ability to determine the type compatibility of expressions at compile time. Strong typing differs from the concept of static and dynamic typing. Static and dynamic typing refer to the time when names are bound to types. Static binding means that the types of all variables and expressions are fixed at the time of compilation. Dynamic binding means that the types of the variables and expressions are not known until runtime. C++ is strongly typed and supports dynamic binding.

REFERENTIAL INTEGRITY. All objects have a unique identity and other objects can refer to that identity. Referential integrity constraints, within the context of an object-oriented database, enforce that objects cannot have references to objects that do not exist in the database.

APPLICATION-SPECIFIC CONSTRAINTS. Many object-oriented databases provide the capability to define application-specific constraints as well as triggers which specify the action to be taken when some condition becomes true. In case of POET, constraints and triggers can be defined as operations, using plain C++, which are checked whenever a persistent object is about to be added to the database, updated, or about to be deleted from the database. Invocation of application-specific operations is not automatic. It is the designer’s responsibility to add the appropriate invocations to the standard POET storage and deletion operations defined for the class.

B.2.3 Operations

DATA MANIPULATION. Data manipulation in an object-oriented database system is accomplished by means of the operations which are part of the class definitions of the objects stored in the database. Some object-oriented databases provide on-line support for ad-hoc queries (cf. section B.2.4).
SCHEMA MANIPULATION. According to Booch (1991), in practice, the following changes to the design are to be expected during the evolution of the database and its associated application(s): adding a new class, changing the implementation of a class, changing the properties of a class, reorganising the class structure, and changing the interface (the operations) of a class. Changing the implementation of a class or its properties is generally not costly for the interface of the class remains unchanged. Apart from making the changes, one only needs to recompile the implementation of the class. The costs of other changes is highly situation-dependent.

B.2.4 Object-oriented query processing

In an object-oriented database environment, despite the effort that is put into the design of the operations on the objects, there is a need for a comprehensive, high-level query language for performing 'ad-hoc' or 'one-time' queries which is both flexible and easy to use and which adheres to the fundamental principles of the object-oriented model [Hughes, 1991]. It is difficult to anticipate all the information requirements that apply to a specific database system and some users simply do not want to interact with the database through a programming language interface. Many object-oriented database systems therefore provide a high-level query language interface, mostly based on SQL.

OBJECT SQL

An example of an object-oriented query language is Object SQL (OSQL) which is part of ONTOS, formerly known as Vbase. OSQL employs the familiar SQL syntax. Given a relation \( R \) defined over the set of attributes \( A = \{ A_1, \ldots, A_n \} \), let \( A^e \subseteq A \), \( A^c \subseteq A \) denote subsets (not necessarily distinct) of the attributes of \( R \). Then, a basic SQL (SELECT ... FROM ... WHERE) statement that produces a projection of \( R \) over \( A^o \) of a selected range of tuples which satisfy certain conditions on the values of \( A^e \) would have the following syntax:

\[
\text{SELECT } A^o_{i_1}, \ldots, A^o_{i_k} \text{ FROM } R \text{ WHERE } A^e_{j_1} \odot c_{j_1}, \ldots, A^e_{j_l} \odot c_{j_l}
\]

where \( \odot \) denotes an (in)equality sign and \( c_i \) a constant value drawn from the appropriate domain. The Object SQL syntax resembles the SQL syntax except that the FROM clause uses the identities of the objects and that the SELECT clause deals with object properties. The main differences between OSQL and relational SQL are [Hughes, 1991]:

- Objects are referenced through their identity rather than via key values. Variables may be bound to objects on creation and retrieval and may then be used to refer to objects in subsequent statements;
- Operations may appear in the SELECT and WHERE clauses.
Simple OSQL (SELECT...FROM...WHERE) statements use the familiar ‘dot’ notation to access information (\( p.p \) denote properties of the object, \( c \) an appropriate constant):

\[
\text{SELECT } object.p \text{ FROM } object \text{ IN class WHERE } object.p \bowtie c
\]

It is difficult to design an object-oriented query language that provides full support to the data that is stored in an object database because the properties of an object may be complex objects themselves. As yet, there is no well-defined, widely accepted query language for object-oriented database systems which fulfils the general purpose role that SQL plays in the relational environment [Hughes, 1991, p153].
CURRICULUM VITAE

Eric Kraan was born in Gouda, the Netherlands, on May 24, 1968. After graduating in 1986 from pre-university education (‘Atheneum’) at the Samenwerkingsschool voor Havo-Atheneum in Waddinxveen, he studied Mathematics at the Delft University of Technology, from which he graduated in 1991 in the Statistics, Stochastics and Operational Research group on a study on the application of the concept of Social Accounting Matrices to business accounting. In September 1992, he became a doctorate student at the Delft University of Technology, Department of Mechanical Engineering and Marine Technology, Transportation Technology group (Logistics Engineering) where he has been working towards his PhD thesis. During 1993 he gave a course of lectures, including practical work, on modelling and simulation of logistics systems at the Hogeschool van Amsterdam, Opleiding Logistiek. As of March 1996, he works as a consultant at Andersen Consulting, the Netherlands.